LEP DLL Manual

April 23, 2007
Document Version: 0.9
DLL Version: 0.9
Preliminary

Contents

1.0 Introduction

This document[1] describes the function of the LEP COM DLL. The LEP COM DLL, herein referred to as the DLL, allows a user's application to interface with the LEP peripheral products over Serial, USB and TCPIP ports. The goal of the DLL is to provide the user with a simple method of communicating to the LEP devices, without the communication overhead. The DLL also includes many useful extended functions that make interfacing with the LEP device(s) simple.

2.0 Requirements

2.1 Supports

Operating System Support:	Windows 98, ME, 2000, 2003 & XP.
LEP Interfaces:	MAC2002, MAC5000 & MAC6000.
DLL's:	Requires standard Windows DLL's.*
Development Platforms:	
Microsoft C/C++:	Version 6+
Microsoft VB:	Version 6+
Microsoft .Net	Any

*The standard LepCom.DLL is dynamically link to Window's DLL. A static link LepComStatic.DLL is also available for those who don't have the windows DLL environment. Note this file is quite large.

2.2 Files

To use the DLL you will need the following files.

LEPCOM.dll – This is the actual DLL file and must located in the directory your application is running from or the windows default DLL directory. The windows default DLL directory is usually C:\WINDOWS\SYSTEM.

LEPCOM.lib – This is the library file, which your C linker will use to properly link your application to the DLL. This file must be located in the directory of your source code. This file is only required for C applications.

LEPCOM.h – This is the C header file, which has the function prototyping in and useful constants that will be used when communicating with the DLL. This file must be located in the directory of your C source code. This file is only required for C applications.

LEPCOM.bas – This is the VB header file, which has the function prototyping in and useful constants that will be used when communicating with the DLL. This file must be located in the directory of your VB project and added to the project as a module or included in the source code file. This file is only required for VB applications.

LEPCOM.vbn – This is the VB.net header file, which has the function prototyping in and useful constants that will be used when communicating with the DLL. This file must be located in the directory of your VB.net project and added to the project as a module or included in the source code file. This file is only required for VB.net applications.

LEPCOM.cs – This is the C# header file, which has the function prototyping in and useful constants that will be used when communicating with the DLL. This file must be located in the directory of your C# project and added to the project as a module or included in the source code file. This file is only required for C# applications.

3.0 Using the DLL in your applications.

3.1 C Applications

There are two ways to use the DLL in C applications; they are Link Implicitly and Link Explicitly.

Implicitly link is the easiest to use. In Implicit linking the DLL is loaded when your application loads. To implicitly link to the DLL add the DLL library file (LepCom.lib) to your project and include the DLL header file (LepCom.h) in your source code file. Then simply call the DLL functions. All C code examples in this manual use the implicit method.

Explicitly linking allows your application to load the DLL at a defer time. For your application to Explicitly link to the DLL you will have to load the DLL with LoadLibrary() function. LoadLibrary loads the DLL, if not already loaded, and returns a handle to the DLL. Then using the DLL handle and the DLL function name of interest, call GetProcAddress() to obtain the address of the DLL function. GetProcAddress returns the function address, which you can use to call the DLL functions. When your done using the DLL you should call FreeLibrary() to release the DLL resources.

3.2 VB Applications

When using the DLL in a VB application you must declare the DLL functions. To declare DLL procedures use the Declare statement to prototype each function. Or you can just include the lepcom.bas file in your source code.

Declare Function *publicname* **Lib** "*libname*" **[Alias** "*alias*"**] [([[ByVal]** *variable* **[As** *type*] **[,[ByVal]** *variable* **[As** *type*]**...**])] **As Type**

Example

```
Declare Function LepComRead() Lib “LEPCOM.dll”  (ByVal buffer As String, ByVal size As Long) As Long
```

All the DLL functions have been declared in the LEPCOM.bas file. Simply add this file to your VB project and then call the DLL function from your code as you would any other function.

3.3 VB.Net Applications

VB.Net is similar to VB description above, except that in VB.Net LONG’s are now INT’s. And you need to use StringBuilder when defining your strings. For function prototyping you can either use the old VB declaration style (described above) or you can use the .NET ‘DllImport’. The ‘DllImport is the better method because it provides better parameter checking and gives you those nice auto fill drop down boxes in Visual Studio .NET.

Example

```
Imports System.Runtime.InteropServices
<DllImport(“LepCom.Dll”) > _
Public Shared Function LepComOpen() As Integer
End Function
```

3.4 C# Applications

When using the DLL in a C# application you must declare the DLL functions. To declare DLL procedures use the Declare statement to prototype each function. Or you can just include the LepCom.csc file in your source code or include the function declarations in your source file.

Example

```
// LepCom DLL simple Example
using System.Text;                          // for StringBuilder
using System.Runtime.InteropServices;   // DLL Import
public class LEP              // declare the LepCom DLL functions
{
      [DllImport("LepCom.Dll")] public static extern int LepComOpen();
}
LEP.LepComOpen();
```

4.0 Quick Start Example

This sections illustrates a simply example using the DLL. In all examples the following quick start code examples the program tries to: Open the LEP device. If the port cannot be opened the DLL will automatically display the 'Communication Setup' dialog. The program then sends a 'VER' command, waits for the return data, prints the returned data and closes the connection

4.1 C Example

Source Code:

```
#include "stdafx.h"
#include "string.h"
#include "conio.h"
#include "LepCom.h"


int main(int argc, char* argv[])
{
	printf("LepCom DLL simple example.\n");

	int status = LepComOpen();		// Open the LEP device

	if( status) {
		char buffer[1024];
		strcpy(buffer,"ver\r");				// Setup the Command
		LepComWrite(buffer, strlen(buffer));	// Send the Command
		LepComWait4LF(2000);				// Wait for the answer(LineFeed), up to 2s
		int size = LepComGetRxLen(0);		// Get the length of the rx
		LepComReadString(buffer, size);		// Get the rx buffer
		LepComClose();						// Close the port
		printf("%s", buffer);				// print the results
		}
	else
		printf("Unable to open LEP device.\n");

	printf("Press any key to continue.");
	while(!kbhit());					// wait for the anykey

	return 0;
}
```

Output:

```
LepCom DLL simple example.
Version no. :	8.300	MAC5000 Interface with USB

Press any key to continue.
```

4.2 VB Example

Source Code:

```
‘ DLL Functions Declarations
Private Declare Function LepComOpen Lib "LEPCOM.DLL" () As Long
Private Declare Function LepComClose Lib "LEPCOM.DLL" () As Long
Private Declare Function LepComWrite Lib "LEPCOM.DLL" (ByVal str As String, ByVal size As Long) As Long
Private Declare Function LepComWriteString Lib "LEPCOM.DLL" (ByVal str As String) As Long
Private Declare Function LepComRead Lib "LEPCOM.DLL" (ByVal str As String, ByVal size As Long) As Long
Private Declare Function LepComReadString Lib "LEPCOM.DLL" (ByVal str As String, ByVal size As Long) As Long
Private Declare Function LepComGetRxLen Lib "LEPCOM.DLL" (ByVal mode As Long) As Long
Private Declare Function LepComWait4LF Lib "LEPCOM.DLL" (ByVal TimeOut As Long) As Long


Sub Form_Load()         ' runs on form load

Dim buffer As String * 1042    ' All string passed to the DLL must be memory allocated
Dim status, size As Long

status = LepComOpen()                                   ' Open the LEP device
If status Then
   buffer = "ver" + Chr$(13)                    ' Setup the command
   status = LepComWrite(buffer, 4)                      ' Send the Command, 4 bytes long
   status = LepComWait4LF(2000)                         ' Wait for the LineFeed, wait up to 2seconds
   size = LepComGetRxLen(0)                             ' Get the Rx length
   status = LepComReadString(buffer, size)  ' read the Rx String
   status = LepComClose()                               ' Close the device
   Label1.Caption = buffer                              ' print the results
   Else                                                 ' Failed to open LEP device
      Label1.Caption = "Failed to open LEP device"
   End If

End Sub
```

4.3 VB.Net Example

Source Code:

```
Imports System.Text 'required for StringBuilder

Module Module1

‘ LepCom DLL Function Declaration
Private Declare Function LepComShowSetup Lib "LEPCOM.DLL" () As Integer
Private Declare Function LepComOpen Lib "LEPCOM.DLL" () As Integer
Private Declare Function LepComClose Lib "LEPCOM.DLL" () As Integer
Private Declare Function LepComWrite Lib "LEPCOM.DLL" (ByVal str As StringBuilder, ByVal size As Integer) As
Integer
Private Declare Function LepComWriteString Lib "LEPCOM.DLL" (ByVal str As StringBuilder) As Integer
Private Declare Function LepComRead Lib "LEPCOM.DLL" (ByVal str As StringBuilder, ByVal size As Integer) As Integer
Private Declare Function LepComReadString Lib "LEPCOM.DLL" (ByVal str As StringBuilder, ByVal size As Integer) As
Integer
Private Declare Function LepComGetRxLen Lib "LEPCOM.DLL" (ByVal mode As Integer) As Integer
Private Declare Function LepComWait4LF Lib "LEPCOM.DLL" (ByVal TimeOut As Integer) As Integer


    Sub Main()
        System.Console.WriteLine("LepCom DLL Simple VB.net Example")
        Dim buffer As New StringBuilder(1024)
        Dim status, size As Integer

        LepComShowSetup()               ' Show the setup dialog
        status = LepComOpen()           ' Open the LEP device
        If status Then
            buffer.Insert(0, "ver")             ' Setup the command
            buffer.Append(Chr(13))              ' Add the CR
            status = LepComWrite(Buffer, 4)     ' Send the Command, 4 bytes long
            status = LepComWait4LF(2000)        ' Wait for the LineFeed, wait up to 2seconds
            size = LepComGetRxLen(0)            ' Get the Rx length
            status = LepComReadString(Buffer, size) ' read the Rx String
            status = LepComClose()                  ' Close the device
            System.Console.WriteLine(buffer)        ' print the results

             Else
            System.Console.WriteLine("Unable to open LEP device.")
        End IF

        System.Console.WriteLine("Press Enter to Continue")
        System.Console.Read()


    End Sub

End Module
```

4.4 C# Example

Source Code:

```
// LepCom DLL simple Example
using System;
using System.Text;                          // for StringBuilder
using System.Runtime.InteropServices;   // DLL Import

public class LEP              // declare the LepCom DLL functions
{
      [DllImport("LepCom.Dll")] public static extern int LepComShowSetup();
      [DllImport("LepCom.Dll")] public static extern int LepComOpen();
      [DllImport("LepCom.Dll")] public static extern int LepComClose();
      [DllImport("LepCom.Dll")] public static extern int LepComWrite(StringBuilder str,   uint len);
      [DllImport("LepCom.Dll")] public static extern int LepComRead(StringBuilder buffer, uint size);
      [DllImport("LepCom.Dll")] public static extern int LepComWait4LF(uint TimeOut);
}

namespace LepComCS
{
      class Class1
      {
            static void Main(string[] args)
            {
            StringBuilder str = new StringBuilder(1024);                  // You must must predefined string with the DLL
            System.Console.WriteLine("LepCom DLL Simple C# example.");      // Print title

            LEP.LepComShowSetup();                                    // Open the setup dialog, comment out if not wanted
            int status = LEP.LepComOpen();                             // Open the LEP device
            if(status > 0)                                                  // If openned
                  {
                  str.Insert(0,"ver\r");                                 // Setup the command string
                  LEP.LepComWrite(str, (uint)str.Length);         // Send the command to the LEP device
                  uint size = (uint)LEP.LepComWait4LF(2000);      // Wait for LF
                  LEP.LepComRead(str, size);                              // Read the received data
                  System.Console.WriteLine(str);                     // Print the recevice data
                  LEP.LepComClose();                                          // Close the Lep device
                  }
            else
                  System.Console.WriteLine("Unable to open LEP device.");

            System.Console.WriteLine("Press Enter to Continue");
            System.Console.Read();
            }
      }
}
```

Output:

```
LepCom DLL Simple C# example.
Version no. :      8.300   MAC5000 Interface with USB

Press Enter to Continue
```

5.0 Functions

The DLL functions are divided into three groups they are Common, CAN and Extended functions. Common functions are the basic functions to open, setup, write, read and close the communication port. The extended functions are device specific, examples are read or write motor position.

For simplicity, prototyping for the function listed here are shown as follows. For specific language prototype and declaration see the LepCom.h, LepCom.bas, LepCom.vbn or LepCom.cs files.

U8	Unsigned Integer 1 Byte.
U16	Unsigned Integer 2 Bytes.
U32	Unsigned Integer 4 Bytes.
S8	Signed Integer 1 Byte.
S16	Signed Integer 2 Bytes.
S32	Signed Integer 4 Bytes.
D64	Floating point number 8 Bytes (64bit).

* If the data type end with a '*' it's a pointer to that data type.
e.g. U8* A pointer to an unsigned 1 Byte

All DLL functions are of _stdcall calling conventions and are unmanaged.

IMPORTANT: All data passed by reference/pointer must be pre-allocated in the users code. With .net application you must use 'StringBuilder' define in System.Text.

5.1 Common Functions

D64 LepComGetDLLVersion()
DLL Version v0.1
This function returns the version of the LepCom DLL. The return value is a 64 bit float point number.

S32 LepComOpen()
DLL Version v0.1
This function opens an LEP communication port. The function returns nonzero if the function succeeded, otherwise zero. If this function fails to open the communication port or if the configuration data stored in the registry is invalid or absent then the LEP Communication Setup Dialog is displayed (see LepComShowSetup() function below).

S32 LepComOpenEx(U8* LepComStruct, U32 Mode)
DLL Version v0.5
This function opens an LEP communication port with a passed set of parameters. The function returns nonzero if the function succeeded, otherwise zero. If this function fails to open the communication port or if the configuration data stored in the registry is invalid or absent and the Mode value is true then the LEP Communication Setup Dialog is displayed (see LepComShowSetup() function below). The LepComStruct is a data structure containing all the communication settings. See appendix for data structure.

S32 LepComClose()
DLL Version v0.1
This function closes the LEP communication port. Returns zero. It's important to close the communication port when you're done using it. If you don't you will have to reboot in order to reopen the port.

S32 LepComIsOpen()
DLL Version v0.1
This function return non-zero if opened otherwise zero.

S32 LepComWrite(U8* Buffer, U32 Length)
DLL Version v0.1
This function sends a byte string to the communication port. The Buffer parameter is a pointer to a byte string and the Length parameter tells the DLL how many bytes to send. If this function fails to write the data with in TxTimeOut microseconds the function will return zero. Otherwise the function will return the number of bytes written.

S32 LepComWriteString(U8* Buffer)
DLL Version v0.1
This function sends a NULL terminated byte string to the communication port. The Buffer parameter is a pointer to a NULL terminated byte string. This function is useful when working with NULL terminated strings. If this function fails to write the data with in TxTimeOut microseconds the function will return zero. Otherwise the function will return the number of bytes written.

S32 LepComRead(U8* Buffer, S32 Length)
DLL Version v0.1
This function reads data from the communication port. The Buffer parameter is a pointer to a byte string and MUST be pre-allocated (must be at least the size of Length parameter). This function will try to read Length bytes with in the RxTimeOut microseconds. Returns the length of the byte returned in the Buffer. When wait for returned data it best to use LepComGetRxLen() or LepComWait4LF() .

S32 LepComReadString(U8* Buffer, S32 Length)

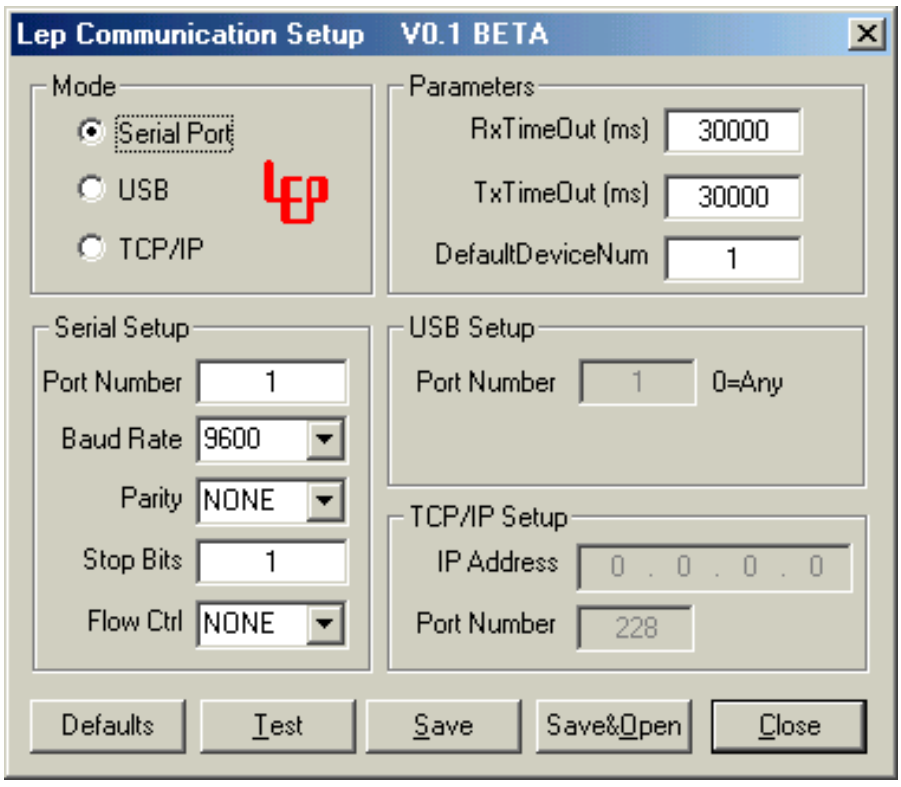
DLL Version v0.1
This function is the same as LepComRead() function except that it appends a NULL to the end of the returned string. This is help when dealing with NULL terminated strings. Note memory allocation must be at least the size of Length +1 parameter.

S32 LepComGetRxLen(U8 Mode)
DLL Version v0.1
This function returns the length of data in the received data buffer depending on the Mode type. If the Mode type is zero it returns the length of all the received data currently in the receive data buffer. If the Mode type is 10 (decimal for LineFeed) it returns the length of the received data till and including the present of the LineFeed character. This is very useful in high-level mode.

S32 LepComWait4LF(U32 TimeOut)
DLL Version v0.1
This functions returns when either a LineFeed has been found in the received data or the TimeOut period as expired. The function returns the length of the received data till and including the present of the LineFeed character. Returns zero if LineFeed is not found or TimeOut has expired. TimeOut is in Microseconds. This function is very useful in high-level mode.

LepComDumpRx()
DLL Version v0.2
This functions clears all the receive data in the receive data buffer.

S32 LepComShowSetup()
DLL Version v0.1
This functions displays the LEP Communication Setup Dialog. This Dialog allow the user to configure the communication port parameters. The configuration parameter are stored in the registry. Registry key HKEY_LOCAL_MACHINE\SOFTWARE\Ludl Electronic Products\LEP_DLL. Calling this function will close any pervious open port first. The functions returns non-zero if a communication port is opened, otherwise zero



The dialog allows the user to choose between the Serial, USB and TCP/IP ports. Within each port type the port parameters can be configured.

Default – Defaults the communication parameters.

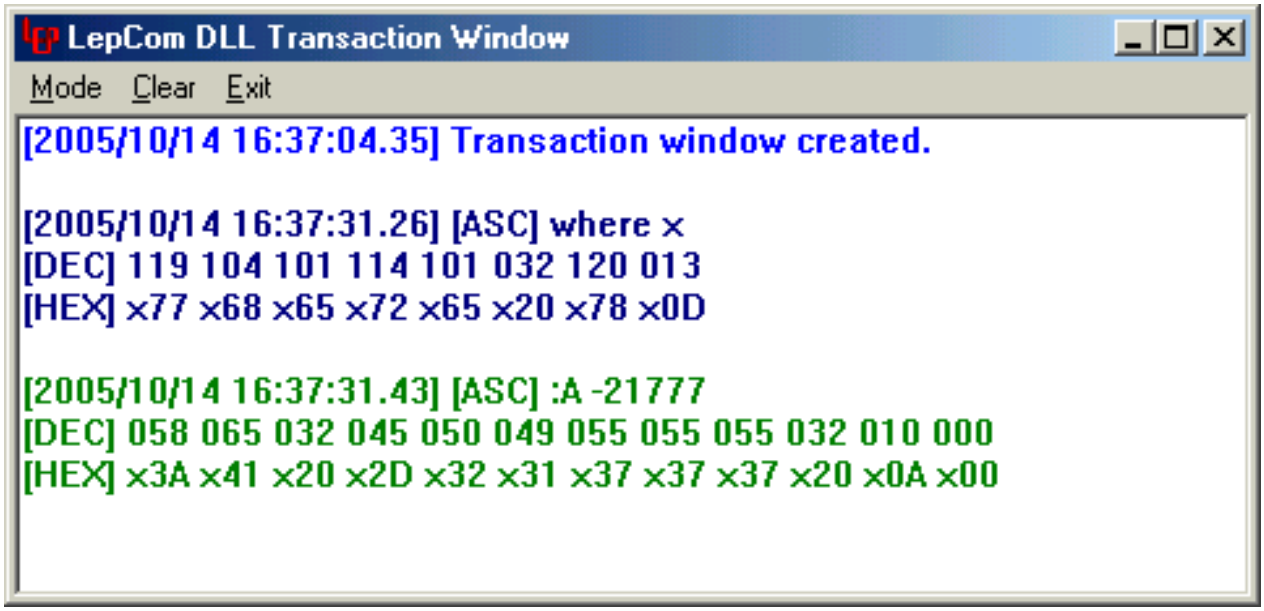
Test – Test is the port can be opened.

Save – Saves the parameters to the registry.

Save&Open – Saves the parameters to the registry and tries to open the communication port.

Close – Closes the dialog with out saving or opening.

LepComDebug(S32 Mode)
DLL Version v0.2
This functions displays the DLL Transaction Window. The transaction window show all error/warning messages as well as all the communications between the your program and the LEP device. Communications can be displayed in ASCII, DEC and HEX modes. To open the window set the Mode parameter to non-zero, to close it set the mode to zero (window can also be closed by the user). The non-zero Mode parameters are; ASCII = 0x01, DEC = 0x02 and HEX = 0x04, these modes can be or'ed together. If you pass a mode of '-1' then the last mode used will be used. These modes can also be changed by user when the window is opened. History is not saved when the transaction window is closed.



Color Syntax:

Red	- Errors and Warning Messages
Blue	- General Messages
Dark Gray	- User messages
Dark Blue	- Transactions sent to the LEP device
Green	- Transactions received from the LEP device

LepComUserMessage(U8* string)
DLL Version v0.4
Writes a user character string to the transaction window. Useful for debugging. Message string should be terminated with CR and LF. User messages are time stamped and displayed in dark gray.

S32 LepComShowDownLoad()
DLL Version v0.9
The function displays the LEP Firmware Down Load Dialog box. Using this dialog box the user can up date the firmware on the LEP hardware. While firmware downloading the user must terminate all communications with the LEP devices. This dialog opens as a modal dialog, therefore control is not passed back to the users code till the dialog is closed.

5.2 MAC6000 Commands

These functions send and receive the new MAC6000 commands. These commands are only available on the LEP MAC6000 products. See LEP MAC6000 programming manual for more information. When the user connects to the LEP hardware via the DLL for the first time, the DLL will auto detect* the LEP hardware type and interface version. With this information the DLL will choose the best command set to use. If the DLL detects the MAC6000 hardware it will enable Wrapper mode and Report mode. Wrapper mode encapsulates all non-MAC6000 commands (LEP Low Level and LEP High Level ASCII) in to the MAC6000 command protocol. The Wrapper mode allows the interface and DLL (or other programs) to send asynchronous commands between each other and therefore allows for an event driven architecture.
* Auto detect maybe disable in the LepComSetup Dialog.

S32 LepComSendCAN(U32 Module, U32 CmdNum, U32 Index, U32 Data)
DLL Version v0.7, MAC6000
Setup's a CAN message and sends it to a device. Return zero on success.

S32 LepComWait4CAN(U32 Module, U32 CmdNum, U32* IndexOut, U32 *DataOut)
DLL Version v0.7, MAC6000
Waits for a received CAN message who's Module, CmdNum and Index* matches. Note the index match test is only checked if the index is non-zero. So if you know the index you want, preset the IndexOut to that value. Otherwise you must preset it to zero. Return zero on success and nonzero of timeout or non-match error. Use standard DLL Rx TimeOut value.

S32 LepComSetIndex(U32 Module, U32 Index, U32 Data)
DLL Version v0.7, MAC6000
Sets a CAN index data value. Return zero on success.

S32 LepComGetIndex(U32 Module, U32 Index, U32* DataOut)
DLL Version v0.7, MAC6000
Gets a CAN index value. Return zero on success.

S32 LepComWriteM6(U32 IndexNum, U32 Length, U8* pData)
DLL Version v0.9, MAC6000
Sends a LEP Low Level or LEP High Level ASCII command to the interface. This function is the same as the LepComWrite() functions. The advantage of this functions over the LepComWrite() functions is that this function allows the user to set the index number. The index number can then be used in the LepComWaitM6() and LepComReadM6() commands to read the response data. The Index Num is optional and does not have to be used (when not using set index to zero). For a more robust design it is recommended that the user increment the index number each time the function is called. Then when reading the command response, use the Index number to match the correct response. This command is recommend for new designs.

S32 LepComReadM6(U32 IndexNum, U32 Length, U32 Mode, U8* pData)
DLL Version v0.9, MAC6000
This command is similar to the LepComRead() functions but only returns data in a response to the LepComWrietM6() command. If the IndexNum is zero the index match test is ignored. User should call the LepComWait4M6() functions before calling this function to determine data length. If mode is set to 1, a NULL character will be appended to the end of the data. This function will return when all data length is found or the time out period has expired. Note user must allocate data for the pData pointer in the user code. See the LepComRead() for more information. Returns number of bytes read or negative number on error.

S32 LepComWait4M6(U32 IndexNum)
DLL Version v0.9, MAC6000
This command is similar to the LepComWait4() functions but only returns data length on a command sent via the LepComWriteM6() command. If index number match is not desired set IndexNum to zero. Returns the length of data found for matching command.

MAC6000 Reports

S32 LepComGetReport(U32 Module, U32 ReportType, U32 Mode, U32 *IndexOut, U32* DataOut, U32* TimeOut)
DLL Version v0.9, MAC6000
Gets an asynchronous report from Module and ReportType passed (see MAC6000 command manual for report types).
Returns a negative number if the report never existed, returns zero is the report is new and has never been read and returns 1 if the report present but has been read before.
If mode is 1 the report is deleted. If mode is zero the report is mark read if it was present.
The IndexOut and DataOut are the index and data of the report. TimeOut is the age of the report in milliseconds.

S32 LepComReportRegister(U32 hWnd, U32 WMCode)
DLL Version v0.9, MAC6000
Configures the DLL to send Windows Messages to the users application. When enabled the DLL will send asynchronous MAC6000 Reports to the users application via the Windows Messaging system. This allows the user application to be event driver from the LEP hardware. For LEP report description see the MAC6000 command manual. The LEP Reports must be enabled by the user. The hWnd parameter must be set to the windows handle of your application. The WMCode is the Windows Message base code number, this number must be greater than or equal to 0x0400 (or WM_USER). WMCode below 0x0400 are reserved for Windows. Once enabled your application will receive these Windows Message on any LEP Report. The Window Message code will be equal to the LEP Report Number plus the WMCode passed when setting up this command up. The wParam MSB byte will be ModuleId, the next byte will be the Command Number and the last two bytes will be the report index. The lParam will be set to the report data value.

S32 LepComReportUnRegister(U32 hWnd)
DLL Version v0.9, MAC6000
Un-configures the DLL to send Windows Messages to the users application. This should be called when your program does not want LEP Report sent via the Windows Message system. Cancels the above command.

5.3 Extended Functions

These functions are device specific, they only work on specific modules. See LEP programming manual to see which commands can be used on which modules. Depending on the hardware you have, the DLL will automatically choose the best command method to execute the command. For example if your hardware is a MAC6000 then the new MAC6000 commands will be used. Otherwise the best LEP low level or LEP high level ASCII command will be used.

New functions will be constantly added, so check the LEP website for new releases of the DLL.

S32 LepComGetPosition(S32 Module)
DLL Version: v0.2
Uses Commands: 97
Returns: Position.
This function returns the current position.

S32 LepComSetPosition(S32 Module, S32 Position)
DLL Version: v0.2
Uses Commands: 65
Returns: 0 on Success, Non-Zero on error.
This function set the current position.

S32 LepComGetTarget(S32 Module)
DLL Version: v0.2
Uses Commands: 116
Returns: Target Position
This function returns the target position.

LepComSetTarget(S32 Module, S32 Target)
DLL Version: v0.2
Uses Commands: 84
Returns: none
This function set the target position.

S32 LepComGoto(S32 Module, S32 Target)
DLL Version: v0.2
Uses Commands: 74, 7
Returns: 0 on Success, Non-Zero on error.
This function move the module to the target position.

S32 LepComIsModulePresent(S32 Module, S32 Rescan)
DLL Version: v0.4
Uses Commands: Interface 8.0+ ISAVAIL, otherwise RCONFIG and 105 command.
Returns: 0 if not present, otherwise the module version times 10.
This function tests if the module is present. If the interface version is 8.0 or greater the ISAVAIL command is used, otherwise the RCONFIG and 105 commands are used. Once a module's version has been determined it is stored in the DLL, so that future calls to this functions are immediate. The stored value is saved while the DLL is opened by the user program. Closing the user program and therefore closing the DLL will reset this value. The user can, also force an active scan each time, by setting the rescan parameter to non-zero, default is zero.

S32 LepComIsBusy(S32 Module)

DLL Version: v0.2
Uses Commands: 63
Returns: 0 on Not Busy, Non-Zero on Busy.
This function returns the busy status of the module. If the module is not present, this command will always return non-zero (busy).

S32 LepComGetModuleStatus(S32 Module)
DLL Version: v0.4
Uses Commands: 126
Returns: Motor Status
This function returns the module status byte

S32 LepComGetMotorStatus(S32 Module)
DLL Version: v0.4
Uses Commands: 136
Returns: Motor Status
This function returns the motor status byte.

S32 LepComStop(S32 Module)
DLL Version: v0.2
Uses Commands: 66
Returns: 0 on Success, Non-Zero on error.
This function sends a stop command to the module.

S32 LepComGetVelocity(S32 Module)
DLL Version: v0.2
Uses Commands: 115
Returns: Top Speed
This function returns the current top speed of the module

S32 LepComSetVelocity(S32 Module, S32 Velocity)
DLL Version: v0.2
Uses Commands: 83
Returns: Top Speed
This function returns the top speed from the module.

S32 LepComGetAccel(S32 Module)
DLL Version: v0.2
Uses Commands:
Returns: 0 on Success, Non-Zero on error.
This function set the acceleration of the module.

S32 LepComSetAccel(S32 Module, S32 Accel)
DLL Version: v0.2
Uses Commands: 81
Returns: Acceleration (Garbis Units)
This function returns the acceleration from the module.

D64 LepComGetInterfaceVersion()
DLL Version: v0.2
Uses Commands: VER

Returns: Interface Version, Zero on failure.
This function the interface version.

S32 LepComGetIds(CString* cstr)
DLL Version: v0.2
Uses Commands: 105
Returns: The number of modules found
This functions returns array of CStrings. This function is only available to C++ programs because it uses the CStrings class. The strings contain the module number and module Id.

S32 LepComGet1Wire(S32 Module, S32 Device, U8* Buffer)
DLL Version: v0.2
Uses Commands: 205,206
Returns: 0 on Success, Non-Zero on error.
This functions reads the One Wire data from the Module and the OneWire Device. Module is any valid module number. Device is any valid OneWire device range is 0-3. The buffer is 40 byte and must be allocated before calling this function. See one wire data structure in appendix.

S32 LepComWriteRaw(S32 Module, S32 Command, S32 Size, S32 Data)
DLL Version: v0.5
Uses Commands: User defined
Returns: 0
This function writes zero to four data byte low level command to the module.

S32 LepComWriteCommand(S32 Module, S32 Command)
DLL Version: v0.4
Uses Commands: User defined
Returns: 0
This function writes a zero data byte low level command to the module.

S32 LepComWriteByte(S32 Module, S32 Command, S32 Data)
DLL Version: v0.4
Uses Commands: User defined
Returns: 0
This function writes a one data byte low level command to the module.

S32 LepComWriteWord(S32 Module, S32 Command, S32 Data)
DLL Version: v0.4
Uses Commands: User defined
Returns: 0
This function writes a two data byte low level command to the module.

S32 LepComWrite3Byte(S32 Module, S32 Command, S32 Data)
DLL Version: v0.4
Uses Commands: User defined
Returns: 0
This function writes a three data byte low level command to the module.

S32 LepComWriteDWord(S32 Module, S32 Command, S32 Data)
DLL Version: v0.4

Uses Commands: User defined
Returns: 0
This function writes a four data byte low level command to the module.

S32 LepComReadRaw(S32 Module, S32 Command , S32 Size)
DLL Version: v0.4
Uses Commands: User defined
Returns: Result of low level read.
This function reads a one to four data byte low level command from the module.

S32 LepComReadByte(S32 Module, S32 Command)
DLL Version: v0.4
Uses Commands: User defined
Returns: Result of low level read.
This function reads a one data byte low level command from the module.

S32 LepComReadWord(S32 Module, S32 Command)
DLL Version: v0.4
Uses Commands: User defined
Returns: Result of low level read.
This function reads a two data byte low level command from the module.

S32 LepComRead3Byte(S32 Module, S32 Command)
DLL Version: v0.4
Uses Commands: User defined
Returns: Result of low level read.
This function reads a three data byte low level command from the module.

S32 LepComReadDWord(S32 Module, S32 Command)
DLL Version: v0.4
Uses Commands: User defined
Returns: Result of low level read.
This function reads a four data byte low level command from the module.

6.0 Release Notes

Version	Date
0.1	Beta Release
0.2	Beta Release Added Extended functions
0.3	Second Beta Release.
0.4	Added Rx timeout to RS232 port. Updated the USB time out on change. Default the Rx and Tx timeouts to 3000ms. Made Transaction window smaller. Fixed DEC and HEX mode display in Transaction window. Added USB Driver Version option in Transaction window. Added Release Notes option in Transaction window. Added low level commands: LepComWriteByte,LepComReadByte,...
0.5	Added LepComWriteCommand, LepComWriteRaw and LepComReadRaw. Added LepComOpenEx. Fixed LepComUserMessage. Added Documentation Link.
0.6	Fixed USB Open&Save Error.
0.7	Added MAC6000 CAN Command Support.

If you would like to see a DLL function added to the DLL please email your request to support@ludl.com. If the request is merited we will add your requested function to the DLL.

Your code should always check the DLL version to see if your program will be compatible with the DLL version found on your machine.

Latest LEPDLL version can be found at http://www.ludl.com/beta/LepDLL.zip

7.0 Errata

None ☺

7.1 Coming soon:

1 – Error generation. So that you can use C:TRY()/CATCH() or VB:OnError().
2 – Extended Filter Wheel command support.
3 – MAC6000 CAN report support.

8.0 Sample Code

8.1 Simple C Example using Extended Functions.

This examples increments the motor position by 1000 and prints out the motor position while it's moving.

```
#include "stdio.h"
#include "LepCom.h"
int main(int argc, char* argv[])
{
	int Module = 1;		// requires a motor module at address one
	printf("Incrementing motor by 1000.\n");
	int status = LepComOpen();		// Open the LEP device
	if( status) {
		LepComGoto( LepComGetPosition(Module) + 1000 );		// Increment the motor
		while( LepComIsBusy(Module) ) ;		// loop while busy
			printf("Position = %d.\n", LepGetPosistion(Module));		// print the position
		printf("Done. Final Position = %d.\n", LepGetPosistion(Module));		// print the final position
		LepComClose();		// Close the port
		}

	else	printf("Unable to open LEP device.\n");
}
```

9.0 Appendix

OneWire Data Structure:

```
#pragma pack(1)
typedef struct _OneWireStruct {
    unsigned char       RomSerialNumber[8];     // ROM Area
    unsigned int        SerialNumber            // Commom Area
    unsigned int        MfrDate;
    unsigned int        MntDate;
    unsigned short      DeviceType;
    unsigned char       Version;
    unsigned char       Future[3];
    // Device Area
        union {
            unsigned char Raw[13];
            struct  _Stage {        // Stage
                unsigned short      AxisTravel;
                unsigned short      LeadScrewPitch;
                unsigned char       EncoderType;
                unsigned short      Resolution;
                unsigned char       MotorType;
                unsigned char       AuxEncoderType;
                unsigned short      AuxEncoderRes;
                unsigned char       LimitType;
                unsigned char       AxisId;
                } Stage;
            struct _FilterWheel{    // Filterwheel
                unsigned short      ApertureNumber;
                unsigned short      ApertureDiameter;
                unsigned char       OpenKick;
                unsigned short      HomeOffset;
                unsigned char       MotorType;
                unsigned char       OpenCoast;
                unsigned char       CloseKick;
                unsigned char       CloseCoast;
                unsigned char       RecordId;
                unsigned char       AxisId;
                } FilterWheel;
            struct _Focus{          // Focus
                unsigned short      ApertureNumber;
                unsigned short      LeadScrewPitch;
                unsigned char       EncoderType;
                unsigned short       Resolution;
                unsigned char       MotorType;
                unsigned char       AuxEncoderType;
                unsigned short      AuxEncoderRes;
                unsigned char       LimitType;
                unsigned char       AxisId;
                } Focus;
            struct _Indexer{        // Indexer
                unsigned short      IndexSize;
                unsigned short       LeadScrewPitch;
                unsigned char       EncoderType;
                unsigned short      Resolution;
                unsigned char       MotorType;
                unsigned char       AuxEncoderType;
                unsigned short      AuxEncoderRes;
                unsigned char       LimitType;
                unsigned char       AxisId;
                } Indexer;
            struct _Carousel {      // Carousel
                unsigned short      CarousalSize;
                unsigned short      LeadScrewPitch;
                unsigned char       EncoderType;
                unsigned short      Resolution;
                unsigned char       MotorType;
                unsigned char       AuxEncoderType;
                unsigned short      AuxEncoderRes;
                unsigned char       LimitType;
                unsigned char       AxisId;
                } Carousel;
        } As;       // end of union

        unsigned char       CRC;
    } OneWireStruct, *pOneWireStruct;
```

LepComStruct Data Structure

```
typedef struct _LepComStruct {
	BYTE Type;			// 0=uninitialized, 1 = RS232, 2=USB, 3=TCPIP
	BYTE RS232Port;

	BYTE Baud;			// by index
	BYTE Parity;
	BYTE StopBits;
	BYTE FlowCtrl;
	BYTE DeviceNum;

	BYTE USBPort;

	DWORD IPAddress;
	BYTE IPPort;

	int	RxTimeOut;
	int	TxTimeOut;

	BYTE Verbose;
	BYTE DebugWindow;

} LepComStruct, *pLepComStruct;
```