LADYBUG TECHNOLOGIES LLC

LB_API2 Supplemental Interface

X86 Interface for 64 bit systems

Jon Sigler

8/20/2013

Test system developers may be required to transition from 32 bit to 64 bit systems. Ladybug has supported this transition in the past by having direct support for both 32 bit and 64 bit environments. However, cross platform support has been limited. This supplement addresses the need for cross bitness platform support for LadyBug products.

Contents

Overview

This is intended be a practical response to the cross platform requirements.

The following definitions apply:

- 32/32 – refers to a 32 bit application running in a 32 bit environment (such as XP)
- 64/64 – refers to a 64 bit application running in a 64 bit environment (such as 64 bit Windows 7)
- 32/64 – refers to a 32 bit application running in a 64 bit environment

I may use the term “conventional” to refer to 32/32 and 64/64 solutions collectively. And I may refer to 32/64 as “cross-bit” or “cross-bitness” solutions.

Sometimes I’ll refer to the INI or ini file. By this I mean the file named LB_API2_32_64.ini

Using the Solution

System Requirements and Preparation

- Windows 7, 64 bit or later
- LadyBug applications (64 bit driver) must be installed and functioning
- Microsoft Visual Studio 2010 (optional)
- The zipped file must be unzipped to directory of your choosing
- LadyBug power sensor
- RF Power source

Quick Start

If you have a 32 bit solution and you want to run on Windows 7, 64 bit this outlines what you’ll need to do. It should be something like this:

1. Install the LadyBug power meter application on the target Windows 7 64-bit computer. Make sure the power meter application works before going on to the next steps.
2. Get your executable and its development environment on the 64 bit computer.
3. Copy the “necessities” directory into your executable directory
4. Run the VB test harness to check things out (it should be in the executable directory). If it runs then the system is working.
5. Change your application so it points to LB_API2_32_64.dll instead of LB_API2.dll. LB_API2_32_64.dll adds three calls to the power meter interface.
6. Call StartSocket() before calling anything else (e.g. LB_SensorCnt()). See the VB.Net or C# code for examples. It’s pretty simple.
7. Your code should now work.
8. Call StopSocket() when you are shutting down.

There is an Excel example and we have NI LabVIEW stuff coming in the next few days.

More on Getting up and Running

Note: The test harness applications can be run as stand-alone applications. To do so, go to the following directory and run:
<unzipped location>\Sockets\TestHarness_VB_NET\bin\Debug\ TestHarness_VB_NET.exe
…or…
<unzipped location>\Sockets\TestHarness_CSharp\bin\Debug\ TestHarness_CSharp.exe

If you have Visual Studio 2010, open the previously unzipped "LB_Sockets" solution. The solution contains five projects. One project is the socket client (32 bit side). One is the socket server (64 bit side), two are test harnesses and one is the RegsitryEditor. We'll be running one of the test harnesses. Run either the following two projects:

- Test Harness_VB_Net (more complete)
- Test Harness_CSharp

After starting the project of your choosing use the following process:

- Click the `LB_SensorCnt()` button in the upper left. If you have a sensor connected the count should show a positive number.
- Click the `LB_SensorList()` button. A list of sensors should appear in adjacent list box.
- Select a sensor by clicking on one of the sensors in the list. The current device information should be filled in and appear to the right of the list box.
- Select the "**`Init. CW & Pulse Mod`**" tab.
- In the upper left area of the tab click the "`LB_InitializeSensor_Addr()`" button. Wait about 5 seconds. The word "Success" should appear next to right of the button.
- Locate the **"LB_MeasureCW()"** button on the right side of the same tab.
- Click the button. The measure power level should appear in the text box below the button.

Additional test harness information is available in the documentation accompanying these examples (see documentation folder in the VB.NET or C# projects) or refer to documentation shipped with the original sample code supplied.

Next Steps

There are two additional items which need attention.

- Ensuring that the paths are set correctly using the best scheme.
- Modifying your code to start the sockets

Paths and more Paths

If you examine the description at the end of the document you'll note that this solution is composed of a few pieces. And these pieces must work together. As you'll see in the next section, your call to StartSockets starts the solution. For the solution to work the following has to occur:

- A “StartSockets…” call must be made from your application.
- LB_API2_32_64.dll must know where LB_SocketServer.exe is located. It assumes that LB_SocketServer.exe is in the same directory in which it is located.
- LB_API2_32_64.dll and LB_SocketServer.exe must use the same socket. The default socket or port is 27713. This is modifiable.
- The INI file should be in the same directory (LB_API2_32_64.INI)
- LB_API2.dll must be located in the same directory as LB_SocketServer.exe

Under normal circumstances if you co-locate these pieces things will work just fine. This is accomplished by asking the system for the current execution path. However, not all development systems play well with others. Some systems bias or change the execution path.

To resolve these issues we have provided three means of startup. The three means of startup are:

1. INI file initialization – this requires the INI file to be properly set and then you must start by calling StartSocket. This will start by looking for specific registry entries. If these entries do not exist or the use INI file bit is set, then the INI file is used.
2. Explicit initialization – requires that you call StartSocketExplicit. In this case the registry and INI file is ignored.
3. Registry initialization – this requires registry entries to be set properly and then call StartSocket. Note that if the use INI file bit is set, the registry is ignore and INI file is used.

First is the easiest and is summarized below:

- Co-locate all the various pieces of the solution in your executable directory.
- Fill out the INI file to your liking.
- Call StartSocket (see next section) – then proceed normally (your old code will work)

However, this method can fail because some development systems (most notably, NI LabVIEW) resets the executable path to Windows\System32 (or other another path) when loaded. This is most problematic during development. So that when LB_API2_32_64.dll tries to start LB_SocketSever.exe it won’t be able to find LB_SocketServer because the path is set to the Windows\System32 directory.

This means that you must override the path information normally obtained from the operating system. You can use either of the next two methods to accomplish this.

Note: Clearly you can choose to relocate the items to the path preferred by your development environment. However, I’ll leave such exercises to those more familiar with the development environment in question.

Instead of the normal call, StartSocket, call StartSocketExplicit. This call allows you to pass the path and socket number explicitly to LB_API2_32_64.dll. The path must be the correct path and all the various pieces are co-located in this path.

Another alternative is to use the registry. We have provided a small application that allows you to setup the registry (yes we've provide the code). If you choose this method you can call StartSocketl. All of the path information will be obtained from the registry. This alternative is very flexible in that you can relocate and executable without having to rebuild – very handy.

One of these methods should allow you to start up regardless of your environment.

Code Modifications

These test harnesses are the same test harnesses that have been previously provided with a few vital exceptions. The first call is made when the application starts and the form is loaded. A final call is made when the application ends.

VB.NET

```
Change #1:
Private Sub RgrTest_Tabbed_Load(sender As Object, e As System.EventArgs) Handles
Me.Load
        Dim rslt As Integer = StartSocket()
End Sub

Change #2:
Private Sub RgrTest_Tabbed_FormClosing(sender As Object, e As
System.Windows.Forms.FormClosingEventArgs) Handles Me.FormClosing
        Dim rslt As Integer = StopSocket()
End Sub
```

C#

```
Change #1:
private void RgrTest_Tabbed_Load(object sender, EventArgs e)
{
        int rslt = LB_API2_Declarations.StartSocket();
}

Change #2:
private void RgrTest_Tabbed_FormClosing(object sender, FormClosingEventArgs e)
{
        int rslt = LB_API2_Declarations.StopSocket();
}
```

The StartSocket call must be made prior to any call to the LadyBug API. Also, when the application closes call StopSocket for an orderly shutdown.

```
Change #3:
```

Finally, the DLL called by the 32 bit application has been change from LB_API2.dll to LB_API2_32_64.dll. Below is a typical declaration for this application:

```
Public Declare Function LB_SensorCnt Lib "LB_API2_32_64.dll" () As Integer
```

For those familiar with the 32/32 or the 64/64 solutions, you'll note the calls are identical except for the library name contained in the declaration of the API in the using application. As stated previously, the library name has changed from **`LB_API2.dll`** to **`LB_API2_32_64.dll and three calls have been added`**. In all other respects the code is identical. The three calls are StartSocket, StartSocketExplicit and

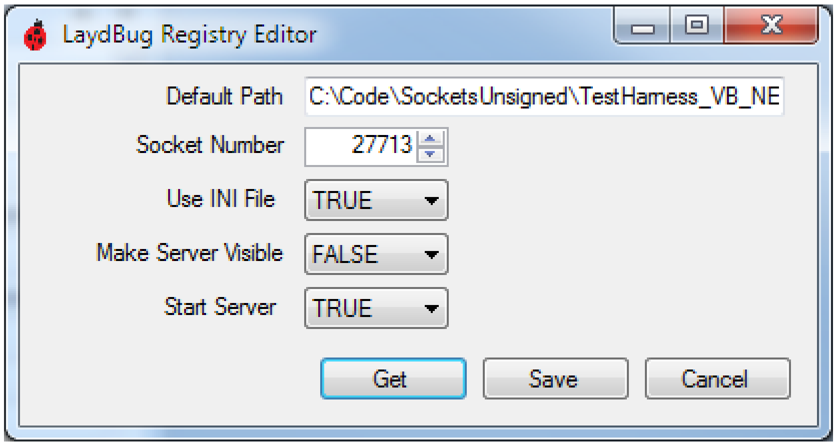
Stop Socket. Either StartSocket or StartSocketExplicit must be called before any other call is made to LB_API2_32_64.dll. StopSocket should be called when shutting down.

So software used to communicate with the LadyBug sensors via 32/32 or 64/64 can be used as is with the following modifications excepting the previously noted changes. Again, those modifications are the changing of the library to `LB_API2_32_64.dll` and the need to call "`StartSocket`" before making any calls to the LadyBug API and "`StopSocket`" on shut down.

Over 100 calls have been ported - virtually all of the power meter or LB_* calls. The pulse profiling (PP_*) calls have not been ported. However, the source code for the sockets interface is provided and you are free to add any calls you wish. The calls provided in this interface are listed in the appendix.

Using the Registry

If you have the VS 2010 environment open you'll see the RegistryEditor project. If you run this project you'll see the following window:



Default Path	C:\Code\SocketsUnsigned\TestHamess_VB_NE
Socket Number	27713
Use INI File	TRUE
Make Server Visible	FALSE
Start Server	TRUE

The application is pretty selfexplanatory. The code is available and is very straight forwared. But you can get help while running the app by getting focus on the control and then pressing F1. Here is a summary:

- **Default Path** – location of the executables and the LB* files
- **Socket Number** (or Port Number) – normally it can be between 1 and 65535. I've constrained it to between 10,000 and 60,000.
- **Use INI file** – if true it will use the INI file instead of the registry. This allows you to retain registry while temporarily using an INI file. If it is false it will use the values save in the registry.
- **Make Server Visible** – if true the command line interface will be visible. If false (normal operation) it will not be visible but you may get a flash as it appears and disappears quickely.
- **Start Server** – if true (default) it will start the socket server normally. If, usually for diagnostic purposes, it is false then you'll need to start LB_SocketServer manually or by some other means.

Get button retrieves the settings from the registry. Save, saves the settings to the registry. And Cancel exits the application without saving.

Deployment

All of the files must be collocated to function properly. These include the following as a minimum assuming all have been compiled as release. If some have been compiled as debug additional files may be required :

- LadyBug applications or driver installed
- Your executable and any additional supporting files
- LB_API2.dll (64 bit version)
- LB_SocketServer.exe
- LB_API_32_64.dll
- lb_api2_32_64.ini

Possible Issues

- One problem has already been covered in detail. This is related to the executable path. We've provided three methods of starting the system. One of these methods should overcome any issue assuming all of the various pieces are co-located.
- The server application is a console application with the console being hidden. You may see a quick flash when the socket application loads. This is the loading and unloading of the console window. If the console remains visible check the following two items:
  - Lb_api2_32_64.ini is collocated
  - The value of [VISIBLE]COMM = 0
- When communicating via sockets, port conflicts invariably arise. The port number used by the LadyBug 32/64 solution in an INI file. This INI file is accessed by both the socket client and socket server. Change or revise the port number as required. The contents of **`lb_api2_32_64.ini`** are shown below:

  ```
  [COMM]
  SOCKET=27713
  [VISIBLE]
  STATE=1
  ```

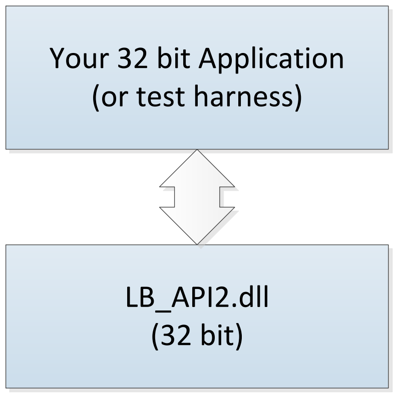
  - The [COMM] SOCKET entry controls the port number used.
  - The [VISIBLE] STATE variable controls whether the server console is visible. Making it visible can be handy for diagnosis.
- Since this application uses sockets (localhost only) you'll need to ensure it is enabled. For this you'll need the following entry in "C:\WINDOWS\System32\drivers\etc\host" shown below:

  ```
  # localhost name resolution is handled within DNS itself.
  127.0.0.1       localhost
  ```

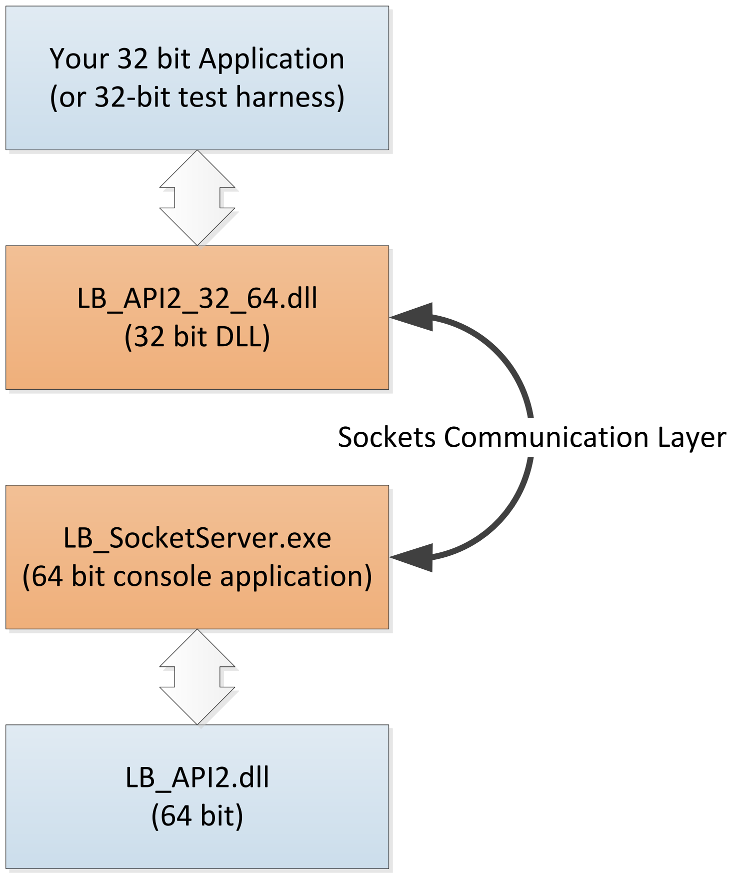
- IIS (internet information services) must be enabled, on and unblocked by your virus program.
- In the extreme, you may notice some measurement speed degradation. This is normal and unavoidable. If you must have the speed then you'll need to employ conventional solutions.

Description

Accompanying this document is a pair of test harnesses (VB.Net and C#) and components needed to resolve this issue. Below is a depiction of how our 32/32 bit and 64/64 bit (same general diagram applies to both):



And what follows is a depiction of how 32/64 (cross-bit) support is achieved:



The solution we chose was to interpose a sockets communication layer between the 32 bit application and the 64 bit LadyBug API. The interface seen by the 32-bit application is, in general, identical to LB-API2.dll interface. Three calls have been added to the interface:

- StartSocket() which starts the socket server and initializes the sockets communication layer. This call uses the INI file or registry settings. This (or StartSocketExplicit) must be called once before any other calls are made.
- StartSocketExplicit(path, socket) starts up the socket server and initializes the sockets communication layer. This call uses the path and socket information passed to it thus overriding INI file or registry settings. This (or StartSocket) must be called once before any other calls are made.
- StopSocket() which closes the sockets communication layer in an orderly fashion.

Source for LB_API2_32_64.dll and Socket Server have been provided. So that if you need to supplement the current API you may do so.

Appendix – List of Function Calls

StartSocket
StartSocketExplicit
StopSocket
LB_DriverVersion
LB_SensorCnt
LB_BlinkLED_SN
LB_BlinkLED_Addr
LB_BlinkLED_Idx
LB_SensorList
LB_GetAddress_SN
LB_GetAddress_Idx
LB_SetAddress_SN
LB_AddressConflictExists
LB_WillAddressConflict
LB_GetSerNo_Addr
LB_ChangeAddress
LB_GetModelNumber_SN
LB_GetModelNumber_Addr
LB_IsSensorConnected_SN
LB_IsSensorConnected_Addr
LB_SetAutoPulseEnabled
LB_GetAutoPulseEnabled
LB_SetCWReference
LB_GetCWReference
LB_SetDoubleSidedLimit
LB_GetDoubleSidedLimit
LB_SetDutyCycleEnabled
LB_GetDutyCycleEnabled
LB_SetDutyCyclePerCent
LB_GetDutyCyclePerCent
LB_GetFirmwareVersion
LB_GetIndex_SN
LB_GetIndex_Addr
LB_SetLimitEnabled
LB_GetLimitEnabled
LB_SetMeasurementPowerUnits
LB_GetMeasurementPowerUnits
LB_GetModelNumber_Idx

LB_SetOffsetEnabled
LB_GetOffsetEnabled
LB_SetPulseReference
LB_GetPulseReference
LB_SetResponseEnabled
LB_GetResponseEnabled
LB_SetResponse
LB_SetResponse_Flat
LB_GetResponse
LB_GetResponse_Flat
LB_GetSerNo_Idx
LB_SetSingleSidedLimit
LB_GetSingleSidedLimit
LB_InitializeSensor_Idx
LB_MeasureCW_PF
LB_MeasurePulse_PF
LB_ReadStateFromINI
LB_WriteStateToINI
LB_Recall
LB_Store
LB_ResetCurrentState
LB_ResetRegStates
LB_SetAddress_Idx
LB_SetOffset
LB_GetOffset
LB_SetCalDueDate
LB_GetCalDueDate
LB_GetCalOptExpDate
LB_GetWtyOptExpDate
LB_GetCalAndWtyOption
LB_GetRecorderOutOption
LB_GetBestMatchOpt
LB_GetTriggerOpt
LB_GetFilterOpt
LB_InitializeSensor_SN
LB_InitializeSensor_Addr
LB_GetFrequency
LB_SetFrequency
LB_GetAverages
LB_SetAverages
LB_GetPulseCriteria
LB_SetPulseCriteria

LB_GetTTLTriggerInEnabled
LB_SetTTLTriggerInEnabled
LB_GetTTLTriggerInInverted
LB_SetTTLTriggerInInverted
LB_GetTTLTriggerInTimeOut
LB_SetTTLTriggerInTimeOut
LB_GetTTLTriggerOutEnabled
LB_SetTTLTriggerOutEnabled
LB_GetTTLTriggerOutInverted
LB_SetTTLTriggerOutInverted
LB_GetAntiAliasingEnabled
LB_SetAntiAliasingEnabled
LB_MeasureCW
LB_MeasurePulse
LB_GetExtendedAveragingEnabled
LB_SetExtendedAveragingEnabled
LB_GetExtendedAveraging
LB_SetExtendedAveraging
LB_ResetExtendedAveraging
LB_GetMaxHoldEnabled
LB_SetMaxHoldEnabled
LB_ResetMaxHold
LB_Get75OhmsEnabled
LB_Set75OhmsEnabled
LB_IsDeviceInUse_Addr
LB_IsDeviceInUse_SN
LB_MeasureBurst_DBM