TEC-IT

WWW.TEC-IT.COM

TConnector

Data Acquisition ActiveX Control

Version 2.4

User Documentation

17 September 2013

TEC-IT Datenverarbeitung GmbH
Wagnerstrasse 6
A-4400 Steyr, Austria

t ++43 (0)7252 72720
f ++43 (0)7252 72720 77
office@tec-it.com
www.tec-it.com

1 Content

1.1 Table of Figures

1.2 List of Tables

2 Disclaimer

The actual version of this product (document) is available as is. TEC-IT declines all warranties which goes beyond applicable rights. The licensee (or reader) bears all risks that might take place during the use of the system (the documentation). TEC-IT and its contractual partners cannot be penalized for direct and indirect damages or losses (this includes non-restrictive, damages through loss of revenues, constriction in the exercise of business, loss of business information or any kind of commercial loss), which is caused by use or inability to use the product (documentation), although the possibility of such damage was pointed out by TEC-IT.

We reserve all rights to this document and the information contained therein. Reproduction, use or disclosure to third parties without express authority is strictly forbidden.

Für dieses Dokument und den darin dargestellten Gegenstand behalten wir uns alle Rechte vor. Vervielfältigung, Bekanntgabe an Dritte oder Verwendung außerhalb des vereinbarten Zweckes sind nicht gestattet.

© 1998-2013
TEC-IT Datenverarbeitung GmbH
Wagnerstr. 6

A-4400 Austria
t.: +43 (0)7252 72720
f.: +43 (0)7252 72720 77
http://www.tec-it.com

3 Introduction

3.1 About TConnector

TConnector is a software tool, which allows communication with various external devices with a very slim and unified programming interface.

TConnector conforms to Microsoft® ActiveX® specifications and can therefore be used in many standard applications (like Microsoft® Excel®, Microsoft Access, Microsoft Word, …) and in most Windows® programming environments (like Microsoft Visual Basic®, Microsoft Visual Studio® (C/C++/C#), .NET, C#, Delphi, …).

TConnector supports serial interfaces, parallel ports and even TCP/IP connections. It can also be used to read or write from simple files.

TConnector is controlled by a set of properties which are used to adjust basic interface parameters (like connection speed or IP-address). In addition a few programming methods are used to connect to the device and read or write data.

A unique feature of *TConnector* is the unified programming interface. In other words: *TConnector* offers the user always the same program methods– regardless of the physical interface used at the moment.

3.2 Supported Interfaces

- Null device
- Serial communication (COM)
- Parallel communication (LPT) – limited read functionality
- File-Input/Output
- TCP/IP (Client)

3.3 System Requirements

TConnector may be used with the following operating systems:

- Windows 98
- Windows ME
- Windows NT (version 4.x)
- Windows 2000
- Windows 2003
- Windows XP
- Windows Vista

► Please note: Windows 95 is not supported.

3.4 Version History

The product history containing version info, evolution in functionality and changes in the COM interface can be accessed at our web site http://www.tec-it.com (Software ► Data Acquisition ► TConnector ► More Info ► Version History).

4 Installation

Execute following steps to install TConnector to your PC:

1. Download the demo version from the TEC-IT Web Site www.tec-it.com
2. If you downloaded a ZIP file, extract the zipped files into an arbitrary directory
3. Execute the setup application and follow the instructions
4. *TConnector* can now be used by any application that supports Microsoft ActiveX technology

5 Using TConnector

TConnector is an ActiveX object and can be inserted in various applications supporting the ActiveX technology. *TConnector* is implemented as invisible ActiveX control – meaning that no window or visual effects are displayed during runtime.

Calling the `Open` method starts a communication. `Open` has to be called after the properties are set but before any other IO-operation is executed. To close an active connection call `Close`. Each connection opened with `Open` must be closed with `Close`.

You can communicate with the device in synchronous mode (`Read, Write…`) or in asynchronous mode (`StartListen, StopListen, OnData, OnStatusChange…`).

- **Synchronous** means that a function call returns only after data was read (written) or a timeout occurred.
- **Asynchronous** means that the called function returns immediately. Whenever data was received an event is fired. An event handler provided by the user processes the event (and the data).

5.1 Synchronous mode

After `Open` has been called, data can be written and received using the `Write` and `Read` methods.

When `Write` or `Read` is called, the application waits until data was written (read) or a timeout occurred and the method returns.

5.1.1 Visual Basic Example

A typical sequence looks like this:

```
Dim Data As String          'string for data
Dim Bytes As Long           'number of bytes to receive or to send
Dim Timeout As Long         'timeout for the read/write method

' Open connection

Connector2.Open

' Receive data

Bytes = 100                 'receive 100 bytes
Timeout = 10000             'in 10 sec
Data = Connector2.Read (Bytes, Timeout)

' Data contains the received data
' Send data

NoBytesToWrite = 100         'send 100 bytes
Timeout = 10000              'in 10 sec
Data = "Hello World!"        'Text to send
NoBytesWritten = Connector2.Write (NoBytesToWrite, Timeout, Data)

' Close connection

Connector2.Close
```

► "Connector2" is the name of the TConnector2 object in this sample code.

5.2 Asynchronous Mode

After `Open` has been called, you can activate the Asyncronous Mode with `StartListen`. The method `StartListen` instructs TConnector to perform continuous reads in the background. `StartListen` returns immediately.

When data is received from the device, the application is informed by an `OnData` event. If status lines are changing, the `OnStatusChange` event is fired. `StopListen` ends the asynchronous mode.

With this mode you don't have to perform multiple reads to check always for new data – TConnector does this for you.

> ► You can call the `Write` method to send data. But you can't use the `Read` method, which is reserved for the Synchronous Mode.

5.2.1 Example in Visual Basic

A typical sequence looks like this:

```
' Open connection

Connector2.Open

' Start listen mode

Connector2.StartListen

' From now on OnData events will be sent if data is received!
' During listen mode also OnError and OnStatusChanged events can be fired.

...

' Stop listen mode

Connector2.StopListen

' Close connection

Connector2.Close
```

Sample of a simple event handler:

```
'This method is called if data is received in listen mode

Private Sub Connector2_OnData (ByVal Data As String)
      MsgBox "OnData event occurred, received data:" + Data
End Sub
```

5.3 Simulating Keystrokes

In Asynchronous Mode *TConnector* is able to translate incoming data to keystrokes. That means that the received data is directly sent to the currently active window, so as if the data was entered by the user on the keyboard.

> ► To enable this feature you have to select the `Send Keystrokes` check box in the `Transmission` property page.

In Synchronous mode you could call the `EmulateKeys()` method after data was received with `Read()`.

5.3.1 Application Scenario

Some data should be read by a bar code scanner. The scanners are connected via the serial port. It is not desired to integrate the communication process directly into your applications. So just plug in *TConnector* ActiveX into a small program, that does nothing but communicate with the serial scanners. The received data is automatically transmitted to the active window – without changing your application.

You can insert special actions after each `OnData` event by customizing the prefix and postfix property - e.g. you can simulate pressing `F10`, `Tab` or `Return` keys (and many more).

If supported by the used scanner, you can also configure it to send <`CR`> or <`tab`> at the end of a data stream. Doing this you can instruct the system to automatically jump from one edit field to the next without additionally programming.

If you enable the keystroke feature, all events in the asynchronous mode (`OnData`, `OnError`, `OnStatusChange`) are **not** disabled. They can be handled as usual.

► You may also give another product of TEC-IT a try. It is called *TWedge* and performs the above mentioned functionality as ready to run application. Download a demo of TWedge from www.tec-it.com

6 Property Pages

6.1 Introduction

TConnector is a full featured ActiveX control. It provides so-called “property pages”. These property pages offer the possibility to change all *TConnector* related options without programming.

In most applications the property pages can be accessed with a right mouse-click directly onto the object. After right clicking, the appearing menu [*TConnector*-Object] - [Properties] offers access to the settings of the object.

The property pages are described in the following sections.

> ► Beside the property pages you can use the menu option "Properties" in some Microsoft Office applications - in this window you have a list of “property - value” pairs, which allows you to change the characteristics of the control.

6.2 Property Page Connection

In the `Connection` page you can set all properties necessary to open a connection via the specified interface. The properties you have to enter depend on the selected device type.

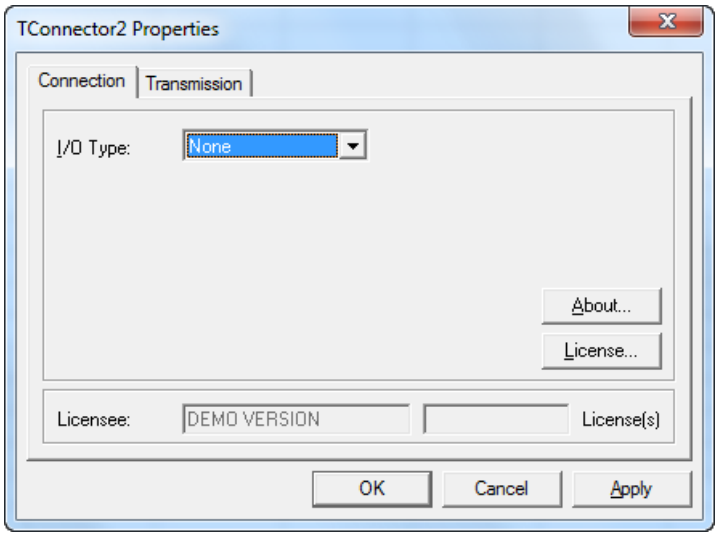

Fig. 1: Property Page Connection

6.2.1 IOType

This setting adjusts the type of the used interface.

Interface Type	Description
None	no connection specified
NULL	null device
File	reading from and writing to files
Serial	Serial device (COM1…)
Parallel	Parallel device (LPT1…)
TCP	TCP/IP connection
Bluetooth	Bluetooth connection

Table 1: Supported Interface Types

Depending on the selected type, the appearance of the property dialog changes.

6.2.2 Button About

Pressing this button opens the About-dialog where the version information is displayed.

6.2.3 Button license

This button opens the License dialog. Enter your license data in this dialog to unlock the demo version and to remove any demo-restrictions.

6.3 Property Page Connection for IOType None

`IOType` is set to None. In this mode no communication is possible.

6.4 Property Page Connection for IOType NULL

`IOType` is set to NULL (null device). In this mode the control connects to a dummy device, which has no functionality. The `NULL` mode is usually used for testing only.

6.5 Property Page Connection for IOType File

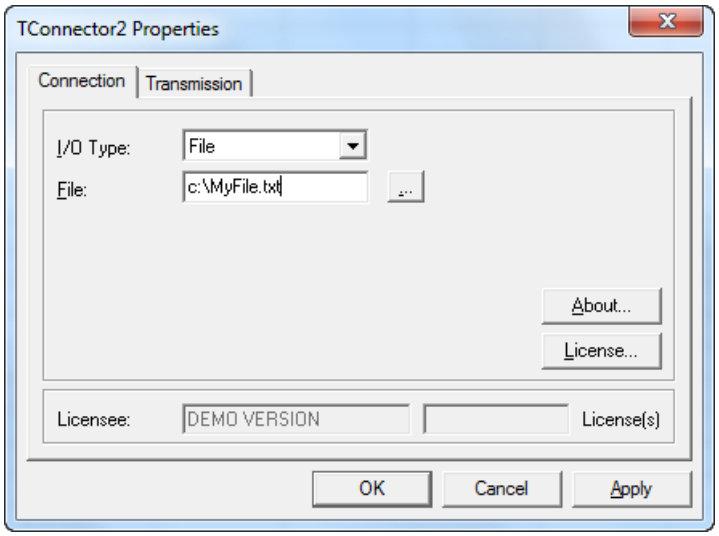

Fig. 2: IO-Type File

`IOType` is set to File – file input/output. In this mode TConnector connects to a file that can be read or written to.

6.5.1 File

Specifies the full filename including the path; with ... you can browse for an existing file.

6.6 Property Page Connection for IOType Serial

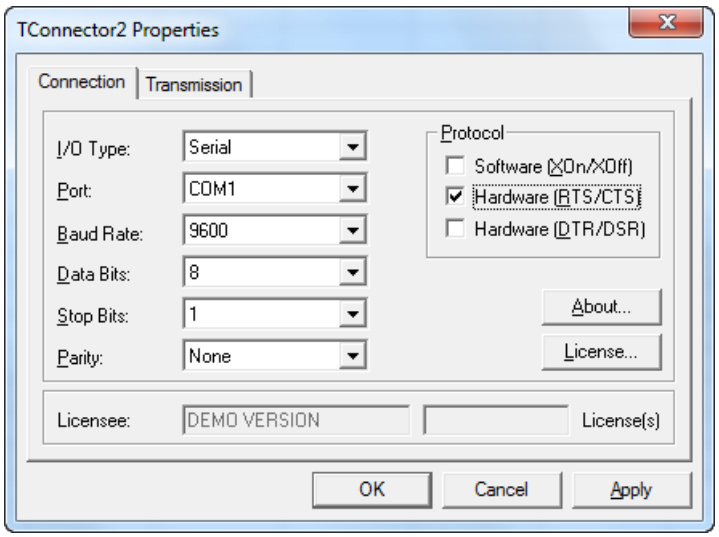

Fig. 3: IO-Type Serial

`IOType` is set to Serial – serial communication. In this mode TConnector is able to communicate via a serial port ("COM Port"). You can specify usually used settings for this kind of communication (port, baud rate…) including support for software and hardware handshake.

6.6.1 Port

Specifies the name of the serial port (COM1, COM2…). Port names not available in the list box can be entered directly – see How can I specify a COM port > COM9?

Baud rate

Specifies the communication speed used for this serial line (in bits per second). It is important to choose exactly the same value as adjusted in your external device (110, …, 9600,…, 115200, …, 256000).

6.6.2 Data Bits

Specifies the number of bits used for one data-word. It is important to choose exactly the same value as adjusted in your external device.

6.6.3 Stop Bits

Specifies the number of bits used for marking the end of a data-word. It is important to choose exactly the same value as adjusted in your external device.

6.6.4 Parity

Specifies if a parity bit is used and which type of parity (even/odd). It is important to choose exactly the same value as adjusted in your external device.

6.6.5 Protocol

The handshake modes listed below can be configured. Please not that it is possible to combine multiple handshake methods. You should select the same handshake protocols as used in your external device.

Protocol	Description
XOn/XOff	use XOn/XOff software handshake
RTS/CTS	use RTS/CTS hardware handshake
DTR/DSR	use DTR/DSR hardware handshake

Table 2: Supported Handshake Protocols

6.7 Property Page Connection for IOType Parallel

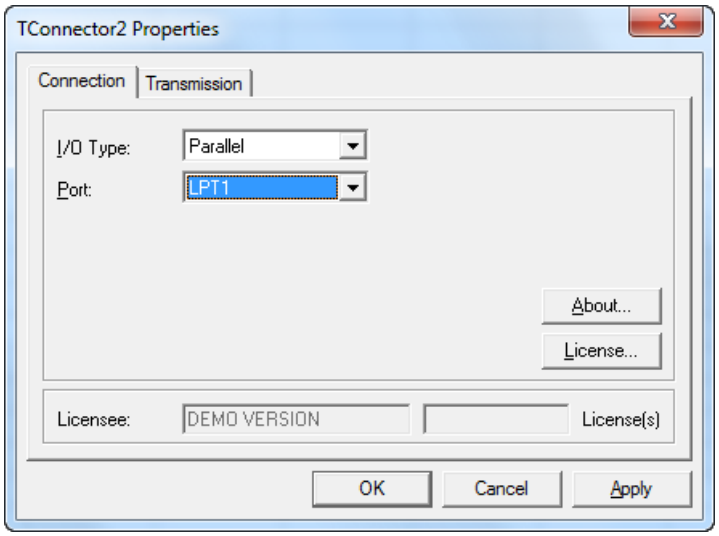

Fig. 4: IO-Type Parallel

`IOType` is set to Parallel – communication over a parallel port. In this mode TConnector opens a connection to a parallel port, usually for writing (for example to a printer).

► Please note: Reading from a parallel port is not fully supported (depends on Bios, Windows Driver, communication chip set and the connected devicce).

6.7.1 Port

Specifies the name of the parallel port (LPT1, LPT2…). Port names not available in the list box can be entered directly – see How can I specify a COM port > COM9?

6.8 Property Page Connection for IOType TCP

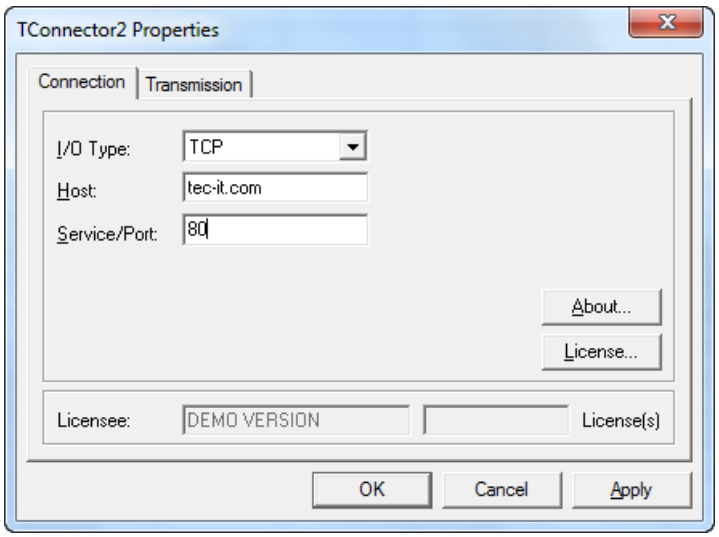

Fig. 5: IO-Type TCP

`IOType` is set to TCP – communication over TCP/IP. In this mode TConnector opens a connection to an IP port. The user can specify host name and service name (or port number).

- TConnector can act as a "client", who connects to a server – i.e. the remote device (or computer) must act as a server. It is not possible to connect to another client.
- Please note: For TCP connections you can additionally specify a keep-alive time. This allows TConnector to detect a lost connection within a reasonable time. The keep-alive time must be appended in the field Service/Port. Add a hash and then the keep-alive time in milliseconds (e.g. 80#1000 for 1 sec. keep-alive).

6.8.1 Host (or IP-Address)

Host name of a TCP/IP server (e.g. *www.tec-it.com*), or a TCP address (like '127.0.0.1' = local host)

6.8.2 Service/Port

Specifies a service name or port number (e.g. "http", "80"…)

6.9 Property Page Connection for IOType Bluetooth

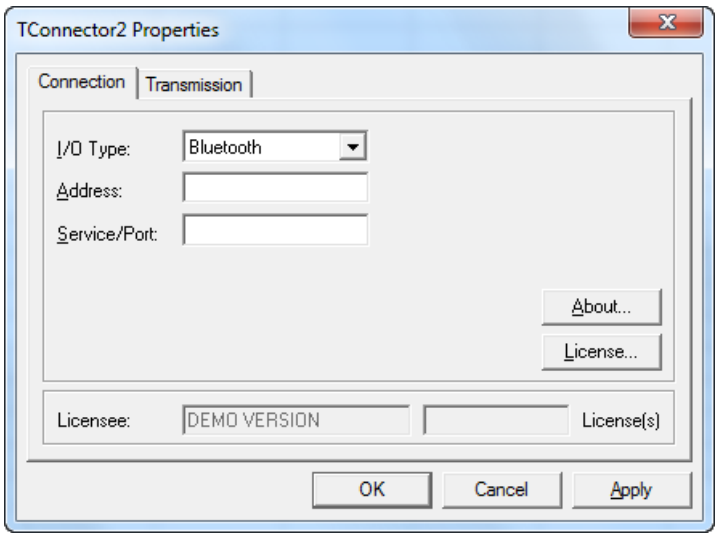

Fig. 6: IO-Type Bluetooth

`IOType` is set to Bluetooth. In this mode TConnector opens a connection to a Bluetooth device. The user can specify the unique address of the Bluetooth device (a series of hex codes).

6.9.1 Address

Bluetooth address (like '1B:F3:E1:10:01:21'). Retrieve the unique Bluetooth address from your device. For programmatically access: The address is stored in the COM Property "Host".

6.9.2 Service/Port

Specifies a service name or port number, enter '0' as default.

6.10 Property Page Transmission

In the `Transmission` page you can specify all properties that have to do with data transmission.

> ► Only methods using the asynchronous mode are concerned by these properties (`StartListen`, `StopListen`, `OnData`,...). Methods using the synchronous calls (like `Read`, `Write`) are not concerned.

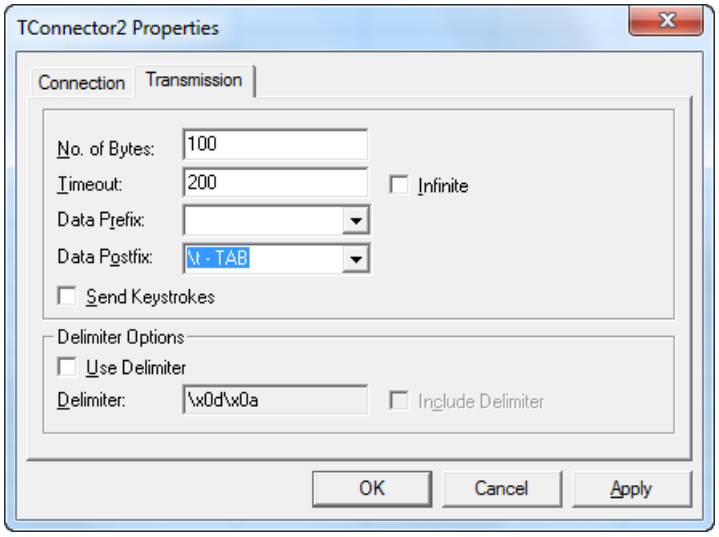

Fig. 7: Property Page Transmission

6.10.1 No. of Bytes

Maximum number of bytes, which should be receive in one turn.

6.10.2 Timeout

Timeout for one asynchronous read (only available if property Infinite is not checked)

6.10.3 Infinite

Determine if time out should be infinite or the value set in the property `Timeout`.

6.10.4 Data Prefix

String prefixes the incoming data stream (values can be selected from the list or typed in directly)

6.10.5 Data Postfix

String postfixes the incoming data stream (values can be selected from the list or typed in directly)

6.10.6 Send Keystrokes

If enabled, the incoming data is translated into corresponding keystrokes (can be used to enter data automatically into input fields).

6.10.7 Use Delimiter

You can use a “Delimiter” to trigger an OnData event, if special characters occur in the input data. A delimiter consists of one or more characters and is used to separate blocks of data not by length, but by the given character.

6.10.8 Delimiter

Delimiter characters (e.g. \x0d for Carriage Return)

6.10.9 Include Delimiter

Include delimiter into byte block or strip off from the input data. If this option is checked, the received text contains the delimiter character(s) otherwise the delimiter is removed before firing the OnData event.

7 ActiveX Programming Interface

7.1 General

Most programming environments support the use of ActiveX objects. *TConnector* conforms to Microsoft's ActiveX specification and can therefore be used in a very comfortable and easy way.

The *TConnector* object can be inserted on a form (e.g. in Visual Basic) but it can also be created as (invisible) instance of an object (without needing a form). *TConnector* is not limited to Microsoft applications – it can be used with all development environments in common use.

To learn more about object-oriented programming languages and get further information about COM objects we refer to the appropriate technical literature. For specific questions you can also contact our support.

7.1.1 Prog ID, Class ID

```
Prog ID TConnector2 = „ TConnector2.TConnector2"
Class ID TConnector2 = {126C289A-607B-4251-BF31-1555A5951948}
```

7.2 Properties

Object properties are essentially identical to those, which are used in the Property Pages – so we recommend the reading of the section Property Pages too.

The object characteristics (Properties) are discussed below.

7.2.1 Connection Properties+

This section contains all properties that are necessary to open a connection to a specified device.

7.2.1.1 IOType

`IOType` is the main property in this section that determines which kind of connection should be opened. In dependence of `IOType`, some properties get different meanings. Particular properties are used only with certain communication types and are ignored by others (details see below).

Default Value	eNone
Get/Set	Get/Set
Data Type	Enumeration e_DeviceType
Value Range	eNone no connection eNULL null device eFILE file eCOM serial port eLPT parallel port eTCP TCP eBTH Bluetooth
See also	Device, Host, Service

7.2.1.2 Device

This property specifies the name of the interface or target file. The value is used only with the communication types File, Serial, and Parallel.

Default Value	Empty

Get/Set	Get/Set
Value Range	Any valid filename or port-name
See also	IOType

The exact meaning depends on the currently selected IOType:

IOType	Meaning	Example
eFILE	File name (full path)	C:\Test.txt
eCOM	Serial port name	COM1, COM2…
eLPT	Parallel port name	LPT1, LPT2…
Else	Not used	-

7.2.1.3 Baud

This property is only used for IOType Serial.

Specifies the communication speed used for this serial line (in bits per second). It is important to choose exactly the same value as adjusted in your external device (110, …, 9600,…, 115200, …, 256000).

Default Value	9600
Get/Set	Get/Set
Data Type	String
Value Range	110, …, 9600,…, 115200, …, 256000
See also	

7.2.1.4 Data

This property is only used for IOType Serial.

Specifies the number of bits used for one data-word. It is important to choose exactly the same value as adjusted in your external device.

Default Value	8
Get/Set	Get/Set
Data Type	String
Value Range	5, 6, 7 or 8
See also	

7.2.1.5 Stop

This property is only used for IOType Serial.

Specifies the number of bits used for marking the end of a transmitted data-word. It is important to choose exactly the same value as adjusted in your external device.

Default Value	1
Get/Set	Get/Set
Data Type	String
Value Range	1, 1.5 or 2
See also	

7.2.1.6 Parity

This property is only used for IOType Serial.

Specifies the calculation method used for the parity bit. It is important to choose exactly the same value as adjusted in your external device.

Default Value	None (N)
Get/Set	Get/Set
Data Type	String
Value Range	N None O Odd E Even M Mark S Space
See also	

7.2.1.7 XonXOff

This property is only used for IOType Serial.

Xon/XOff is a software protocol for data flow-control (handshaking)[1]. If your external device uses this protocol, enable it. Software handshaking is usually disabled if hardware handshaking (DTRDSR and/or RTSCTS) is used. To enable Xon/Xoff handshaking set the property to `True`, to disable set it to `False`. It is important to choose exactly the same flow control as adjusted in your external device.

► If you want to transmit binary data, use hardware handshaking. Xon/XOff are control characters, which may interfere with binary data.

Default Value	False
Get/Set	Get/Set
Data Type	Boolean
Value Range	True False
See also	

7.2.1.8 DTRDSR

This property is only used for IOType Serial.

DTR (Data Terminal Ready) and DSR (Data Set Ready) are lines of the RS-232 Interface. Sometimes these lines are used for flow-control - in this case, enabled this property. To enable DTR/DSR hardware handshaking set the property to `True`, to disable set it to `False`. It is important to choose exactly the same flow control as adjusted in your external device.

► Be aware that some serial devices need a specific state of the DTR line. If DTR/DSR handshake is not used, the DTR line is controlled by the property DTRDefault.

Default Value	False
Get/Set	Get/Set
Data Type	Boolean
Value Range	True False
See also	

[1] Flow control means the ability to slow down the flow of Bytes in a wire. For serial ports this means the ability to stop and then restart the flow without any loss of Bytes.

7.2.1.9 RTSCTS

This property is only used for IOType Serial.

RTS (Ready To Send) and CTS (Clear To Send) are lines for hardware based flow control. Sometimes RTS/CTS is combined with DTR/DSR handshaking. To enable RTS/CTS hardware handshaking set the property to `True`, to disable set it to `False`. It is important to choose exactly the same flow control as adjusted in your external device!

► If RTS/CTS handshake is not used, the RTS line is controlled by the property RTSDefault.

Default Value	False
Get/Set	Get/Set
Data Type	Boolean
Value Range	True False
See also	

7.2.1.10 DTRDefault

This property is only used for IOType Serial.

If no DTR/DSR handshake is set, this property controls the state of the DTS line after the `Open()` command. The default setting is true, so the DTS-line will be high (if no handshaking is set).

Default Value	True
Get/Set	Get/Set
Data Type	Boolean
Value Range	True False
See also	

7.2.1.11 RTSDefault

This property is only used for IOType Serial.

If no RTS/CTS handshake is set, this property controls the state of the RTS line after the `Open()` command. The default setting is true, so the RTS-line will be high (if no handshaking is set).

Default Value	True
Get/Set	Get/Set
Data Type	Boolean
Value Range	True False
See also	RTSCTS, DTRDSR

7.2.1.12 Host (Address)

For IOType TCP this property specifies the hostname or IP-address (www.tec-it.com, “127.0.0.1“…). For IOType Bluetooth (BTH) this property stores the Bluetooth address.

► You can connect to an IP address as a client (like a telnet client) - the remote address must act as a server. Connecting to another client is not possible.

Default Value	Empty

Get/Set	Get/Set
Data Type	String
Value Range	- Any valid IP-address or (resolvable) hostname - Bluetooth address: nn:nn:nn:nn:nn:nn
See also	

7.2.1.13 Service

This property is used for IOType TCP and BTH.

Specifies the service name (e.g. telnet) or port-number (e.g. 22) when using IOType TCP.

Default Value	Empty
Get/Set	Get/Set
Data Type	String
Value Range	Any valid service name or port-number
See also	

7.2.2 Transmission Properties

This section contains the transmission properties for TConnector's asynchronous mode.

> ► Only methods using the asynchronous mode (like `StartListen`, `StopListen`, `OnData`) are concerned by these properties. Methods using the synchronous calls (like `Read`, `Write`) are not concerned.

7.2.2.1 Data Collection during Async Mode

You have different options to control data collection during the asynchronous mode. In Asynchronous mode, input data is collected in the background and is passed to the application by the OnData event.

Usually it is not intended that the user receives the data byte for byte; so there are options to collect the input data and transmit a "bunch of bytes" at one time. Several conditions can be configured to limit data collection and trigger the OnData event.

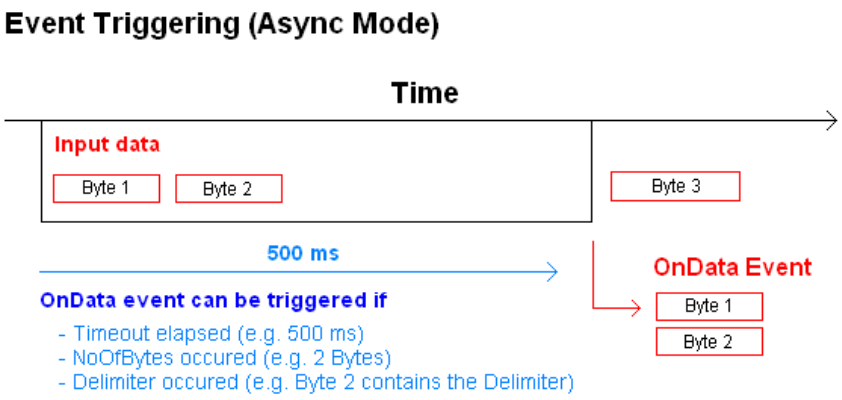

Fig. 8: Event Triggering in Async Mode

In the figure above Byte 1 and Byte 2 are passed to the application within the OnData event. Byte 3 will be passed in the next event.

The OnData event can be triggered after a specific Timeout elapsed since the first byte was received. Also TConnector reserves an input buffer with the size of *NoOfBytes*. and fires always an event if this limitation has been reached.

7.2.2.2 NoOfBytes

The maximum number of bytes to receive before an `OnData` event is fired. If `NoOfBytes` is 0 then all bytes received within the timeout period will be sent with the next `OnData` event.

Default Value	100
Get/Set	Get/Set
Data Type	Long
Value Range	0…64000
See also	Timeout

7.2.2.3 Timeout

The timeout period measured in 0.001 seconds. During the timeout period all data is collected (until `NoOfBytes` is reached) and then an `OnData` event is fired. The event will be fired , even if less than `NoOfBytes` bytes were received.

To suppress the timeout period, you can set the property `Timeout_Infinite` to false.

Default Value	1000
Get/Set	Get/Set
Data Type	Long
Value Range	0…64000
See also	Timeout_Infinite

7.2.2.4 Timeout_Infinite

If set to true, there is no timeout period in asynchronous mode. The events are fired when a certain amount of bytes (specified in `NoOfBytes`) was received. This mode is not supported with HTTP connections (here always a timeout is required).

Default Value	True
Get/Set	Get/Set
Data Type	Boolean
Value Range	True False
See also	Timeout

7.2.2.5 Prefix

`Prefix` defines a string that is placed in front of all other data characters sent by the `OnData` event. `Prefix` can contain arbitrary ASCII characters and predefined escape sequences.

Please see `Keyboard Emulation – Escape Sequences` for more information about possible virtual key codes.

Default Value	Empty
Get/Set	Get/Set
Data Type	String
Value Range	Any string
See also	Postfix

7.2.2.6 Postfix

Type: String

`Postfix` defines a string that is appended to a data package sent by the `OnData` event. `Postfix` can contain arbitrary ASCII characters and predefined escape sequences.

Default Value	Empty
Get/Set	Get/Set
Data Type	String
Value Range	Any string
See also	Prefix

Please see `Keyboard Emulation – Escape Sequences` for more information about possible virtual key codes.

7.2.2.7 SendKeyStrokes

If this value is true, the incoming data is translated into corresponding keystrokes. These characters are transmitted as if they have been entered on the keyboard.

This feature can be used to enter data automatically into input fields. The incoming data characters are passed to the input field of the current active foreground window.

Default Value	False
Get/Set	Get/Set
Data Type	Boolean
Value Range	True False
See also	Prefix, Postfix

Please see `Keyboard Emulation – Escape Sequences` for more information about possible virtual key codes.

7.2.2.8 UseDelimiter

Type: Boolean

If this value is true, the data read is not only limited by length (or by timeout), but also by one or more delimiter characters (see property `Delimiter`). If a delimiter is detected in the input data read by *TConnector*, the read data is returned by the `OnData` event immediately.

Default Value	False
Get/Set	Get/Set
Data Type	Boolean
Value Range	True False
See also	Delimiter

7.2.2.9 Delimiter

Delimiter String - used to fire the `OnData` event if this character combination occurs in the data read from the device. For special ASCII characters (e.g. CR or LF) use escape sequences with hexadecimal codes. The additional property `IncludeDelimiter` adjusts whether the delimiter string itself is passed as part of the data package by `OnData` or not.

Default Value	Empty
Get/Set	Get/Set
Data Type	String
Value Range	Any string

See also	UseDelimiter, IncludeDelimiter

Samples:

Delimiter sequence	Meaning
\x0d	Hexadezimal 0D = decimal 13 = Carriage Return (CR) The OnData event will be fired after CR occurred in the input data.
\x0a	Hexadezimal 0A = decimal 10 = Line Feed (LF) The OnData event will be fired after LF occurred in the input data.
\x0d\0xa	Hexadecimal 0d0a is ASCII CR/LF The OnData event will be fired after CR/LF occurred in the input data.

7.2.2.10 IncludeDelimiter

If the value of this property is `true`, the delimiter characters (property `Delimiter`) are included in the data passed over by the `OnData` event. Is set to false delimiter characters are stripped from the input data.

Default Value	False
Get/Set	Get/Set
Data Type	Boolean
Value Range	True False
See also	Delimiter

Can be used to strip off the CR/LF characters after reading one line (one shot with a bar code scanner) of data.

7.2.3 Other properties

7.2.3.1 StateAsTxt

```
StateAsTxt
(
eState        As e_TC_State
) As String
```

The `StateAsTxt` property translates a state number (received from `OnStatusChange` event or from `GetStates / GetSingleState` method) to its text representation. See also section Status Enumeration.

Default Value	Empty
Get/Set	Get
Data Type	String
Value Range	See section Status Enumeration
See also	See section Status Enumeration

7.3 Methods

7.3.1 Connection Methods

7.3.1.1 Open

Opens a connection to a device by using the interface parameters currently adjusted in the property settings. Changing the properties after calling `Open` will not effect the active connection.

`Open` must be the called before any other method that uses a connection is called. Otherwise an error occurs.

```
Open ()
```

Return Value	None
Exceptions	On error an exception is thrown

7.3.1.2 Close

Closes a connection and releases the opened port.

This method should always be called when you want to close an open connection (at least before object destruction).

```
Close ()
```

Return Value	None
Exceptions	On error an exception is thrown

7.3.2 Synchronous Methods

7.3.2.1 Read

Used for reading data.

Reads maximal `nNoOfBytes` bytes from the active port within the period defined by `nTimeout` (synchronous).

Read returns as soon as the specified number of bytes has been received from the device - or when the timeout period has expired. If no bytes were received during the timeout period, an exception (refer to the error codes below) will be released. To get the number of bytes received during the last successful read, use the `GetNoOfBytesRead()` method.

```
Read
(
nNoOfBytes    As Long,
nTimeout      As Long
) As String
```

nNoOfBytes	Number of bytes to receive
nTimeout	Timeout period in milliseconds
Return Value	Data received from the device
Exceptions	On error an exception is thrown
See also	*GetNoOfBytesRead, ClearBuffer*

Possible errors (constants):

Error Code	Meaning
E_TIMEOUT (-2147023436)	Timeout-Error: no data was read before timeout
E_END_OF_FILE (-2147024858)	End of File-Error: end of file reached

7.3.2.2 GetNoOfBytesRead

Return the number of bytes received during the last `Read` method. If a timeout occurred the return value can be 0, if an error occurred, it is –1.

```
GetNoOfBytesRead () As Long
```

Return Value	-1 in case of error

	0..x otherwise
Exceptions	None
See also	*Read, UseCodePage*

> ► Depending on the selected code page (see UseCodePage()) the number of bytes can be different from the number of characters in the received string. MultiByte character sets may use more than one byte for a single character.

7.3.2.3 ClearBuffer

Deletes all data from the input buffer.

Most interfaces can receive data in the background by caching in an internal system buffer. This data is also read in the `Read()` call.

Place `ClearBuffer()` before `Read()` if you want to ensure that you receive only "new" data (and not reading "old" data).

```
ClearBuffer ()
```

Return Value	None
Exceptions	None
See also	*Read*

7.3.2.4 Write

Used for sending data.

Transmits `nBytes` bytes of the specified string `bstrData` within the period defined by `nTimeout` to the device (or writes them to the file).

```
Write
(
nBytes        As Long,
nTimeout      As Long,
bstrData      As String
)
As Long
```

nBytes	Number of bytes to send
nTimeout	Timeout period in milliseconds
bstrData	Data to send
Return Value	Number of bytes sent/written within the timeout period
Exceptions	On error an exception is thrown
See also	*Read*

Possible errors (constants):

Error Code	Meaning
E_TIMEOUT (-2147023436)	Timeout-Error: no data was sent before timeout

> ► Depending on the selected Code Page (see `UseCodePage()`) the number of bytes sent/written can be different from the number of characters in bstrData.

7.3.3 Asynchronous Methods

7.3.3.1 StartListen

Starts the asynchronous mode. `OnData` events will be generated if data has been received. You can handle the `OnStatusChange` event if you want to know more about status line changes.

```
StartListen ()
```

Return Value	None
Exceptions	On error an exception is thrown
See also	Read, Write

7.3.3.2 StopListen

Stops the asynchronous mode. Should be called after `StartListen` if the asynchronous mode should be stopped.

```
StopListen ()
```

Return Value	None
Exceptions	On error an exception is thrown
See also	Read, Write

7.3.4 Other methods

7.3.4.1 AboutBox

Shows information dialog (containing the version number) about *TConnector*.

```
AboutBox ()
```

Return Value	None
Exceptions	None
See also	

7.3.4.2 EscapeZeroBytes

Strings or Data containing Zero Bytes (binary value = 0x00) can be truncated by some COM wrappers (such as used by Visual Basic). To avoid the truncation of data with Zero Bytes you can escape them with backslash zero: “\0”.

- This works for the OnData event (and in V2.4.3 also for Read() and Write() method).
- Your software must convert the \0 back to the binary value during processing of the data.

Turn on the escaping before you call StartListen()

```
EscapeZeroBytes (bEnable)
```

bEnable	True: Enable Escaping of Zero Bytes in the OnData event False: Disable Escaping of Zero Bytes (Default)
Return Value	None
Exceptions	None
See also	

7.3.4.3 LicenseMe

Licenses *TConnector* with the license key provided by TEC-IT. Licensing removes any demo-mode restrictions. You receive the license key from TEC-IT Datenverarbeitung GmbH after you have ordered a license. This method enables the user to license TConnector automatically from the applications source code (for a more detailed description see chapter Licensing).

TEC-IT suggests calling this method every time your application starts up. For license keys not starting with "`Mem:`" this call can be executed once (when installing your product).

If the license key starts with "`Mem: xxxxxxxx`" the product stays licensed until *TConnector* will be unloaded from memory.

```
LicenseMe
(
bstrLicensee          As String,
eKind                 As e_LicenseKind,      /* As Long if enumeration is not available */
nLicenses             As Long,
bstrLicenseKey        As String,
eProductID            As e LicenseProduct    /* As Long if enumeration is not available */
)
```

bstrLicensee	String containing the name of the Licensee. Provided by TEC-IT after ordering.
eKind	The type of the license. Provided by TEC-IT after ordering. Possible values: eLicKindSingle (1) eLicKindSite (2) eLicKindDeveloper (3)
nLicenses	The number of available licenses. Provided by TEC-IT after ordering.
bstrLicenseKey	String containing the license key Provided by TEC-IT after ordering.
eProductID	The type of the licensed product variant. Always eLicprodStd
Return Value	None
Exceptions	None
See also	

- If the license key received by TEC-IT looks like "`HKCU:xxxxxxxx`" the product is licensed in the section "HKEY Current User" of the Windows registry.
- If the license key received by TEC-IT looks like "`xxxxxxxx`" or "`HKLM:xxxxxxxx`" the product is licensed within the section "HK Local Machine" of the Windows Registry (this is the default).

7.3.4.4 Licensing

Call the dialog-box for manual licensing of *TConnector*. License data can be entered in the upcoming dialog.

```
Licensing ()
```

Return Value	None
Exceptions	None
See also	

7.3.4.5 TranslateErrorNo

Returns the error description text for the error number given by the `OnError` event.

```
TranslateErrorNo
(
hr            As Long
)
```

```
As String
```

Hr	Error number
Return Value	Error text
Exceptions	None
See also	

7.3.4.6 SetDebugLevel

Enables tracing of TConnector's internal activities into a log-file. A higher debug level logs more details.

The created log file is named "TConnector.log" and is placed within the actual users temporary directory (path depends on operating system and settings).

Example Log File Path : `C:\Documents and Settings\Susan\Local Settings\Temp`

```
SetDebugLevel
(
nLevel        As long                                    /* debugging level */
)
```

nLevel	Possible values:	
	0:	No debug (default)
	1:	Tracing errors within the ActiveX and other important information
	2:	1 + low level errors (API functions → here called LIB)
	3:	2 + ActiveX function-calls…
	4:	3 + LIB information
	5:	4 + all other information (this will result in big data amount for the log file).
Return Value	None	
Exceptions	-	
See also		

► Make sure that you disable the log level after debugging. Logging can fill your disk and slow down execution of programs.

7.3.4.7 GetStates

This function may be called in synchronous and asynchronous mode (after a device has been opened). It returns an array of actually set states. Unset states are not included. See chapter Status Enumeration to get the list of possible states. The Variant data type is VT_ARRAY | VT_I4.

```
GetStates () As Variant
```

Return Value	Array of state flags (as numbers)
Exceptions	-
See also	

7.3.4.8 GetStateArraySize

Returns the size of the State Array passed in the `OnStatusChange()` event or from the `GetStates()` method. This function is used together with `GetStateFromArray()`.

```
GetStateArraySize
(
vStates              As Variant
) As Long
```

vStates	Variant array with the status info

Return Value	Size of vStates array (max index = size - 1)
Exceptions	-
See also	*GetStates, GetStateFromArray*

7.3.4.9 GetStateFromArray

This function is used to retrieve a single element of the variant array passed in the `OnStatusChange()` event or from the `GetStates()` method. See chapter Status Enumeration to get the list of possible states.

```
GetStateFromArray
(
nIndex              As Long,
vStates             As Variant
) As etag_TC_State
```

nIndex	Index of the array element to be retrieved
vStates	Variant array with the status info
Return Value	Enumeration of state info (enum type long int)
Exceptions	-
See also	*GetStates, GetStateArraySize*

7.3.4.10 SetStates

This function may be called as soon a device has been opened (in synchronous and asynchronous mode). Fill in the states to be set into an array of 4-byte integer and pass it as argument to the `SetState` function.

```
SetStates
(
vStates             As Variant
)
```

Parameters: vStates - array of states to be set (element type integer, element size 4)

vStates	Array of states to be set (element type integer, element size 4), see chapter State Enumeration to get the list of possible states. Supported Variant types = VT_ARRAY | VT_I4 and VT_ARRAY | VT_VARIANT.
Return Value	-
Exceptions	-
See also	SetSingleState

7.3.4.11 SetSingleState

Like `SetStates()`, but only to set a single state.

```
SetSingleState
(
eState              As etag_TC_State
)
```

eState	Enumeration of the state that you want to set (enum type = long int), see chapter State Enumeration to get the list of possible states.
Return Value	-
Exceptions	-
See also	SetStates

7.3.4.12 EmulateKeys

Use this method to send keystrokes and keystroke combinations to the active application. `EmulateKeys` translates an input string into virtual key strokes and puts them into the keyboard events message queue.

For virtual key codes (e.g. function key `F10`) you can enable `bTransEscSequ`. For an overview about the possible escape sequences, please check out #Keyboard Emulation – Escape Sequences

```
EmulateKeys
(
strData              As String,
bTransEscSeq         As Bool,
nDelay               As Long
) As Long
```

strData	The string of keystrokes to send
bTransEscSeq	Translate escape sequences on/off
nDelay	Delay in milliseconds (100 = 0.1 sec) If not zero, an inter character or "inter key stroke" delay will be added (sometimes programs need slow typing and don't like fast input)
Return Value	Number of generated key strokes
Exceptions	See below
See also	

- Don't generate key strokes in synchronous and asynchronous mode at the same time. The function will raise an exception if called during asynchronous mode (`StartListen`) with the property `SendKeystroke` activated.
- When using escape sequences (`bTransEscSeq=True`), check your input data for backslashes. Convert a '\' into '\\' if it is no escape sequence.

7.3.4.13 UseCodePage

Adjusts Codepage to be used for Unicode-MultiByte string conversion.

All string-parameters for Read/Write/OnData are using Unicode characters (16 Bit per character according to ActiveX/COM Standard). But the available input/output channels are Byte-based (8-Bit). The value range 0x00 - 0xff in the Unicode standard corresponds to the ASCII table and is transmitted as a Single Byte. If you use characters outside of the ASCII range it may be necessary to perform a Unicode-MultiByte conversion, which can be enabled by selecting a specific CodePage.

The default codepage is ANSI (0). For a complete list of codepage Identifiers use this URL:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/intl/unicode_81rn.asp

- The available Codepages can be dependent on your operating system.

```
UseCodePage
(
nCodePage            As UINT
)
```

nCodePage	CodePage number for Unicode - Multibyte conversion.
Return Value	Number of generated key strokes
Exceptions	See below
See also	

Codepage Examples:

Codepage number	Description
437	OEM - United States
932	ANSI/OEM - Japanese, Shift-JIS
949	ANSI/OEM - Korean (Unified Hangeul Code)
950	ANSI/OEM - Traditional Chinese (Taiwan; Hong Kong SAR, PRC)
1250	ANSI - Central European
20932	JIS X 0208-1990 & 0121-1990
28592	ISO 8859-2 Central Europe
65000	Unicode UTF-7
65001	Unicode UTF-8

Table 3: Codepage List (partial)

Beside this, the following values are supported by Windows (C/C++):

```
//  Code Page Default Values.

#define CP_ACP                0       // default to ANSI code page
#define CP_OEMCP              1       // default to OEM  code page
#define CP_MACCP              2       // default to MAC  code page
#define CP_THREAD_ACP         3       // current thread's ANSI code page
#define CP_SYMBOL4            2       // SYMBOL translations

#define CP_UTF7               65000   // UTF-7 translation
#define CP_UTF8               65001   // UTF-8 translation
```

► The methods `GetNoOfBytesRead()` and `Write()` return the number of bytes before/after conversion. Depending on the selected code page the return value can be different from the count of characters in the input/result string.

7.4 Events

7.4.1.1 OnClose

This event is used in IOType TCP. It informs the client (=*TConnector*) that the server has just terminated the connection. The `Close()` method is called automatically and the connection will be reset.

```
OnClose
```

7.4.1.2 OnData

This event is used in the asynchronous mode. Each time when data is received an `OnData` event is thrown. The data read from the device is returned with the `Data` argument.

```
OnData
(
Data                  As String
)
```

Data	Data received from the connected device
See also	Asynchronous Mode

Depending on the selected code page (see `UseCodePage()`) the number of bytes transmitted can be different from the number of Characters in the received string. During conversion (Multi-Byte to Unicode) several bytes can be combined to one character in the Unicode string.

7.4.1.3 OnError

This event will be fired when an error occurs (only available during Asynchronous mode). The value passed is the error number to be converted to the error text using TranslateErrorNo.

```
OnError
(
hr                  As Long
)
```

Data	Error number
See also	TranslateErrorNo

7.4.1.4 OnStatusChange

This event is used in the asynchronous mode. Each time when the state of the currently opened interface changes an `OnStatusChange` event is thrown. The list of state flags is returned with the `vStates` argument. If your application or your scripting language hasn't sufficient support for the used Variant data type, you can use the methods `GetStateArraySize()` and `GetStateFromArray()` to access the status info.

To make it easier for the user to handle the values of a state array, an enumeration type `e_TC_State` is defined that has a 1:1 relationship with the states returned. To convert the state flags to text representation use the property `StateAsTxt`.

```
OnStatusChange
(
vStates             As Variant
)
```

vStates	Array of state flags that show which states have been changed.
See also	GetStates, SetStates, SetState

8 Licensing

When you download TConnector from http://www.tec-it.com you will get a demo version of this product. During unlicensed mode this version inserts demo data at random periods of time.

To switch the demo into the full version you have to license TConnector by applying a license key. This can be done manually in the license dialog or by program code from within your application.

The license data (including the license key to unlock the demo) can be ordered TEC-IT Datenverarbeitung GmbH (www.tec-it.com) or any reseller of TEC-IT software.

> ► Licensing is not the same as Registering. “Licensing” means that you enable the full-featured version by applying a license key. “Registering” means to register the “tconnector2.dll” file (TConnector ActiveX) in your Windows system. All ActiveX Controls have to be “registered” in the system before you can use them. Registering is usually done with the setup tool (but can also be done manually using regsvr32.exe).

8.1 Manual Licensing

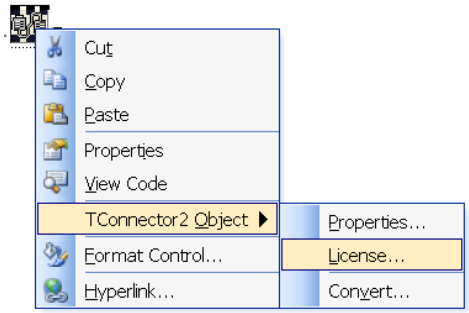

Fig. 9: Open the license dialog

To license manually you have to execute following steps:

1. Insert TConnector ActiveX into your application (when not already done)
2. Right click the inserted object and select TConnector2-Object/License… from the popup menu.
3. The license dialog (Fig. 10) should now open
4. Enter the license data, that you get from TEC-IT Datenverarbeitung GmbH, into the according fields

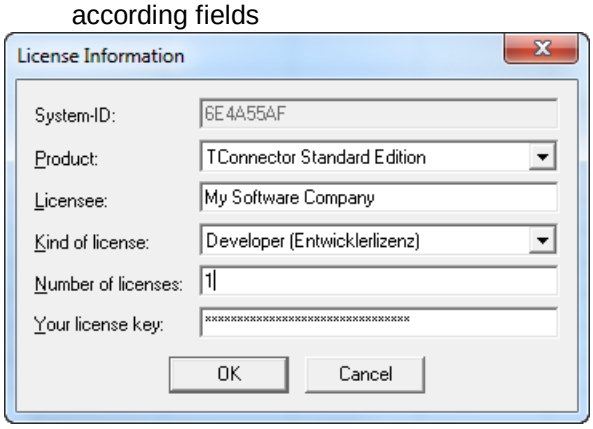

Fig. 10: License dialog

8.2 Automatic Licensing

If you want to distribute TConnector with your self-written applications you should license the control programmatically.

Include a statement like the following into your program code (the sample is written in Visual Basic, the license key is abbreviated).

```
TC2.LicenseMe ("LicenseeName", eLicKindDeveloper, 1, "1E4D21…", eLicProdStd)
```

► Do not use the license data exactly as shown in this sample but use the license data that TEC-IT Datenverarbeitung GmbH sends/will send you after you have ordered a license for TConnector ActiveX.

If your programming language doesn't support enumerations, use these values:

Enumeration Name	Value
ELicKindSingle	1
eLicKindSite	2
eLicKindDeveloper	3
eLicProdStd	7

Table 4: License Enumeration Equivalents

9 Redistribution

This chapter explains what is important when redistributing a custom application that uses the TConnector ActiveX control.

> ➢ In most cases you need a developer license for re-distribution of *TConnector* (except for in-house applications which are bound to one or more sites).

9.1 Dependencies

An application that uses *TConnector* requires the following files:

File	Description
TConnector2.dll	This is the ActiveX DLL. This file is mandatory
TConnectorps.dll	This file is a proxy file required for the OnData event.

These files are located in the folder *C:\Program Files\TEC-IT\TConnector2*.

TConnector requires also the **Visual Studio Runtime Components** (Microsoft VC90 CRT and ATL DLLs). See next section how to distribute them with your application.

9.2 Redistribution

When redistributing a custom application the files described above need to be redistributed together with the application. The DLLs should be located in the same folder as the executable. Other files than those listed above must not be redistributed.

You may have to redistribute the **Visual C++ 2008 SP1 runtime components** (MS CRT 9.0 and ATL 9.0 DLLs) with your application and ensure they are installed on the target computer.

There are two options to install them:

- You can install these components with the *Microsoft Visual C++ 2008 SP1 Redistributable Package (x86)* available at http://www.microsoft.com/downloads/details.aspx?familyid=A5C84275-3B97-4AB7-A40D-3802B2AF5FC2&displaylang=en.
  The package installs all required runtime DLLs.
- If you use a setup tool and your installer supports merge modules (*.msm files) you can add the required msm packages to your setup:
  - `Microsoft_VC90_CRT_x86.msm`
  - `policy_9_0_Microsoft_VC90_CRT_x86.msm`
  - `Microsoft_VC90_ATL_x86.msm`
  - `policy_9_0_Microsoft_VC90_ATL_x86.msm`

> ➢ The most simply way to deploy *TConnector* along with your application is to use the original MSI setup you can download from TEC-IT. The MSI package includes all dependencies (also Visual Studio Runtimes). MSI packages also allow a “silent setup” mode (no dialogs).
>
> ➢ Please contact TEC-IT Support if you need help.

10 Sample Applications

10.1 Sample code

Additional sample code for Microsoft Access®, Borland®/Inprise® Delphi®, Internet Explorer® (Javascript®/VBScript®), ASP® (VBScript) can be downloaded from TEC-IT's web site:

http://www.tec-it.com/download/ ► Data Acquisition.

Samples for Microsoft Excel® and Visual Basic® are included in the setup of TConnector.

11 Troubleshooting / FAQ

11.1 How can I eliminate the CR/LF after each data input?

You are attempting to route scanned data into an existing application, so you need just the raw data, no “Carriage Return” or “Line Feed”.

Follow the steps below to clip CR/LF at the end of your data (these steps apply if you are working in asynchronous mode).

- Open the property page `Configuration` ► `Interface` and change to the tab `Transmission`
- For the delimiter enter: \x0d\x0a (these are the hex codes for CR+LF). Enter only \x0d if you need only CR stripped off.
- Leave the “`Include Delimter`” option unchecked.

Now if a CR/LF is found in the input data the data will be processed immediately but the CR/LF will be filtered out (because “Include Delimiter” is unchecked).

11.2 How TConnector supports debugging?

For easier problem solving you can instruct *TConnector* to write a log file. Call the method `SetDebugLevel` in your program code (at the very first beginning at startup) and then *TConnector* writes internal trace information into the file “TConnector.log”.

Follow this link to read more about SetDebugLevel.

11.3 What can I do if the COM port is not accessible?

Make sure that the COM port is really free and not occupied by another process (= another application or driver, e.g. mouse driver).

It can be that you need to disable the "Direct Connection Between 2 Computers" device AND uninstall it.

If you can’t get it working, we suggest writing a log (trace) file:

Place this command as the first command, which is performed by TConnector in your program.

```
[TConn2ObjectID].SetDebugLevel (5)
```

Next open the port. A `TConnector2.log` file will be created in the actual user temp directory (use file search if you don’t find it)

Follow this link to read more about SetDebugLevel.

The log file can tell you more, why a connection can’t be established. Also our support can help you.

11.4 How can I specify a COM port > COM9?

First you have to edit the properties through the ActiveX property dialog available from your application (not the through the Property Pages of TConnector itself). Then you can enter the name of the port without restriction.

You need to specify COM ports > COM9 as follows:

Com port	String passed to Property Port
COM1	COM1
COM10	\\.\COM10
COM255	\\.\COM255

► On systems running Windows 95, Windows 98 or Windows ME this feature is not possible. These systems are limited to 9 com-ports.

11.5 Zero Bytes are truncated in my input data!

Some COM wrappers truncate strings if they contain a binary zero. So the data string won't pass the COM interface of TConnector without being truncated.

TConnector offers a workaround for this problem by using Escape Sequences. You can convert Bytes with binary values of 0x00 to an Escape Sequence like "\0". This workaround is available for the OnData event and starting with V2.4.3 also for Read() and Write() method.

Enable the escaping feature with *EscapeZeroBytes (true)* – for more details see 7.3.4.2.

12 Contact and Support Information

TEC-IT Datenverarbeitung GmbH

Address:	Wagnerstr. 6 AT-4400 Steyr Austria/Europe
Phone:	+43 / (0)7252 / 72 72 0
Fax:	+43 / (0)7252 / 72 72 0 – 77
Email:	mailto:support@tec-it.com
Web:	http://www.tec-it.com

AIX is a registered trademark of IBM Corporation.
HTML, DHTML, XML, XHTML are trademarks or registered trademarks of W3C, World Wide Web Consortium, Laboratory for Computer Science NE43-358, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139.
JAVA® is a registered trademark of Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, CA 94303 USA.
JAVASCRIPT® is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.
Microsoft®, Windows®, Microsoft Word®, Microsoft Excel® are registered trademarks of Microsoft Corporation.
Navision is a registered trademark of Microsoft Business Solutions ApS in the United States and/or other countries.
Oracle® is a registered trademark of Oracle Corporation.
PCL® is a registered trademark of the Hewlett-Packard Company.
PostScript is a registered trademark of Adobe Systems Inc.
SAP, SAP Logo, R/2, R/3, ABAP, SAPscript are trademarks or registered trademarks of SAP AG in Germany (and in several other countries).

All other products mentioned are trademarks or registered trademarks of their respective companies. If any trademark on our web site or in this document is not marked as trademark (or registered trademark), we ask you to send us a short message (mailto:office@tec-it.com).

Appendix A: State Enumerations

This chapter offers a list of possible state values. Use this enumerations for `GetStates()`, `SetStates()`, `OnStatusChange()` and related state functions.

A.1 Serial States

A.1.1 OnStatusChange

Name	Number	Description
e_State_COM_EV_BREAK	1000	A break was detected on input.
e_State_COM_EV_CTS	1001	The CTS (clear-to-send) signal changed state
e_State_COM_EV_DSR	1002	The DSR (data-set-ready) signal changed state
e_State_COM_EV_ERR	1003	A line-status error occurred
e_State_COM_EV_PERR	1004	A printer error occurred
e_State_COM_EV_RING	1005	A ring indicator was detected
e_State_COM_EV_RLSD	1006	The RLSD (receive-line-signal-detect) signal changed state
e_State_COM_EV_RXCHAR	1007	A character was received and placed in the input buffer
e_State_COM_EV_RX80FULL	1008	The receive buffer is 80% full
e_State_COM_EV_TXEMPTY	1009	The last character in the output buffer was sent

Table 5: Serial State Enumerators (OnStatusChange)

A.1.2 GetState

Name	Number	Description
e_State_COM_GET_CTS	1100	The CTS (clear-to-send) signal is on
e_State_COM_GET_DSR	1101	The DSR (data-set-ready) signal is on
e_State_COM_GET_RING	1102	The ring indicator signal is on
e_State_COM_GET_RSLD	1103	The RLSD (receive-line-signal-detect) signal is on

Table 6: Serial State Enumerators (GetState)

A.1.3 SetState

Name	Number	Description
e_State_COM_SET_DTRCLR	1200	Clears the DTR (data-terminal-ready) signal
e_State_COM_SET_RTSCLR	1201	Clears the RTS (request-to-send) signal
e_State_COM_SET_DTRSET	1202	Sends the DTR (data-terminal-ready) signal
e_State_COM_SET_RTSSET	1203	Sends the RTS (request-to-send) signal
e_State_COM_SET_XOFF	1204	Causes transmission to act as if an XOFF character has been received
e_State_COM_SET_XON	1205	Causes transmission to act as if an XON character has been received
e_State_COM_SET_BREAKCLR	1206	Restores character transmission and places the transmission line in a nonbreak state
e_State_COM_SET_BREAKSET	1207	Suspends character transmission and places the transmission line in a break state

Table 7: Serial State Enumerators (SetState)

A.2 TCP States

A.2.1 OnStatusChange

Name	Number	Description
e_State_TCP_EV_READ	2000	Data received
e_State_TCP_EV_WRITE	2001	Data sent
e_State_TCP_EV_OOB	2002	Out of band data received
e_State_TCP_EV_ACCEPT	2003	Notification of incoming connections
e_State_TCP_EV_CONNECT	2004	Notification of completed connection
e_State_TCP_EV_CLOSE	2005	Connection was closed
e_State_TCP_EV_QOS	2006	Quality of service changed
e_State_TCP_EV_GROUP_QOS	2007	Not implemented yet.
e_State_TCP_EV_ROUTING_INTERFACE_CHANGE	3008	Routing interface changed
e_State_TCP_EV_ADDRESS_LIST_CHANGE	2009	Address list changed

Table 8: TCP State Enumerators

A.3 Parallel States

A.3.1 GetState

Name	Number	Description
e_State_PAR_GET_INIT	3000	Device is initializing
e_State_PAR_GET_AUTOFEED	3001	Auto feed line set
e_State_PAR_GET_PAPER_EMPTY	3002	Out of paper
e_State_PAR_GET_OFF_LINE	3003	Device is offline
e_State_PAR_GET_POWER_OFF	3004	Device is shut off
e_State_PAR_GET_NOT_CONNECTED	3005	No device connected
e_State_PAR_GET_BUSY	3006	Device is busy
e_State_PAR_GET_SELECTED	3007	Selected line set

Table 9: Parallel State Enumerators

Appendix B: Escape Sequences

Following escape sequences can be used together with keyboard emulation:

Sequence	Meaning
\n	Line feed (LF)
\r	Carriage return (CR)
\t	Tab
\xnn	ASCII value in hex (2 digits)
\0nnn	ASCII value in octal (3digits)

Table 10: Escape Sequences

Key codes can be used with keyboard emulation (`SendKeyStrokes` / `EmulateKeys`)

> ► Note: if used in the prefix/suffix with the `OnData` event, they are simulated as key events but filtered out in the data string passed to the event.

Sequence	Meaning
\1	Cursor up
\2	Cursor down
\3	Cursor left
\4	Cursor right
\5	Page up
\6	Page down
\Vxnn	nn … Virtual Key Code (in Hex format) Example: \Vx79 = F10 function key Please use this link to get an overview of possible virtual key codes: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winui/winui/windowsuserinterface/userinput/virtualkeycodes.asp

Table 11: Keycode Table

Appendix C: Wiring and Pin Out

C.1 RS232 Connector Signal Description

25 Pin DCE	9 Pin DTE	Signal	Direction	Description	Note
1		FG		Frame Ground	
2	3	TD	PC --> Device	Transmitted Data (TXD)	required
3	2	RD	Device -> PC	Received Data (RXD)	required
4	7	RTS	PC --> Device	Request to Send	for hardware handshaking
5	8	CTS	Device -> PC	Clear to Send	for hardware handshaking
6	6	DSR	Device -> PC	Data Set Ready	for hardware handshaking
7	5	SG		Signal Ground	required
8	1	DCD	Device -> PC	Data Carrier Detect	used for modem control
9			Device -> PC	Positive DC Test Voltage	not EIA standard
10			Device -> PC	Negative DC Test Voltage	not EIA standard
11					
12		SDCD	Device -> PC	Sec. Data Carrier Detect	
13		SCTS	Device -> PC	Sec. Clear to Send	
14		STD	Device -> PC	Sec. Transmitted Data	
15		TC	Device -> PC	Negative DC Test Voltage	optional (not EIA)
16		SRD	Device -> PC	Sec. Received Data	
17		RC (DD)	Device -> PC	Receiver Clock	synchronous communication
18					
19		SRTS	PC --> Device	Sec. Request to Send	
20	4	DTR	PC --> Device	Data Terminal Ready	for hardware handshaking
21		SQ	Device -> PC	Signal Quality Detector	
22	9	RI	Device -> PC	Ring Indicator	used for modem control
23		CI	PC --> Device	Data Ready Selector (Data Signal Rate Selector)	optional (not EIA)
24		TC (TA)	PC --> Device	Transmitter Clock Timing	synchronous communication
25					

Table 12: RS232 Pin Description

C.2 Parallel Port DB-25 Pin Out

Line	In/Out	Signal
1	Out	Strobe
2	Out	Data 0
3	Out	Data 1
4	Out	Data 2
5	Out	Data 3
6	Out	Data 4
7	Out	Data 5
8	Out	Data 6
9	Out	Data 7
10	In	!ACK

11	In	BUSY
12	In	Paper End
13		SLCT Select
14	Out	Autofeed (active on low signal! low = signal on)
15	In	Error (active on low signal! low = signal on)
16	Out	Reset (active on low signal! low = signal on)
17		SLCTIN (Select In)
18-25		Signal GND

Table 13: Parallel (Centronics) Pin Description

C.3 Related Links

C.3.1 RS232

C.3.1.1 Wiring and Pin Out Reference

http://www.arcelect.com/rs232.htm
http://www.airborn.com.au/rs232.html
http://www.commlinx.com.au/RS232_pinouts.htm

C.3.2 Parallel Port

C.3.2.1 Wiring and Pin Out Reference

http://www.ctips.com/spp.html
http://www.lvr.com/files/pppinout.pdf (Acrobat PDF document)

C.3.2.2 General Overview

http://www.lvr.com/parport.htm