

CAN-4-USB/FX~~FX~~/MCP2515

Using ZCAN4USB DLL

Written by Steven D. Letkeman, B.Sc.

©2006 Zanthic Technologies Inc.

All rights reserved.

V 1.0.0

Last modified May 24, 2006

Copyright © 2006 by Zanthic Technologies Inc.
Other products are the trademarks of their respective manufacturers

All rights reserved. No part of this manual may be reproduced, in any form or by any means, without the permission in writing from Zanthic Technologies Inc.

Zanthic Technologies Inc. reserves the right to make changes without further notice to any products herein to improve reliability, function or design. Although the information in this document has been carefully reviewed and is believed to be reliable, Zanthic Technologies Inc. does not assume any liability arising out of the application or use of any product or circuit described herein. Zanthic Technologies Inc. products are not authorized for use as components in life support devices wherein a failure or malfunction of the component may directly threaten life or injury. Any software or firmware included with or embedded into this product may contain trade secrets and in order to protect them you may not decompile, reverse engineer, disassemble, or otherwise reduce the software to human-perceivable form.

Introduction:	4
SetCANBaud	5
Declaring this function in Visual Basic	5
Calling this function from Visual Basic	5
ResetInterface	6
Declaring this function in Visual Basic	6
Calling this function from Visual Basic	6
SendCANMess(age)	7
Declaring this function in Visual Basic	8
Calling this function from Visual Basic	8
Returned Values	9
RecCANMess(age)	10
Declaring this function in Visual Basic	11
Calling this function from Visual Basic	11
GetCANMess(age)	12
Declaring this function in Visual Basic	12
Calling this function from Visual Basic	12
Returned Values	13
ReadReg(ister)	15
Declaring this function in Visual Basic	15
Note: Make sure buffer is declared as ByRef	15
Calling this function from Visual Basic	15
WriteReg(ister)	17
Declaring this function in Visual Basic	17
Calling this function from Visual Basic	17
GetNumCANMess(ages)	19
Declaring this function in Visual Basic	19
Calling this function from Visual Basic	19
Returned Values	19
ClearCANMess(ages)	20
Declaring this function in Visual Basic	20
Calling this function from Visual Basic	20
GetInfo	21
Declaring this function in Visual Basic	21
Calling this function from Visual Basic	21
SetFilters	24
Declaring this function in Visual Basic	25
Calling this function from Visual Basic	25
New Commands to the CAN-4-USB FX interface	26
EnableTS	26
Declaring this function in Visual Basic	26
Calling this function from Visual Basic	26
DisableTS	27
Declaring this function in Visual Basic	27
Calling this function from Visual Basic	27

Introduction:

The Zanthic Technologies Inc. CAN-4-USB/FX device was designed to give a high speed, low cost, reliable method of allowing your PC to communicate on a Controller Area Network (CAN) Bus. This document describes the higher level interface that is used to transmit and receive commands to the CAN-4-USB/FX device across the USB port. In order for your software to send and receive CAN messages, there must first be a method to send and receive commands to the device. Packets of data are sent from the PC to the CAN-4-USB/FX/FX device through the USB port that contain the commands and data required to use the device. This lower level protocol is independent of USB in that it simply uses the USB structure to delivery the message and could just as easily be RS-232 or some other transmission medium. Likewise, the protocol is independent of any higher level CAN protocol, the data is simply passed through this layer regardless of what the CAN data means to either your software or another CAN node on the CAN bus. On top of this layer of commands is a DLL that converts functions like SendCANMess(age) to the protocol required by the CAN-4-USB/FX device. This document describes those functions. A sample VB6 program that demonstrates these can be found on the distribution disk. The source code for all of the PC programs and DLL's are included for your use when using the CAN-4-USB/FX device.

Common Error Messages

Each of the following functions will return a value (where applicable) within the following definitions. Note that some of these errors apply to other Zanthic applications that use threading and other features that are not used in the ZCAN4USBFX DLL

Value Returned	Description
1	Success
-1	for generic fail (no board present)
-2	for no response
-3	for improper calling parameter
-4	for improper response
-5	if more than one unit responded
-6	if system is busy at this time
-7	for failed in bulk write
-8	for failed in bulk read
-9	for interface not responding
-10	if interface config file not loaded properly
-11	start driver routine failed
-12	load driver routine failed
-13	could not open device
-14	could not open ISR device
-15	could not open ISR thread
-16	driver already loaded

SetCANBaud

This function will set the CAN controller to a standard CAN baud rate. Should your requirements not be met by these standard settings you have the option of setting the configuration registers manually.

The C function within the DLL looks like this

```
short int __stdcall SetCANBaud(unsigned char InterfaceNum, unsigned char DeviceNum, short int Baud)
```

InterfaceNum(ber) will be a value from 0-9 for depending on how many CAN-4-USB/FX devices are connected to the same computer. If there is only one device, this value should be set to zero.

DeviceNum(ber) is a future option for devices that will incorporate more than one CAN controller on a single device. For now, use the value zero.

Baud is a value given by the following:

Baud	Rate	# tq	BRP	Tseg1	Tseg2	Sample
10	10kbps	16	\50	8	4	75%
20	20kbps	16	\25	8	4	75%
50	50kbps	16	\10	8	4	75%
100	100kbps	16	\5	8	4	75%
125	125kbps	16	\4	8	4	75%
250	250kbps	16	\2	8	4	75%
500	500kbps	16	\1	8	4	75%
800	800kbps	10	\1	6	2	80%
1000	1000kbps	8	\1	4	2	75%

Declaring this function in Visual Basic

```
Private Declare Function SetCANBaud Lib "ZCAN4USBFX.dll" (ByVal InterfaceNum As _ Byte, ByVal DeviceNum As Byte, ByVal Baud As Integer) As Integer
```

Calling this function from Visual Basic

```
result = SetCANBaud(InterfaceNum, DeviceNum, baud)
```

where InterfaceNum, DeviceNum are declared as Byte variables and result and baudlist are declared as an integer variable

example:

```
result = SetCANBaud(0,0, 500) ‘ set CAN baud to 500kbps
```

ResetInterface

This function can be used to reset the CAN controller to a known state should your software put it in a confused condition. This call will also clear and reset the incoming message buffers.

The C function within the DLL looks like this

```
short int __stdcall  
ResetInterface(  
    unsigned char InterfaceNum,  
    unsigned char DeviceNum  
)
```

InterfaceNum(ber) will be a value from 0-9 for depending on how many CAN-4-USB/FX devices are connected to the same computer. If there is only one device, this value should be set to zero.

DeviceNum(ber) is a future option for devices that will incorporate more than one CAN controller on a single device. For now, use the value zero.

Declaring this function in Visual Basic

```
Private Declare Function ResetInterface Lib "ZCAN4USBFX.dll" _  
    (ByVal InterfaceNum As Byte, ByVal DeviceNum As Byte) As Integer
```

Calling this function from Visual Basic

```
result = ResetInterface(InterfaceNum, DeviceNum)
```

where InterfaceNum, DeviceNum are declared as Byte variables and result is declared as an integer variable

example:

```
result = ResetInterface(0, 0)
```

SendCANMess(age)

This function will send a CAN message using the ID, data and message object that you specify. Before using this function you must have already executed the SetCANBaud function.

The C function within the DLL looks like this

```
short int __stdcall
SendCANMess(
    unsigned char InterfaceNum,
    unsigned char DeviceNum,
    long ID,
    unsigned char CB1,
    unsigned char CB2,
    unsigned char * databytes
)
```

InterfaceNum(ber) will be a value from 0-9 for depending on how many CAN-4-USB/FX devices are connected to the same computer. If there is only one device, this value should be set to zero.

DeviceNum(ber) is a future option for devices that will incorporate more than one CAN controller on a single device. For now, use the value zero.

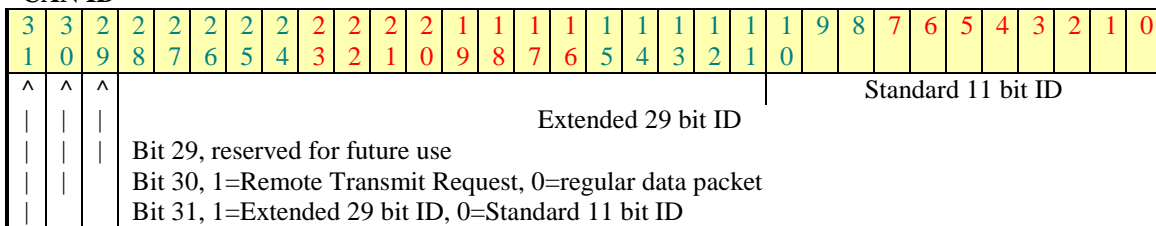
ID is the CAN ID (right justified) with the top 3 bits defined as shown below

CB1 is short for Command Byte 1 and is shown below

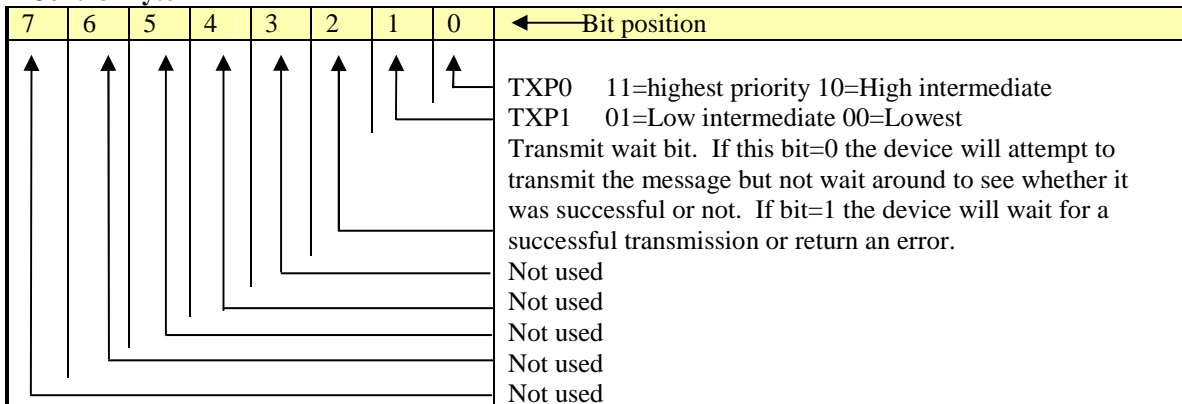
CB2 is short for Command Byte 2 and is shown below

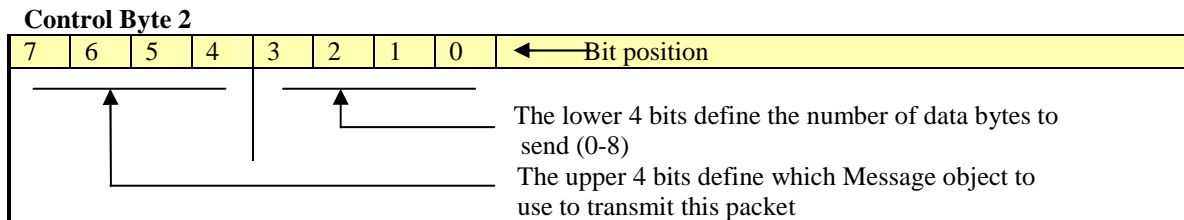
Databytes is an unsigned char buffer that stores the data bytes (0 to 8 max)

CAN ID



Control Byte 1





Declaring this function in Visual Basic

```
Private Declare Function SendCANMess Lib "ZCAN4USBFX.dll" (
    ByVal InterfaceNum As Byte,
    ByVal DeviceNum As Byte,
    ByVal ID As Long,
    ByVal CB1 As Byte,
    ByVal _ CB2 As Byte,
    ByVal buffer As Byte)
    As Integer
```

NOTE: The databuffer must be declared as a ByRef because error codes will be saved to this buffer when returned. See Returned Values for more details.

Calling this function from Visual Basic

result = SendCANMess(InterfaceNum, DeviceNum, ID, CB1, CB2, databytes(0))
 where InterfaceNum, DeviceNum, CB1, CB2 are declared as Byte variables and result is declared as an integer variable. ID is a Long and databytes is declared as a byte array

example:

```
Dim ID As Long
Dim CB1,CB2 As Byte
Dim databytes(8) As Byte
Dim result As Integer
```

ID = &H80004120 ' example CAN ID with 29 bit format

CB1 = 4 ' enable txwait bit

CB2 = &H03 'use Message object 0 and send three bytes

databytes(0) = &HFF

databytes(1) = &HA

databytes(2) = &H50

result = SendCANMess(InterfaceNum, DeviceNum, ID, CB1, CB2, databytes(0))

Returned Values

This function will return a value of

- a) 1 for success
- b) Negative value for error (see table at the beginning of this document)
- c) A value of zero if an error occurred within the actual transmission of the CAN message. If this is the case, the first two locations of the databytes buffer will contain 2 error codes as follows
 - 1) MCP2515 TXBnCTRL register
 - 2) MCP2515 TEC register

See the MCP2515 data sheet for more details.

RecCANMess(age)

This command will set up a receive message object using the message object specified. Any messages received through this message object will be stored in a buffer until the Get CAN Message command is used. The Receive CAN Message command is only used to set the message object up with the desired CAN ID.

The C function within the DLL looks like this

```
short int __stdcall
RecCANMess(
    unsigned char InterfaceNum,
    unsigned char DeviceNum,
    long ID,
    unsigned char CB1
)
```

InterfaceNum(ber) will be a value from 0-9 for depending on how many CAN-4-USB/FX devices are connected to the same computer. If there is only one device, this value should be set to zero.

DeviceNum(ber) is a future option for devices that will incorporate more than one CAN controller on a single device. For now, use the value zero.

ID is the CAN ID (right justified) with the top 3 bits defined below

CB1 is short for Command Byte 1 and is shown below

CAN ID

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0									
^	^	^	Extended 29 bit ID																		Standard 11 bit ID									
			Bit 29, reserved for future use																											
			Bit 30, reserved for future use																											
			Bit 31, 1=Extended 29 bit ID, 0=Standard 11 bit ID																											

Control Byte 1

7	6	5	4	3	2	1	0	← Bit position
↑	↑	↑	↑	↑	↑	↑	↑	
								BUKT: Rollover enable. If the message object being set is message object zero within the MCP2515 and you want any incoming messages to 'rollover' to message object 1 if message object 0 is full, set this bit.
								Bits 1 and 2 control the mode with the following:
								11 = Turn mask/filters off, receive all messages
								10 = Receive only 29 bit IDs that meet filter req.
								01 = Receive only 11 bit IDs that meet filter req.
								00 = Receive either 11 or 29 bit ID's that meet filt.
								1=Extended 29 bit ID, 0=Standard 11 bit ID
								Message object to use (0 or 1 for MCP2515)

NOTE: Due to a previous discrepancy between two firmware versions in regards to the setting of the flag to indicate a 29 bit ID's, you can now set either bit (bit 31 in

the ID AND/OR bit 3 in Control Byte 1. If both bits are set to zero the resulting CAN packet will be 11 bit.

Declaring this function in Visual Basic

```
Private Declare Function RecCANMess Lib "ZCAN4USBFX.dll" (  
    ByVal InterfaceNum As Byte,  
    ByVal DeviceNum As Byte,  
    ByVal ID As Long,  
    ByVal CB1 As Byte)  
    As Integer
```

Calling this function from Visual Basic

result = RecCANMess(InterfaceNum, DeviceNum, ID, CB1)

where InterfaceNum, DeviceNum, CB1 are declared as Byte variables and result is declared as an integer variable. ID is a Long.

example:

```
Dim ID As Long
```

```
Dim CB1 As Byte
```

```
ID = &H80004130 ' made up 29 bit ID
```

```
CB1 = 0 ' message object 0, no BUKT, receive either 11 or 29 bit
```

```
result = RecCANMess(InterfaceNum, DeviceNum, ID, CB1)
```

GetCANMess(age)

After a receive message object has been set up and an incoming message has been received and stored into the RAM buffer within the CAN-4-USB/FXdevice, this function will retrieve the oldest message.

The C function within the DLL looks like this

```
short int __stdcall  
GetCANMess(  
    unsigned char InterfaceNum,  
    unsigned char DeviceNum,  
    unsigned char *Buffer  
)
```

InterfaceNum(ber) will be a value from 0-9 for depending on how many CAN-4-USB/FXdevices are connected to the same computer. If there is only one device, this value should be set to zero.

DeviceNum(ber) is a future option for devices that will incorporate more than one CAN controller on a single device. For now, use the value zero.

Buffer is your user buffer that will be filled with the oldest received message (if present). Make sure this buffer is pre-defined to be at least **20 bytes long** or the DLL will over write your memory space.

Note: The data returned from this function is increased from previous versions of the DLL due to the addition of the optional Time Stamp data being returned.

Declaring this function in Visual Basic

```
Private Declare Function GetCANMess Lib "ZCAN4USBFX.dll" (  
    ByVal InterfaceNum As Byte,  
    ByVal DeviceNum As Byte,  
    ByRef buffer As Byte)  
As Integer
```

Note: Make sure buffer is declared as ByRef.

Calling this function from Visual Basic

result = GetCANMess(InterfaceNum, DeviceNum, buffer(0))

where InterfaceNum and DeviceNum are declared as Byte variables and result is declared as an integer variable. Buffer is a Byte declared array of at least 16 bytes

example:

```
Dim buffer(65) As Byte  
Dim result As Integer
```

```
result = GetCANMess(InterfaceNum, DeviceNum, buffer(0))
```

Returned Values

The following values are returned

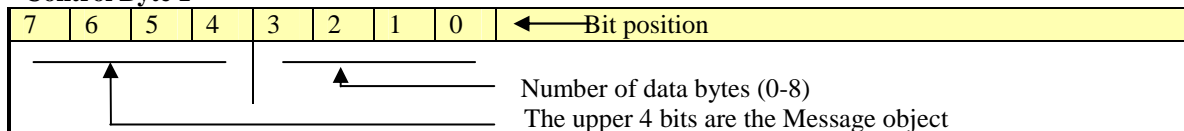
Negative value (see error table at beginning of document)

0= no new messages in buffer

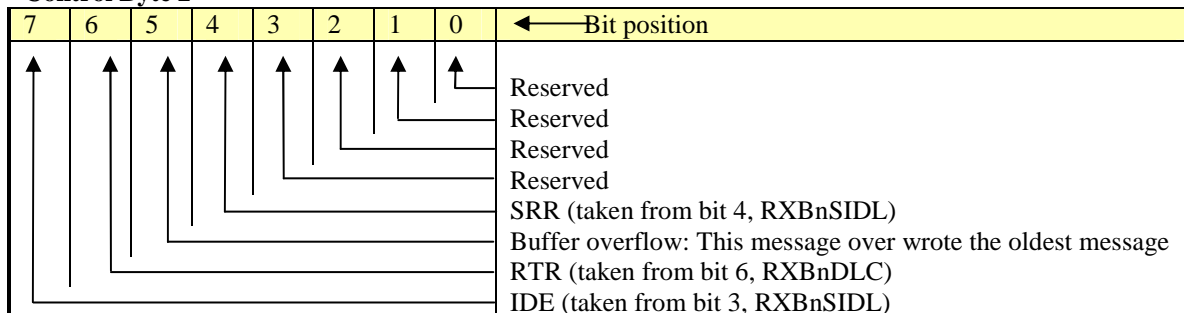
1= means the oldest message has been saved to your buffer according to the following format

Byte loc.	Name
0	Number of bytes in this message including this byte
1	Control Byte 1
2	Control Byte 2
3	CAN ID 1 st (MS) Byte according to format below
4	CAN ID 2 nd Byte
5	CAN ID 3 rd Byte
6	CAN ID 4 th Byte
7...	Data bytes from 0-8 if present
n	3 Byte optional Time Stamp data

Control Byte 1



Control Byte 2



Please see the MCP2515 data sheets for more information on these status bits.

Note: Bit 5 is new for the CAN-4-USB/FX model. The incoming CAN message buffer will eventually loop around and overwrite the oldest message that has not been downloaded to the PC. When a new message is written over the old message, the new message has bit 5 set to show that an old message was lost. The status LED will also blink a quick red pulse to show that at least one message was lost and will continue to blink until the Reset command is used or the interface is unplugged.

CAN ID

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0									
^	^	^	Extended 29 bit ID																		Standard 11 bit ID									
			Bit 29, reserved for future use																											
			Bit 30, reserved for future use																											
			Bit 31, 1=Extended 29 bit ID, 0=Standard 11 bit ID																											

Note: The IDE bit (bit 7) from Control Byte 2 and the Bit 31 from the CAN ID are the same bit.

Time Stamp

If the optional time stamp feature is enabled, there will be an additional 3 bytes that represent the time stamp for the received CAN message in 8 microsecond increments. This value will increment to 100 seconds and then roll over to zero. In order to determine whether the Time Stamp data is present in the packet, your software can compare the number of bytes returned (first byte of the returned buffer) and the number of bytes in the CAN message. For example, if the number of bytes in the entire packet is equal to the number of bytes in the CAN message plus 10, then the time stamp is present. (because there are 10 bytes in the header + time stamp)

One method for calculating the Time Stamp value would be to take the 3 bytes of data and use the following VB formula

NumBytes = DataBuf(1) And &HF ' get DLC, actual number of bytes in packet

If DataBuf(0) = NumBytes + 10 Then ' check if there are three extra bytes (timestamp)

' because packet will consist of 1Numbytes+Data+4ID+2CB+3 time stamp

TimeStamp = DataBuf(7 + NumBytes) * 0.524288

TimeStamp = TimeStamp + (DataBuf(8 + NumBytes) * 0.002048)

TimeStamp = TimeStamp + (DataBuf(9 + NumBytes) * 0.000008)

ReadReg(ister)

This function will read one or more registers from the CAN controller up to a maximum of 62 bytes. By sending a starting address, the number of bytes and an empty Byte array, the DLL will fill your array with the values read back from the CAN controller.

The C function within the DLL looks like this

short int __stdcall

ReadReg(

unsigned char InterfaceNum,

unsigned char DeviceNum,

unsigned char Address,

unsigned char Numbytes,

*unsigned char *buffer*

)

InterfaceNum(ber) will be a value from 0-9 for depending on how many CAN-4-USB/FX devices are connected to the same computer. If there is only one device, this value should be set to zero.

DeviceNum(ber) is a future option for devices that will incorporate more than one CAN controller on a single device. For now, use the value zero.

Address is the start address of the CAN register to be read from.

Numbytes is the number of bytes to read back from 1-62.

Buffer is your user buffer that will be filled with the data read. Make sure this buffer is pre-defined to be at least as long as the number of bytes that are being requested, if not, the DLL will over write your memory space.

Declaring this function in Visual Basic

```
Private Declare Function ReadReg Lib "ZCAN4USBFX.dll" (
```

```
    ByVal InterfaceNum As Byte,
```

```
    ByVal DeviceNum As Byte,
```

```
    ByVal Address As Byte,
```

```
    ByVal NumBytes As Byte,
```

```
    ByRef buffer As Byte)
```

```
As Integer
```

Note: Make sure buffer is declared as ByRef.

Calling this function from Visual Basic

result = ReadReg(InterfaceNum, DeviceNum, Address, NumBytes, buffer(0))

where InterfaceNum, DeviceNum, Address & Numbytes are declared as Byte variables and result is declared as an integer variable. Buffer is a Byte declared array of at least as large as the number of bytes being requested.

example:

Dim buffer(65) As Byte 'we'll send an empty buffer and the function will fill it

Dim result As Integer

Dim Address As Byte

Dim NumBytes As Byte

Address = 0 ' starting address within the CAN controller 0-255

NumBytes = 4 ' hardcoded as an example

result = ReadReg(InterfaceNum, DeviceNum, Address, NumBytes, buffer(0))

WriteReg(ister)

This function will write one or more registers into the CAN controller up to a maximum of 62 bytes. By sending a starting address, the number of bytes and a Byte array with data, the DLL will write the data to the CAN controller. Note that the function will not read back or verify the data written due to the fact that for some registers the value read back may not match the value written due to changing bits or non write-able bits. Also note that the CAN controller will be put into the proper configuration mode in order to write to registers that are normally protected.

The C function within the DLL looks like this

short int __stdcall

```
WriteReg(
    unsigned char InterfaceNum,
    unsigned char DeviceNum,
    unsigned char Address,
    unsigned char Numbytes,
    unsigned char *buffer
)
```

InterfaceNum(ber) will be a value from 0-9 for depending on how many CAN-4-USB/FX devices are connected to the same computer. If there is only one device, this value should be set to zero.

DeviceNum(ber) is a future option for devices that will incorporate more than one CAN controller on a single device. For now, use the value zero.

Address is the start address of the CAN register to write to.

Numbytes is the number of bytes to write to from 1-62.

Buffer is your user buffer that will contain the data to be written.

Declaring this function in Visual Basic

```
Private Declare Function WriteReg Lib "ZCAN4USBFX.dll" (
    ByVal InterfaceNum As Byte,
    ByVal DeviceNum As Byte,
    ByVal Address As Byte,
    ByVal NumBytes As Byte,
    ByVal buffer As Byte)
    As Integer
```

Calling this function from Visual Basic

result = WriteReg(InterfaceNum, DeviceNum, Address, NumBytes, buffer(0))

where InterfaceNum, DeviceNum, Address & Numbytes are declared as Byte variables and result is declared as an integer variable. Buffer is a Byte declared array.

example:

Dim buffer(65) As Byte

Dim result As Integer

Dim Address As Byte

Dim NumBytes As Byte

Address = 0 ' starting address within the CAN controller 0-255

NumBytes = 4 ' hardcoded as an example

buffer(0) = &H11

buffer(1) = &H22

buffer(2) = &H33

buffer(3) = &H44

result = WriteReg(InterfaceNum, DeviceNum, Address, NumBytes, buffer(0))

GetNumCANMess(ages)

This function will return a value representing the number of messages that are currently being stored in the RAM buffer. This function does not return any of the messages.

The C function within the DLL looks like this

```
short int __stdcall  
GetNumCANMess(  
    unsigned char InterfaceNum,  
    unsigned char DeviceNum  
)
```

InterfaceNum(ber) will be a value from 0-9 for depending on how many CAN-4-USB/FX devices are connected to the same computer. If there is only one device, this value should be set to zero.

DeviceNum(ber) is a future option for devices that will incorporate more than one CAN controller on a single device. For now, use the value zero.

Declaring this function in Visual Basic

```
Private Declare Function GetNumCANMess Lib "ZCAN4USBFX.dll" (  
    ByVal InterfaceNum As Byte,  
    ByVal DeviceNum As Byte)  
    As Integer
```

Calling this function from Visual Basic

```
result = GetNumCANMess(InterfaceNum, DeviceNum)
```

where InterfaceNum & DeviceNum as Byte variables and result is declared as an integer variable.

example:

```
Dim result As Integer
```

```
result = GetNumCANMess(InterfaceNum, DeviceNum)
```

Returned Values

A negative returned value is an error according to the table at the beginning of this document.

0 means no messages in buffer

>0 is the number of messages in buffer

ClearCANMess(ages)

This function will clear the receive buffer of all messages. This function does not return any of the messages.

The C function within the DLL looks like this

```
short int __stdcall  
ClearCANMess(  
    BYTE InterfaceNum,  
    BYTE DeviceNum  
)
```

InterfaceNum(ber) will be a value from 0-9 for depending on how many CAN-4-USB/FX devices are connected to the same computer. If there is only one device, this value should be set to zero.

DeviceNum(ber) is a future option for devices that will incorporate more than one CAN controller on a single device. For now, use the value zero.

Declaring this function in Visual Basic

```
Private Declare Function ClearCANMess Lib "ZCAN4USBFX.dll" (  
    ByVal InterfaceNum As Byte,  
    ByVal DeviceNum As Byte)  
    As Integer
```

Calling this function from Visual Basic

```
result = ClearCANMess(InterfaceNum, DeviceNum)
```

where InterfaceNum & DeviceNum as Byte variables and result is declared as an integer variable.

example:

```
Dim result As Integer
```

```
result = ClearCANMess(InterfaceNum, DeviceNum)
```

GetInfo

This function will retrieve information about this device including firmware version, feature bits, manufacture's name and type(s) of CAN controllers. New for this version is the ability to show whether the device is running in USB Full Speed mode or USB High Speed mode (480Mbps)

The C function within the DLL looks like this

```
short int __stdcall
GetInfo(
    unsigned char InterfaceNum,
    unsigned char DeviceNum,
    unsigned char *Version,
    unsigned char Feature ,
    unsigned char *Manufact,
    unsigned char *CANCont
)
```

InterfaceNum(ber) will be a value from 0-9 for depending on how many CAN-4-USB/FX devices are connected to the same computer. If there is only one device, this value should be set to zero.

DeviceNum(ber) is a future option for devices that will incorporate more than one CAN controller on a single device. For now, use the value zero.

Version is a pointer to a three byte array.

Feature is a pointer to a single byte variable that the function will write to based on the available features of this interface.

Manufact(ure) is a pointer to a byte array that will be filled with up to a 20 characters of the name of the manufacture. This is in an effort to create a more standard protocol. This array should be pre-defined to be at least 21 characters long.

CANCont(rollers) is a byte array that will contain the number of CAN controllers and the type of CAN controllers. This array should be pre-defined to be at least 10 characters long.

Declaring this function in Visual Basic

```
Private Declare Function GetInfo Lib "ZCAN4USBFX.dll" (
    ByVal InterfaceNum As Byte,
    ByVal DeviceNum As Byte,
    ByRef Version As Byte,
    ByRef Feature As Byte,
    ByRef Manufact As Byte,
    ByRef CANCont As Byte)
    As Integer
```

Calling this function from Visual Basic

result = GetInfo(InterfaceNum, DeviceNum, Version(0), Feature, Manufact(0), CANCont(0))

where InterfaceNum & DeviceNum as Byte variables and result is declared as an integer variable. Version is a 3 byte array, feature is a byte variable, manufacture is a 21 byte array and CANCont is a 10 byte array.

example:

Dim result As Integer

Dim Version(3) As Byte

Dim Feature As Byte

Dim Manufact(22) As Byte ' make sure this is at least 21 bytes long

Dim CANCont(10) As Byte ' first byte is number of CAN controllers, 2nd+=type

result = GetInfo(InterfaceNum, DeviceNum, Version(0), Feature, Manufact(0), CANCont(0))

See sample program for complete details.

Version

Byte loc.	Name
0	Major version number
1	Minor version number

Feature Flags:

7	6	5	4	3	2	1	0	← Bit position
↑	↑	↑	↑	↑	↑	↑	↑	If =1 then capable of time stamping incoming CAN message (CAN-4-USB/FX-MCP2515 is capable of this)
↑	↑	↑	↑	↑	↑	↑	↑	If =1 then capable of generating an interrupt upon reception of a CAN message (this applies to a RS232 device, not a USB device)
↑	↑	↑	↑	↑	↑	↑	↑	If =1 then capable of full duplex communication (this applies to a RS232 device, not a USB device)
↑	↑	↑	↑	↑	↑	↑	↑	Reserved
↑	↑	↑	↑	↑	↑	↑	↑	Reserved
↑	↑	↑	↑	↑	↑	↑	↑	Reserved
↑	↑	↑	↑	↑	↑	↑	↑	Reserved
↑	↑	↑	↑	↑	↑	↑	↑	Reserved

Manufacturer

The CAN-4-USB/FX-MCP2515 will return one of two normal messages or two error messages, depending on the current speed of the device which is determined by what kind of USB port it is plugged into

Normal:

For USB Hi Speed port “Zanthic FX@HS” (480Mbps)

For USB Full Speed “Zanthic FX@FS” (12Mbps)

Error:

RAM buffer failure will return “Zanthic FX RAM FAIL”

CAN controller failure will return “Zanthic CAN FAIL”

CAN Controller

Byte loc.	Name
0	Number of CAN controllers in this device
1	Type of CAN controller from following list
2...	If more than one CAN controller present, list the type in next bytes

Type of CAN Controller: (This list is taken from the CANPIE specification)

Value returned	CAN Controller	Manufacturer
0	82C200	Philips
1	SJA1000	Philips
2	80C591	Philips
3	80C592	Philips
16 (0x10)	MCP2510	MicroChip
21 (0x15)	MCP2515	MicroChip
20	C505C	Infineon
22	C161	Infineon
23	C164	Infineon
24	C167	Infineon
25	81C90	Infineon
26	81C91	Infineon
40	AN82527	Intel
41	AN87C196CA	Intel
42	AN87C196CB	Intel
60	68HC05	Motorola
61	68HC08	Motorola
62	68HC912	Motorola
63	68376	Motorola
64	MPC555	Motorola

SetFilters

This function allows you to program the acceptance filters and masks within the CAN controller. Although you could write directly to the registers using the WriteReg, the SetFilters function will format the data from a 4 byte ID to the proper configuration within the registers of the MCP2515 (saving you the hassle). The low level protocol actually allows us to write up to 8 filters with one command but this function is fixed as one at a time. The reason is because you don't need to do this more than once at startup (usually) so speed is not required. Note that the filter you are programming is identified with the value in FilterNum, FilterNum=0 for RXF0, =1 for RXF1 etc. Note also that the filterVal is either an 11 bit value (right justified) or a 29 bit value (right justified). If a 29 bit value is used, set the MSB. The function will look after putting the bits in the correct registers for you as noted above. If you read the values back directly using the ReadReg command, they may not be the same as the MCP2515 has a non-standard way of storing the values.

The C function within the DLL looks like this

short int __stdcall

SetFilters(

unsigned char InterfaceNum,

unsigned char DeviceNum,

unsigned char FilterNum,

long FilterVal

)

InterfaceNum(ber) will be a value from 0-9 for depending on how many CAN-4-USB/FX devices are connected to the same computer. If there is only one device, this value should be set to zero.

DeviceNum(ber) is a future option for devices that will incorporate more than one CAN controller on a single device. For now, use the value zero.

FilterNum(ber) will be a value between 0 and 5 for the MCP2510

FilterVal(ue) will be an 11 or 29 bit filter value as shown below

FilterNum Values

Byte loc.	Name
0	Acceptance Filter RXF0
1	Acceptance Filter RXF1
2	Acceptance Filter RXF2
3	Acceptance Filter RXF3
4	Acceptance Filter RXF4
5	Acceptance Filter RXF5

FilterVal			3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0												
^	^	^	Extended 29 bit ID																			Standard 11 bit ID											
			Bit 29, reserved for future use																														
			Bit 30, reserved for future use																														
			Bit 31, 1=Extended 29 bit ID, 0=Standard 11 bit ID																														

Declaring this function in Visual Basic

```
Private Declare Function SetFilters Lib "ZCAN4USBFX.dll" (
    ByVal InterfaceNum As Byte,
    ByVal DeviceNum As Byte,
    ByVal FilterNum As Byte,
    ByVal FilterVal As Long)
    As Integer
```

Calling this function from Visual Basic

```
result = SetFilters(InterfaceNum, DeviceNum, FilterNum, FilterVal)
```

where InterfaceNum & DeviceNum as Byte variables and result is declared as an integer variable. Filternum is a byte variable and FilterVal is a long.

example:

```
Dim result As Integer
Dim FilterVal As Long
Dim FilterNum As Byte
```

```
FilterVal = &H80000000 ' set high to mark this as 29 bit
FilterVal = FilterVal + &H11223344 ' 29 bit
'FilterVal=&H7FFF ' 11 bit
FilterNum = 0 ' hardcoded as an example
result = SetFilters(InterfaceNum, DeviceNum, FilterNum, FilterVal)
```

New Commands to the CAN-4-USB FX interface

EnableTS

This function will enable the time stamp feature. Each incoming CAN message will be assigned a timer value with a 8 micro second resolution. This 24bit value will count to 100 seconds and then roll over to zero

The C function within the DLL looks like this

short int __stdcall

```
EnableTS(  
    BYTE InterfaceNum,  
    BYTE DeviceNum  
)
```

InterfaceNum(ber) will be a value from 0-9 for depending on how many CAN-4-USB/FX devices are connected to the same computer. If there is only one device, this value should be set to zero.

DeviceNum(ber) is a future option for devices that will incorporate more than one CAN controller on a single device. For now, use the value zero.

Declaring this function in Visual Basic

```
Private Declare Function EnableTS Lib "ZCAN4USBFX.dll" (  
    ByVal InterfaceNum As Byte,  
    ByVal DeviceNum As Byte)  
    As Integer
```

Calling this function from Visual Basic

result = EnableTS(InterfaceNum, DeviceNum)

where InterfaceNum & DeviceNum as Byte variables and result is declared as an integer variable.

example:

```
Dim result As Integer
```

```
result = EnableTS(InterfaceNum, DeviceNum)
```

DisableTS

This function will disable the time stamp feature.
The C function within the DLL looks like this

```
short int __stdcall  
DisableTS(  
    BYTE InterfaceNum,  
    BYTE DeviceNum  
)
```

InterfaceNum(ber) will be a value from 0-9 for depending on how many CAN-4-USB/FX devices are connected to the same computer. If there is only one device, this value should be set to zero.

DeviceNum(ber) is a future option for devices that will incorporate more than one CAN controller on a single device. For now, use the value zero.

Declaring this function in Visual Basic

```
Private Declare Function DisableTS Lib "ZCAN4USBFX.dll" (  
    ByVal InterfaceNum As Byte,  
    ByVal DeviceNum As Byte)  
    As Integer
```

Calling this function from Visual Basic

```
result = DisableTS(InterfaceNum, DeviceNum)
```

where InterfaceNum & DeviceNum as Byte variables and result is declared as an integer variable.

example:

```
Dim result As Integer
```

```
result = DisableTS(InterfaceNum, DeviceNum)
```