Windows Driver DLL

Overview

Mantracourt supply a standard Windows driver DLL which can be used by many development tools. This DLL has been created to simplify communications with the T24 range of telemetry devices. This handles circular buffers for incoming data and provides signaling to the host application of incoming packets via a callback mechanism.
The DLL can be used with a single serial port or a USB bus.
Where strings are used these are in a format that can be used by Visual Basic. If used in Delphi or C Builder these functions need to be declared as StdCall.
Below you will find the API declarations for Visual Basic which may be used as a guide for other languages:

```
Public Declare Function VERSION Lib "T24Drv.dll" () As Single

Public Declare Sub INITIALISE Lib "T24Drv.dll" (ByRef iCallbackAddress As Long)

Public Declare Function OPENPORT Lib "T24Drv.dll" (ByVal ComPort As Integer, ByVal Baudrate
As Long) As Integer

Public Declare Function CLOSEPORT Lib "T24Drv.dll" () As Integer

Public Declare Function OPENUSB Lib "T24Drv.dll" () As Integer

Public Declare Sub ENUMUSB Lib "T24Drv.dll" ()

Public Declare Function BIND Lib "T24Drv.dll" (ByVal BaseStation As Byte, ByVal
UseRemoteSettings As Byte, ByVal ConfigMode As Byte, ByVal Duration As Byte, ByRef ID As
Long, ByRef DataTag As Long) As Integer

Public Declare Function BINDASYNC(ByVal BaseStation As Byte, ByVal UseRemoteSettings As
Byte, ByVal ConfigMode As Byte, ByVal Duration As Byte) As Integer

Public Declare Function BINDASYNCPOLL(ByRef ID As Long, ByRef DataTag As Long) AS Integer

Public Declare Function READREMOTE Lib "T24Drv.dll" (ByVal BaseStation As Byte, ByVal ID As
Long, ByVal Command As Byte, ByRef sData As Byte, ByRef Length As Long, ByRef RSSI As
Integer, ByRef CV As Integer, ByRef flags As Integer) As Integer

Public Declare Function WRITEREMOTE Lib "T24Drv.dll" (ByVal BaseStation As Byte, ByVal ID
As Long, ByVal Command As Byte, ByRef sData As Byte, ByVal Length As Long, ByRef RSSI As
Integer, ByRef CV As Integer, ByRef flags As Integer) As Integer

Declare Function WRITEPACKET Lib "T24Drv.dll" (ByVal BaseStation As Byte, ByRef sData As
Byte, ByVal Length As Long) As Integer
```

Functions and Subs

INITIALISE

Initialise the DLL and setup the callback address so that when packets arrive the hosting application receives a callback.

```
SUB INITIALISE(BYREF iCallbackAddress AS LONG)
```

Where:

Parameter	Description
iCallbackAddress	long pointer to the address of the callback procedure.

Required before OPENUSB or OPENPORT is called. Pass the address of the callback function. The format of the function (In Visual Basic) is

```
Sub CallBack(ByRef StringPtr As Long, Length As Long)
```

To extract the packet data the following **Visual Basic** example may prove useful:

```
Sub CallBack(ByRef StringPtr As Long, Length As Long)
    Dim PacketString As String
    Dim PacketByteArray(128) As Byte
    'To get the packet into a byte array
    CopyMemory PacketByteArray(0), ByVal StringPtr, Length
    'To get the packet into a string
    PacketString = Left$(StrConv(PacketByteArray(), vbUnicode),
Length)
End Sub
```

VERSION

Return the driver version.

```
FUNCTION VERSION() AS SINGLE
```

Returns:
The version number in floating point format.

OPENPORT

Open a serial port for communications.

```
FUNCTION OPENPORT(BYVAL ComPort AS INTEGER, BYVAL Baudrate AS LONG) AS
INTEGER
```

Where:

Parameter	Description
ComPort	Long variable indicating the COM port to open.
Baudrate	Long value representing the actual required baudrate. The PC port will generate to closest available baudrate to this value.

Returns:

Integer Value	Description
0	Port opened OK.
-1	Invalid settings. The serial port exists but could not be configured to the required settings including the requested baudrate.
-2	Could not open serial port at all. Either the port does not exist or another application has opened this port.

This function must be called before transmitting any packets and once called will activate the callbacks as packets are received by the base station.

OPENUSB

Opens communications with the USB bus. This function does not rely on a base station being present to be a success.

```
FUNCTION OPENUSB() AS INTEGER
```

Returns:

Integer Value	Description
0	USB bus opened OK.
-2	Could not initialize the USB bus.

This function must be called before transmitting any packets and once called will activate the callbacks as packets are received by the base station.

ENUMUSB

This sub **can** be called every few seconds to allow changes on the USB bus to be enumerated. Most of the time this will change nothing but if a device has been plugged or unplugged from the bus calling this will enumerate the remaining devices and gracefully cope with additions or removals of the base stations. As this can take up to 250mS to complete (When a change has been detected) it is recommended that this be called only every few seconds.

```
SUB ENUMUSB()
```

This should only be called after OPENUSB has been successful. If a base station is present when OPENUSB is called and it is going to stay connected then there is no need to call this method.

CLOSEPORT

Close any open serial ports or USB bus connection.

```
FUNCTION CLOSEPORT() AS INTEGER
```

Returns:

Integer Value	Description
0	Closed OK.
-2	An error occurred while closing the ports.

This function should be called before closing the hosting application.

BIND

Also known as **Pairing**. The BIND function allows connection information to be retrieved from an unknown remote device and to configure the communications settings between that device and the base station. Most devices activate their binding mechanisms by being power cycled but refer to the device manual for details.
This function is blocking and does not return until a bind is successful or the duration has expired. For a non blocking Bind function see BINDASYNC.

```
FUNCTION BIND(BYVAL BaseStation AS BYTE, BYVAL UseRemoteSettings AS BYTE,
BYVAL ConfigMode AS BYTE, BYVAL Duration AS BYTE, BYREF ID AS LONG, BYREF
DataTag AS LONG)  AS INTEGER
```

Where:

Parameter	Description
BaseStation	Represents the base station address. This should be 1.
UseRemoteSettings	Determines whether the remote device will be configured to the base station communications settings or vice versa. Set to 1 to change the base station settings to match those of the remote device or zero to change the remote device settings to match the base station.
ConfigMode	Determines whether the remote device will enter configuration mode. This mode is dependent on the actual device but will generally mean it will stop any automatic transmissions, inhibit low power modes and not act on any automatic sleep mechanisms. This ensures that the binding application can communicate and configure it.
Duration	Sets the duration of the bind attempt in seconds. i.e. how long the base station will wait for the remote device to enter bind mode.
ID	The ID of the bound device.
DataTag	The default Data Tag of the bound device.

Returns:

Integer Value	Description
0	Bind was successful.
1	Bind was not successful. No remote device was detected.
99	Thread conflict detected! Yield (sleep, doevents etc) and retry the function.

Once a successful bind has occurred the hosting application may communicate with the device using READREMOTE and WRITEREMOTE using the ID returned from the bind function.

BINDASYNC

This bind function is non blocking and is useful when you need to control the power supplied to your device to trigger the bind. This function is called first then use BINDASYNCPOLL to test the status and outcome of the bind.

```
FUNCTION BINDASYNC(BYVAL BaseStation As BYTE, BYVAL UseRemoteSettings As
BYTE, BYVAL ConfigMode As BYTE, BYVAL Duration As BYTE) As Integer
```

Where:

Parameter	Description
BaseStation	Represents the base station address. This should be 1.
UseRemoteSettings	Determines whether the remote device will be configured to the base station communications settings or vice versa. Set to 1 to change the base station settings to match those of the remote device or zero to change the remote device settings to match the base station.
ConfigMode	Determines whether the remote device will enter configuration mode. This mode is dependent on the actual device but will generally mean it will stop any automatic transmissions, inhibit low power modes and not act on any automatic sleep mechanisms. This ensures that the binding application can communicate and configure it.
Duration	Sets the duration of the bind attempt in seconds. i.e. how long the base station will wait for the remote device to enter bind mode.

Returns:

Integer Value	Description
0	Bind initiation was successful.
99	Thread conflict detected! Yield (sleep, doevents etc) and retry the function.

Now call BINDASYNCPOLL to determine when the bind has completed or failed.

BINDASYNCPOLL

Called after BINDASYNC to determine whether the bind is busy or has completed.

```
FUNCTION BINDASYNCPOLL(BYREF ID As LONG, BYREF DataTag As LONG) As INTEGER
```

Where:

Parameter	Description
ID	The ID of the bound device.
DataTag	The default Data Tag of the bound device.

Returns:

Integer Value	Description
0	Bind was successful.
1	Bind was not successful. No remote device was detected.
99	Busy. Binding is still in progress.

If successful the ID and DataTag parameters will contain the ID and Data Tag of the bound device.

READREMOTE

Reads a parameter from a remote device. The radio modules transparently handle retries. This function is blocking and execution will not continue until a response has been received or the function has timed out.

```
FUNCTION READREMOTE(BYVAL BaseStation AS BYTE, BYVAL ID AS LONG, BYVAL
Command AS BYTE, BYREF sData AS BYTE, BYREF Length AS LONG, BYREF RSSI AS
INTEGER, BYREF CV AS INTEGER, BYREF Flags AS INTEGER)  AS INTEGER
```

Where:

Parameter	Description
BaseStation	The address of the base station through which to route this packet.
ID	The ID of the remote device.
Command	The command number of the parameter to read.
sData	Pointer to the result data. There must be enough bytes allocated to avoid buffer overflows. Recommend 128 bytes. The first byte indicates the data type. This will be set by the device and you cannot request data of a specific type. See Data Types and Formats later.
Length	The number of bytes returned.
RSSI	The radio signal level in dB of the received packet. This parameter will be set on return from this function.
CV	The correlation value of the received packet. A value of 55 is a poorly formed signal whereas 110 is a perfectly formed signal. This parameter will be set on return from this function.
Flags	Contains extra information regarding this packet. The binary value of the flags indicate the following: 1=This packet was broadcast. 2=Remote device reports low battery. 4=Remote device reports an error. This parameter will be set on return from this function.

Returns:

Integer Value	Description
0	Received response OK
1	No response from base station.
2	No response from remote device.
3	NAK response from remote device.
99	Thread conflict detected! Yield (sleep, doevents etc) and retry the function.

WRITEREMOTE

Writes a parameter to a remote device. This function is blocking and execution will not continue until a response has been received or the function has timed out.

```
FUNCTION WRITEREMOTE(BYVAL BaseStation AS BYTE, BYVAL ID AS LONG, BYVAL
Command AS BYTE, BYREF sData AS BYTE, BYVAL Length AS LONG, BYREF RSSI AS
INTEGER, BYREF CV AS INTEGER, BYREF Flags AS INTEGER)  AS INTEGER
```

Where:

Parameter	Description
BaseStation	The address of the base station through which to route this packet.
ID	The ID of the remote device.
Command	The command number of the parameter to read.
sData	Pointer to the data to write. The first byte indicates the data type. See Data Types and Formats later. You can write any data type to any parameter and if possible the data will be converted when written. For example you could use strings to write all data if desired.
Length	The number of bytes returned.
RSSI	The radio signal level in dB of the received packet. This parameter will be set on return from this function.
CV	The correlation value of the received packet. A value of 55 is a poorly formed signal whereas 110 is a perfectly formed signal. This parameter will be set on return from this function.
Flags	Contains extra information regarding this packet. The binary value of the flags indicate the following: 1=This packet was broadcast. 2=Remote device reports low battery. 4=Remote device reports an error. This parameter will be set on return from this function.

WRITEPACKET

Transmit a custom T24 packet.
WARNING: Be very careful with this command as this command has the capability to write data to modules or alter their operational state.
This can be used to generally provide data provider packets or can be used to send basic commands via the Data Provider Control Interface. To send a Data Provider Packet the data pointed to by pData would contain the packet type byte and all data.

```
WORD WRITEPACKET(BYTE bBaseStation, BYTE *pData, DWORD dwLength)
```

Where:

Parameter	Description
`bBaseStation`	The address of the base station through which to route this packet.
`pData`	Pointer to the data to transmit. (The first byte must indicate the data type)
`dwLength`	The number of bytes to transmit.

Returns:

Integer Value	Description
0	Received response OK
1	No response from base station.
2	No response from remote device.
3	NAK response from remote device.
4	Invalid Data response from remote device.
99	Thread conflict detected! Yield (sleep, doevents etc) and retry the function.

Example
To send a Data Provider packet that looks to be from Data Tag 0x1234 and contains a floating point number of 1.234
The structure of a Data Provider Packet is

Packet Type	Data Tag		Status	Data Type	Data
03	00	00	00	00	[…]

So we can emulate this with this command by sending

Packet Type	Data Tag		Status	Data Type	Data	Data	Data	Data
03	12	34	00	04	3F	9D	F3	B6

As Data Type of 4 means a float and the value of 1.234 = 3F9DF3B6
You would point pData to the above bytes and set the length to 9.

Using the Data Provider Interface

When acquisition devices are operating in low power mode it is not easy to communicate using the full read/write packets as most of the time the device is asleep. Also in some cases the consumer of the data only knows the Data Tag from the Data Provider packet and does not know the ID of the sender. Therefore we need to utilise a control interface within the Data Provider packet scheme whereby devices such as a handheld can perform rudimentary control on another device while knowing no more than that devices default Data Tag.
Each device supplying data to a consumer only has one defined default Data Tag. We reuse that tag to enable communicating back to the data provider. This will not affect other consumers of the data as the data provider packet will contain a data type of **FF** which indicates our internal control interface.
All other consumers will automatically reject the FF data type anyway.
So to control the provider we simply send a data provider packet using the same data tag but containing data of type **FF** the data consists of a single **Function Byte** which has fixed functionality depending on its value. The status byte is not used and may be left at zero.
To use this interface the sender must reply with the control interface packet within 8 milliseconds of receiving a Data Provider packet. The format of the transmitted packet would be as follows.

Packet Type	Data Tag		Status	Data Type	Function Byte
03	00	00	**00**	**FF**	00

The Function Byte can have the following values. Note that not all modules may support all commands.

Value	Function
0	None
1	SLEEP
2	PAUSE
3	STAYAWAKE
4	CONTINUE
5	DOSYSTEMZERO
6	REMOVESYSTEMZERO
7	SHUNTCALON
8	SHUNTCALOFF
9	DOTARE
A	REMOVETARE
B	LEDONUNTILNEXTTX

Using a Data Tag of FFFF will act as a broadcast data provider control interface and all recipients of an FFFF data tag will check the data type and if this is FF the device may perform the specified function.
So to pause a module sending Data Provider packets with a Data Tag of 0x1234 you would need to send the following data within 8mS of receiving the Data Provider Packet.

Packet Type	Data Tag		Status	Data Type	Function Byte
03	12	34	**00**	**FF**	02

To send this packet use WRITEPACKET where **pData** would point to the above bytes and the **dwLength** would be 6.

Data Types and Formats

The first byte in the data written to and read from devices indicates the data type and thus the format of the data that follows.

Value	Data Type
0	Unknown data type. Can be used when executing commands.
1	UINT8
2	UINT16
3	INT32
4	FLOAT
5	STRING
5	BINARY

See **Appendix A** – **Data Type Formats** in the T24 Technical Manual for details on the formatting.

Thread Conflicts

The DLL has been designed to allow the host IDE to debug through the callback routines. To achieve this there is a situation where sometimes calls to the functions will not able to be handled correctly (i.e. a callback to the host IDE is in progress and to continue would violate the integrity of the threading). In this case the returned value will be 99 and the host program needs to yield processing if it to succeed in getting a response.
The suggested technique is as follows:

```
Variable = 99
While Variable = 99
       Variable = DLLFUNCTION()
       If Variable = 99 Then Yield
Wend
```

Where **`DLLFUNCTION()`** is where you would place the **`ReadRemote`** or **`WriteRemote`** call. You will need to find the appropriate command in your language to Yield. In **Visual Basic** this is **`DoEvents`** and in **Delphi** this is **`Application.ProcessMessage`**.

Notes

You should only use data from a function if it has returned a zero (0). All other responses indicate an error.

DLL Limitations

- When connecting via USB only one base station is supported and this must be address 1 (set by DIP switches on industrial version).
- This DLL can only open one serial port per host thread.