SILICON ENGINES

LIN/USB CONVERTER

MODEL 9011

MODEL 9004

PROGRAMMER'S REFERENCE MANUAL

MODEL 9011 REPLACES MODEL 9004
For a comparison of features, please see the User's Guide for the Model 9011.

This document covers both the newer Model 9011 and the older Model 9004.
This document replaces PRM9004 Rev. B.

WINDOWS® COMPATIBILITY
The Model 9011 is compatible with Windows 2000, Windows XP, Windows Vista, and Windows 7.

The Model 9004 is compatible with Windows 2000 and Windows XP.

COMMENTS

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

SILICON ENGINES

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

CONTENTS

1. OVERVIEW 4
1.1. INTRODUCTION 4
1.2. USER'S GUIDE 4
2. DESCRIPTION OF ROUTINES 4
2.1. LIN USB OPEN 4
2.2. LIN USB IS OPEN 6
2.3. LIN USB QUERY BAUD RATE 7
2.4. LIN USB SET BUS CHARACTERISTICS 8
2.5. LIN USB SERIAL NUMBER 9
2.6. LIN USB INIT ENUM NODE NAMES 9
2.7. LIN USB ENUM NODE NAME 10
2.8. LIN USB INIT ENUM SCHED NAMES 11
2.9. LIN USB ENUM SCHED NAME 12
2.10. LIN USB OKAY TO TRANSMIT 13
2.11. LIN USB STATUS MSG WAITING 14
2.12. LIN USB DATA MSG WAITING 14
2.13. LIN USB READ DATA MSG 15
2.14. LIN USB ACK WAITING 17
2.15. LIN USB WRITE DATA MSG 18
2.16. LIN USB READ STATUS 19
2.17. LIN USB STATUS 21
2.18. LIN USB WAKEUP BUS 22
2.19. LIN USB SET BAUD RATE 22
2.20. LIN USB SET MAX BUF SIZE 23
2.21. LIN USB SET CONTINUAL WAKEUP 24
2.22. LIN USB SET PROTOCOL 25
2.23. LIN USB SET WAKEUP ENABLE 26
2.24. LIN USB SET STOP ON ERROR 27
2.25. LIN USB SET MODE 28
2.26. LIN USB SET FAST 1 STOP BIT 29
2.27. LIN USB SET INCREMENTING PAYLOAD 30
2.28. LIN USB TEST FRAM 31
2.29. LIN USB CLEAR BUFFERS 32
2.30. LIN USB READ FIRMWARE REVISION 33
2.31. LIN USB CLOSE 34
2.32. LIN USB UNLOAD 35
2.33. LIN USB BAUD RATE 35
3. LIN USB DLL RETURN CODES 37
3.1. DLL RETURN CODES 37
4. LIN USB DEVICE ERROR CODES 38
4.1. DEVICE ERROR CODES 38
5. ACKNOWLEDGEMENT COMMAND BYTES 39
5.1. ACKNOWLEDGEMENT COMMAND BYTES 39
6. REVISION HISTORY 41
6.1. REVISION C 41
6.2. REVISION B 41
6.3. REVISION A 41

1. OVERVIEW

1.1. INTRODUCTION

This document describes the LINUSB DLL and the information necessary to write an application program that uses the Model 9011 or Model 9004 Silicon Engines LIN/USB Converter.

Note that on the installation CD for this product, there is included full source code for the Silicon Engines LIN/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 LIN/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 for your product—GD9011 or GD9004.

2. DESCRIPTION OF ROUTINES

2.1. LIN USB OPEN

Function: LinUsbOpen

Library: LINUSB.DLL

C/C++ Calling Format:

```
int __stdcall LinUsbOpen(
short *handle,
char SN[6],
short baudrate,
short actively_change_mode,
char *CLD,
char *name,
char *table,
char errorstring[128]);
```

VB6 Calling Format:

```
Public Declare Function LinUsbOpen _
    Lib "linusb.dll" _
     (ByRef handle As Integer, _
     ByRef SerialNumber As Byte, _
     ByVal BaudRate As Integer, _
     ByVal actively_change_mode As Boolean, _
     ByRef CLD As Byte, _

     ByRef Node_Name As Byte, _
```

```
            ByRef Schedule_Table As Byte, _
            ByRef errorstring As Byte) _
        As Integer 'returns 1 if ok
```

VB.NET Calling Format:

```
32-bit:   Public Declare Function LinUsbOpen Lib "linusb.dll"
(ByVal handle As System.IntPtr, ByVal SerialNumber As System.IntPtr,
ByVal BaudRate As System.Int16, ByVal actively_change_mode As
System.Int16, ByVal CLD As System.IntPtr, ByVal Node_Name As
System.IntPtr, ByVal Schedule_Table As System.IntPtr, ByVal
errorstring As System.IntPtr) As System.Int32 'returns 1 if ok

64-bit:   Public Declare Function LinUsbOpen Lib
"linusb64.dll" (ByVal handle As IntPtr, ByVal SerialNumber As
IntPtr, ByVal BaudRate As Int16, ByVal actively_change_mode As
Boolean, ByVal CLD As IntPtr, ByVal Node_Name As IntPtr, ByVal
Schedule_Table As IntPtr, ByVal errorstring As IntPtr) As Int32
'returns 1 if ok
```

Description: Use this function to (a) discover if any LIN/USB devices are connected to the computer and if so, obtain a handle to one of them, or (b) discover whether a LIN/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 LIN/USB device whose handle has already been obtained is still connected to the computer (although you can use the less intrusive function LinUsbIsOpen for the same purpose), or (d) change the active mode of a LIN/USB device between PC-controlled mode, Master mode, or Slave mode, where the last two are specified fully by a CLD file in the format of the Configuration Language Description of the LIN 1.2/2.0 specifications.

Return Code: This function returns a standard LIN/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 LIN/USB device being interrogated or set up. If the user wishes to direct the action of this routine to a specific LIN/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 LIN/USB device being interrogated or set up. If the user wishes to direct the action of this routine to a LIN/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.

BaudRate [input] - This specifies the baud rate of the LIN bus communications link that this LIN/USB device will talk at. Note that if a CLD file is opened in the same command, the baud rate defined here is overridden by the baud rate from the CLD file. To override the baud rate from the CLD file, simply make another call to LinUsbOpen with a specified baud rate in this argument and the argument actively_change_mode set to false. Valid baud rates are 1000 baud to 21000 baud. Note that once this baud rate is set, it is remembered in non-volatile memory and used on the next power-up of the LIN/USB device.

actively_change_mode [input] – This is a Boolean that governs whether the mode of the LIN/USB device needs to change. Possible mode settings are PC-controlled, Master mode, or Slave mode. If this flag is False, then the arguments CLD, name, and table are not used. If this flag is True, then the arguments CLD, name, and table can specify the necessary parameters to set up the proper mode.

CLD [input] – This is a standard C null-terminated ASCII string that contains the full path and file name of a CLD file. This argument is only used if the user intends to set the LIN/USB device to either a Master node or a Slave node and not a PC-controlled node. If the user wants to set the LIN/USB device to a PC-controlled node, then this argument should be NULL (a pointer to the address 0). Note that all settings of mode, whether PC-controlled, master node, or slave node, are stored in non-volatile memory along with all necessary data to perform the node emulation even after a power-cycle without having to call this routine again.

Node_Name [input] – This argument is only used if the CLD argument is not equal to NULL. This argument is a standard C null-terminated ASCII string that contains the full node name as it appears in the CLD file for the node that the user wants this LIN/USB device to emulate. This name can refer to either a Master node or a Slave node. If the user does not want to emulate a Master or Slave node, and the CLD argument is NULL, then they should specify NULL for this argument too.

Schedule_Table [input] – This argument is only used if the CLD argument and the Node_Name arguments are both not equal to NULL and the user wants to have this LIN/USB device emulate a Master node. This argument is a standard C null-terminated ASCII string that contains the full name of the Master node schedule table to execute from the CLD file. If the user does not want this device to emulate a Master node, then they should specify a NULL for this argument.

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. LIN USB IS OPEN

Function: LinUsbIsOpen

Library: LINUSB.DLL

C/C++ Calling Format:

```
int _stdcall LinUsbIsOpen(
short handle);
```

VB6 Calling Format:

```
Public Declare Function LinUsbIsOpen _
   Lib "linusb.dll" _
   (ByVal handle As Integer) _
```

```
        As Integer 'returns 1 if open
```

VB.NET Calling Format:

```
     32-bit:        Public Declare Function LinUsbIsOpen Lib
"linusb.dll" (ByVal handle As Int16) As Int32 'returns 1 if open

     64-bit:    Public Declare Function LinUsbIsOpen Lib
"linusb64.dll" (ByVal handle As Int16) As Int32 'returns 1 if open
```

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

Return Code: This function returns a standard LIN/USB return code. A return code of "1" indicates that the LIN/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 LIN/USB device being checked. Only valid handles obtained using LinUsbOpen are accepted by this routine.

2.3. LIN USB QUERY BAUD RATE

Function: LinUsbQueryBaudRate

C/C++ Calling Format:

```
int __stdcall LinUsbQueryBaudRate(
      short handle,
      unsigned int *baudrate);
```

VB6 Calling Format:

```
Public Declare Function LinUsbQueryBaudRate _
    Lib "linusb.dll" _
    (ByVal handle As Integer, _
     ByRef baudrate As Integer) _
  As Integer 'returns 1 if ok
```

VB.NET Calling Format:

```
32-bit:      Public Declare Function LinUsbQueryBaudRate Lib
"linusb.dll" (ByVal handle As Int16, ByVal busbaud As IntPtr) As
Int32 'returns 1 if ok

64-bit:       Public Declare Function LinUsbQueryBaudRate Lib
"linusb64.dll" (ByVal handle As Int16, ByVal busbaud As IntPtr) As
Int32 'returns 1 if ok
```

Description: Use this routine to determine the current bus baud rate of the LIN/USB device specified by the handle argument. This has been added because there can be some confusion because if LinUsbOpen is called with reference to a CLD or LDF file, it can change the baud rate automatically. Of course you can set the baud rate to any valid value with the LinUsbSetBaudRate function.

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

Parameter Description:

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

baudrate [output] – On successful return, this value will contain the bus baud rate that the LIN/USB module is using to talk/listen on the LIN bus.

2.4. LIN USB SET BUS CHARACTERISTICS

Function: LinUsbSetBusCharacteristics

Library: LINUSB.DLL

C/C++ Calling Format:

```
int __stdcall LinUsbSetBusCharacteristics(
short handle,
unsigned char flags);
```

VB6 Calling Format:

```
Public Declare Function _
     LinUsbSetBusCharacteristics _
    Lib "linusb.dll" _
(ByVal handle As Integer, _
ByVal flags As Byte) _
As Integer 'returns 1 if ok
```

VB.NET Calling Format:

32-bit: Public Declare Function LinUsbSetBusCharacteristics Lib "linusb.dll" (ByVal handle As Int16, ByVal flags As Byte) As Int32 'returns 1 if ok

64-bit: Public Declare Function LinUsbSetBusCharacteristics Lib "linusb64.dll" (ByVal handle As Int16, ByVal flags As Byte) As Int32 'returns 1 if ok

Description: Use this routine to set the Bus Load Resistor for a specified LIN/USB device.

Return Code: This function returns a standard LIN/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 LIN/USB device being modified. Only valid handles obtained using LinUsbOpen are accepted by this routine.

flags [input] - The byte value passed into this argument is interpreted on a bit-by-bit basis. Bits 7-2 are currently unused and should be set to 0. Bit 1 indicates whether to force the resistance on the bus to what has been specified in bit 0. If bit 1 is 1, then the resistance on the bus is dictated by bit 0. If bit 1 is 0, then the resistance on the bus is dictated by whether the LinUsb device is in master mode or slave mode. If the LinUsb device is in PC mode, then bit 0 automatically is used to determine resistance on the bus. If bit 0 is active, then a 1 indicates use Master mode resistance and a 0 indicates use Slave mode resistance.

2.5. LIN USB SERIAL NUMBER

Function: LinUsbSerialNumber

Library: LINUSB.DLL

C/C++ Calling Format:

```
int _stdcall LinUsbSerialNumber(
     short handle,
     unsigned char SN[6]);
```

VB6 Calling Format:

```
Public Declare Function LinUsbSerialNumber _
  Lib "linusb.dll" _
  (ByVal handle As Integer, _
  ByRef SerialNumber As Byte) _
 As Integer 'returns 1 if successful
```

VB.NET Calling Format:

```
32-bit:      Public Declare Function LinUsbSerialNumber Lib
"linusb.dll" (ByVal handle As Int16, ByVal SerialNumber As IntPtr)
As Int32 'returns 1 if successful

64-bit:      Public Declare Function LinUsbSerialNumber Lib
"linusb64.dll" (ByVal handle As Int16, ByVal SerialNumber As IntPtr)
As Int32 'returns 1 if successful
```

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

Return Code: This function returns a standard LIN/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 LIN/USB device being queried. Only valid handles obtained using LinUsbOpen 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 LIN/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. LIN USB INIT ENUM NODE NAMES

Function: LinUsbInitEnumNodeNames

Library: LINUSB.DLL

C/C++ Calling Format:

```
int _stdcall LinUsbInitEnumNodeNames(
      char *CLD,
     char errorstring[128]);
```

VB6 Calling Format:

```
Public Declare Function LinUsbInitEnumNodeNames _
   Lib "linusb.dll" _
   (ByRef CLD As Byte, _
   ByRef errorstring As Byte) _
 As Integer 'returns 1 if successful
```

VB.NET Calling Format:

```
32-BIT:    Public Declare Function LinUsbInitEnumNodeNames
Lib "linusb.dll" (ByVal CLD As IntPtr, ByVal errorstring As IntPtr)
As Int32 'returns 1 if successful else fills errorstring

64-bit:    Public Declare Function LinUsbInitEnumNodeNames Lib
"linusb64.dll" (ByVal CLD As IntPtr, ByVal errorstring As IntPtr) As
Int32 'returns 1 if successful else fills errorstring
```

Description: Use this routine to initialize enumeration of all node names that exist in a specified CLD file. After calling LinUsbInitEnumNodeNames successfully, subsequent calls to LinUsbEnumNodeName will get the names of each node in the CLD file one-by-one. This is useful to go through all node names that can be used in the Node_Name argument to LinUsbOpen for an application to decide which node name to use.

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

Parameter Description:

CLD [input] - This is a standard C null-terminated string containing the full path and file name of the CLD file to analyze.

errorstring [output] – This argument should contain a pointer to an empty 128-byte long or greater buffer to hold a standard C null-terminated string. If this routine fails to return 1 indicating success, the string errorstring will contain a verbose description of the error encountered. Note that this may contain an error message that was generated while reading the CLD file and may refer to specific syntax in the CLD file that is unrecognized.

2.7. LIN USB ENUM NODE NAME

Function: LinUsbEnumNodeName

Library: LINUSB.DLL

C/C++ Calling Format:

```
int _stdcall LinUsbEnumNodeName(
      char nodename[128]);
```

VB6 Calling Format:

```
Public Declare Function LinUsbEnumNodeName _
  Lib "linusb.dll" _
  (ByRef nodename As Byte) _
 As Integer 'returns 1 if successful
```

VB.NET Calling Format:

```
32-bit:      Public Declare Function LinUsbEnumNodeName Lib
"linusb.dll" (ByVal nodename As IntPtr) As Int32 'returns 1 and
nodename filled if successful

64-bit:    Public Declare Function LinUsbEnumNodeName Lib
"linusb64.dll" (ByVal nodename As IntPtr) As Int32 'returns 1 and
nodename filled if successful
```

Description: Before calling this routine, make sure to call LinUsbInitEnumNodeNames first. Each successive call of this routine will retrieve a new node name that is present in the CLD file that was specified in the call to LinUsbInitEnumNodeNames. This is useful to go through all node names that can be used in the Node_Name argument to LinUsbOpen for an application to decide which node name to use.

Return Code: This function returns a standard LIN/USB return code. A return code of "1" indicates success. A return code of "4" (RANGE) indicates no more enumerated node names exist. Other values indicate specific errors (see Part 3).

Parameter Description:

nodename [output] – The user should pass a pointer to a buffer of at least 128 bytes in this argument. If this routine returns successful, then this buffer will be filled with a standard C null-terminated ASCII string containing the next successive node name in the CLD file specified in the call to LinUsbInitEnumNodeNames.

2.8. LIN USB INIT ENUM SCHED NAMES

Function: LinUsbInitEnumSchedNames

Library: LINUSB.DLL

C/C++ Calling Format:

```
int _stdcall LinUsbInitEnumSchedNames(
      char *CLD,
     char errorstring[128]);
```

VB6 Calling Format:

```
Public Declare Function LinUsbInitEnumSchedNames _
   Lib "linusb.dll" _
   (ByRef CLD As Byte, _
   ByRef errorstring As Byte) _


 As Integer 'returns 1 if successful
```

VB.NET Calling Format:

```
     32-bit:  Public Declare Function LinUsbInitEnumSchedNames Lib
"linusb.dll" (ByVal CLD As IntPtr, ByVal errorstring As IntPtr) As
Int32 'returns 1 if successful else fills errorstring

     64-bit: Public Declare Function LinUsbInitEnumSchedNames Lib
"linusb64.dll" (ByVal CLD As IntPtr, ByVal errorstring As IntPtr) As
Int32 'returns 1 if successful else fills errorstring
```

Description: Use this routine to initialize enumeration of all schedule table names that exist in a specified CLD file. After calling LinUsbInitEnumSchedNames successfully, subsequent calls to LinUsbEnumSchedName will get the names of each schedule table in the CLD file one-by-one. This is useful to go through all schedule table names that can be used in the Schedule_Table argument to LinUsbOpen for an application to decide which schedule table to use for Master node emulation.

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

Parameter Description:

CLD [input] - This is a standard C null-terminated string containing the full path and file name of the CLD file to analyze.

errorstring [output] – This argument should contain a pointer to an empty 128-byte long or greater buffer to hold a standard C null-terminated string. If this routine fails to return 1 indicating success, the string errorstring will contain a verbose description of the error encountered. Note that this may contain an error message that was generated while reading the CLD file and may refer to specific syntax in the CLD file that is unrecognized.

2.9. LIN USB ENUM SCHED NAME

Function: LinUsbEnumSchedName

Library: LINUSB.DLL

C/C++ Calling Format:

```
int _stdcall LinUsbEnumSchedName(
     char schedname[128]);
```

VB6 Calling Format:

```
Public Declare Function LinUsbEnumSchedName _
  Lib "linusb.dll" _
  (ByRef schedname As Byte) _
 As Integer 'returns 1 if successful
```

VB.NET Calling Format:

```
     32-bit:     Public Declare Function LinUsbEnumSchedName Lib
"linusb.dll" (ByVal schedname As IntPtr) As Int32 'returns 1 and
nodename filled if successful

     64-bit:       Public Declare Function LinUsbEnumSchedName Lib
"linusb64.dll" (ByVal schedname As IntPtr) As Int32 'returns 1 and
nodename filled if successful
```

Description: Before calling this routine, make sure to call LinUsbInitEnumSchedNames first. Each successive call of this routine will retrieve a new schedule table name that is present in the CLD file that was specified in the call to LinUsbInitEnumSchedNames. This is useful to go through all schedule table names that can be used in the Schedule_Table argument to LinUsbOpen for an application to decide which schedule table to use for Master node emulation.

Return Code: This function returns a standard LIN/USB return code. A return code of "1" indicates success. A return code of "4" (RANGE) indicates no more enumerated schedule tables exist. Other values indicate specific errors (see Part 3).

Parameter Description:

schedname [output] – The user should pass a pointer to a buffer of at least 128 bytes in this argument. If this routine returns successful, then this buffer will be filled with a standard C null-terminated ASCII string containing the next successive schedule table name in the CLD file specified in the call to LinUsbInitEnumSchedNames.

2.10. LIN USB OKAY TO TRANSMIT

Function: LinUsbOkayToTransmit

Library: LINUSB.DLL

C/C++ Calling Format:

```
int __stdcall LinUsbOkayToTransmit(
short handle);
```

VB6 Calling Format:

```
Public Declare Function LinUsbOkayToTransmit _
    Lib "linusb.dll" _
    (ByVal handle As Integer) _
 As Integer 'returns 1 if ok to transmit
```

VB.NET Calling Format:

```
32-bit:        Public Declare Function LinUsbOkayToTransmit
Lib "linusb.dll" (ByVal handle As Int16) As Int32 'returns 1 if ok
to transmit

64-bit:        Public Declare Function LinUsbOkayToTransmit
Lib "linusb64.dll" (ByVal handle As Int16) As Int32 'returns 1 if ok
to transmit
```

Description: This routine is only applicable to the PC-controlled mode in which the PC is capable of transmitting messages through the LIN/USB device. This routine will return a code of 1 if the transmitter is idle and all messages commanded to transmit through the LIN/USB device and on to the LIN bus have been conveyed successfully to the LIN/USB device. It basically returns 1 if the transmit queue from the PC to the LIN/USB device of LIN messages that will go on the LIN bus is empty (all messages have been transmitted). This is a good indication that it is safe to send a new message on the LIN bus using LinUsbWriteDataMsg.

Return Code: This function returns a standard LIN/USB return code. A return code of "1" indicates that the current transmit queue is empty and that it is okay to add another message to the LIN transmit queue. Other values indicate specific errors (see Part 3) although a code of "2" could also mean simply that the transmit queue is not empty.

Parameter Description:

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

2.11. LIN USB STATUS MSG WAITING

Function: LinUsbStatusMsgWaiting

Library: LINUSB.DLL

C/C++ Calling Format:

```
int __stdcall LinUsbStatusMsgWaiting(
short handle);
```

VB6 Calling Format:

```
Public Declare Function LinUsbStatusMsgWaiting _
 Lib "linusb.dll" _
 (ByVal handle As Integer) _
 As Integer 'returns 1 if status message is waiting
```

VB.NET Calling Format:

```
32-bit:       Public Declare Function LinUsbStatusMsgWaiting
Lib "linusb.dll" (ByVal handle As Int16) As Int32 'returns 1 if
status message is waiting

64-bit:      Public Declare Function LinUsbStatusMsgWaiting
Lib "linusb64.dll" (ByVal handle As Int16) As Int32 'returns 1 if
status message is waiting
```

Description: This function should be used to determine if any status messages are waiting to be read by the PC from the specified LIN/USB device. Status messages are error conditions that can arise that are not directly caused by a PC command such as buffer overflows. If there are one or more status messages waiting to be read by the PC, then this routine will return a code of 1; otherwise it returns a code of 2. Read the status message(s) using the routine LinUsbStatus.

Return Code: This function returns a 1 if there are one or more status messages waiting to be read; otherwise it returns 2.

Parameter Description:

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

2.12. LIN USB DATA MSG WAITING

Function: LinUsbDataMsgWaiting

Library: LINUSB.DLL

C/C++ Calling Format:

```
int __stdcall LinUsbDataMsgWaiting(
short handle);
```

VB6 Calling Format:

```
Public Declare Function LinUsbDataMsgWaiting _
  Lib "linusb.dll" _
 (ByVal handle As Integer) _
 As Integer 'returns 1 if message is waiting
```

VB.NET Calling Format:

```
32-bit:      Public Declare Function LinUsbDataMsgWaiting Lib "linusb.dll" (ByVal handle As Int16) As Int32 'returns 1 if message is waiting

64-bit:       Public Declare Function LinUsbDataMsgWaiting Lib "linusb64.dll" (ByVal handle As Int16) As Int32 'returns 1 if message is waiting
```

Description: This function should be used to determine if any data messages are waiting to be read by the PC from the specified LIN/USB device. Data messages are valid LIN messages that have been seen on the LIN bus. If there are one or more data messages waiting to be read by the PC, then this routine will return a code of 1; otherwise it returns a code of 2. Read the data message(s) using successive calls to the routine LinUsbReadDataMsg.

Return Code: This function returns a 1 if there are one or more data messages waiting to be read. It will return 2 if there are no more data messages to be read or the module could not communicate.

Parameter Description:

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

2.13. LIN USB READ DATA MSG

Function: LinUsbReadDataMsg

Library: LINUSB.DLL

C/C++ Calling Format:

```
int __stdcall LinUsbReadDataMsg(
short handle,
unsigned char protocol,
     unsigned short *count,
unsigned char *buffer,
unsigned char *error_code);
```

VB6 Calling Format:

```
Public Declare Function LinUsbReadDataMsg _
    Lib "linusb.dll" _
   (ByVal handle As Integer, _
   ByVal protocol As Byte, _
   ByRef count As Integer, _
   ByRef buffer As Byte, _
```

```
        ByRef ErrorCode As Byte) _
      As Integer 'returns 1 if successful
```

VB.NET Calling Format:

```
32-bit:     Public Declare Function LinUsbReadDataMsg Lib
"linusb.dll" (ByVal handle As Int16, ByVal protocol As Byte, ByVal
count As IntPtr, ByVal buffer As IntPtr, ByVal ErrorCode As IntPtr)
As Int32 'returns 1 if successful

64-bit:     Public Declare Function LinUsbReadDataMsg Lib
"linusb64.dll" (ByVal handle As Int16, ByVal protocol As Byte, ByVal
count As IntPtr, ByVal buffer As IntPtr, ByVal ErrorCode As IntPtr)
As Int32 'returns 1 if successful
```

Description: Use this function to read messages that were received on the LIN bus. Normally a call to LinUsbDataMsgWaiting is made first to see if this routine should or should not be called.

Return Code: This function returns a standard LIN/USB return code. A return code of "1" indicates success. A code of "2" can indicate that there are no received messages queued up currently. It will return "17" if there was a data message, the buffer has been filled with the data message, but the checksum did not match the protocol specified.

Other values indicate specific errors (see Part 3).

Parameter Description:

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

protocol [input] – This is a variable that indicates which LIN protocol to expect for the LIN bus. This is used for checking the checksum in the LIN message. Valid values are 0 (for LIN 1.x), 1 (for LIN 2.0), and 2 (for MIXED (LIN 1.x or LIN 2.0 allowed)).

count [input/output] – This argument points to an integer value that contains a count for the number of message bytes as well as the size of the buffer in the buffer argument. The value of this integer when this routine is called should be set to the maximum number of bytes that can be stored in the buffer in the buffer argument. This should really be 9 or greater to receive the longest LIN message which is 1 ID byte and 8 data bytes. When this routine returns, this integer value will contain the number of bytes in the LIN message that are now stored in the buffer pointed to by the buffer argument. The maximum value that count will have is 9; 1 for the ID byte and 8 is the maximum for the number of data bytes.

buffer [output] – This argument should contain a pointer to a buffer that will contain the LIN message when this routine returns. The number of bytes allocated for this buffer should be put into the integer pointed to by the count argument. Normally this should be 9 or greater and the buffer pointed to by the buffer argument should be allocated for 9 bytes or greater. On return from this routine, the buffer will contain the LIN message. The first byte in the buffer will contain the 6-bit ID of the LIN message and the subsequent bytes will contain the data bytes of the LIN message, not including the checksum.

ErrorCode [output] – This argument should point to a byte that can be written by the routine. The routine writes the final error code of the LIN/USB device to this byte. 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, but it also can be a "leftover" error code from some error condition that occurred previously. To clear leftover error codes, you need to call LinUsbClearBuffers. Error codes will persist until cleared or overridden by a different error code.

2.14. LIN USB ACK WAITING

Function: LinUsbAckWaiting

Library: LINUSB.DLL

C/C++ Calling Format:

```
int __stdcall LinUsbAckWaiting(
short handle,
unsigned char *msg_acked,
     unsigned char *error_code);
```

VB6 Calling Format:

```
Public Declare Function LinUsbAckWaiting _
  Lib "linusb.dll" _
  (ByVal handle As Integer, _
  ByRef msg_acked As Byte, _
  ByRef error_code As Byte) _
 As Integer 'returns 1 if ack received
```

VB.NET Calling Format:

```
32-bit:  Public Declare Function LinUsbAckWaiting Lib "linusb.dll" (ByVal handle As Int16, ByVal msg_acked As IntPtr, ByVal error_code As IntPtr) As Int32 'returns 1 if ack received

64-bit:  Public Declare Function LinUsbAckWaiting Lib "linusb64.dll" (ByVal handle As Int16, ByVal msg_acked As IntPtr, ByVal error_code As IntPtr) As Int32 'returns 1 if ack received
```

Description: This function should be used to determine if any command acknowledgements are waiting to be read by the PC from the specified LIN/USB device. Command acknowledgements are messages from the LIN/USB device that a specific command has just been executed. Normally this routine is not necessary since specific operations wait for the acknowledgement back before returning. This routine would only be used to determine if a "Lost Command" has finally executed. It can also be used to silently "kill" acknowledgement messages that get queued up that were orphaned from their original commands due to exceptional circumstances in order to keep the queue empty and free. If there are one or more acknowledgement messages waiting to be read by the PC, then this routine will return a code of 1; otherwise it returns a code of 2. If this routine returns 1 indicating an ACK is waiting, the ACK is automatically removed from the acknowledgement queue and the specific ACK message is stored in the msg_acked argument. Additionally, if this routine returns 1, then the current LIN/USB device error code is also loaded into the error_code argument.

Return Code: This function returns a 1 if there are one or more acknowledgement messages waiting to be read; otherwise it returns 2.

Parameter Description:

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

msg_acked [output] – This argument should point to a byte. This byte is untouched unless the return code is 1. If the return code is 1, then this byte is filled with the FIFO acknowledgement queue first-in acknowledgement byte. The list of possible acknowledgement bytes is in Part 5. Note that if the most 2 significant bits of the acknowledgement byte are 0, then this is an acknowledgement of sending a LIN message whose command ID is equal to the acknowledgement byte.

error_code [output] – This argument should point to a byte that can be written by the routine. The routine writes the final error code of the LIN/USB device to this byte. 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 acting on the command/message acknowledged by the msg_acked argument's acknowledgement byte, but it also can be a "leftover" error code from some error condition that occurred previously. To clear leftover error codes, you need to call LinUsbClearBuffers. Error codes will persist until cleared or overridden by a different error code.

2.15. LIN USB WRITE DATA MSG

Function: LinUsbWriteDataMsg

Library: LINUSB.DLL

C/C++ Calling Format:

```
int __stdcall LinUsbWriteDataMsg(
short handle,
unsigned short count,
     unsigned char *buffer,
unsigned char *error_code,
short wait_for_ack);
```

VB6 Calling Format:

```
Public Declare Function LinUsbWriteDataMsg _
 Lib "linusb.dll" _
 (ByVal handle As Integer, _
 ByVal count As Integer, _
 ByRef buffer As Byte, _
 ByRef ErrorCode As Byte, _
 ByVal Wait_For_Ack As Boolean) _
As Integer 'returns 1 if successful
```

VB.NET Calling Format:

```
32-bit:        Public Declare Function LinUsbWriteDataMsg Lib
"linusb.dll" (ByVal handle As Int16, ByVal count As Short, ByVal
buffer As IntPtr, ByVal ErrorCode As IntPtr, ByVal Wait_For_Ack As
Boolean) As Int32 'returns 1 if successful

64-bit:        Public Declare Function LinUsbWriteDataMsg Lib
"linusb64.dll" (ByVal handle As Int16, ByVal count As Short, ByVal
buffer As IntPtr, ByVal ErrorCode As IntPtr, ByVal Wait_For_Ack As
Boolean) As Int32 'returns 1 if successful
```

Description: This function is used to write a message to the LIN bus. Note that this function can only be used if the LIN/USB device is in PC-Controlled mode. If the LIN/USB device is in Master mode or Slave mode, then message transmission from that device is controlled by the CLD file that the device was initialized with. Note that if the LIN/USB device is in PC-Controlled mode and you send just a Master Task message (single byte message with count=1 and the first byte of buffer containing just a 6-bit command ID) then you can use LinUsbDataMsgWaiting and LinUsbReadDataMsg to read back the Slave task portion of the message.

Return Code: This function returns a standard LIN/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 LIN/USB device being used. Only valid handles obtained using LinUsbOpen are accepted by this routine.

count [input] – This is the number of bytes of the LIN message to send including the message ID. The range for this value is 1-9.

buffer [input] – This is the LIN message buffer for the message to send. This buffer should be allocated and filled with the number of bytes specified in the count argument. Note that the first byte in the buffer, the message ID, should only have the least 6 significant bits valid – the most significant 2 bits should be 0. The correct parity check bits will automatically be added to the message ID byte before the byte is sent. Note also that you do not need to include the LIN checksum; this is computed automatically and added to the message automatically, along with synchronization header and other elements of the LIN message frame. You can modify how the LIN checksum is computed, either the Classic Checksum method or the LIN 2.0 Checksum method, by calling the LinUsbSetProtocol command, or by opening the LIN/USB device with a CLD file which automatically sets the protocol based on the contents of the CLD file.

ErrorCode [output] – This argument should point to a byte that can be written by the routine. The routine writes the final error code of the LIN/USB device to this byte. 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 trying to send the specified message, but it also can be a "leftover" error code from some error condition that occurred previously. To clear leftover error codes, you need to call LinUsbClearBuffers. Error codes will persist until cleared or overridden by a different error code.

Wait_For_Ack [input] – If this is True, then this routine will assure that the command to send the message is acknowledged successfully before returning a successful return code. You can also set this input parameter to False indicating that the command to send the message should be sent but that the routine should return immediately. This might be used for sending lots of fast messages on the LIN bus and then using the routine LinUsbAckWaiting to confirm proper transmission of the messages afterwards.

2.16. LIN USB READ STATUS

Function: LinUsbReadStatus

Library: LINUSB.DLL

C/C++ Calling Format:

```
int __stdcall LinUsbReadStatus(
short handle,
unsigned char *code,
```

```
    unsigned char *awake,
    unsigned char *mode);
```

VB6 Calling Format:

```
    Public Declare Function LinUsbReadStatus _
        Lib "linusb.dll" _
        (ByVal handle As Integer, _
        ByRef status As Byte, _
        ByRef awake As Byte, _
        ByRef mode As Byte) _
      As Integer 'returns 1 if successful
```

VB.NET Calling Format:

```
    32-bit:     Public Declare Function LinUsbReadStatus Lib "linusb.dll" (ByVal handle As Int16, ByVal status As IntPtr, ByVal awake As IntPtr, ByVal mode As IntPtr) As Int32 'returns 1 if successful

    64-bit:     Public Declare Function LinUsbReadStatus Lib "linusb64.dll" (ByVal handle As Int16, ByVal status As IntPtr, ByVal awake As IntPtr, ByVal mode As IntPtr) As Int32 'returns 1 if successful
```

Description: This function is used to get a status update on a LIN/USB device that is connected to the PC. It is a good command to send to get an overall picture of what state the LIN/USB device is in. It tells what current error code is present in the LIN/USB device, whether the LIN bus is awake or asleep, and what mode the LIN/USB device is in (whether Master mode, Slave mode, or PC-controlled mode).

Return Code: This function returns a standard LIN/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 LIN/USB device being used. Only valid handles obtained using LinUsbOpen are accepted by this routine.

status [output] - This argument should point to a byte that can be written by the routine. The routine writes the current error code of the LIN/USB device to this byte. This error code is equal to 4F hex if there is no error. For a list of error codes, see Part 4. To clear this error code, you need to call LinUsbClearBuffers. Error codes will persist until cleared or overridden by a different error code.

awake [output] – This argument should point to a byte that can be written by the routine. This routine writes a 1 to this byte if the LIN bus is currently "Awake", as defined by the LIN specifications, or writes a 0 to this byte if the LIN bus is currently "Asleep", as defined by the LIN specifications. If Continual Wakeups are not currently set (using the LinUsbSetContinualWakeup function) and the bus is idle (not being used), then the bus will be sensed as asleep and this byte will contain 0.

mode [output] – This argument should point to a byte that can be written by the routine. This routine writes a 0 to this byte if the LIN/USB device is currently in the "PC-controlled" mode. This routine writes a 1 to this byte if the LIN/USB device is currently in the "CLD Slave Emulation" mode. This routine writes a 2 to this byte if the LIN/USB device is currently in the "CLD Master Emulation" mode.

2.17. LIN USB STATUS

Function: LinUsbStatus

Library: LINUSB.DLL

C/C++ Calling Format:

```
int __stdcall LinUsbStatus(
short handle,
unsigned char *code);
```

VB6 Calling Format:

```
Public Declare Function LinUsbStatus _
    Lib "linusb.dll" _
    (ByVal handle As Integer, _
    ByRef status As Byte) _
  As Integer 'returns 1 if successful
```

VB.NET Calling Format:

```
32-bit:       Public Declare Function LinUsbStatus Lib "linusb.dll"
(ByVal handle As Int16, ByVal status As IntPtr) As Int32 'returns 1
if successful

64-bit:      Public    Declare    Function    LinUsbStatus    Lib
"linusb64.dll" (ByVal handle As Int16, ByVal status As IntPtr) As
Int32 'returns 1 if successful
```

Description: This function is used to retrieve a status message from a LIN/USB device that is connected to the PC. The user should call the routine LinUsbStatusMsgWaiting first to see if a status message is waiting to be read using this routine. This is used for status notifications from the LIN/USB device to the PC.

Return Code: This function returns a standard LIN/USB return code. A return code of "1" indicates success indicating that the status argument has been filled with the previously queued status code. A return code of "2" indicates that there are no pending status messages in the status message queue.

Parameter Description:

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

status [output] - This argument should point to a byte that can be written by the routine. The routine writes the current error code of the LIN/USB device to this byte. This error code is equal to 4F hex if there is no error. For a list of error codes, see Part 4. To clear this error code, you need to call LinUsbClearBuffers. Error codes will persist until cleared or overridden by a different error code.

2.18. LIN USB WAKEUP BUS

Function: LinUsbWakeupBus

Library: LINUSB.DLL

C/C++ Calling Format:

```
int __stdcall LinUsbWakeupBus(
short handle);
```

VB6 Calling Format:

```
Public Declare Function LinUsbWakeupBus _
  Lib "linusb.dll" _
  (ByVal handle As Integer) _
 As Integer 'returns 1 if successful
```

VB.NET Calling Format:

```
32-bit: Public Declare Function LinUsbWakeupBus Lib "linusb.dll"
(ByVal handle As Int16) As Int32 'returns 1 if successful

64-bit: Public Declare Function LinUsbWakeupBus Lib "linusb64.dll"
(ByVal handle As Int16) As Int32 'returns 1 if successful
```

Description: This function is used to send a single wake-up pulse on the bus. Note that this wake-up pulse is retried 3 times if there is no further activity on the bus. After the third wake-up pulse is sent, another length of time is waited defined by the LIN specification and a decision is made by the LIN/USB device whether to continue sending wake-up pulses. This is based on the current setting set by the LinUsbSetContinualWakeup routine. If Continual Wakeups are enabled, then wake-up pulses will continue to be sent as long as the LIN/USB device is powered-on and is considered the LIN Bus Master (the last node that transmitted a Master Task message on the bus is considered the LIN Bus Master). If Continual Wakeups are disabled, wake-up pulses terminate being sent after the third try and the bus is asleep again. This function is useful to test the LIN bus with a minimal amount of signal or possibly to activate a sleeping LIN node. This function is not strictly necessary during normal transmission since a wakeup pulse is automatically sent at the start of every transmitted message by the LIN/USB device as long as the bus was asleep.

Return Code: This function returns a standard LIN/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 LIN/USB device being used. Only valid handles obtained using LinUsbOpen are accepted by this routine.

2.19. LIN USB SET BAUD RATE

Function: LinUsbSetBaudRate

Library: LINUSB.DLL

C/C++ Calling Format:

```
int __stdcall LinUsbSetBaudRate(
short handle,
```

```
unsigned short baudrate);
```

VB6 Calling Format:

```
Public Declare Function LinUsbSetBaudRate _
  Lib "linusb.dll" _
  (ByVal handle As Integer, _
  ByVal BaudRate As Integer) _
 As Integer 'returns 1 if successful
```

VB.NET Calling Format:

32-bit: **Public Declare Function LinUsbSetBaudRate Lib "linusb.dll" (ByVal handle As Int16, ByVal BaudRate As Int16) As Int32 'returns 1 if successful**

64-bit: **Public Declare Function LinUsbSetBaudRate Lib "linusb64.dll" (ByVal handle As Int16, ByVal BaudRate As Int16) As Int32 'returns 1 if successful**

Description: This function is used to set the baud rate of the LIN bus communications link. Typical baud rates are 4800, 9600, 19200, and 20000. Valid values range anywhere from 1000 to 21000. Note that if a baud rate was specified in the CLD file that the LIN/USB device was initialized with, then this command will override that baud rate and the baud rate specified by this command will be used. Note that after this baud rate is set, it will be used every time the selected module is powered up at all times until this command or LinUsbOpen with a different baud rate is executed.

Return Code: This function returns a standard LIN/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 LIN/USB device being used. Only valid handles obtained using LinUsbOpen are accepted by this routine.

BaudRate [input] – This argument contains the new baud rate in units of bits per second. It can range from 1000 to 21000.

2.20. LIN USB SET MAX BUF SIZE

Function: LinUsbSetMaxBufSize

Library: LINUSB.DLL

C/C++ Calling Format:

```
int __stdcall LinUsbSetMaxBufSize(
unsigned short xmitsize,
     unsigned short recvsize);
```

VB6 Calling Format:

```
Public Declare Function LinUsbSetMaxBufSize _

 Lib "linusb.dll" _
  (ByVal xmitsize As Integer, _
```

```
  ByVal recvsize As Integer) _
 As Integer 'returns 1 if successful
```

VB.NET Calling Format:

```
32-bit:    Public Declare Function LinUsbSetMaxBufSize Lib
"linusb.dll" (ByVal xmitsize As Int16, ByVal recvsize As Int16) As
Int32 'returns 1 if successful

64-bit:    Public Declare Function LinUsbSetMaxBufSize Lib
"linusb64.dll" (ByVal xmitsize As Int16, ByVal recvsize As Int16) As
Int32 'returns 1 if successful
```

Description: This function sets the maximum number of data buffers for any subsequent devices that are opened after this command is finished. Devices that were open previously will use the old maximum number of data buffers settings. The maximum number of buffers defaults to 4 for the transmit buffer queue (xmitsize) and 16384 for the receive buffer queue (recvsize).

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

Parameter Description:

xmitsize [input] – This parameter controls the number of transmit buffers that will be kept for each LIN/USB device. The transmit buffer holds messages that have not been sent yet and are waiting for the first available moment to be transmitted to the LIN/USB device. The default value is 4; it is assumed that for the most part messages are transmitted one at a time until the acknowledgement comes back so there is not usually a need for a large number of transmit buffers.

recvsize [input] – This parameter controls the number of receive buffers that will be kept for each LIN/USB device. The receive buffer holds messages that have not been processed yet by the PC. To prevent a receive buffer overflow, the program must constantly check to see if any messages (status, data, and ack's) are waiting and then to read them out so as to keep the receive queue clear. The number of receive messages that can be queued defaults to 16384; a large number that allows for the many time-consuming latencies of the Windows OS to make sure the messages are not lost before the Windows OS finally has time to process them.

2.21. LIN USB SET CONTINUAL WAKEUP

Function: LinUsbSetContinualWakeup

Library: LINUSB.DLL

C/C++ Calling Format:

```
int __stdcall LinUsbSetContinualWakeup(
short handle,
unsigned char cw);
```

VB6 Calling Format:

```
Public Declare Function LinUsbSetContinualWakeup _
  Lib "linusb.dll" _
  (ByVal handle As Integer, _
  ByVal cw as Byte) _
 As Integer 'returns 1 if successful
```

VB.NET Calling Format:

```
     32-bit:      Public Declare Function LinUsbSetContinualWakeup
Lib "linusb.dll" (ByVal handle As Int16, ByVal cw As Byte) As Int32
'returns 1 if successful

     64-bit:        Public             Declare             Function
LinUsbSetContinualWakeup Lib "linusb64.dll" (ByVal handle As Int16,
ByVal cw As Byte) As Int32 'returns 1 if successful
```

Description: This function is used to set the behavior of the LIN/USB device after it has finished transmitting its Master Task messages and has already sent 3 wakeup pulse retries. If the value of cw is set to 1, then this indicates that it should continue to keep the bus awake by repeating the transmission of a set of 3 wakeup pulses indefinitely. If the value of cw is set to 0, then this indicates that the LIN/USB device will then let the bus lapse into sleep at this point. The value of 0 is certainly the default behavior of the bus, but keep in mind that whatever you set the LIN/USB device continual wakeup flag to, it will be remembered across power cycles and the LIN/USB device will continue to use the last cw setting it was sent on each new boot-up.

Return Code: This function returns a standard LIN/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 LIN/USB device being used. Only valid handles obtained using LinUsbOpen are accepted by this routine.

cw [input] – This argument contains the new Continual Wakeup flag. A value of 1 indicates that the bus should be kept awake for as long as the LIN/USB device is the Bus Master. A value of 0 indicates that the bus should not be kept awake indefinitely but allowed to go to sleep after the 3 wakeup retries when all else is idle.

2.22. LIN USB SET PROTOCOL

Function: LinUsbSetProtocol

Library: LINUSB.DLL

C/C++ Calling Format:

```
int __stdcall LinUsbSetProtocol(
short handle,
unsigned char prot);
```

VB6 Calling Format:

```
Public Declare Function LinUsbSetProtocol _
  Lib "linusb.dll" _
  (ByVal handle As Integer, _
  ByVal prot as Byte) _
 As Integer 'returns 1 if successful
```

VB.NET Calling Format:

```
     32-bit:       Public Declare Function LinUsbSetProtocol Lib
"linusb.dll" (ByVal handle As Int16, ByVal prot As Byte) As Int32
'returns 1 if ok
```

```
     64-bit:       Public Declare Function LinUsbSetProtocol Lib
"linusb64.dll" (ByVal handle As Int16, ByVal prot As Byte) As Int32
'returns 1 if ok
```

Description: This function is used to set the behavior of the LIN/USB device based on the LIN 1.3 protocol versus the LIN 2.0 protocol. There are different wakeup timing characteristics between LIN 1.3 and LIN 2.0 as well as a differently calculated checksum on most message types.

Return Code: This function returns a standard LIN/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 LIN/USB device being used. Only valid handles obtained using LinUsbOpen are accepted by this routine.

prot [input] – This argument contains the new Protocol flag. A value of 1 indicates that the LIN/USB device should conform to LIN 2.0. A value of 0 indicates that the LIN/USB device should conform to LIN 1.3.

2.23. LIN USB SET WAKEUP ENABLE

Function: LinUsbSetWakeupEnable

Library: LINUSB.DLL

C/C++ Calling Format:

```
int __stdcall LinUsbSetWakeupEnable(
short handle,
unsigned char wake);
```

VB6 Calling Format:

```
Public Declare Function LinUsbSetWakeupEnable _
  Lib "linusb.dll" _
  (ByVal handle As Integer, _
  ByVal wake as Byte) _
 As Integer 'returns 1 if successful
```

VB.NET Calling Format:

```
     32-bit:     Public Declare Function LinUsbSetWakeupEnable Lib
"linusb.dll" (ByVal handle As Int16, ByVal wake As Byte) As Int32
'returns 1 if ok

     64-bit:       Public Declare Function LinUsbSetWakeupEnable
Lib "linusb64.dll" (ByVal handle As Int16, ByVal wake As Byte) As
Int32 'returns 1 if ok
```

Description: This function is used primarily to enable/disable the transmission of the standard LIN specification wakeup pulses. While enabled, the wakeup pulses behave normally, but are defined differently in the LIN 1.3 specs as opposed to the LIN 2.0 specs. Make sure to set the protocol using LinUsbSetProtocol to achieve the desired wakeup pulse functionality. If for some reason the modules that you are communicating with are not designed to handle the standard LIN wakeup pulses, then you can disable the transmission of wakeup pulses using this function and a "wake" argument of 0. This can be useful during development or working with non-standard LIN modules. Note that this parameter is stored non-volatile in the LIN/USB device and will be used when the LIN/USB device powers on each time.

Return Code: This function returns a standard LIN/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 LIN/USB device being used. Only valid handles obtained using LinUsbOpen are accepted by this routine.

wake [input] – This argument contains the Wakeup Enable flag. If this is set to 1, then standard wakeup pulses will be generated as defined by the LIN specifications (either 1.3 or 2.0 depending on which protocol is currently enabled). If this is set to 0, then all wakeup pulses will be suppressed and the bus will be left idle during the times that wakeup pulses are normally sent. The rest of the LIN/USB device will act as if the wakeup pulses were sent and the same rules for determining whether a bus is awake or asleep will be used for purposes of reporting whether awake or asleep by the function

LinUsbReadStatus. The purpose of disabling wakeup pulses is for supporting non-standard LIN devices that do not recognize standard specification LIN pulses or for development purposes. Normally you should call this routine with the wake argument set to 1.

2.24. LIN USB SET STOP ON ERROR

Function: LinUsbSetStopOnError

Library: LINUSB.DLL

C/C++ Calling Format:

```
int __stdcall LinUsbSetStopOnError(
short handle,
unsigned char serr);
```

VB6 Calling Format:

```
Public Declare Function LinUsbSetStopOnError _
  Lib "linusb.dll" _
  (ByVal handle As Integer, _
  ByVal serr as Byte) _
 As Integer 'returns 1 if successful
```

VB.NET Calling Format:

32-bit: Public Declare Function LinUsbSetStopOnError Lib "linusb.dll" (ByVal handle As Int16, ByVal serr As Byte) As Int32 'returns 1 if successful

64-bit: Public Declare Function LinUsbSetStopOnError Lib "linusb64.dll" (ByVal handle As Int16, ByVal serr As Byte) As Int32 'returns 1 if successful

Description: This function is used to set a useful feature that applies when the LIN/USB device is in Master emulation mode or Slave emulation mode. This feature, "stop on error", allows the LIN/USB device to become passive and idle (will not transmit any more on the LIN bus) the moment any type of error is detected. The idea is that an automated test can be started and left alone and if it ever stops it is because an error occurred. The error will remain in the "current error code" which can be read with LinUsbReadStatus. The "stop on error" flag of course does not apply when the LIN/USB device is in PC-controlled mode. Keep in mind that this flag is stored in non-volatile memory aboard the LIN/USB device and it will use whatever it was last set to each time it powers up.

Return Code: This function returns a standard LIN/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 LIN/USB device being used. Only valid handles obtained using LinUsbOpen are accepted by this routine.

serr [input] – This argument contains the Stop on Error flag. If this is set to 1, the LIN/USB device in Master emulation mode or Slave emulation mode will stop transmitting anything on the LIN bus the moment an error is detected. The error that is detected will stick in the Error Code register and be available for reading with calls to LinUsbReadStatus. If this is set to 0, then a LIN/USB device in Master emulation mode or Slave emulation mode will continue to emulate the Master or Slave (continue executing the schedule table forever if in Master emulation mode for example) regardless of whether an error occurred. Errors can still be detected with calls to LinUsbReadStatus but will not stop the transmission if this flag is 0.

2.25. LIN USB SET MODE

Function: LinUsbSetMode

Library: LINUSB.DLL

C/C++ Calling Format:

```
int __stdcall LinUsbSetMode(
short handle,
unsigned char mode);
```

VB6 Calling Format:

```
Public Declare Function LinUsbSetMode _
  Lib "linusb.dll" _
  (ByVal handle As Integer, _
  ByVal mode as Byte) _
 As Integer 'returns 1 if successful
```

VB.NET Calling Format:

```
 32-bit:    Public Declare Function LinUsbSetMode Lib "linusb.dll"
(ByVal handle As Int16, ByVal mode As Byte) As Int32 'returns 1 if
successful
```

```
64-bit:    Public Declare Function LinUsbSetMode Lib "linusb64.dll"
(ByVal handle As Int16, ByVal mode As Byte) As Int32 'returns 1 if
successful
```

Description: This function is used to set the mode of operation of the LIN/USB device; whether PC-controlled, Master emulation, or Slave emulation. Note that if set to Master emulation or Slave emulation, the device will use the last loaded CLD configuration from the last call to LinUsbOpen with the mode indicated by that call. If this is a new instance of a Master or Slave emulation mode, then you should use LinUsbOpen instead which calls LinUsbMode already to set the mode already based on your argument selection to LinUsbOpen.

Return Code: This function returns a standard LIN/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 LIN/USB device being used. Only valid handles obtained using LinUsbOpen are accepted by this routine.

mode [input] – This argument contains the mode setting. Note that bits 7-2 of the mode input byte should be set to 0. Bits 0 and 1 completely specify the mode as follows: a value of 1 indicates Slave emulation mode; a value of 2 indicates Master emulation mode; a value of 0 indicates PC-controlled mode; and a value of 3 is not used and is reserved.

2.26. LIN USB SET FAST 1 STOP BIT

Function: LinUsbSetFast1StopBit

Library: LINUSB.DLL

C/C++ Calling Format:

```
int __stdcall LinUsbSetFast1StopBit(
short handle,
unsigned char fast);
```

VB6 Calling Format:

```
Public Declare Function LinUsbSetFast1StopBit
  Lib "linusb.dll" _
  (ByVal handle As Integer, _
  ByVal fast As Byte) _
  As Integer 'returns 1 if successful
```

VB.NET Calling Format:

```
64-bit: Public Declare Function _
        LinUsbSetFast1StopBit _
        Lib "linusb64.dll" _
        (ByVal handle As Int16, _
        ByVal fast As Byte) _
```

```
            As Int32 'returns 1 if ok
        'fast==0 --> 2 stop bits
        'fast==1 --> 1 stop bit
32-bit:    Public Declare Function _
            LinUsbSetFast1StopBit _
            Lib "linusb.dll" _
            (ByVal handle As Int16, _
            ByVal fast As Byte) _
            As Int32 'returns 1 if ok
        'fast==0 --> 2 stop bits
        'fast==1 --> 1 stop bit
```

Description: Normally, in order to be sure to provide enough processing time to handle fast bus traffic and USB traffic, the LIN/USB sends 2 stop bits after each byte. However, in order to create fast as possible transmissions, you can set the LIN/USB Converter to send 1 stop bit after each byte. The only cost is that some USB transmissions will take longer and possibly fail in the worst case latency situations on the PC. But this may be worthwhile (an occasional hiccup in the test procedure) for faster transmission of data if this is a requirement.

Return Code: This function returns a standard LIN/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 LIN/USB device being used. Only valid handles obtained using LinUsbOpen are accepted by this routine.

fast [input] – This argument contains the stop bit setting. A value of 0 indicates to use 2 stop bits. A value of 1 indicates to use 1 stop bit.

2.27. LIN USB SET INCREMENTING PAYLOAD

Function: LinUsbSetIncrementingPayload

Library: LINUSB.DLL

C/C++ Calling Format:

```
int __stdcall LinUsbSetIncrementingPayload(
short handle,
unsigned char inc);
```

VB6 Calling Format:

```
Public Declare Function LinUsbSetIncrementingPayload
  Lib "linusb.dll" _
  (ByVal handle As Integer, _
  ByVal inc As Byte) _
  As Integer 'returns 1 if successful
```

VB.NET Calling Format:

```
32-bit:      Public Declare Function _
         LinUsbSetIncrementingPayload _
         Lib "linusb.dll" _
         (ByVal handle As Int16, _
         ByVal inc As Byte) _
         As Int32 'returns 1 if ok
     'inc==0 --> static master payload
     'inc==1 --> incrementing master payload
64-bit:     Public Declare Function _
         LinUsbSetIncrementingPayload _
         Lib "linusb64.dll" _
         (ByVal handle As Int16, _
         ByVal inc As Byte) _
         As Int32 'returns 1 if ok
     'inc==0 --> static master payload
     'inc==1 --> incrementing master payload
```

Description: This feature allows the LINUSB device to send an incrementing count in the Master Task message first 2 bytes of the payload. If the setting is "0" indicating not to use an incrementing payload, then the LINUSB acts as normal – sending the static data from the master payload as defined in the CLD file. However, if the setting is "1" then the static data from the master payload defined in the CLD file defines the first value that is sent in the first transmission of the master task message. Each subsequent transmitted message increments this count. Note that this count is 2 bytes and that the Least Significant Byte (LSB) goes in the first position and the Most Significant Byte (MSB) goes in the second position. When the count reaches FFFF, it rolls over to 0000 again. This feature can be useful for making sure that an ECU receives each and every message sent from a master. Note this feature only comes into play when the LIN/USB device is setup as a Master Node from a CLD file.

Return Code: This function returns a standard LIN/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 LIN/USB device being used. Only valid handles obtained using LinUsbOpen are accepted by this routine.

inc [input] – This argument contains the incrementing payload setting. A value of 0 indicates a static payload. A value of 1 indicates an incrementing payload.

2.28. LIN USB TEST FRAM

Function: LinUsbTestFRAM

Library: LINUSB.DLL

C/C++ Calling Format:

```
int __stdcall LinUsbTestFRAM(
```

```
short handle);
```

VB6 Calling Format:

```
Public Declare Function LinUsbTestFRAM
  Lib "linusb.dll" _
  (ByVal handle As Integer)
  As Integer 'returns 1 if successful returns 12 if FRAM bad
```

VB.NET Calling Format:

32-bit: Public Declare Function LinUsbTestFRAM Lib "linusb.dll" (ByVal handle As Int16) As Int32 'returns 1 if ok or 18 if FRAM bad or 2 if cannot communicate

64-bit: Public Declare Function LinUsbTestFRAM Lib "linusb64.dll" (ByVal handle As Int16) As Int32 'returns 1 if ok or 18 if FRAM bad or 2 if cannot communicate

Description: This feature allows the LINUSB device's internal storage device (FRAM) to be tested. After extremely heavy use for a very extended period of time, the FRAM may wear out. It is rated for 1 billion write cycles. Send this command to test the FRAM. This command will take about 1-2 minutes to execute. Please be patient. It will return 1 if the FRAM is okay and it will return 12 if the FRAM is detected to be bad. Other error codes are possible (see Part 3).

Return Code: This function returns a standard LIN/USB return code. A return code of "1" indicates success. A return code of "12" indicates FRAM bad. Other values indicate specific errors (see Part 3).

Parameter Description:

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

2.29. LIN USB CLEAR BUFFERS

Function: LinUsbClearBuffers

Library: LINUSB.DLL

C/C++ Calling Format:

```
int __stdcall LinUsbClearBuffers(
short handle);
```

VB6 Calling Format:

```
Public Declare Function LinUsbClearBuffers _
  Lib "linusb.dll" _
  (ByVal handle As Integer) _
As integer 'returns 1 if successful
```

VB.NET Calling Format:

```
          32-bit: Public Declare Function LinUsbClearBuffers Lib
"linusb.dll" (ByVal handle As Int16) As Int32 'returns 1 if
successful

           64-bit:        Public Declare Function LinUsbClearBuffers
Lib "linusb64.dll" (ByVal handle As Int16) As Int32 'returns 1 if
successful
```

Description: This function has multiple purposes. It performs the following operations:

(1) It clears all receive and transmit buffers at all stages of the Windows PC DLL and LIN/USB device. This flushes out all pending requests and transmissions. (2) It clears the current device Error Code. It is reset to "No Error" which is a 4F error code. (This is the error code that can be read using LinUsbReadStatus). (3) For Master/Slave emulation mode, it re-starts the emulation from the beginning. This is useful if the emulation has stopped because the Stop on Error flag is set and you want to restart emulation. So it is sort of a "restart" function as well.

Return Code: This function returns a standard LIN/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 LIN/USB device being used. Only valid handles obtained using LinUsbOpen are accepted by this routine.

2.30. LIN USB READ FIRMWARE REVISION

Function: LinUsbReadFirmwareRevision

Library: LINUSB.DLL

C/C++ Calling Format:

```
int __stdcall LinUsbReadFirmwareRevision(
short handle,
unsigned char *rev_major,
unsigned char *rev_minor);
```

VB6 Calling Format:

```
Public Declare Function LinUsbReadFirmwareRevision _
Lib "linusb.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:

```
     32-bit:        Public             Declare             Function
LinUsbReadFirmwareRevision Lib "linusb.dll" (ByVal handle As Int16,
ByVal major_version As IntPtr, ByVal minor_version As IntPtr) As
Int32 'returns 1 if successful
```

```
64-bit:   Public Declare Function LinUsbReadFirmwareRevision
Lib "linusb64.dll" (ByVal handle As Int16, ByVal major_version As
IntPtr, ByVal minor_version As IntPtr) As Int32 'returns 1 if
successful
```

Description: This function reads the major and minor firmware version of the LIN/USB device. The major version refers to the version of the USB microcontroller whereas the minor version refers to the version of the LIN microcontroller. Versions range from 1-255.

Return Code: This function returns a standard LIN/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 LIN/USB device being used. Only valid handles obtained using LinUsbOpen are accepted by this routine.

major_version [output] – This argument should point to a byte that the routine will write to. The routine will write the current major version number to this byte which corresponds to the USB microcontroller firmware version.

minor_version [output] – This argument should point to a byte that the routine will write to. The routine will write the current minor version number to this byte which corresponds to the LIN microcontroller firmware version.

2.31. LIN USB CLOSE

Function: LinUsbClose

Library: LINUSB.DLL

C/C++ Calling Format:

```
int __stdcall LinUsbClose(
short handle,
unsigned char SN[6]);
```

VB6 Calling Format:

```
Public Declare Function LinUsbClose _
    Lib "linusb.dll" _
    (ByVal handle As Integer, _
     ByRef SerialNumber As Byte) _
  As Integer 'returns 1 if ok
```

VB.NET Calling Format:

```
32-bit:          Public  Declare  Function  LinUsbClose  Lib
"linusb.dll" (ByVal handle As Int16, ByVal SerialNumber As IntPtr)
As Int32 'returns 1 if ok

64-bit:        Public   Declare   Function   LinUsbClose   Lib
"linusb64.dll" (ByVal handle As Int16, ByVal SerialNumber As IntPtr)
As Int32 'returns 1 if ok
```

Description: This function closes a handle to a specified LIN/USB device and allows other programs to connect to the device. It should be called when the user is done working with a specific LIN/USB device but still needs to use the LINUSB DLL for other operations, possibly with other LIN/USB devices. Note that if a program is completely done with all LIN/USB device handling, the user should call LinUsbUnload instead.

Return Code: This function returns a standard LIN/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 LIN/USB device being closed. Only valid handles obtained using LinUsbOpen are accepted by this routine.

SerialNumber [input] – This argument can be filled with the serial number of the device being accessed. This argument is not really used however so it is not real critical to fill out this argument.

2.32. LIN USB UNLOAD

Function: LinUsbUnload

Library: LINUSB.DLL

C/C++ Calling Format:

```
int __stdcall LinUsbUnload(void);
```

VB6 Calling Format:

```
Public Declare Function LinUsbUnload _
    Lib "linusb.dll" _
    () _
  As Integer 'returns 1 if ok
```

VB.NET Calling Format:

```
    32-bit:         Public   Declare   Function   LinUsbUnload   Lib
"linusb.dll" () As Int32 'returns 1 if ok

64-bit:    Public Declare Function LinUsbUnload Lib "linusb64.dll"
() As Int32 'returns 1 if ok
```

Description: This function MUST be called when the application is done using the LINUSB DLL. This terminates all open handles and all threads that are used to monitor those handles. It allows other applications in the future to use the LIN/USB devices that were used by the current application.

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

Parameter Description:

None.

2.33. LIN USB BAUD RATE

Function: LinUsbBaudRate

Library: LINUSB.DLL

C/C++ Calling Format:

```
int __stdcall LinUsbBaudRate(
short handle,
short *baudrate);
```

VB6 Calling Format:

```
Public Declare Function LinUsbBaudRate _
   Lib "linusb.dll" _

   (ByVal handle As Integer, _
   ByRef BaudRate As Integer) _
   As Integer 'returns 1 if ok
```

VB.NET Calling Format:

```
32-bit:        Public Declare Function LinUsbBaudRate Lib "linusb.dll" (ByVal handle As Int16, ByVal BaudRate As IntPtr) As Int32 'returns 1 if ok and BaudRate filled in

64-bit:        Public Declare Function LinUsbBaudRate Lib "linusb64.dll" (ByVal handle As Int16, ByVal BaudRate As IntPtr) As Int32 'returns 1 if ok and BaudRate filled in
```

Description: This function can be used at any time if the user wants to know what baud rate the LIN/USB device is currently operating at. Upon successful return, the BaudRate argument is filled in with the current baud rate of the LIN bus that the LIN/USB device is configured at.

Return Code: This function returns a standard LIN/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 LIN/USB device being closed. Only valid handles obtained using LinUsbOpen are accepted by this routine.

BaudRate [output] – Upon successful completion, the integer pointed to by this argument will be filled in by the routine to contain the baud rate that the LIN/USB device is currently set to (in units of bits per second).

3. LIN USB DLL RETURN CODES

3.1. 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 ROUNTINE IN LINUSB DLL
5	MEMERR – OUT OF MEMORY
6	NODEV – NO LIN/USB DEVICE DETECTED
7	NONFCE – NO LIN/USB DEVICE INTERFACE DETECTED, IMPROPER INSTALLATION
8	NODETL – NO LIN/USB DEVICE DETAIL DETECTED, SYSTEM ERROR
9	NODETL2 – NO LIN/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 – FATAL ERROR – COULD NOT START LIN READ THREAD
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 MUTEX'S
17	WRONGPR – WRONG LIN PROTOCOL SPECIFIED (A MESSAGE WAS RECEIVED WHOSE CHECKSUM MATCHED A CHECKSUM THAT WOULD INDICATE A DIFFERENT PROTOCOL SETTING SHOULD BE USED TO RECEIVE THIS TYPE OF MESSAGE)
18	FRAMBAD – THE FRAM HAS BEEN TESTED AND IS BAD.

4. LIN USB DEVICE ERROR CODES

4.1. DEVICE ERROR CODES

00h Failed to bring the LIN bus line low. The LIN bus line was stuck high.

This usually indicates that the LIN bus is shorted to a high voltage; or that the LIN/USB device is incapable of transmitting due to a hardware error; or there might be too much capacitance on the bus causing the LIN signals to move too gradually to the proper levels. Sometimes putting a 1K or 10K resistor between the bus and 12 volts fixes this problem.

01h Inconsistent PARITY FIELD. The command ID is a 6 bit identifier that is protected by an extra 2 bits of parity. If these 2 bits of parity don't match, the entire message can be considered to be an invalid LIN message or possibly that noise or bus contention on the bus has altered the message and made it corrupt. In this case, the rest of the message is not received.

02h Inconsistent SYNC FIELD. The synchronization field was not detected to be the proper value, which is 55 hex. This usually indicates a wrong baud rate somewhere, bus contention, noise on the bus, or possibly bus capacitance or other hardware fault.

03h FRAMING ERROR. This indicates that the stop bit was not detected properly for a byte of the LIN message frame. This usually indicates a wrong baud rate somewhere, bus contention, or noise on the bus.

04h Failed to bring the LIN bus line high. The LIN bus line was stuck low. This usually indicates that there is too much capacitance on the bus causing the LIN signals to move too gradually to the proper levels. Sometimes putting a 1K or 10K resistor between the bus and 12 volts fixes the problem. This can also occur due to bus contention (two or more devices trying to talk at once) or noise on the bus. It can also indicate that the LIN bus line is shorted to ground in some cases.

05h RECEIVED ILLEGAL DATA WHILE THE BUS WAS ASLEEP. The LIN/USB device did not receive the proper wake-up pulse before active LIN message data was transmitted on the bus. This can occur sometimes if wakeups are disabled with the LinUsbSetWakeupEnable command or if a LIN node has sent a message without sending a wakeup pulse.

06h SLAVE NOT RESPONDING. This indicates a Master task message was sent (a single command ID) but no Slave task was ever sent from any source to complete the message. Note that according to the LIN specification, no LIN frame can be more than 1.4 times the total time for transmission of data so that there is very little latency that is allowed for the Slave task to respond in time to the Master task. Often if this is a real problem this is because the Slave task is not responding quick enough.

07h RX BAD LIN CHECKSUM. This indicates that a bad LIN checksum was received indicating that the LIN message is corrupt. This can sometimes happen when the baud rate is incorrect, bus contention, or noise on the bus.

08h BUS IS BUSY. This indicates the bus line was low every time the module tried to transmit on the bus. This could be an indication that the bus is shorted to ground or has an unusually high amount of traffic.

20h INVALID CHECKSUM FROM PC. This indicates the PC has sent the wrong checksum on an internal command frame. This could indicate a faulty LINUSB DLL or an abrupt termination of a LINUSB 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 LINUSB DLL or an abrpupt termination of a LINUSB DLL that was then restarted.

30h RECEIVE BUFFER OVERFLOW. This indicates the LIN/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 heavy, heavy bus traffic for a long period of time.

31h TRANSMIT BUFFER OVERFLOW. This indicates the LIN/USB device transmit buffer (containing commands and requests) has overflowed. If this is occurring then the user should take care to wait for a LinUsb.. routine to return before sending the next LinUsb.. command.

4Fh NO ERROR. Everything is okay.

50h TRANSMIT LIN BUS OVERFLOW. This indicates that the data queued up to send on the LIN/BUS has overflowed the LIN bus transmit queue. If this has occurred, the user should take care to wait for a LinUsb… routine to return before sending the next LinUsb… command.

51h RECEIVE LIN BUS OVERFLOW. This indicates the receive queue for the LIN bus has overflowed. This is an extremely rare occurrence. This would indicate that too much data has been received on the LIN bus before the LIN/USB device has been able to process it, which has never been observed to happen under normal circumstances.

60h EEPROM WRITE VERIFY ERROR. This indicates that after writing to Non-Volatile memory aboard the LIN/USB device, a subsequent check of the write failed to verify the data that was written indicating that the LIN/USB device should be replaced (or at least the small Non-Volatile memory chip connected to the GR16 should be replaced).

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 EEPROM LOCKOUT ERROR. This indicates the firmware is busy reading/writing to the EEPROM and cannot fulfill the current request. If this error condition persists even after repeated LinUsbClearBuffers calls then the module should be returned to Silicon Engines for analysis with explicit steps necessary to recreate the problem included.

5. ACKNOWLEDGEMENT COMMAND BYTES

5.1. ACKNOWLEDGEMENT COMMAND BYTES

These numbers might be seen during a call to LinUsbAckWaiting. They indicate Acknowledgements from the LIN/USB device to commands that were sent long ago. Each LinUsb 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 LinUsb… routine has allocated for waiting and the routine will return a FAILURE but the LIN/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 LIN/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 If the two most signicant bits are 0, then this corresponds to a LIN

	message transmission command where the message ID is equal to this value (not including parity check bits)
81h	Corresponds to LinUsbClearBuffers command
82h	Corresponds to LinUsbWakeupBus command
83h	Corresponds to LinUsbReadStatus command
84h	Corresponds to LinUsbReadFirmwareRevision command
85h	Corresponds to LinUsbQueryBaudRate command
91h	Corresponds to LinUsbSetContinualWakeup command
92h	Corresponds to LinUsbSetBaudRate command
93h	Corresponds to LinUsbSetMode or LinUsbOpen command
94h	Corresponds to LinUsbSetStopOnError command
95h	Corresponds to LinUsbSetWakeupEnable command
96h	Corresponds to LinUsbSetProtocol command
97H	Corresponds to LinUsbSetFast1StopBit command
98H	Corresponds to LinUsbSetIncrementingPayload command
A0h	Corresponds to a write to EEPROM command such as is done by LinUsbOpen
A1h	Corresponds to a read EEPROM command such as is done by LinUsbOpen
A2h	Corresponds to a read EEPROM single byte command
A3h	Corresponds to FRAM tested bad
A4h	Corresponds to FRAM tested good
A5h	Corresponds to FRAM test command
A6h	Corresponds to a program firmware operation, such as from the LIN USB Message Center
ADh	Corresponds to a prepare program firmware operation, such as from the LINUSB Message Center
AEh	Corresponds to a program JB16 firmware operation such as from the LINUSB Message Center
AFh	Corresponds to a program GR16 firmware operation such as from the LINUSB Message Center

6. REVISION HISTORY

6.1. REVISION C

Document renamed to show that it applies both the Model 9011 and to the Model 9004. Revision advanced from *PRM9004 Rev. B* to *PRM9011-9004 Rev. C*.

Information on Windows compatibility added to first page.

Calling formats updated throughout to include VB.NET 32-bit and 64-bit.

6.2. REVISION B

• **Sec 2.26-2.28:** Added.

• **Sec 5.1:** Added acknowledgement codes 97H, 98H, A3H, A4H, and A5H.

• **Sec 3.1:** Added return code 18.

6.3. REVISION A

Initial release.

☑