SILICON ENGINES

CAN/USB CONVERTER

MODEL 9012

PROGRAMMER'S REFERENCE MANUAL

WINDOWS® COMPATIBILITY

The Model 9012 is compatible with Windows 2000, Windows XP, Windows Vista, and Windows 7.

COMMENTS

We would appreciate receiving corrections and suggestions regarding this document and the product it describes. Please email to sales@siliconengines.net

SILICON ENGINES

3550 W. Salt Creek Lane
Suite 105
Arlington Heights, IL 60005 USA
847-637-1180
FAX 847-637-1185
www.siliconengines.net

CONTENTS

1. OVERVIEW

1.1. INTRODUCTION

This document describes the CANUSB DLL and the information necessary to write an application program that uses the Silicon Engines Model 9012 CAN/USB Converter.

Note that on the installation CD for this product, there is included full source code for the Silicon Engines CAN/USB Message Center, which contains a lot of usage examples and concrete code that can be re-used in a customer's own application program. The programmer who must write their own application for use with the CAN/USB Converter is highly encouraged to start with the full source code provided on the installation CD as a starting point.

1.2. USER'S GUIDE

For general operating instructions, please refer to the User's Guide—GD9012.

2. DESCRIPTION OF ROUTINES

2.1. CAN USB OPEN

Function: CanUsbOpen

Library: CANUSB.DLL

C/C++ Calling Format:

```
int __stdcall CanUsbOpen(S16 * handle, /* [input/output] unique id */
char SN[6], /* [input/output] module serial num */
char errorstring[128]); /* [output] error string if error */
```

VB6 Calling Format:

```
Public Declare Function CanUsbOpen _
Lib "CANUSB.DLL" (ByRef handle As Integer, _
ByRef SerialNumber As Byte, _
ByRef errorstring As Byte) _
As Integer 'returns 1 if ok
' put -1 in handle before call to get first available device or handle of device
' to connect to or to check connectivity to also can fill in SN with SN to connect to
' put 0 in first byte of SerialNumber array to indicate connect to first available serial number
' returns 1 if valid handle to connected device otherwise errorstring populated
' if connected SN buffer is populated with SN of device connected to
```

VB .NET Calling Format:

```
Public Declare Function CanUsbOpen Lib "CANUSB.DLL" (ByVal
handle As IntPtr, ByVal SerialNumber As IntPtr, ByVal
errorstring As IntPtr) As Short 'returns 1 if ok

  ' put -1 in handle before call to get first available device
or handle of device

  ' to connect to or to check connectivity to, also can fill in
SN with SN to connect to

  ' put 0 in first byte of SerialNumber array to indicate
connect to first available serial number

  ' returns 1 if valid handle to connected device otherwise
errorstring populated

' if connected SN buffer if populated with SN of device
connected to

' handle is 16-bit

' SerialNumber is 8-bit wide

  ' errorstring is 8-bit wide
```

Description: Use this function to (a) discover if any CAN/USB devices are connected to the computer and if so, obtain a handle to one of them, or (b) discover whether a CAN/USB device with a specified serial number is connected to the computer, and if so, obtain a handle to it, or (c) discover whether a CAN/USB device whose handle has already been obtained is still connected to the computer (although you can use the less intrusive function CanUsbIsOpen for the same purpose). You should populate the value of handle with -1 to obtain the first available handle or a non-negative handle to open a previously opened device. You should make a null-terminated C string with the SerialNumber and set the very first byte (at array location 0) to a null character if you don’t already know the serial number of the device you are trying to connect to.

Return Code: This function returns a standard CAN/USB return code. A return code of “1” indicates success. Other values indicate specific errors (see Part 3).

Parameter Description:

handle [input/output] - This is the handle of the CAN/USB device being interrogated or set up. If the user wishes to direct the action of this routine to a specific CAN/USB device with a known handle, then that handle number should be put into this argument. If the user wishes to attach to the first available device then they should use a value of -1 for this parameter. On successful return, the variable pointed to by this argument will be set to the handle of the device that was connected to. All positive (zero or non-zero) integer values are legal values for the handle.

SerialNumber [input/output] – This is the serial number string of the CAN/USB device being interrogated or set up. If the user wishes to direct the action of this routine to a CAN/USB device with a specific serial number, then that serial number should be put into a standard C null-terminated ASCII string and pointed to by this argument. If the user wishes to attach to the first available device, then this argument should point to a byte buffer of at least 6 bytes long (5 byte long string with 1 byte null terminator). The serial number is at most 5 bytes long. On successful return, the string pointed to by this argument will contain the serial number of the device that was connected to.

errorstring [output] – The user should allocate a string of at least 128 single-byte characters and have a pointer to this string passed in this parameter. If this routine returns a code other than 1, then errorstring will contain a human-readable standard C null-terminated ASCII string containing a specific error message describing why the routine failed to execute the commanded operation.

2.2. CAN USB IS OPEN

Function: CanUsbIsOpen

Library: CANUSB.DLL

C/C++ Calling Format:

```
int __stdcall CanUsbIsOpen(S16 handle); /* returns 1=OK if
device handle is open */
```

VB6 Calling Format:

```
Public Declare Function CanUsbIsOpen _
Lib "CANUSB.DLL" _
(ByVal handle As Integer) _
As Integer 'returns 1 if open
```

VB .NET Calling Format:

```
Public Declare Function CanUsbIsOpen Lib "CANUSB.DLL" (ByVal
handle As Short) As Short 'returns 1 if open
```

Description: Use this routine to determine whether a given CAN/USB device handle still points to a valid, connected, working CAN/USB device.

Return Code: This function returns a standard CAN/USB return code. A return code of "1" indicates that the CAN/USB device specified by the handle input argument is connected and working properly and an open USB connection exists to this device. Other values indicate specific errors (see Part 3).

Parameter Description:

handle [input] - This is the handle of the CAN/USB device being checked. Only valid handles obtained using CanUsbOpen are accepted by this routine.

2.3. CAN USB SETUP

Function: CanUsbSetup

C/C++ Calling Format:

```
USB_RC __stdcall CanUsbSetup(S16 handle, /* [input] unique id
from CanUsbOpen */
 U8 BS1, /* [input] time for seg 1 (1-16) */
 U8 BS2, /* [input] time for seg 2 (1-8) */
 U8 SJW, /* [input] time for SJW (1-4) */
 S16 Prescaler, /* [input] prescaler for TQ (1-1024) */
S32 Flags, /* [input] mode flags (bit 0: 0=regular 1=automatic
bus off recovery, */
/* bit 1: 0=no termination 1=120 ohm termination, */
/* bit 2: 0=regular 1=auto-incrementing payload in byte 8 for testing) */

S32 XmitTimeout); /* [input] transmit timeout, milliseconds (1-65535) */

/* formula for CAN bit rate is 30000000/(Prescaler*(BS1+BS2+1)) (30 million) */

/* example Prescaler = 2, BS1 = 11, BS2 = 3, SJW = 1 : 1 Mbps (max rate) */

/* returns 1=OK if successful */
```

VB6 Calling Format:

```
Public Declare Function CanUsbSetup _
Lib "CANUSB.DLL" (ByVal handle As Integer, _
ByVal BS1 As Byte, _
ByVal BS2 As Byte, _
ByVal SJW As Byte, _
ByVal Prescaler As Integer, _
ByVal Flags As Long, _
ByVal XmitTimeout As Long) As Integer ' returns 1 if ok
'BS1 = 1-16, BS2 = 1-8, Prescaler = 1-1024, Flags: bit 0=1 for automatic bus-off handling
' bit 1=1 for 120-ohm termination of CAN bus
' bit 2=1 for automatic incrementing payload in 8th byte of transmitted recurring CAN payload
' formula for CAN bit rate is 30000000/(Prescaler*(BS1+BS2+1)) (30 million)
' example Prescaler = 2, BS1 = 11, BS2 = 3 : 1 Mbps
' XmitTimeout is in units of milliseconds
```

VB .NET Calling Format:

```
Public Declare Function CanUsbSetup Lib "CANUSB.DLL" (ByVal handle As Short, ByVal BS1 As Byte, ByVal BS2 As Byte, ByVal SJW As Byte, ByVal Prescaler As Short, ByVal Flags As Integer, ByVal XmitTimeout As Integer) As Short ' returns 1 if ok
'BS1 = 1-16, BS2 = 1-8, Prescaler = 1-1024, Flags: bit 0=1 for automatic bus-off handling
' bit 1=1 for 120-ohm termination of CAN bus
' bit 2=1 for automatic incrementing payload in 8th byte of transmitted recurring CAN payload
' formula for CAN bit rate is 30000000/(Prescaler*(BS1+BS2+1)) (30 million)
' example Prescaler = 2, BS1 = 11, BS2 = 3 : 1 Mbps
' XmitTimeout is in units of milliseconds
```

Description: Use this routine to set up the CAN bus parameters. Typically a call to CanUsbQueryBitRate occurs after this setup call to ensure that the parameters were set up properly. These settings are stored in non-volatile memory in the CAN/USB device so that after reset it will continue to use the same parameters.

Return Code: This function returns a standard CAN/USB return code. A return code of “1” indicates that the CAN/USB device specified by the handle input argument is connected and working properly and an open USB connection exists to this device. Other values indicate specific errors (see Part 3).

Parameter Description:

handle [input] - This is the handle of the CAN/USB device being set up. Only valid handles obtained using CanUsbOpen are accepted by this routine.

BS1 [input] – This is a value from 1-16 indicating the number of TQ (time quanta) for Bit Segment 1. The CAN bit rate is determined by 1,000,000 / (Prescaler * (BS1 + BS2 + 1)). The sampling point is BS1 / (BS1 + BS2 + 1).

BS2 [input] – This is a value from 1-8 indicating the number of TQ (time quanta) for Bit Segment 2. The CAN bit rate is determined by 1,000,000 / (Prescaler * (BS1 + BS2 + 1)). The sampling point is BS1 / (BS1 + BS2 + 1).

SJW [input] – This is a value from 1-4 indicating the Synchronization Jump Width in number of TQ (time quanta). This indicates how much variation the CAN peripheral will allow.

Prescaler [input] – This is a value from 1-1024. The CAN bit rate is determined by 1,000,000 / (Prescaler * (BS1 + BS2 + 1)).

Flags [input] – This is a bit field of flags to use for the CAN peripheral. Bit 0 = 1 for automatic bus-off handling/recovery. Bit 1 = 1 to place a 120 ohm termination resistor across the CAN high and low terminals. If Bit 1 = 0 then no CAN bus terminator is used. A bus terminator should be used if this is an end node in the network. Typically there are 2 devices with CAN bus terminators on either end of the network. Bit 2 = 1 for testing throughput. If Bit 2 = 1 then the 8th byte of the payload automatically increments with each successfully transmitted message. This can be used to test that each message got through properly. Bit 2 = 0 for normal operation.

XmitTimeout [input] – This is the number of milliseconds to allow before giving up for every CAN message transmission.

2.4. CAN USB TRIGGER

Function: CanUsbTrigger

Library: CANUSB.DLL

C/C++ Calling Format:

```
USB_RC __stdcall CanUsbTrigger(S16 handle, /* [input] unique
id from CanUsbOpen */

S32 *TriggerTime); /* [output] timestamp (microseconds) of
rising edge of sent 2 millisecond trigger pulse */
```

VB6 Calling Format:

```
Public Declare Function CanUsbTrigger _

Lib "CANUSB.DLL" (ByVal handle As Integer, _

ByRef Stamp As Long) As Integer 'returns 1 if ok
```

```
'Stamp is in units of microseconds and is the timestamp of the rising edge of the trigger

'signal that is sent on the trigger line in response to this command
```

VB .NET Calling Format:

```
Public Declare Function CanUsbTrigger Lib "CANUSB.DLL" (ByVal handle As Short, ByVal Stamp As IntPtr) As Short 'returns 1 if ok

'Stamp is in units of microseconds and is the timestamp of the rising edge of the trigger

'signal that is sent on the trigger line in response to this command

'Stamp is 32-bits (31 bits used)
```

Description: Use this routine to send a trigger pulse on the trigger output pin of the CAN/USB device. The internal microsecond timestamp that the trigger is sent out is also sent back.

Return Code: This function returns a standard CAN/USB return code. A return code of “1” indicates success. Other values indicate specific errors (see Part 3).

Parameter Description:

handle [input] - This is the handle of the CAN/USB device being accessed. Only valid handles obtained using CanUsbOpen are accepted by this routine.

TriggerTime [output] - This is the internal microsecond timestamp of when the rising edge of the trigger signal was sent out. This can be compared with microsecond timestamps from CAN messages.

2.5. CAN USB SERIAL NUMBER

Function: CanUsbSerialNumber

Library: CANUSB.DLL

C/C++ Calling Format:

```
USB_RC __stdcall CanUsbSerialNumber(S16 handle, U8 SN[6]);
```

VB6 Calling Format:

```
Public Declare Function CanUsbSerialNumber _
Lib "CANUSB.DLL" _
(ByVal handle As Integer, _
ByRef SerialNumber As Byte) _
As Integer 'returns 1 if successful
```

VB .NET Calling Format:

```
Public Declare Function CanUsbSerialNumber Lib "CANUSB.DLL" ( _
ByVal handle As Short, ByVal SerialNumber As IntPtr) As Short
'returns 1 if successful
'SerialNumber is 8-bits wide
```

Description: Use this routine to find out the serial number of a CAN/USB device that has already been opened by CanUsbOpen.

Return Code: This function returns a standard CAN/USB return code. A return code of “1” indicates success. Other values indicate specific errors (see Part 3).

Parameter Description:

handle [input] - This is the handle of the CAN/USB device being queried. Only valid handles obtained using CanUsbOpen are accepted by this routine.

SN [output] – This is a pointer to a buffer of at least 6 bytes to put the standard C null-terminated ASCII string containing the serial number of the CAN/USB device being queried. Upon successful completion of this routine, this field will be written by the routine with the serial number of the device.

2.6. CAN USB STATUS MSG WAITING

Function: CanUsbStatusMsgWaiting

Library: CANUSB.DLL

C/C++ Calling Format:

```
USB_RC __stdcall CanUsbStatusMsgWaiting(S16 handle);

/* returns 1 if a status message is waiting or 0 if no status
message waiting. */
```

VB6 Calling Format:

```
Public Declare Function CanUsbStatusMsgWaiting _
Lib "CANUSB.DLL" _
(ByVal handle As Integer) _
As Integer 'returns 1 if status message is waiting
```

VB .NET Calling Format:

```
Public Declare Function CanUsbStatusMsgWaiting Lib
"CANUSB.DLL" (ByVal handle As Short) As Short 'returns 1 if
status message is waiting
```

Description: Use this routine to determine if a status message is waiting to be read by the application. If so, call CanUsbStatus to read the status message.

Return Code: A return code of “1” indicates that a status message is waiting. A return code of “0” indicates that no status message is waiting. Other values indicate specific errors (see Part 3).

Parameter Description:

handle [input] - This is the handle of the CAN/USB device being queried. Only valid handles obtained using CanUsbOpen are accepted by this routine.

2.7. CAN USB STATUS

Function: CanUsbStatus

Library: CANUSB.DLL

C/C++ Calling Format:

```
USB_RC __stdcall CanUsbStatus(S16 handle, U8 *code);
```

VB6 Calling Format:

```
Public Declare Function CanUsbStatus _
 Lib "CANUSB.DLL" _
 (ByVal handle As Integer, _
 ByRef status As Byte) _
 As Integer 'returns 1 if successful
 'fills status with CAN status code
```

VB .NET Calling Format:

```
Public Declare Function CanUsbStatus Lib "CANUSB.DLL" ( _
ByVal handle As Short, ByVal status As IntPtr) As Short
'returns 1 if successful
'fills status with CAN status code
 'status is 8-bits
```

Description: This routine unobtrusively sees if a status code is waiting in the DLL for this device. The status codes can be found in Part 4. Use CanUsbStatusMsgWaiting first to see if this message is waiting. To actively read the device for a fresh and current status code, use CanUsbReadStatus instead.

Return Code: This function returns a standard CAN/USB return code. A return code of “1” indicates success. A return code of “2” will indicate that no status message is waiting to be read. Other values indicate specific errors (see Part 3).

Parameter Description:

handle [input] - This is the handle of the CAN/USB device being queried. Only valid handles obtained using CanUsbOpen are accepted by this routine.

code [output] – If the routine returns successfully, this will contain one of the codes from Part 4 indicating the current status of the device.

2.8. CAN USB DATA MSG WAITING

Function: CanUsbDataMsgWaiting

Library: CANUSB.DLL

C/C++ Calling Format:

```
USB_RC __stdcall CanUsbDataMsgWaiting(S16 handle);
/* returns 1 if a data message is waiting or 0 if no data
message is waiting.
* returns positive error code greater than 1 if there was an
error.
*/
```

VB6 Calling Format:

```
Public Declare Function CanUsbDataMsgWaiting Lib "CANUSB.DLL"
(ByVal handle As Integer) _
```

```
As Integer 'returns 1 if ok
```

VB .NET Calling Format:

```
Public Declare Function CanUsbDataMsgWaiting Lib "CANUSB.DLL"
(ByVal handle As Short) As Short 'returns 1 if ok
```

Description: Use this routine to determine if a CAN message is waiting to be read. Note that it is usually more efficient to call CanUsbNumReceiveMsgs to see how many CAN messages are waiting so that the appropriate number of read data message calls can be made without losing time.

Return Code: This function returns "1" to indicate a CAN message has already been received and is waiting to be read from the buffer. This function returns 0 if no CAN messages have been received. Other values indicate specific errors (see Part 3).

Parameter Description:

handle [input] - This is the handle of the CAN/USB device being queried. Only valid handles obtained using CanUsbOpen are accepted by this routine.

2.9. CAN USB NUM RECEIVE MSGS

Function: CanUsbNumReceiveMsgs

Library: CANUSB.DLL

C/C++ Calling Format:

```
USB_RC    __stdcall    CanUsbNumReceiveMsgs(S16    handle,    S16
*NumRecvMsgs);

/* if return code is OK then NumRecvMsgs contains number of
CAN data messages waiting to be read */
```

VB6 Calling Format:

```
Public Declare Function CanUsbNumReceiveMsgs Lib "CANUSB.DLL"
(ByVal handle As Integer, _
ByRef NumRecvMsgs As Integer) As Integer 'returns 1 if ok
'returns number of CAN messages waiting to be read in
NumRecvMsgs
```

VB .NET Calling Format:

```
Public Declare Function CanUsbNumReceiveMsgs Lib "CANUSB.DLL"
( _
ByVal handle As Short, ByVal NumRecvMsgs As IntPtr) As Short
 'returns 1 if ok
'returns number of CAN messages waiting to be read in
NumRecvMsgs
'NumRecvMsgs is 16-bits
```

Description: This routine will tell how many CAN messages have already been received in the buffers and are waiting to be read by the application. In cases of high bus utilization this routine is useful to quickly determine how much of the receive buffer is used up and how many read data messages are required to clear the receive buffer. You can use CanUsbReadDataMsg to read each message. If you read more than 100 messages at a time, there may be significant delays noticeable in the user interface, but it may be necessary to pull in many messages to avoid an application receive buffer overflow condition depending on the application's needs and environment.

Return Code: This function returns a standard CAN/USB return code. A return code of "1" indicates success. Other values indicate specific errors (see Part 3).

Parameter Description:

handle [input] - This is the handle of the CAN/USB device being queried. Only valid handles obtained using CanUsbOpen are accepted by this routine.

NumRecvMsgs [output] - If this routine returns successfully, this will contain the number of receive messages that are waiting to be read by the application. Use CanUsbReadDataMsg to read each CAN message.

2.10. CAN USB READ DATA MSG

Function: CanUsbReadDataMsg

Library: CANUSB.DLL

C/C++ Calling Format:

```
USB_RC __stdcall CanUsbReadDataMsg(S16 handle,

U8 *IDE, S32 *ID, U8 *RTR, S32 *timestamp, U8 *count, U8
*buffer, U8 *error_code);

/* count ranges from 0-8 and is output only for
CanUsbReadDataMsg */

/* IDE is either 0 (standard ID) or 1 (extended ID) */

/* RTR is either 0 or 1 */

/* IDE and RTR values are returned */

/* buffer should be allocated for 8 bytes */

/* timestamp units microseconds */

/* IDE, RTR, or timestamp parameter can be NULL if you do not
need this information */

/* and don't want it returned */
```

VB6 Calling Format:

```
Public Declare Function CanUsbReadDataMsg Lib "CANUSB.DLL"
(ByVal handle As Integer, _

ByRef IDE As Byte, ByRef ID As Long, ByRef RTR As Byte, _

ByRef Timestamp As Long, ByRef Count As Byte, ByRef buffer As
Byte, _

ByRef error_code As Byte) As Integer 'returns 1 if ok
```

VB .NET Calling Format:

```
Public Declare Function CanUsbReadDataMsg Lib "CANUSB.DLL" ( _
    ByVal handle As Short, ByVal IDE As IntPtr, _
    ByVal ID As IntPtr, ByVal RTR As IntPtr, _
    ByVal Timestamp As IntPtr, ByVal Count As IntPtr, _
    ByVal buffer As IntPtr, ByVal error_code As IntPtr) As Short
'returns 1 if ok
'IDE is 8-bits (value of 0 or 1)
'ID is 32-bits
'RTR is 8-bits (value of 0 or 1)
'Timestamp is 32-bits (31-bits used)
'Count is 8-bits (value from 0-8)
'buffer is 8-bits wide
'error_code is 8-bits
```

Description: This routine will read the next sequential CAN message that had appeared on the bus. **Return Code:** This function returns a standard CAN/USB return code. A return code of “1” indicates that the routine completed successfully. Other values indicate specific errors (see Part 3) although a code of “2” could also mean simply that there were no messages to receive.

Parameter Description:

handle [input] - This is the handle of the CAN/USB device being checked. Only valid handles obtained using CanUsbOpen are accepted by this routine.

IDE [output] - This is returned to indicate if the received CAN messages had a standard identifier (=0) or an extended identifier (=1).

ID [output] - This is returned to indicate the Identifier of the received CAN message. This is 11 bits for a standard ID or 29 bits for an extended ID.

RTR [output] - This indicates if the received CAN message was a remote frame (=1) or a standard frame (=0).

Timestamp [output] - This is the microsecond timestamp that the CAN/USB device recognized the CAN message on the bus.

Count [output] - This is returned to indicate the number of payload bytes in the received CAN message.

buffer [output] – On calling this function, the pointer passed in this parameter must point to a previously allocated buffer of at least 8 bytes. This will be filled with the payload of the CAN message.

error_code [output] - This is returned to indicate if there was an error while receiving this message. The error codes are listed in Part 4.

2.11. CAN USB WRITE DATA MSG

Function: CanUsbWriteDataMsg

Library: CANUSB.DLL

C/C++ Calling Format:

```
USB_RC __stdcall CanUsbWriteDataMsg(S16 handle,
U8 IDE, S32 ID, U8 RTR, S32 RecurrenceTime, S32 TriggerTime,
U8 count, U8 *buffer, U8 *error_code, S16 wait_for_ack,
S32 *Stamp);
/* IDE = 0 for standard ID or 1 for extended ID
* RTR = 0 or 1
 * RecurrenceTime=0 for single (try-til-ack) send or non-zero for recurrence interval in microsecond units
* TriggerTime=0 for no trigger or non-zero for time in microseconds between rising edge of trigger (which lasts 2 milliseconds) until (first) message transmission
* Stamp will return with timestamp in microseconds for when either the Trigger was sent or the message was sent if no trigger
* count=0 to 8 (DLC)
* wait_for_ack=0 or 1: if 0 sends and returns immediately, if 1 then error_code will also contain whether the message was transmitted successfully
* error_code = returned error code
*/
```

VB6 Calling Format:

```
Public Declare Function CanUsbWriteDataMsg _
Lib "CANUSB.DLL" (ByVal handle As Integer, _
ByVal IDE As Byte, ByVal ID As Long, ByVal RTR As Byte, _
ByVal RecurrenceTime As Long, ByVal TriggerTime As Long, _
ByVal Count As Byte, ByRef buffer As Byte, _
ByRef error_code As Byte, wait_for_ack As Integer, ByRef Stamp As Long) _
As Integer 'returns 1 if ok
' IDE = 0 for standard ID or 1 for extended ID
' RTR = 0 for standard frame 1 for remote frame
' RecurrenceTime=0 for single (try-til-ack) send or non-zero for recurrence interval in microsecond units
' TriggerTime=0 for no trigger or non-zero for time in microseconds between rising edge of trigger (which lasts 2 milliseconds) until (first) message transmission
' Stamp will return with timestamp in microseconds for when either the Trigger was sent or the message was sent if no trigger
' count=0 to 8 (DLC) number of bytes in payload, buffer is
array of count bytes containing transmit payload

' wait_for_ack=0 or 1: if 0 sends and returns immediately, if
1 then error_code will also contain whether the message was
transmitted successfully

' error_code = returned error code
```

VB .NET Calling Format:

```
Public Declare Function CanUsbWriteDataMsg Lib "CANUSB.DLL" ( _
    ByVal handle As Short, ByVal IDE As Byte, _
    ByVal ID As Integer, ByVal RTR As Byte, _
    ByVal RecurrenceTime As Integer, _
    ByVal TriggerTime As Integer, ByVal Count As Byte, _
    ByVal buffer As IntPtr, ByVal error_code As IntPtr, _
  ByVal wait_for_ack As Short, ByVal Stamp As IntPtr) As Short
'returns 1 if ok

' IDE = 0 for standard ID or 1 for extended ID

' RTR = 0 for standard frame 1 for remote frame

' RecurrenceTime=0 for single (try-til-ack) send or non-zero
for recurrence interval in microsecond units

' TriggerTime=0 for no trigger or non-zero for time in
microseconds between rising edge of trigger (which lasts 2
milliseconds) until (first) message transmission

' Stamp (32-bit, 31 bits used) will return with timestamp in
microseconds for when either the Trigger was sent or the
message was sent if no trigger

' count=0 to 8 (DLC) number of bytes in payload, buffer is
array of count bytes containing transmit payload

' wait_for_ack (16-bit)=0 or 1: if 0 sends and returns
immediately, if 1 then error_code will also contain whether
the message was transmitted successfully

' error_code = returned error code (8-bit)
```

Description: This function is used to transmit a message or periodic message on the CAN bus. A non-zero RecurrenceTime value will indicate the number of microseconds between transmitted messages on the CAN bus. A zero RecurrenceTime value will indicate to send a message just a single time. A wait_for_ack argument also allows the message to be a non-blocking call if this is set to 0. An error_code is also returned to indicate any error that has occurred during the message transmission. You can specify a non-zero Trigger time to specify to activate the trigger output, creating a pulse on the trigger output pin of the connector, a specified microsecond interval of time before the CAN message is actually transmitted. A Stamp value in microseconds will be returned indicating the time that the message was successfully transmitted on the bus.

Return Code: This function returns a 1 on success. Other codes (error codes) that it can return are defined in Part 3.

Parameter Description:

handle [input] - This is the handle of the CAN/USB device being commanded. Only valid handles obtained using CanUsbOpen are accepted by this routine.

IDE [input] - This indicates if the CAN message should use a standard identifier (=0) or an extended identifier (=1). A standard identifier has an 11 bit ID and an extended identifier has a 29 bit ID.

ID [input] - This is the identifier to be sent with the CAN message. This is 11 bits for a standard ID or 29 bits for an extended ID.

RTR [input] - This is whether the CAN message being transmitted is a remote frame (=1) or a standard frame (=0).

RecurrenceTime [input] – This value is either 0 to indicate that the message should be transmitted just a single time, or a non-zero value to indicate the number of microseconds between each transmitted message. If a non-zero value is used, you can only stop sending the message by executing a CanUsbStopWriteDataMsg command with the identifier of this message to stop it, or by resetting the CAN/USB device.

TriggerTime [input] – This value is either 0 to indicate no special action or non-zero to indicate the number of microseconds between the rising edge of an active-high trigger signal sent on the trigger output pin and the execution of the CAN transmission.

Count [input] – This is the number of data bytes in the payload of the CAN message (0-8).

buffer [input] – This is a pointer to a buffer of bytes that contain the payload of the CAN message to be transmitted, up to 8 bytes long. If CanUsbSetup was called with bit 2 of the Flags field set to 1, then the 8th byte of the payload will automatically increment after each successfully transmitted message.

error_code [output] - This is returned to indicate if there was an error while receiving this message. The error codes are listed in Part 4.

wait_for_ack [input] – This should be set to 0 to indicate a non-blocking call that returns immediately and does not wait for the acknowledgement message (this can be received separately) or 1 to indicate a blocking call that waits for the acknowledgement of the message (or the acknowledgement of the first message in the case of a recurring message). The time that the routine will wait in the case that this argument was set to 1 is up to the transmit timeout (XmitTimeout) defined in the CanUsbSetup call.

Stamp [output] - This is the microsecond timestamp that the CAN/USB device successfully sent the CAN message on the bus, in the case that this routine returned a 1 success code.

2.12. CAN USB STOP WRITE DATA MSG

Function: CanUsbStopWriteDataMsg

Library: CANUSB.DLL

C/C++ Calling Format:

```
USB_RC __stdcall CanUsbStopWriteDataMsg(S16 handle,
U8 IDE, S32 ID, U8 RTR, U8 *error_code);
/* use this to stop a recurring message */
```

VB6 Calling Format:

```
Public Declare Function CanUsbStopWriteDataMsg _
Lib "CANUSB.DLL" (ByVal handle As Integer, _
ByVal IDE As Byte, ByVal ID As Long, ByVal RTR As Byte, _
ByRef error_code As Byte) As Integer 'returns 1 if ok
```

VB .NET Calling Format:

```
Public Declare Function CanUsbStopWriteDataMsg Lib "CANUSB.DLL" (ByVal handle As Short, ByVal IDE As Byte, ByVal ID As Integer,
ByVal RTR As Byte, ByVal error_code As IntPtr) As Short 'returns 1 if ok
'error_code is 8-bits
```

Description: This function is used to stop transmission of a recurring message or a message that has not yet successfully been transmitted before the XmitTimeout interval. There is an error_code argument that is returned as well to indicate additional error information such as code 53H to indicate that no such CAN message was running so nothing was stopped.

Return Code: This function returns a standard CAN/USB return code. A return code of “1” indicates success. Other values indicate specific errors (see Part 3). Note that if the return code is 1, you should still check the error_code argument for additional errors, such as 53H to indicate that no such CAN message was running so nothing was stopped.

Parameter Description:

handle [input] - This is the handle of the CAN/USB device being checked. Only valid handles obtained using CanUsbOpen are accepted by this routine.

IDE [input] - This indicates if the CAN message being stopped used a standard identifier (=0) or an extended identifier (=1). A standard identifier has an 11 bit ID and an extended identifier has a 29 bit ID.

ID [input] - This is the identifier of the CAN message to stop transmission of. This is 11 bits for a standard ID or 29 bits for an extended ID.

RTR [input] - This is whether the CAN message being transmitted that is being stopped is a remote frame (=1) or a standard frame (=0).

error_code [output] - This is returned to indicate if there was an error while stopping this message, such as 53H to indicate that no such message was running so nothing was stopped. The error codes are listed in Part 4.

2.13. CAN USB ACK WAITING

Function: CanUsbAckWaiting

Library: CANUSB.DLL

C/C++ Calling Format:

```
USB_RC __stdcall CanUsbAckWaiting(S16 handle, U8 *msg_acked,
U8 *error_code, S32 *ID, U8 *MsgProps, S32 *XmitTimestamp);
```

```
* if this routine returns OK, then msg_acked, MsgProps, error_code, and ID are updated.

* if this routine returns GENERR, then no ack message is waiting.

 * If the error_code pointer is not NULL then the error_code is filled with the error code * for the message ID that was acked.  An error code of 4FH means everything    * was okay.

* if msg_acked = CBH this is a CAN message acknowledgement and ID and MsgProps is filled

* MsgProps has 2 bits that are used: bit 0 = 0 for standard IDE or 1 = extended ID

* bit 1 = 0 for data frame or 1 for remote frame

* XmitTimestamp is in units of microseconds

*/
```

VB6 Calling Format:

```
Public Declare Function CanUsbAckWaiting Lib "CANUSB.DLL" (ByVal handle As Integer, _

ByRef msg_acked As Byte, ByRef error_code As Byte, ByRef ID As Long, _

ByRef MsgProps As Byte, ByRef Timestamp As Long) As Integer 'returns 1 if ok

'if this routine returns OK, then msg_acked, MsgProps, error_code, and ID are updated.

'if this routine returns GENERR, then no ack message is waiting.

'If the error_code pointer is not NULL then the error_code is filled with the error code

'for the message ID that was acked.  An error code of 4FH means everything was okay.

'if msg_acked = CBH this is a CAN message acknowledgement and ID and MsgProps is filled

'MsgProps has 2 bits that are used: bit 0 = 0 for standard IDE or 1 = extended ID

'bit 1 = 0 for data frame or 1 for remote frame
```

VB .NET Calling Format:

```
Public Declare Function CanUsbAckWaiting Lib "CANUSB.DLL" ( _

    ByVal handle As Short, _

    ByVal msg_acked As IntPtr, _

    ByVal error_code As IntPtr, _

    ByVal ID As IntPtr, _

    ByVal MsgProps As IntPtr, _
    ByVal Timestamp As IntPtr) As Short 'returns 1 if ok
'if this routine returns OK, then msg_acked, MsgProps, error_code, and ID are updated.
'if this routine returns GENERR, then no ack message is waiting.
'If the error_code pointer is not NULL then the error_code is filled with the error code 'for the message ID that was acked. An error code of 4FH means everything 'was okay.
'if msg_acked = CBH this is a CAN message acknowledgement and ID and MsgProps is filled
'MsgProps has 2 bits that are used: bit 0 = 0 for standard IDE or 1 = extended ID
'bit 1 = 0 for data frame or 1 for remote frame
'msg_acked is 8-bits
'error_code is 8-bits
'ID is 32-bits
'MsgProps is 8-bits
'Timestamp is 32-bits (31-bits used)
```

Description: This function should be called periodically to see if any acknowledgements of messages are waiting, such as acknowledgements of CAN transmissions or commands that have been sent to the CAN/USB device.

Return Code: This function returns a standard CAN/USB return code. A return code of “1” indicates success and that an acknowledgement was waiting (check the msg_acked field). A code of “2” can indicate that there are no acknowledged message notices queued up currently. Other values indicate specific errors (see Part 3).

Parameter Description:

handle [input] - This is the handle of the CAN/USB device being checked. Only valid handles obtained using CanUsbOpen are accepted by this routine.

msg_acked [output] – If the return code from this routine is 1, this will indicate the message that was acknowledged. For a CAN message, this value will be CBH. For a complete list of possible values for this field, see Part 5.

error_code [output] – This error code is equal to 4F hex if there is no error. For a list of error codes, see Part 4. Often this error code reflects an error that occurred while receiving the current message.

ID [output] – If the return code for this routine was 1, and msg_acked = CBH, then this will contain the ID of the CAN message that has been acknowledged as having been transmitted by the CAN/USB device. This was either because a previous call to CanUsbWriteDataMsg was called with the wait_for_ack argument set to 0 or a recurring message is in progress from another all to CanUsbWriteDataMsg.

MsgProps [output] - If the return code for this routine was 1, and msg_acked = CBH, then this will contain the message properties of the acknowledged CAN message in bit format, only 2 bits of significance. Bit 0 is 0 to indicate a standard identifier was transmitted or 1 to indicate an extended identifier was transmitted (see CanUsbWriteDataMsg). Bit 1 is 0 to indicate a standard data frame was transmitted or 1 to indicate a remote frame was transmitted.

Timestamp [output] – This the 32-bit microsecond timestamp that the message was transmitted.

2.14. CAN USB READ STATUS

Function: CanUsbReadStatus

Library: CANUSB.DLL

C/C++ Calling Format:

```
USB_RC __stdcall CanUsbReadStatus(S16 handle, U8 *code);
```

VB6 Calling Format:

```
Public Declare Function CanUsbReadStatus _
Lib "CANUSB.DLL" _
(ByVal handle As Integer, _
ByRef status As Byte) _
As Integer 'returns 1 if successful
'fills status with CAN status code
```

VB .NET Calling Format:

```
Public Declare Function CanUsbReadStatus Lib "CANUSB.DLL" ( _
  ByVal handle As Short, ByVal status As IntPtr) As Short
'returns 1 if successful
'fills status with CAN status code
 'status is 8-bits
```

Description: This routine actively queries the device and returns a fresh status code from the device. The status codes can be found in Part 4. For a more unobtrusive call that just checks to see if a status code is available already having been read, use CanUsbStatusMsgWaiting and CanUsbStatus.

Return Code: This function returns a standard CAN/USB return code. A return code of “1” indicates success. Other values indicate specific errors (see Part 3).

Parameter Description:

handle [input] - This is the handle of the CAN/USB device being queried. Only valid handles obtained using CanUsbOpen are accepted by this routine.

code [output] – If the routine returns successfully, this will contain one of the codes from Part 4 indicating the current status of the device.

2.15. CAN USB SET MAX BUF SIZE

Function: CanUsbSetMaxBufSize

Library: CANUSB.DLL

C/C++ Calling Format:

```
USB_RC __stdcall CanUsbSetMaxBufSize(S16 xmitsize,
S16 recvsize);
```

VB6 Calling Format:

```
'This sets the maximum number of buffers 'for any subsequent devices that are opened
'devices already open will use earlier value
Public Declare Function CanUsbSetMaxBufSize _
Lib "CANUSB.DLL" _
(ByVal xmitsize As Integer, _
ByVal recvsize As Integer) _
As Integer 'returns 1 if successful
```

VB .NET Calling Format:

```
'This sets the maximum number of buffers for any subsequent devices that are opened
'devices already open will use earlier value
Public Declare Function CanUsbSetMaxBufSize Lib "CANUSB.DLL" (ByVal xmitsize As Short, ByVal recvsize As Short) As Short 'returns 1 if successful
```

Description: This function is used to set the maximum number of receive and transmit buffers for all of the CAN/USB devices used by the DLL. It must be called before the CAN/USB device is opened for it to take effect – otherwise the old values will be used. The default values are 4 for the number of transmit buffers (xmitsize) and 16384 for the number of receive buffers (recvsize).

Return Code: This function returns a standard CAN/USB return code. A return code of "1" indicates success. Other values indicate specific errors (see Part 3).

Parameter Description:

xmitsize [input] - This is the number of transmit buffers to use by the CANUSB DLL. This is the number of transmit messages that can be queued waiting for transmission before they are sent to the CAN/USB device for handling.

recvsize [input] – This is the number of receive buffers to use by the CANUSB DLL. This is the number of receive messages that can be queued waiting to be read by the application before a buffer overflow error occurs.

.

2.16. CAN USB SET STOP ON ERROR

Function: CanUsbSetStopOnError

Library: CANUSB.DLL

C/C++ Calling Format:

```
USB_RC __stdcall CanUsbSetStopOnError(S16 handle, U8 serr);
```

VB6 Calling Format:

```
Public Declare Function CanUsbSetStopOnError _
Lib "CANUSB.DLL" _
(ByVal handle As Integer, _
ByVal serr As Byte) _
As Integer 'returns 1 if successful
```

VB .NET Calling Format:

```
Public Declare Function CanUsbSetStopOnError Lib "CANUSB.DLL"
(ByVal handle As Short, ByVal serr As Byte) As Short 'returns
1 if successful
```

Description: This function is used to set the non-volatile stop-on-error setting inside the CAN/USB device. If serr is set to 1, the CAN/USB device will stop transmission of recurring messages if an error is encountered on the CAN bus. If serr is set to 0, the CAN/USB device will continue to try to send recurring messages indefinitely (until reset) even if errors are encountered.

Return Code: This function returns a standard CAN/USB return code. A return code of "1" indicates success. Other values indicate specific errors (see Part 3).

Parameter Description:

handle [input] - This is the handle of the CAN/USB device being commanded. Only valid handles obtained using CanUsbOpen are accepted by this routine.

serr [input] – If serr is set to 1, then the CAN/USB device will stop transmission of recurring messages on encountering an error on the CAN bus. If serr is set to 0, then the CAN/USB device will continue to send recurring messages even if errors are encountered on the CAN bus. Errors are still reported to the application in either case. This setting is preserved even if the CAN/USB device is reset (it is non-volatile).

2.17. CAN USB QUERY BIT RATE

Function: CanUsbQueryBitRate

Library: CANUSB.DLL

C/C++ Calling Format:

```
USB_RC __stdcall CanUsbQueryBitRate(S16 handle, S32 *bitrate);

returns bit-rate of CAN bus; example 1 Mbps would be 1000000
(one million)

formula for CAN bit rate is 30000000/(Prescaler*(BS1+BS2+1))
(30 million)

example Prescaler = 2, BS1 = 11, BS2 = 3 : 1 Mbps
```

VB6 Calling Format:

```
Public Declare Function CanUsbQueryBitRate _
Lib "CANUSB.DLL" _
(ByVal handle As Integer, _
ByRef bitrate As Long) _
 As Integer 'returns 1 if ok
 ' returns bit-rate of CAN bus example 1 Mbps would be 1000000
 (one million)
 ' formula for CAN bit rate is 30000000/(Prescaler*(BS1+BS2+1))
 (30 million)
 ' example Prescaler = 2, BS1 = 11, BS2 = 3 : 1 Mbps
```

VB .NET Calling Format:

```
Public Declare Function CanUsbQueryBitRate Lib "CANUSB.DLL" ( _
  ByVal handle As Short, ByVal bitrate As IntPtr) As Short
  'returns 1 if ok
  ' returns bit-rate of CAN bus; example 1 Mbps would be 1000000
  (one million)
  ' formula for CAN bit rate is 30000000/(Prescaler*(BS1+BS2+1))
  (30 million)
  ' example Prescaler = 2, BS1 = 11, BS2 = 3 : 1 Mbps
  ' bitrate is 32-bits
```

Description: This function is used to retrieve the current bit rate that the CAN/USB is set for on the CAN bus. This bit rate can be set by the parameters in the CanUsbSetup call. It is equal to 30,000,000 / ((BS1 + BS2 + 1) * Prescaler). BS1, BS2, and Prescaler are all arguments in the CanUsbSetup call. This is also the raw bits per second that the CAN network is set up for. The maximum value is 1,000,000 (1 Mbps).

Return Code: This function returns a standard CAN/USB return code. A return code of "1" indicates success indicating that the bitrate argument has been filled with the current bitrate setting of the CAN/USB device. Other error codes are documented in Part 3.

Parameter Description:

handle [input] - This is the handle of the CAN/USB device being queried. Only valid handles obtained using CanUsbOpen are accepted by this routine.

bitrate [output] - This argument will contain the bit rate of the CAN bus if the return code for this routine was 1. The max allowed bit rate is 1,000,000 (1 Mbps). This parameter is entirely determined by the parameters in the CanUsbSetup call.

2.18. CAN USB CLEAR BUFFERS

Function: CanUsbClearBuffers

Library: CANUSB.DLL

C/C++ Calling Format:

```
USB_RC __stdcall CanUsbClearBuffers(S16 handle);
```

VB6 Calling Format:

```
Public Declare Function CanUsbClearBuffers _
Lib "CANUSB.DLL" _
(ByVal handle As Integer) _
```

```
As Integer 'returns 1 if successful
```

VB .NET Calling Format:

```
Public Declare Function CanUsbClearBuffers Lib "CANUSB.DLL"
(ByVal handle As Short) As Short 'returns 1 if successful
```

Description: This function is used to clear the receive buffer of the CAN/USB device. If automatic bus-off recovery is specified in the flags of CanUsbSetup, this will also serve to return the bus to a working state. After this routine, the error code returned by CanUsbReadStatus should be 4F (indicating OK) unless there is some lingering issue. Note that this function does not stop recurring messages. To stop recurring messages, use CanUsbStopWriteDataMsg or reset the CAN/USB converter.

Return Code: This function returns a standard CAN/USB return code. A return code of “1” indicates success. Other values indicate specific errors (see Part 3).

Parameter Description:

handle [input] - This is the handle of the CAN/USB device being used. Only valid handles obtained using CanUsbOpen are accepted by this routine.

2.19. CAN USB READ FIRMWARE REVISION

Function: CanUsbReadFirmwareRevision

Library: CANUSB.DLL

C/C++ Calling Format:

```
USB_RC __stdcall CanUsbReadFirmwareRevision(S16 handle, U8
*major_rev, U8 *minor_rev);
```

VB6 Calling Format:

```
Public Declare Function CanUsbReadFirmwareRevision _
Lib "CANUSB.DLL" _
(ByVal handle As Integer, _
ByRef major_version As Byte, _
ByRef minor_version As Byte) _
As Integer 'returns 1 if successful
```

VB .NET Calling Format:

```
Public Declare Function CanUsbReadFirmwareRevision Lib
"CANUSB.DLL" ( _
ByVal handle As Short, _
ByVal major_version As IntPtr, _
ByVal minor_version As IntPtr) As Short 'returns 1 if
successful
'major_version is 8-bits
'minor-version is 8-bits
```

Description: This function is used to retrieve the current version of software of the CAN/USB device. If the major version is 0, this indicates that the bootloader is active and that a proper firmware set is not yet loaded on this device. A major version of 1 or greater indicates proper operating firmware.

Return Code: This function returns a standard CAN/USB return code. A return code of "1" indicates success. Other values indicate specific errors (see Part 3).

Parameter Description:

handle [input] - This is the handle of the CAN/USB device being used. Only valid handles obtained using CanUsbOpen are accepted by this routine.

major_version [output] – This argument will contain the major firmware version of the CAN/USB device. A value of 0 indicates the bootloader is active. A value of 1 or greater indicates proper operating CAN/USB firmware.

minor_version [output] – This argument will contain the minor firmware version of the CAN/USB device.

2.20. CAN USB CLOSE

Function: CanUsbClose

Library: CANUSB.DLL

C/C++ Calling Format:

```
USB_RC __stdcall CanUsbClose(S16 handle,U8 SN[6]);
```

VB6 Calling Format:

```
    Public Declare Function CanUsbClose _
    Lib "CANUSB.DLL" _
   (ByVal handle As Integer, _
    ByRef SerialNumber As Byte) _
 As Integer 'returns 1 if ok
```

VB .NET Calling Format:

```
   Public Declare Function CanUsbClose Lib "CANUSB.DLL" ( _
       ByVal handle As Short, _
       ByVal SerialNumber As IntPtr) As Short 'returns 1 if ok
   'SerialNumber is 8-bits wide
```

Description: This function will close the CAN/USB device specified by handle. The argument SerialNumber is not actually used but is provided for future expansion. The DLL will stop listening to this device after this call.

Return Code: This function returns a standard CAN/USB return code. A return code of "1" indicates success. Other values indicate specific errors (see Part 3).

Parameter Description:

handle [input] – This parameter contains the handle of the CAN/USB device that is being closed.

SerialNumber [input] – This parameter is currently not used. It is provided for future expansion.

2.21. CAN USB UNLOAD

Function: CanUsbUnload

Library: CANUSB.DLL

C/C++ Calling Format:

```
USB_RC __stdcall CanUsbUnload(void);
/* call this upon termination from program. */
```

VB6 Calling Format:

```
Public Declare Function CanUsbUnload _
Lib "CANUSB.DLL" _
() As Integer 'returns 1 if ok
'this routine MUST be called at the termination of your CAN USB program
```

VB .NET Calling Format:

```
Public Declare Function CanUsbUnload Lib "CANUSB.DLL" () As Short 'returns 1 if ok
'this routine MUST be called at the termination of your CAN USB program
```

Description: This function must be called when the program terminates or else the CAN/USB devices will become inaccessible by future instances of the CAN/USB DLL. Make sure to call this routine at the exit of your program.

Return Code: This function returns a standard CAN/USB return code. A return code of “1” indicates success. Other values indicate specific errors (see Part 3).

Parameter Description:

No parameters.

3. CAN USB DLL RETURN CODES

1	SUCCESS – OK
2	GENERAL ERROR – USUALLY COMMUNICATIONS ERROR
3	RWERR – DEVICE DRIVER/OS I/O ERROR
4	RANGE – ERROR IN ARGUMENT OUT OF RANGE IN ROUTINE IN CANUSB DLL
5	MEMERR – OUT OF MEMORY
6	NODEV – NO CAN/USB DEVICE DETECTED
7	NONFCE – NO CAN/USB DEVICE INTERFACE DETECTED,

	IMPROPER INSTALLATION
8	NODETL – NO CAN/USB DEVICE DETAIL DETECTED, SYSTEM ERROR
9	NODETL2 – NO CAN/USB DEVICE DETAIL DETECTED, SYSTEM ERROR
10	NOHNDLS – NO MORE OS HANDLES ARE AVAILABLE TO USE
11	NORDHD – READ ERROR, UNUSED ERROR CODE
12	NOWRHD – WRITE ERROR, UNUSED ERROR CODE
13	NCTS – CLEAR TO SEND WAS DEACTIVE FOR A LONG PERIOD OF TIME INDICATING DEVICE IS TERRIBLY BUSY, STUCK, OR NON-FUNCTIONAL
14	NOORIG – UNUSED ERROR CODE.
15	BOLA – THE PC RECEIVE BUFFER OVERFLOWED CONSTANTLY AND AS A RESULT THE ACKNOWLEDGEMENT TO THE COMMAND JUST SENT WAS LOST
16	LOCKERR – INTERNAL OS ERROR USING SYNCHRONOUS MUTEXES
17	RESERVED. NOT USED IN MODEL 9012
18	BADHL – BAD HANDLE SPECIFIED.

4. CAN USB DEVICE ERROR CODES

4.1. DEVICE ERROR CODES

00h	Reserved, not used by Model 9012.
01h	Reserved, not used by Model 9012
02h	Reserved, not used by Model 9012
03h	Reserved, not used by Model 9012.
04h	Reserved, not used by Model 9012.
05h	Reserved, not used by Model 9012.
06h	Reserved, not used by Model 9012.
07h	Reserved, not used by Model 9012.
08h	Reserved, not used by Model 9012..
10h	ILLEGAL OP CODE.
11h	CAN INIT ERROR. The CAN peripheral had an error during initialization.
12h	CAN STUFF ERROR. The CAN message had a stuff error.
13h	CAN FORM ERROR. The CAN message had a format error.
14h	CAN ACK ERROR. The CAN message had an acknowledgement error.
15h	CAN RECESSIVE BIT ERROR. The CAN message had a recessive bit error.
16h	CAN DOMINANT BIT ERROR. The CAN message had a dominant bit error.
17h	CAN CRC ERROR. The CAN message had a CRC error.
18h	SOFTWARE ERROR. The CAN/USB device had a software error. Consult Silicon Engines if this occurs.
19h	ERROR WARNING. The CAN bus has entered an error warning state.
1Ah	CAN ERROR BUS PASSIVE. The CAN bus has entered an error passive state. The CAN/USB device can no longer acknowledge messages.
1Bh	CAN BUS OFF ERROR. The CAN bus has entered a bus off state. Further transmission on the bus is not possible by the CAN/USB device.
1Ch	CAN XMIT FAILURE. The CAN/USB device had a transmission failure transmitting the message.
1Dh	CAN XMIT TIMEOUT. The CAN/USB device retried the message for the entire length of the Xmit Timeout interval but timed out before the CAN message was acknowledged.
20h	INVALID CHECKSUM FROM PC. This indicates the PC has sent the wrong checksum on an internal command frame. This could indicate a faulty CANUSB DLL or an abrupt termination of a CANUSB DLL that was then restarted.

21h	INVALID COMMAND FROM PC. This indicates the PC has sent an invalid command on an internal command frame. This could indicate a faulty CANUSB DLL or an abrupt termination of a CANUSB DLL that was then restarted.
30h	RECEIVE BUFFER OVERFLOW. This indicates the CAN/USB device receive buffer (containing received message data, acknowledgements, and status updates) has overflowed. This can happen when a USB device is opened on a PC but then ignored such as when another USB device is selected, or when a USB device is opened on a PC and then made to listen during dense bus traffic for a long period of time, or that the PC got busy.
31h	TRANSMIT BUFFER OVERFLOW. This indicates the CAN/USB device transmit buffer (containing commands and requests) has overflowed. If this is occurring then the user should take care to wait for a CanUsb routine to return before sending the next CanUsb command.
4Fh	NO ERROR. Everything is okay.
50h	Reserved, not used by Model 9012.
51h	Reserved, not used by Model 9012.
52h	CAN BUS OVERFLOW. The CAN/USB device could not keep up with the CAN traffic. Should not happen. Contact Silicon Engines if this occurs.
53h	NOT STOPPED NO SUCH CAN MESSAGE RUNNING – Returned in response to a CanUsbStopWriteDataMsg to a CAN message that was not currently being transmitted.
60h	Reserved, not used by Model 9012.
E0h	ROM CHECKSUM ERROR. This indicates the firmware was not properly installed on this device. The user should reinstall the firmware on the device.
E1h	Reserved, not used by Model 9012.
F7h	PC XMIT BUF OVERFLOW. The PC could not send the messages to the CAN/USB device fast enough and its transmit buffer overflowed.
F9h	PC RECEIVE BUF OVERFLOW. The PC's application buffers overflowed before the application could retrieve all the messages from the DLL.
FCh	PC OUT OF MEMORY ERROR. The PC ran out of memory needed to complete a CAN/USB task.

5. ACKNOWLEDGEMENT COMMAND BYTES

5.1. ACKNOWLEDGEMENT COMMAND BYTES

These numbers might be seen during a call to CanUsbAckWaiting. They indicate Acknowledgements from the CAN/USB device to commands that were sent long ago or of a CAN message that has been asynchronously received on the bus. Each CanUsb routine waits a certain amount of time for a command to complete. During this time, an acknowledgement is sent and recognized positively by the routine which then returns a successful return code. However there are some situations where the acknowledgement is not received during the time the CanUsb… routine has allocated for waiting and the routine will return a FAILURE but the CAN/USB device still has the command queued and possibly sends the acknowledgement later on. In those cases, this chart will help a user to determine what command was just performed by the CAN/USB device in those cases where it was long stalled and not able to act on the current transmit queue until the moment the acknowledgement come back.

00xxxxxxb	Reserved, not used by Model 9012.
81h	Corresponds to CanUsbClearBuffers command
82h	Reserved, not used by Model 9012.
83h	Corresponds to CanUsbReadStatus command
84h	Corresponds to CanUsbReadFirmwareRevision command
85h	Reserved, not used by Model 9012.
86h	Corresponds to CanUsbReadStatus command
87h	Corresponds to a CanUsbQueryBitRate command
88h	Corresponds to a CanUsbSendTrigger command
91h	Reserved, not used by Model 9012.
92h	Reserved, not used by Model 9012.
93h	Reserved, not used by Model 9012.
94h	Corresponds to CanUsbSetStopOnError command
95h	Reserved, not used by Model 9012.
96h	Reserved, not used by Model 9012.
97H	Reserved, not used by Model 9012.
98H	Reserved, not used by Model 9012.
99H	Corresponds to a CanUsbSetup command.
A0h	Reserved, not used by Model 9012.
A1h	Reserved, not used by Model 9012.
A2h	Reserved, not used by Model 9012.
A3h	Reserved, not used by Model 9012.
A4h	Reserved, not used by Model 9012.
A6h	Corresponds to a program firmware operation, such as from the CAN USB Message Center
ADh	Corresponds to a prepare program firmware operation, such as from the CAN USB Message Center
AEh	Corresponds to a program firmware operation such as from the CAN USB Message Center
AFh	Corresponds to a program firmware operation such as from the CAN USB Message Center
B1h	Corresponds to a Stop CAN message.
CAh	Corresponds to a Transmit CAN message.
CBh	Corresponds to a CAN message that has been asynchronously received.

6. REVISION HISTORY

6.1. REVISION A

Initial release.

☑