Published on *LabJack* (http://labjack.com)

U12 User's Guide

The complete manual for the LabJack U12. To make a PDF of the whole manual, click "Export All" towards the upper-right of this page. If you are looking at a PDF or hardcopy, realize that the original is an online document at http://labjack.com/support/u12/users-guide.
User's Guide

1 - Installation

The LabJack U12 requires a PC running Windows 98SE, ME, 2000, or XP. To determine your operating system version, go to:

Start => Settings => Control Panel => System => General

and make sure the version number is 4.10.2222 or higher (Win98SE=4.10.2222, WinME=4.90.3000, Win2000=5.0.2195, WinXP=5.1.XXXX).

It does not matter if the hardware or software is installed first.

If you experience installation problems on Windows 98 Second Edition, before contacting us, please go the Windows 98 Second Edition support page.

1.1 - Hardware Installation

With the PC on and using the included cable, connect the LabJack U12 to the USB port on the PC or USB hub. The USB cable provides power and communication for the LabJack U12. The status LED should immediately blink 4 times (at about 4 Hz), and then stay off while the LabJack enumerates.

Enumeration is the process where the PC's operating system gathers information from a USB device that describes it and it's capabilities. The low-level drivers for the LabJack U12 come with Windows and enumeration will proceed automatically. The first time a device is enumerated on a particular PC, it can take a minute or two, and Windows might prompt you about installing drivers. Accept all the defaults at the Windows prompts, and reboot the PC if asked to do so. The Windows Installation CD might also be needed at this point. Make sure a CD with the correct version of Windows is provided. Enumeration occurs whenever the USB cable is connected, and only takes a few seconds after the first time.

When enumeration is complete, the LED will blink twice and remain on. This means Windows has enumerated the LabJack properly.

If the LabJack fails to enumerate:

- Make sure you are running Windows OS version 4.10.2222 or higher,
- Try connecting the LabJack to another PC,
- Try connecting a different USB device to the PC,
- Check our online forum and/or contact LabJack.

1.2 - Software Installation

Although, the low-level USB drivers for the LabJack are included with Windows, high-level drivers are needed to send and receive data. Get started by going to labjack.com/support/u12.

When the LabJack installation is finished, it will start the National Instruments LabVIEW Run-Time Engine (LVRTE) setup. The LVRTE is required for the example applications such as LJtest. If prompted to reboot after this installation, go ahead and do so. Virus scanners can often interfere with the installation of the LVRTE. If you have trouble running the example applications, repeat the LabJack software installation to make sure the LVRTE is installed.

To test the installation, start LJtest by selecting

Start => Programs => LabJack => LJtest.

Make sure “Test Fixture Installed” and “Continuous” are not selected, and press the “Run” button. LJtest will step through 8 separate tests and all should pass.

2 - Hardware Description

The external features of the LabJack U12 are:

- USB connector,
- DB25 digital I/O connector,
- Status LED,
- 30 screw terminals.

The USB connection provides power and communication. No external power supply is needed. The +5 volt connections available at various locations are outputs, do not connect a power supply.

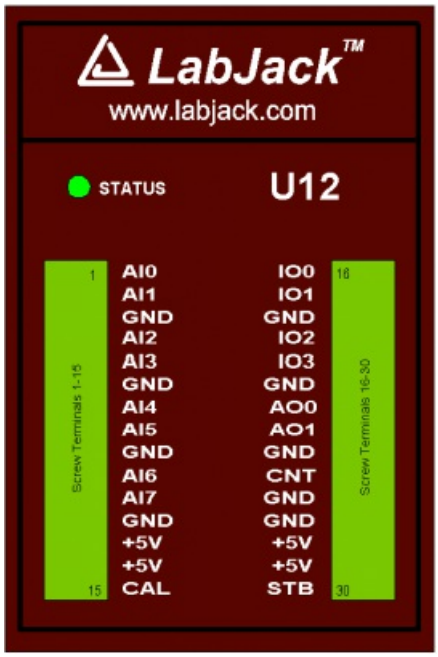


Figure 2-1. LabJack U12 top surface.

Figure 2-1 shows the top surface of the LabJack U12. Not shown is the USB and DB25 connector, which are both on the top edge. The DB25 connector provides connections for 16 digital I/O lines, called D0-D15. It also has connections for ground and +5 volts. All connections besides D0-D15, are provided by the 30 screw terminals shown in Figure 1. Each individual screw terminal has a label, AI0 through STB.

The status LED blinks 4 times at power-up, and then blinks once and stays on after enumeration (recognition of the LabJack U12 by the PC operating system). The LED also blinks during burst and stream operations, unless disabled. The LED can be enabled/disabled through software using the functions AISample, AIBurst, or AIStreamStart. Since the LED uses 4-5 mA of current, some users might wish to disable it for power-sensitive applications.

2.1 - AI0-AI7

Hardware
The LabJack U12 has 8 screw terminals for analog input signals. These can be configured individually and on-the-fly as 8 single-ended channels, 4 differential channels, or combinations in between. Each input has a 12-bit resolution and an input bias current of ±90 µA.

- Single-Ended: The input range for a single-ended measurement is ±10 volts.
- Differential channels can make use of the low noise precision PGA to provide gains up to 20. In differential mode, the voltage of each AI with respect to ground must be between +20 and -10 volts, but the range of voltage difference between the 2 AI is a function of gain (G) as follows:

```
G=1    ±20 volts
```

```
G=2    ±10 volts
G=4    ±5 volts
G=5    ±4 volts
G=8    ±2.5 volts
G=10   ±2 volts
G=16   ±1.25 volts
G=20   ±1 volt
```

The reason the range is ±20 volts at G=1 is that, for example, AI0 could be +10 volts and AI1 could be -10 volts giving a difference of +20 volts, or AI0 could be -10 volts and AI1 could be +10 volts giving a difference of -20 volts.

The PGA (programmable gain amplifier, available on differential channels only) amplifies the AI voltage before it is digitized by the A/D converter. The high level drivers then divide the reading by the gain and return the actual measured voltage.

Figure 2-2 shows a typical single-ended connection measuring the voltage of a battery. This same measurement could also be performed with a differential connection to allow the use of the PGA. In general, any single-ended measurement can be performed using a differential channel by connecting the voltage to an even-numbered analog input, and grounding the associated odd-numbered analog input (as shown by the dashed connection to AI1 in Figure 2-2).

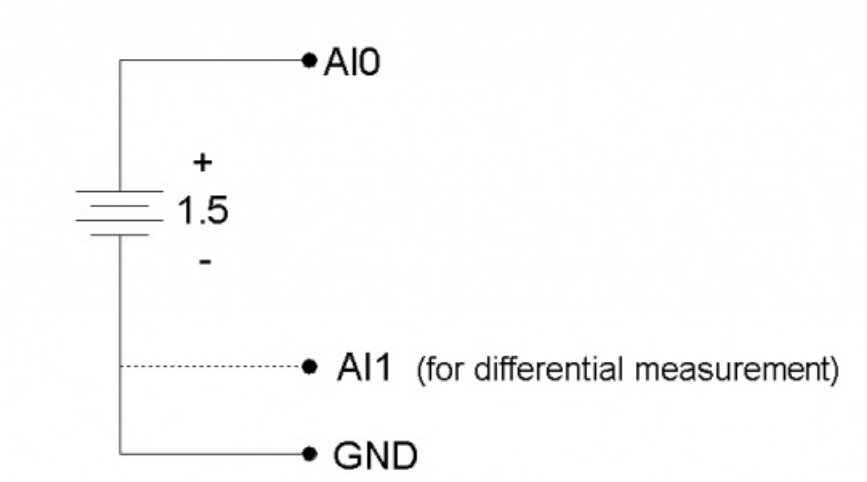


Figure 2-2. Single-ended measurement.

Figure 2-3 shows a typical differential connection measuring the voltage across a current shunt. A differential connection is required when neither leg of the shunt is at ground potential. Make sure that the voltage of both AI0 an AI1 with respect to ground is within ±10 volts. For instance, if the source (Vs) shown in Figure 2-3 is 120 VAC, the difference between AI0 and AI1 might be small, but the voltage from both AI0 and AI1 to ground will have a maximum value near 170 volts, and will seriously damage the LabJack.

Whether or not the ground (GND) connection is needed (Figure 2-3) will depend on the nature of Vs.

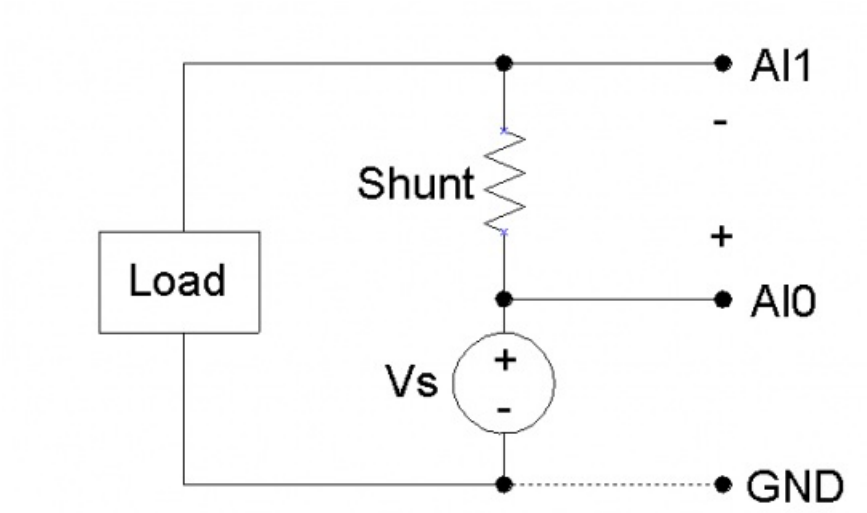


Figure 2-3. Differential measurement.

Figure 2-4 shows a single-ended connection used to measure the output voltage of a typical voltage-divider circuit. The voltage divider circuit is a simple way to convert a varying resistance (thermistor, photoresistor, potentiometer, etc.) to a varying voltage. With nothing connected to Va, the value of the unknown resistance, R2, can be calculated as:

```
R2 = Va*R1 / (Vs-Va)
```

where Vs is the supply voltage (+5V in Figure 2-4).

When Va is connected to AI0, as shown in Figure 2-4, the input bias current of the LabJack affects the voltage divider circuit, and if the resistance of R1 and R2 is too large, this effect must be accounted for or eliminated. This is true for any signal with too high of a source impedance.

All measuring devices have maximum analog input bias currents that very from picoamps to milliamps. The input bias current of the LabJack U12's analog inputs varies from +70 to -94 microamps (µA). This is similar to an input impedance of about 100 kΩ, but because the current is nonzero at 0 volts, it is better to model the analog input as a current sink obeying the following rule:

```
Iin = 8.181*Va - 11.67 µA
```

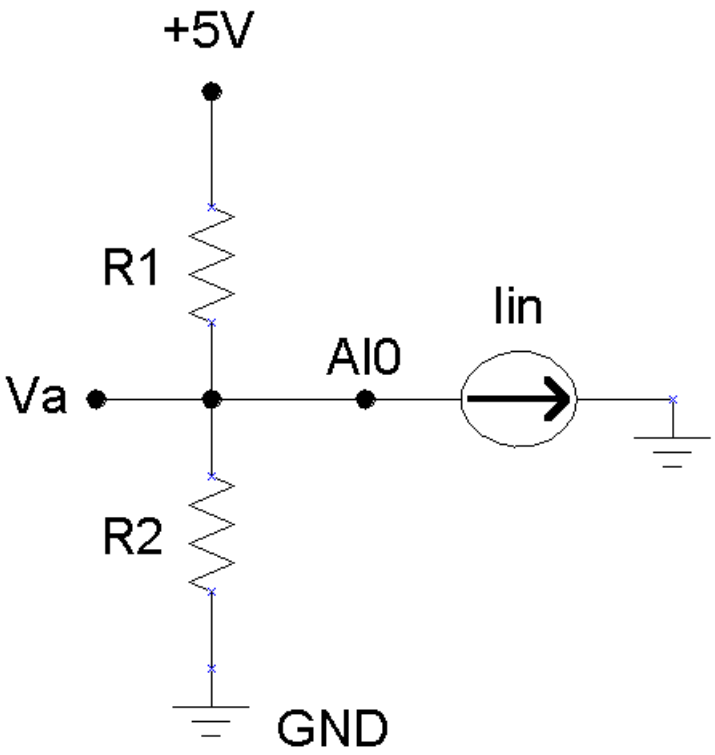


Figure 2-4. Single-ended measurement with voltage divider circuit.

Because the input bias current is known, as a function of input voltage, the simple voltage divider equation can be modified as follows to account for input bias current:

R2 = Va / [((Vs-Va)/R1) – (8.181µ * Va) + 11.67µ]

As an alternative to the equation above, Va can be buffered by a single-supply rail-to-rail operational amplifier, and the original simple voltage divider equation can be used. This solution works for any single-ended signal which stays between 0 and +5 volts. Some op-amp choices are:

- TLV2462
- LMC6482
- MAX4166

Software

Readings from the analog inputs are returned by the functions EAnalogIn, AISample, AIBurst, and AIStreamRead.

EAnalogIn is a simplified (E is for easy) function that returns a single reading from 1 analog input channel. Execution time is up to 20 ms.

AISample returns a single reading of 1-4 channels, and takes up to 20 ms to execute, providing a maximum date rate of about 50

Hz per channel.

AIBurst acquires multiple samples of 1-4 channels at a hardware-timed sample rate of 400-8192 Hz. The acquisition can be triggered based on a change of state on IO0 or IO1. This function also returns the states of the IO pins (which are read every 4 samples).

Internally, the actual number of samples collected and transferred by the LabJack during an AIBurst call is the smallest power of 2, from 64 to 4096, which is at least as big as numSamples. The execution time of this function, in milliseconds, can be estimated as:

```
Turbo (default)  =>  30+(1000*numSamplesActual/sampleRate)+(0.4*numSamplesActual)
Normal           =>  30+(1000*numSamplesActual/sampleRate)+(2.5*numSamplesActual)

numSamples = numScans * numChannels
sampleRate = scanRate * numChannels
```

AIStreamRead is called periodically during a stream acquisition started by AIStreamStart. Each call retrieves multiple samples of 1-4 channels from the LabJack stream buffer, along with the states of the IO pins (read every 4 samples). Hardware-timed sample rates of 200-1200 Hz are available. If any function besides AIStreamRead is called while a stream is in progress, the stream will be stopped.

2.2 - AO0 & AO1

The LabJack U12 has 2 screw terminals for analog output voltages. Each analog output can be set to a voltage between 0 and the supply voltage (+5 volts nominal) with 10-bits of resolution.

The output voltage is ratiometric with the +5 volt supply (+5V), which is generally accurate to ±5% (see Appendix A). If an output voltage of 5 volts is specified, the resulting output will be 100% of the supply voltage. Similarly, specifying 2.5 volts actually gives 50% of the supply voltage. The maximum output voltage is almost 100% of +5V at no-load, and decreases with load. See the specifications in Appendix A relating to maximum output voltage. Also note that loading either analog output will cause an IR drop through the source impedance of each.

If improved accuracy is needed, measure the +5 volt supply with an analog input channel, and the actual output voltage can be calculated. For instance, if an analog output of 2.5 volts is specified and a measurement of +5V returns 5.10 volts, the actual output voltage is 2.55 volts (at no-load). Alternatively (and preferably), the analog output can itself be measured with an analog input.

There is a 1st order low-pass filter on each analog output with a 3dB frequency around 22 Hz.

The analog outputs are initialized to 0.0 volts on power-up or reset.

The analog outputs can withstand a continuous short-circuit to ground, even when set at maximum output.

Voltage should never be applied to the analog outputs, as they are voltage sources themselves. In the event that a voltage is accidentally applied to either analog output, they do have protection against transient overvoltages such as ESD (electrostatic discharge) and continuous overvoltage of a couple volts. An applied voltage that exceeds the capability of this protection will most likely damage the resistor R63 (AO0) or R62 (AO1) on the LabJack U12 PCB. The symptom of such a failure would be reduced voltage from the analog outputs, particularly at load, and could be verified by measuring the resistance of R62/R63 (should be less than 50 ohms but a damaged resistor will measure higher). A simple repair for such damage is to remove the damaged resistor and simply make a short with a blob of solder.

Software

The analog outputs are set using the function EAnalogOut (easy function) or AOUpdate, which take up to 20 ms to execute, providing a maximum update rate of about 50 Hz per channel. AOUpdate also controls/reads all 20 digital I/O and the counter.

2.3 - IO0-IO3

Connections to 4 of the LabJack's 20 digital I/O are made at the screw terminals, and are referred to as IO0-IO3. Each pin can individually be set to input, output high, or output low. These 4 channels include a 1.5 kΩ series resistor that provides overvoltage/short-circuit protection. Each channel also has a 1 MΩ resistor connected to ground.

All digital I/O are set to input on power-up or reset.

One common use of a digital input is for measuring the state of a switch as shown in Figure 2-5. If the switch is open, IO0 reads FALSE. If the switch is closed, IO0 reads TRUE.

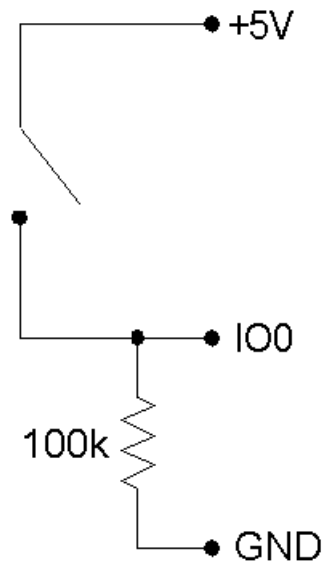


Figure 2-5. IO used to detect the state of a switch.

While providing overvoltage/short-circuit protection, the 1.5 kΩ series resistor on each IO pin also limits the output current capability. For instance, with an output current of 1 mA, the series resistor will drop 1.5 volts, resulting in an output voltage of about 3.5 volts.

Software

The easy functions EDigitalIn or EDigitalOut are used to read or set the state of one digital line, and both take up to 20 ms to execute.

The functions AOUpdate and DigitalIO are used to set the direction, set the state, and/or read the state, of each IO pin. Both of these functions take up to 20 ms to execute, providing a maximum update rate of about 50 Hz per pin.

The function AISample can set/read the state of each IO, but setting the state will have no effect unless the IO have been configured as outputs using another function. The function Counter reads the state of each IO.

The functions AIBurst and AIStreamRead, take a reading of the IO states and return it with the analog data. The states of the 4 IO are read simultaneously every 4 samples, providing a data rate of up to 2048 Hz per pin for burst mode, or 300 Hz per pin for stream mode. For 1 or 2 channel scans, duplicate data (4x or 2x) will be added to the read array such that the size is numScans.

2.4 - D0-D15

Connections to 16 of the LabJack's 20 digital I/O are made at the DB25 connector, and are referred to as D0-D15. These 16 lines have no overvoltage/short-circuit protection, and can sink or source up to 25 mA each (total sink or source current of 200 mA max for all 16). This allows the D pins to be used to directly control some relays. All digital I/O are CMOS output and TTL input except for D13-D15, which are Schmitt trigger input. Each D pin has a 1 MΩ resistor connected to ground.

All digital I/O are set to input on power-up or reset.

DB25 Pinouts:									
1:	D0	6:	D5	11:	+5V	16:	GND	21:	D11
2:	D1	7:	D6	12:	+5V	17:	GND	22:	D12
3:	D2	8:	D7	13:	+5V	18:	D8	23:	D13
4:	D3	9:	NC	14:	GND	19:	D9	24:	D14
5:	D4	10:	+5V	15:	GND	20:	D10	25:	D15

These digital I/O can detect the state of a switch using the same circuit shown in Figure 2-5.

Because the D pins have no overvoltage/short-circuit protection, the user must be careful to avoid damage. A series resistor can provide substantial protection for these pins (see the CB25 datasheet). The following are examples of things that could damage a D pin and/or the entire LabJack:

- Shorting a high output to ground (or any potential other than +5V).
- Shorting a low output to a nonzero voltage (such as +5V).
- Exceeding the voltage limits specified in Appendix A.

Software

The easy functions EDigitalIn or EDigitalOut are used to read or set the state of one digital line, and both take up to 20 ms to execute.

The functions AOUpdate and DigitalIO are used to set the direction, set the state, and/or read the state, of each D pin. In addition, DigitalIO also returns the current state of the direction and output registers. Both of these functions take up to 20 ms to execute, providing a maximum update rate of about 50 Hz per pin.

2.5 - CNT

The input connection to the 32-bit counter is made at screw-terminal CNT. The counter is incremented when it detects a falling edge followed by a rising edge. This means that if you reset the counter while your signal is low, you will not get the first count until it goes high-low-high. In situations where this first count is important, you should simply substract the initial count from the final count, rather than doing a reset.

Software

The functions ECount (easy function), AOUpdate, and Counter are used to reset or read the counter. If a reset is specified, the counter is read first. All of these functions take up to 20 ms to execute, providing a maximum update rate of about 50 Hz.

Counter readings can also be returned in stream mode (AIStreamRead) at up to 300 Hz.

2.6 - CAL - STB

These terminals are primarily used at the factory during testing and calibration.

CAL is a precision 2.5 volt reference, and can be used during normal operation, but care should be taken to observe the current limits specified in Appendix A. The CAL pin is protected from ESD and overvoltage, but severe overvoltage (steady-state or transient) can damage CAL, and result in the failure of all analog inputs.

STB has 2 purposes. One is to disable the watchdog as described in Section 4.35, and the other is to create a rectangular output signal for testing. For the latter, if the U12 is streaming the STB frequency matches the sampling frequency, while if not streaming the STB frequency is about 28 kHz. To get this output signal on STB use the enableSTB parameter in the Counter function.

2.7 - +5V

The LabJack has a nominal +5 volt internal power supply. Power can be drawn from this power supply by connecting to the +5V screw-terminals, or the +5V pins on the DB25 connector. The total amount of current that can be drawn from the +5V pins, analog outputs, and digital outputs, is 450 mA for most desktop computers and self-powered USB hubs. Some notebook computers and bus-powered hubs will limit this available current to about 50 mA.

The USB specification requires all hosts and hubs to have overcurrent protection. If the user puts too large a load on +5V (including a short circuit of +5V to GND) of the LabJack U12 (a USB device), the host or hub is responsible for limiting the current.

2.8 - GND

The GND connections available at the screw-terminals and DB25 connector provide a common ground for all LabJack functions. They are all the same.

Caution should be used whenever making connections with systems that have their own power source. It is normal to connect U12 ground to other grounds to create a common reference, but the risk is that the U12 ground will become the preferred ground for the other systems and they could try to send high currents into the U12. To prevent this it is often a good idea to put a 10-100 ohm resistor (or even a fuse) in series with GND on the U12 and any grounds from active systems.

2.9 - OEM Versions

The LabJack U12 is also available in 2 OEM (original equipment manufacturer) versions:

- **LJU12-PH:** This is a populated LabJack U12 PCB with pin-headers installed (on the component side of the PCB) instead of screw-terminals. Also, the LED is installed on the component side of the PCB, so nothing is installed on the solder side.
- **LJU12-NTH:** This is a populated LabJack U12 PCB with no through-hole components (DB25 connector, USB connector, LED, screw-terminals). This board is meant for OEMs who solder connections directly to the PCB, or wish to install only certain connectors.

Dimensional drawings are available here.

Normally, nothing ships with these OEM LabJacks except for the populated PCB. All software is available online on the U12 Support Page.

3 - Example Applications

The LabJack U12 CD installs 9 example applications:

- LJconfig: Lists all LabJacks connected to the USB and allows the local ID to be set on each.
- LJcounter: Reads the LabJack counter and provides the current frequency or count.
- LJfg (Function Generator): Outputs basic waveforms on AO0 (analog output zero).
- LJlogger: Saves data to disk, writes data to an HTML page on the Internet, and performs various actions (including email) on trigger events.
- LJscope: Simulates an oscilloscope by reading data from 2 AI channels in burst mode.
- LJstream: Uses stream mode to read, graph, and write to file, 4 AI channels.
- LJtest: Runs a sequence of tests on the LabJack itself.
- LJSHT: Retrieves and records data from 1 or 2 EI-1050 digital temperature/humidity probes.
- LJSHTmulti: Displays data from up to 20 EI-1050 digital temperature/humidity probes.

The LabVIEW source code for most of these applications is installed in the examples directory.

3.1 - LJconfig

Every LabJack has a local ID and serial number. The local ID is a value between 0 and 255 that can be changed by the user. The serial number is a value between 256 and 2,147,483,647 that is unique among all LabJacks and cannot be changed by the user. LJconfig is used to set the local ID of a particular LabJack. When using multiple U12s, each should be assigned a unique local ID.

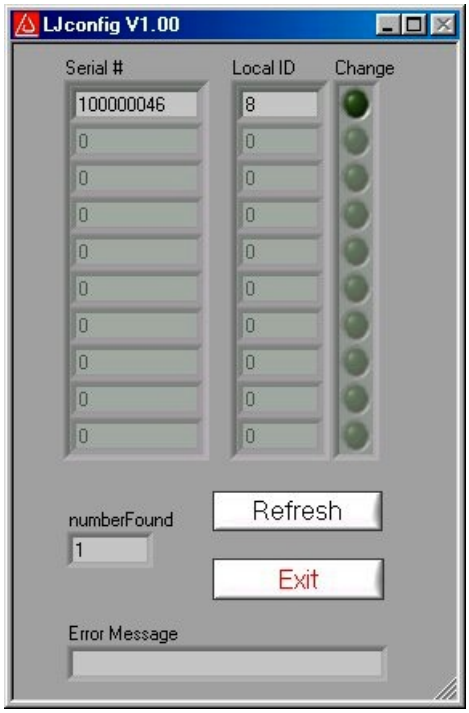


Figure 3-1. LJconfig

Figure 3-1 shows the window that opens when LJconfig is run. Each time the “Refresh” button is pushed, LJconfig will scan the USB for all LabJacks. To change the local ID of a particular LabJack, push the “Change” button next to that LabJack, and the

window shown in Figure 3-2 will appear.

Figure 3-2. LJconfig Change Local ID

Enter a new local ID between 0 and 255 and push the “Change” button. The new local ID will be written and the LabJack will be forced to re-enumerate.

3.2 - LJcounter

Reads the LabJack counter and provides the current frequency or count.

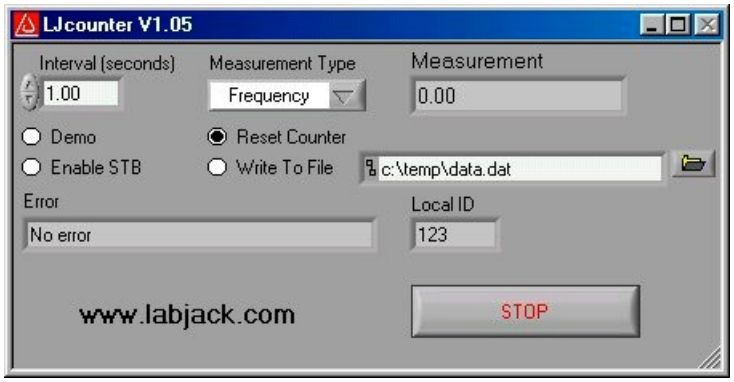


Figure 3-3. LJcounter

Figure 3-3 shows the LJcounter window:

- **Interval (seconds):** Specifies the interval, in seconds, between calls to the DLL function “Counter”.
- **Measurement Type:** If set to “Frequency”, this application divides the count by the interval to determine frequency in Hertz, and automatically resets the counter every read. If set to “Count”, the measurement is simply the current reading from the counter.
- **Measurement:** Displays frequency or count, depending on “Measurement Type”.

3.3 - LJfg

This application allows the LabJack U12 to be used as a simple function generator. The DLL function “AOUpdate” is called every 25 milliseconds providing an update rate of 40 Hz, and thus a maximum reasonable signal frequency of a few Hertz.

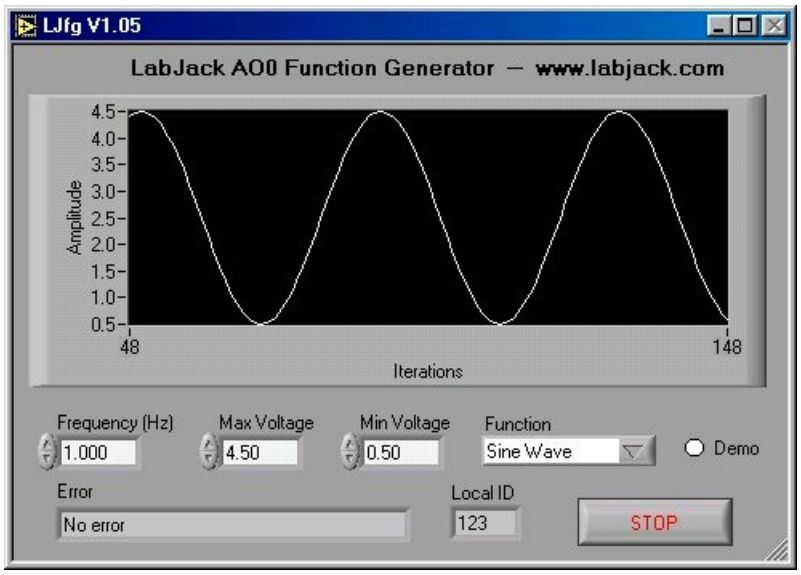


Figure 3-4. LJfg

3.4 - LJlogger

LJlogger sends and receives data in command/response mode by making 2 DLL calls to “AISample” and 1 DLL call to “AOUpdate”. It is capable of saving data to disk (2 Hz max) and performing various actions on trigger events.

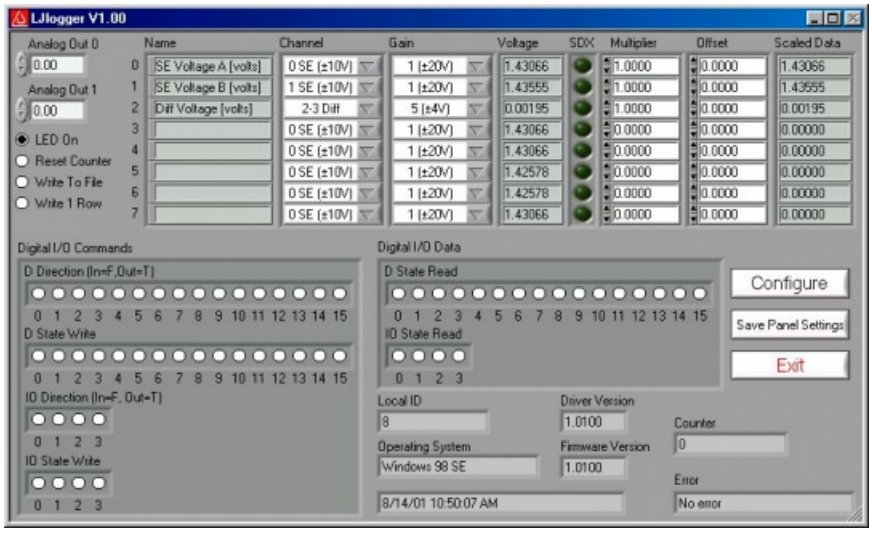


Figure 3-5. LJlogger

The main window for LJlogger is shown in Figure 3-5. The white colored items and the “SDX” buttons are controls to be edited/selected by the user. The grey colored items are indicators which display various information about the LabJack. Clicking the button labeled “Save Panel Settings” will save the current values of the controls as the default values.

If SDX is activated for a given analog input, the corresponding SDX DLL will be used to determine the scaled data. Users can make their own SDX DLLs (see the source code for more information), to provide more complex scaling or scaling that depends on other analog inputs.

Clicking the “Configure” button shown in Figure 3-5 brings up the window shown in Figure 3-6:

- **Working Directory:** This is the directory where data and configuration files will be written.
- **Data File Name:** Determines the name of the data file to which data will be written. New data is appended to the end of this file.
- **Data File Write Interval:** Determines the interval at which a new row of data will be written to the data file. Minimum of 0.5 seconds.
- **HTML Write Interval:** Determines the interval at which the HTML file is rewritten.

Figure 3-6. LJlogger Configuration

Clicking on the “Internet Setup” button in Figure 3-6 brings up the Internet configuration window shown in Figure 3-7. Basic customization of the HTML file can be done by clicking on “Advanced HTML Configuration” which brings up Figure 3-8.

Clicking on the “Trigger Setup” button in Figure 3-6 brings up the window shown in Figure 3-9.

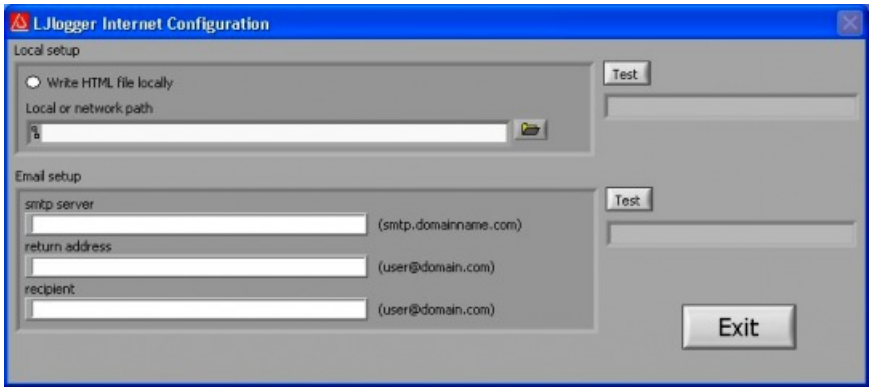


Figure 3-7. LJlogger Internet Configuration

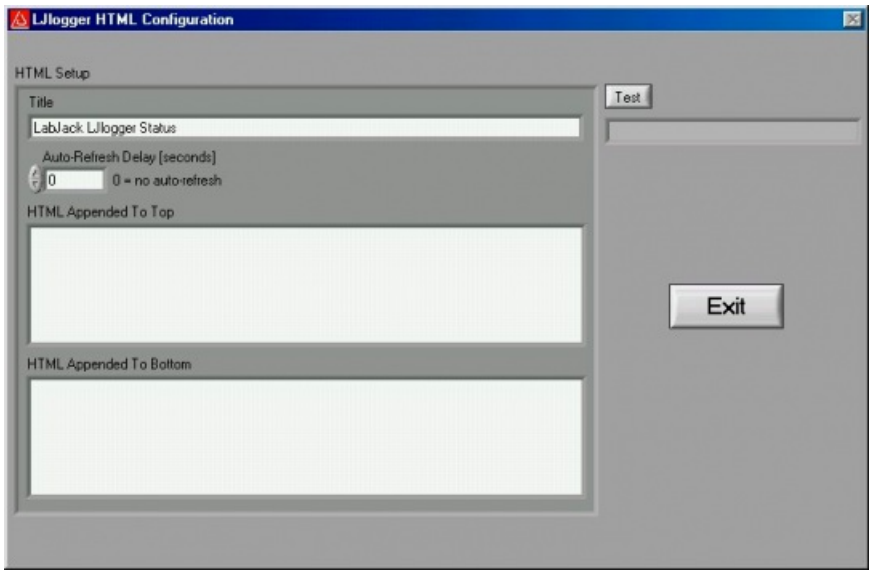


Figure 3-8. LJlogger HTML Configuration

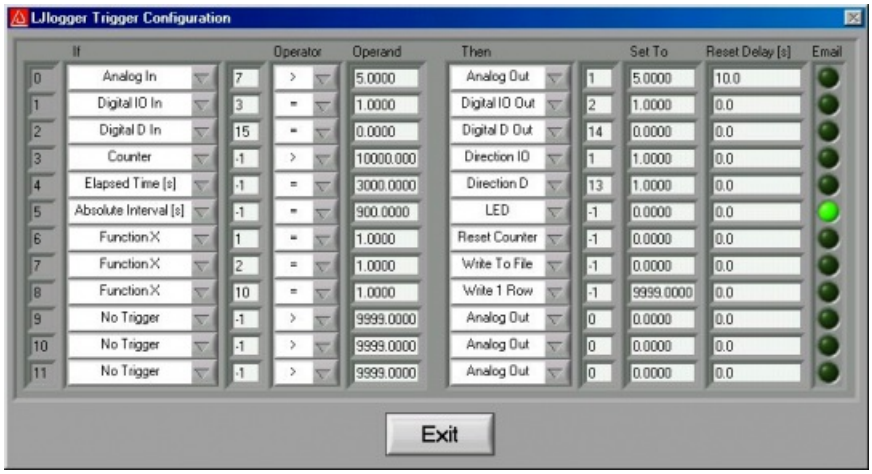


Figure 3-9. LJlogger Trigger Configuration

Figure 3-9 shows 9 example triggers:

- **Trigger #0:** If the scaled data from analog input row 7 (Figure 3-5) is greater than 5, then set AO1 to 5 volts. Once triggered, there is a 10 second delay before it can be triggered again.
- **Trigger #1:** If IO3 is high, set IO2 high. Reset delay is zero so this trigger can occur every iteration (every 0.1 seconds) if IO3 is high.
- **Trigger #2:** If D15 is low, set D14 low.
- **Trigger #3:** If the count is greater than 10,000, set IO1 to an output.
- **Trigger #4:** If it has been 3000 seconds since LJlogger started, set D13 to an output.
- **Trigger #5:** When the PC's clock is at 15 minute intervals, the status LED will be turned off and an email will be sent.
- **Trigger #6:** Calls FunctionX from function1.dll. If the function returns True, reset the counter. Users can make their own FunctionX DLLs. See the source code for more information.
- **Trigger #7:** Calls FunctionX from function2.dll. If the function returns True, stop writing data to file.
- **Trigger #8:** Calls FunctionX from function10.dll. If the function returns True, write 1 row to the data file.

3.5 - LJscope

LJscope simulates an oscilloscope by reading data from 2 analog input channels in burst mode.

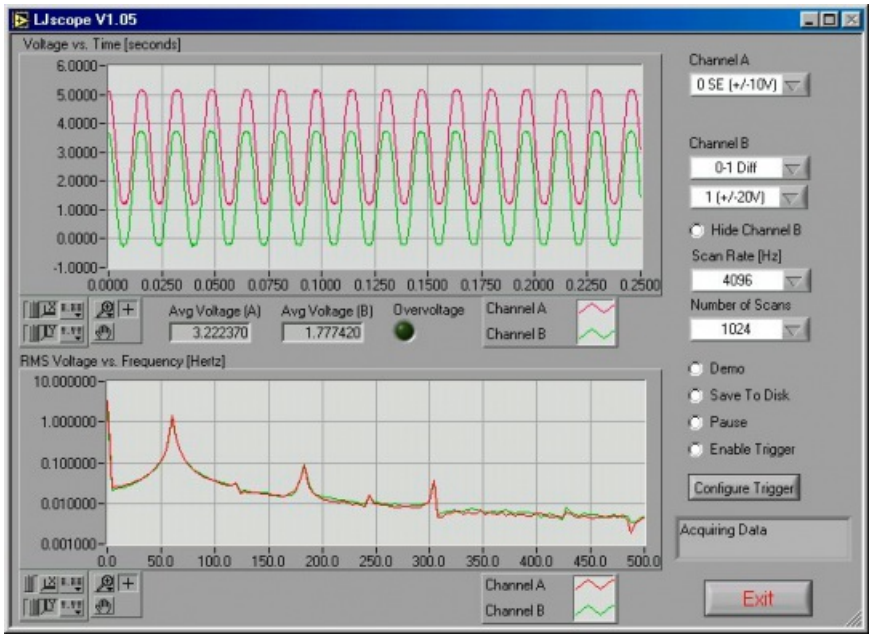
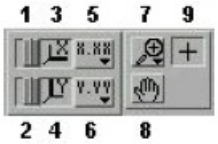


Figure 3-10. LJscope

There are two graphs on the LJscope main window (Figure 3-10), which show voltage versus time and voltage versus frequency. Both graphs have a palette to control various features such as autoscaling and zooming:



1. When you press this button it locks button 3 on (autoscale) position.
2. When you press this button it locks button 4 on (autoscale) position.
3. Pressing this button autoscales the x-axis.
4. Pressing this button autoscales the y-axis.
5. Miscellaneous x-axis formatting options.
6. Miscellaneous y-axis formatting options.
7. Zooming tool. Press this button to see different options for zooming. When collecting data, zooming will not work well unless autoscaling is off.
8. Panning tool. Allows you to drag and scroll around the graph.
9. Not applicable.

Other LJscope controls include:

- **Channel A/B:** Select the two AI channels that will be acquired. If a differential channel, is selected, the gain selection control will appear.
- **Hide Channel B:** When selected channel B will not be shown on the graph.
- **Scan Rate [Hz]:** (256 to 4096) Determines the scans/second for both channels.
- **Number of Scans:** (32 to 2048) Determines the number of scans that will be collected, and thus the total acquisition period. For example, if 1024 scans are collected at 4096 Hz, a quarter second of data will be collected (as shown in Figure 3-10).
- **Demo:** Calls “AIBurst” in demo mode so timing and data is simulated.
- **Save To Disk:** If selected a prompt will appear for a filename, and the current burst of data is saved to a tab-delimited file (time, channel A, channel B).
- **Pause:** Pauses data acquisition.
- **Enable Trigger:** Enables the IO trigger.
- **Configure Trigger:** Brings up the window shown in Figure 3-11. Choose the IO line to trigger on and whether to trigger when it is high or low. Also set the timeout period so the application will continue (with an error) if the trigger is not detected.

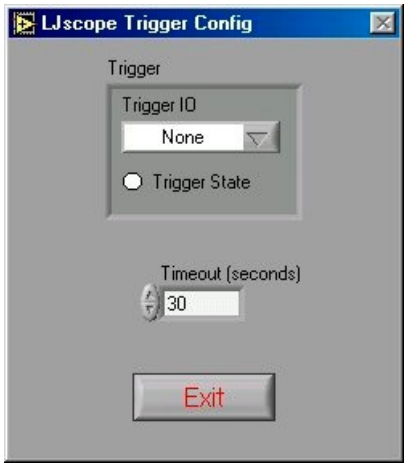


Figure 3-11. Configure Trigger

3.6 - LJstream

Uses stream mode to read, graph, and write to file, 4 AI channels. For more information, read about the stream functions (AIStreamStart, AIStreamRead, and AIStreamClear).

- **Enable Stream:** Starts and stops the stream acquisition.
- **Scan Rate:** Determines the scans/second (50 to 300).
- **Number of Scans:** Determines the number of scans that will be collected each iteration, and thus determines how fast this application iterates.
- **Demo:** Calls the “AIStream” functions in demo mode so timing and data is simulated.
- **Read Counter:** Collects 1 analog input and the counter if selected.
- **Configure Channels:** Click this button to bring up the channel configuration window shown in Figure 3-13.
- **Save Current Settings:** Saves all the current settings, including channel configuration.

- **Graph History:** Determines how much past history appears on the graph.

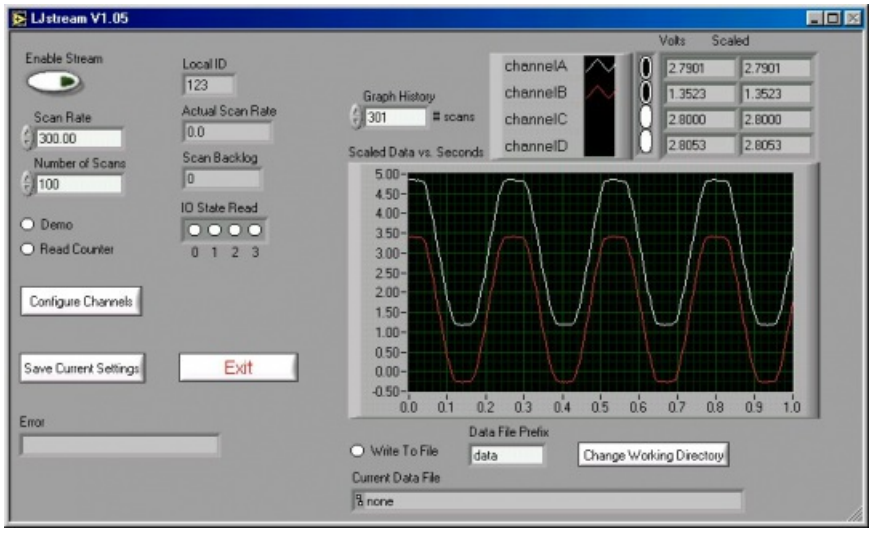


Figure 3-12. LJstream

Figure 3-13 shows the LJstream channel configuration window. Here you can select analog inputs and gains and enter scaling equations. Use “Test Data” to see the effect of the scaling equations (“v” column is the measured voltage and the “y” column is the output of the scaling equations). “Manual/Sampled” determines where the “Test Data” in the “v” column originates.

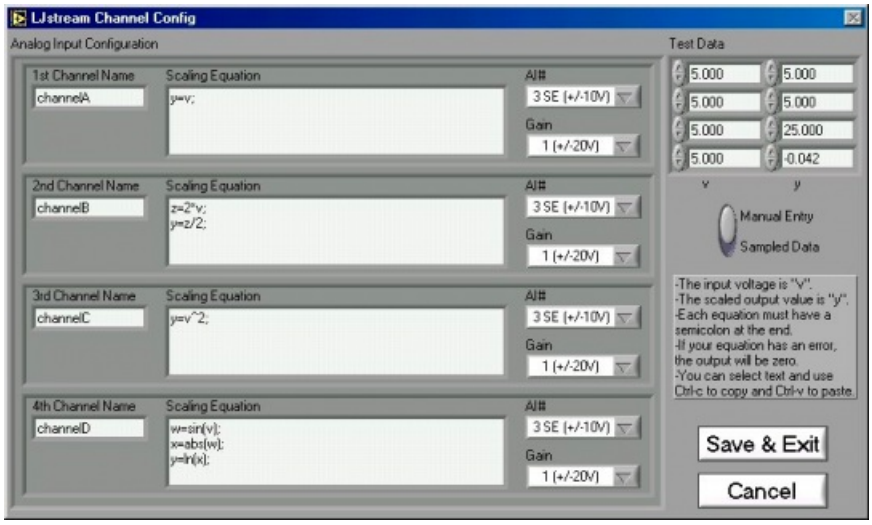


Figure 3-13. LJstream Channel Configuration

3.7 - LJtest

LJtest runs a sequence of tests on the LabJack itself. Users will generally leave “Test Fixture Installed” unselected and execute the tests with nothing connected to the LabJack (except the USB of course).

Note that the U12 is not the “Test Fixture”. Rather, the U12 is the device under test (DUT) and might connect to a test fixture.

LJtest is installed as part of the normal U12 software package. It can typically be run by doing Start => Programs => LabJack U12 Legacy => Legacy U12 Samples => LJtest.

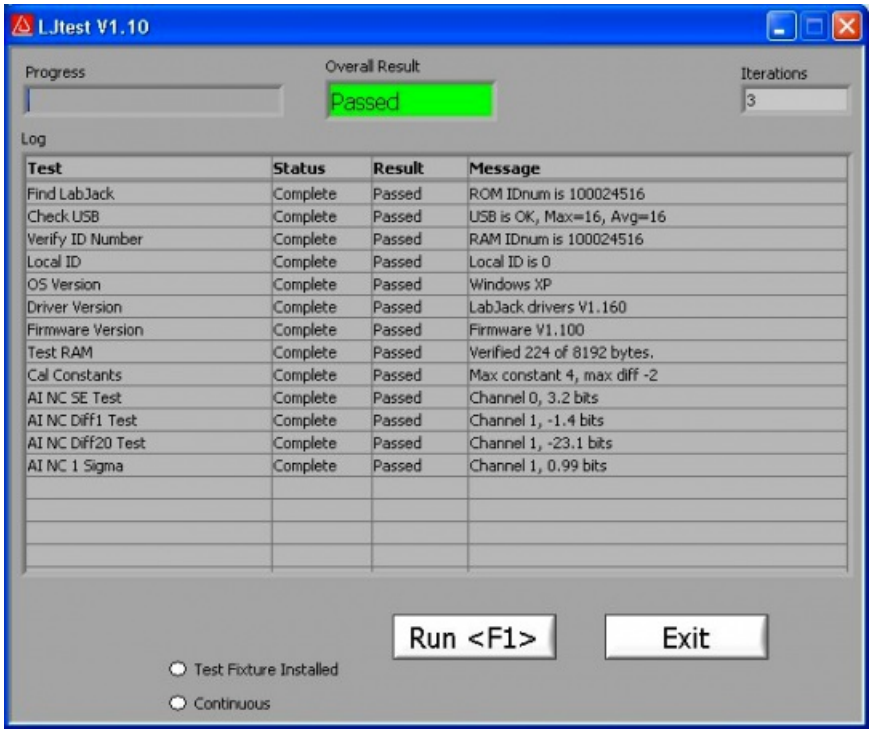


Figure 3-14. LJtest

If all tests fail except “OS Version” and “Driver Version”, it generally indicates LJtest does not detect a LabJack U12 at all. Check for proper blinking of status LED upon power-up, and if using Windows 98 SE check out the related file (Win98sehid.zip) from the downloads page at labjack.com.

If “Find LabJack” is the only failure, it is often because more than 1 LabJack U12 is connected.

“Check USB” performs some basic tests to detect any obvious problems with the Universal Serial Bus. Proper LabJack U12 communication is required for this test.

“Local ID” will show a yellow warning if the Local ID has been changed from the factory default of 0.

Failures from “Test RAM” or any of the “AI …” tests could indicate damage to the unit. Make sure there are no connections to the LabJack U12 (except for the USB cable), and contact LabJack support if the failures continue. Yellow warnings on any of the “AI …” tests (make sure nothing is connected to the AI channels) could indicate that a self-calibration needs to be performed (see below).

A yellow warning from the “Cal Constants” test is usually because the constants have all been set to zero. Most often this is due to selecting “Test Fixture Installed” and running LJtest without the proper connections. Follow the below procedure to correct this issue.

To write new calibration data, a self-calibration should be performed using LJtest and 12 small (1.5” will work) jumper wires:

1. Make the following initial connections:
   - AI0 <=> AI2 <=> AI4 <=> AI6 <=> +5V
   - AI1 <=> AI3 <=> AI5 <=> AI7 <=> +5V
   - IO0 <=> IO1
   - IO2 <=> AO0
   - IO3 <=> AO1
   - CNT <=> STB
2. Start LJtest and select “Test Fixture Installed” and “Prompt During Cal”, and then click on the “Run” button.
3. LJtest will step through various tests and then prompt to connect GND to all 8 AI channels (AI6<=>GND and AI7<=>GND), then to connect CAL to the even channels (AI6<=>CAL), then to connect CAL to all 8 AI channels (AI7<=>CAL), and finally to connect GND to the even channels (AI6<=>GND).
4. When finished, remove all wires and unplug the USB cable. Reconnect the USB cable and the new calibration constants will be loaded at power-up. Run LJtest again with “Test Fixture Installed” unselected, to make sure the unit passes the normal self-test.

3.8 - LJSHT

Reads and records data from one or two EI-1050 digital temperature/humidity probes.

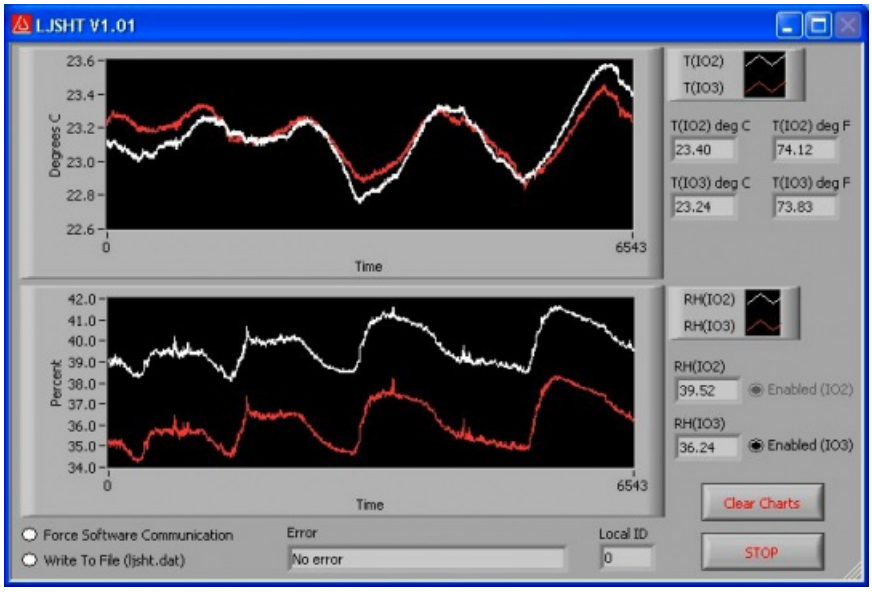


Figure 3-15. LJSHT

Figure 3-15 shows the LJSHT window:

- **Enabled (IO2/IO3)**: At least one EI-1050 must be connected. IO2 will be controlled as the enable line for this probe. If two probes are connected, then enable control of IO3.
- **Force Software Communication:** Forces software based SHT1X communication, even if the LabJack U12 firmware is V1.10 or higher.
- **Write To File:** Appends data to a tab-delimited ASCII file called ljsht.dat in the current directory. Data is written as seconds since 1904, followed by tempC/tempF/RH for each probe.

3.9 - LJSHTmulti

Displays readings from up to 20 EI-1050 digital temperature/humidity probes connected to a single LabJack U12.

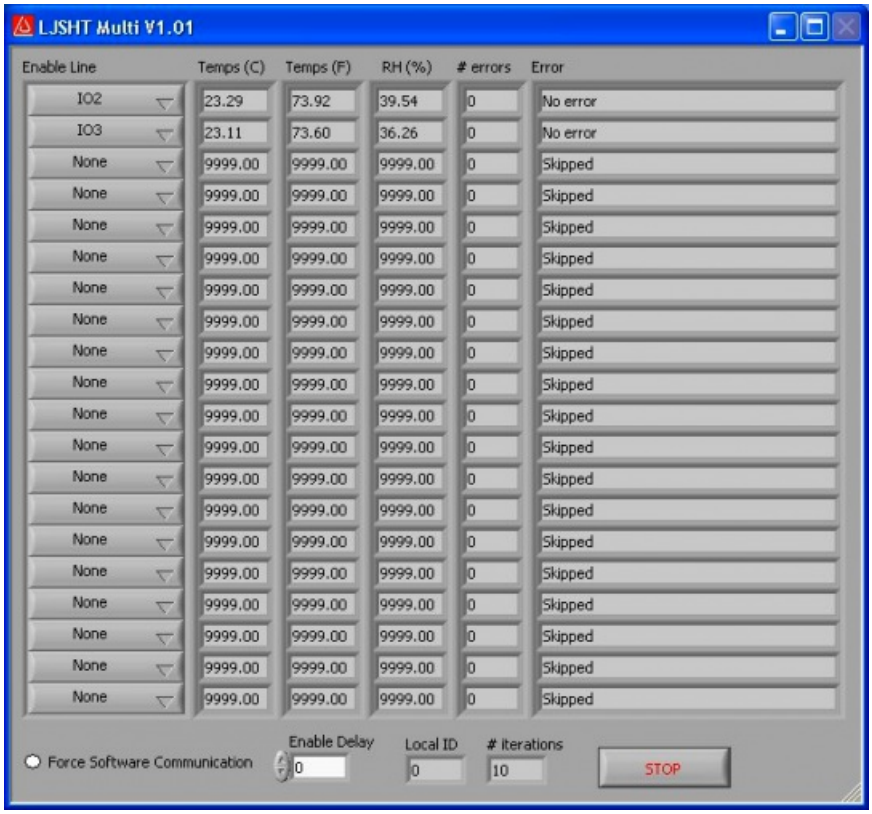


Figure 3-16. LJSHTmulti

Figure 3-16 shows the LJSHTmulti window:

- **Enable Line:** Choose the LabJack U12 output used to control the enable line on each EI-1050 probe.
- **Force Software Communication:** Forces software based SHT1X communication, even if the LabJack U12 firmware is V1.10 or higher.
- **Enable Delay:** Adds a delay between each reading for testing purposes.

4 - Programming Reference

The LabJack U12 CD installs high-level drivers (ljackuw.dll), an ActiveX interface to the high-level drivers (ljackuwx.ocx), and LabVIEW6 (or higher) VIs which call all the DLL functions. The DLL and OCX are installed in the Windows System directory. If the installation program can determine the LabVIEW directory, it copies the LabVIEW VIs into that directory (\vi.lib\addons\) so they show up on the function palette. Otherwise, the LabVIEW drivers are copied into the LabJack installation directory (c:\Program Files\LabJack)\drivers\labview, and can manually be transferred to the LabVIEW directory. LabVIEW5 VIs are also installed and can be found in the \LabJack\examples directory.

There are 38 functions exported by the LabJack DLL, and matching functions in the OCX and LabVIEW VIs. There are two additional support functions in the OCX, provided due to the limitations of ActiveX. All functions are command/response except for AIBurst and AIStreamStart/Read/Clear.

There are 2 parameters that are used by most functions:

- **errorcode** – A LabJack specific numeric error code. 0 means no error and 2 means no LabJacks were found. Use the function “GetErrorString” to get a description of the error, or see the list in Section 4.24 of this document.
- **idnum** – Functions with this input take either a local ID, serial number, or -1. A local ID or serial number will specify a specific LabJack, while –1 means the first found LabJack. Every LabJack has a local ID and serial number. The local ID is a value between 0 and 255 that can be changed by the user. The serial number is a value between 256 and 2,147,483,647 that is unique among all LabJacks and cannot be changed by the user. When using multiple U12s, each should be assigned a unique local ID.

To maintain compatibility with as many languages as possible, the every attempt has been made to keep the parameter types very basic. The declarations that follow, are written in C. If there are any differences in the ActiveX version of a function, they are described.

When a parameter name is preceded by a “*”, it means that it is a pointer. In most cases, this means that the parameter is an input and/or output, whereas a non-pointer parameter is input only. In some cases a pointer points to an array of values rather than a single value, and this will be noted in the parameter description.

Some of the digital I/O parameters contain the information for each bit of I/O in one value, where each bit of I/O corresponds to the same bit in the parameter (i.e. the direction of D0 is set in bit 0 of parameter trisD). For instance, in the function DigitalIO, the parameter *trisD is a pointer to a single memory location that sets/reads the direction of each of the 16 D lines:

- if *trisD points to 0, all D lines are input,
- if *trisD points to 1 (2^0), D0 is output, D1-D15 are input,
- if *trisD points to 5 ($2^0 + 2^2$), D0 and D2 are output, all other D lines are input,
- if *trisD points to 65535 ($2^0 + \ldots + 2^{15}$), D0-D15 are output.

The range of the value pointed to by *trisD is 0 to 65535. When calling DigitalIO, if updateDigital is >1, the D lines will be set to input or output based on the value pointed to by *trisD. When DigitalIO returns, the value pointed to by *trisD will have been set to reflect the status of the direction register in the LabJack U12.

4.1 - EAnalogIn

Easy function. This is a simplified version of AISample. Reads the voltage from 1 analog input. Calling this function turns/leaves the status LED on.

Execution time for this function is 20 milliseconds or less (typically 16 milliseconds in Windows).

Declaration:

```
long EAnalogIn (    long *idnum,
                    long demo,
                    long channel,
                    long gain,
```

```
                    long *overVoltage,
                    float *voltage )
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.
Inputs:

- ***idnum** – Local ID, serial number, or -1 for first found.
- **demo** – Send 0 for normal operation, >0 for demo mode. Demo mode allows this function to be called without a LabJack.
- **channel** – Channel command is 0-7 for single-ended, or 8-11 for differential.
- **gain** – Gain command is 0=1, 1=2, …, 7=20. This amplification is only available for differential channels.

Outputs:

- ***idnum** – Returns the local ID or -1 if no LabJack is found.
- **overVoltage** – If >0, an overvoltage has been detected on one of the selected analog inputs.
- **voltage** – Returns the voltage reading.

LabJackPython Example

```
>>> import u12
>>> d = u12.U12()
open called
Writing: [0x0, 0x0, 0x0, 0x0, 0x0, 0x57, 0x0, 0x0]
Received: [0x57, 0x0, 0x0, 0x0, 0xff, 0xff, 0x0, 0x0]
>>> d.eAnalogIn(0)
Writing: [0x8, 0x9, 0xa, 0xb, 0x1, 0xc0, 0x0, 0x0]
Received: [0x85, 0x0, 0x99, 0x2b, 0x2b, 0x99, 0x2c, 0x26]
{'overVoltage': 0, 'idnum': 12, 'voltage': 1.4599609375}
```

4.2 - EAnalogOut

Easy function. This is a simplified version of AOUpdate. Sets the voltage of both analog outputs.

Execution time for this function is 20 milliseconds or less (typically 16 milliseconds in Windows).

If either passed voltage is less than zero, the DLL uses the last set voltage. This provides a way to update 1 output without changing the other. Note that when the DLL is first loaded, it does not know if the analog outputs have been set, and assumes they are both the default of 0.0 volts. Similarly, there are situations where the LabJack could reset without the knowledge of the DLL, and thus the DLL could think the analog outputs are set to a non-zero voltage when in fact they have been reinitialized to 0.0 volts.

Declaration:

```
long EAnalogOut (    long *idnum,
                     long demo,
                     float analogOut0,
                     float analogOut1 )
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.
Inputs:

- ***idnum** – Local ID, serial number, or -1 for first found.
- **demo** – Send 0 for normal operation, >0 for demo mode. Demo mode allows this function to be called without a LabJack.
- **analogOut0** – Voltage from 0.0 to 5.0 for AO0.
- **analogOut1** – Voltage from 0.0 to 5.0 for AO1.

Outputs:

- ***idnum** – Returns the local ID or –1 if no LabJack is found.

LabJackPython Example

```
>>> import u12
>>> d = u12.U12()
open called
Writing: [0x0, 0x0, 0x0, 0x0, 0x0, 0x57, 0x0, 0x0]
```

```
Received: [0x57, 0x0, 0x0, 0x0, 0xff, 0xff, 0x0, 0x0]
>>> d.eAnalogOut(2, 2)
Writing: [0x0, 0x0, 0x0, 0x0, 0x0, 0x5, 0x66, 0x66]
Received: [0x0, 0x0, 0x0, 0x50, 0xbb, 0x10, 0x0, 0xef]
{'idnum': 12}
```

4.3 - ECount

Easy function. This is a simplified version of Counter. Reads & resets the counter (CNT). Calling this function disables STB (which is the default anyway).

Execution time for this function is 20 milliseconds or less (typically 16 milliseconds in Windows).

Declaration:

```
long ECount (    long *idnum,
                 long demo,
                 long resetCounter,
                 double *count,
                 double *ms )
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.
Inputs:

- ***idnum*** – Local ID, serial number, or -1 for first found.
- **demo** – Send 0 for normal operation, >0 for demo mode. Demo mode allows this function to be called without a LabJack.
- **resetCounter** – If >0, the counter is reset to zero after being read.

Outputs:

- ***idnum*** – Returns the local ID or –1 if no LabJack is found.
- ***count*** – Current count, before reset.
- ***ms*** – Value of Window's millisecond timer at the time of the counter read (within a few ms). Note that the millisecond timer rolls over about every 50 days. In general, the millisecond timer starts counting from zero whenever the computer reboots.

LabJackPython Example

```
>>> import u12
>>> d = u12.U12()
open called
Writing: [0x0, 0x0, 0x0, 0x0, 0x0, 0x57, 0x0, 0x0]
Received: [0x57, 0x0, 0x0, 0x0, 0xff, 0xff, 0x0, 0x0]
>>> d.eCount()
Writing: [0x0, 0x0, 0x0, 0x0, 0x0, 0x52, 0x0, 0x0]
Received: [0x52, 0x0, 0x0, 0x50, 0xbb, 0x10, 0x0, 0xef]
{'count': 3138388207, 'idnum': 12, 'ms': 1273175001372.4438}
```

4.4 - EDigitalIn

Easy function. This is a simplified version of DigitalIO that reads the state of one digital input. Also configures the requested pin to input and leaves it that way.

Execution time for this function is 20 milliseconds or less (typically 16 milliseconds in Windows).

Note that this is a simplified version of the lower level function DigitalIO, which operates on all 20 digital lines. The DLL (ljackuw) attempts to keep track of the current direction and output state of all lines, so that this easy function can operate on a single line without changing the others. When the DLL is first loaded, though, it does not know the direction and state of the lines and assumes all directions are input and output states are low.

Declaration:

```
long EDigitalIn (    long *idnum,
                     long demo,
                     long channel,
                     long readD,
```

```
                    long *state )
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.
Inputs:

- ***idnum** – Local ID, serial number, or -1 for first found.
- **demo** – Send 0 for normal operation, >0 for demo mode. Demo mode allows this function to be called without a LabJack.
- **channel** – Line to read. 0-3 for IO or 0-15 for D.
- **readD** – If >0, a D line is read as opposed to an IO line.

Outputs:

- ***idnum** – Returns the local ID or –1 if no LabJack is found.
- ***state** – The selected line is TRUE/Set if >0. FALSE/Clear if 0.

LabJackPython Example

```
>>> import u12
>>> d = u12.U12()
open called
Writing: [0x0, 0x0, 0x0, 0x0, 0x0, 0x57, 0x0, 0x0]
Received: [0x57, 0x0, 0x0, 0x0, 0xff, 0xff, 0x0, 0x0]
>>> d.eDigitalIn(0)
Writing: [0x0, 0x0, 0x0, 0x0, 0x0, 0x57, 0x0, 0x0]
Received: [0x57, 0x0, 0x0, 0x50, 0xff, 0xff, 0x0, 0x0]
Writing: [0x0, 0x0, 0x0, 0x0, 0xff, 0x57, 0x1, 0x0]
Received: [0x57, 0x0, 0x0, 0x50, 0x0, 0x0, 0x0, 0x0]
{'state': 1, 'idnum': 12}
```

4.5 - EDigitalOut

Easy function. This is a simplified version of DigitalIO that sets/clears the state of one digital output. Also configures the requested pin to output and leaves it that way.

Execution time for this function is 20 milliseconds or less (typically 16 milliseconds in Windows).

Note that this is a simplified version of the lower level function DigitalIO, which operates on all 20 digital lines. The DLL (ljackuw) attempts to keep track of the current direction and output state of all lines, so that this easy function can operate on a single line without changing the others. When the DLL is first loaded, though, it does not know the direction and state of the lines and assumes all directions are input and output states are low.

Declaration:

```
long EDigitalOut (    long *idnum,
                      long demo,
                      long channel,
                      long writeD,
                      long state )
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.
Inputs:

- ***idnum** – Local ID, serial number, or -1 for first found.
- **demo** – Send 0 for normal operation, >0 for demo mode. Demo mode allows this function to be called without a LabJack.
- **channel** – Line to read. 0-3 for IO or 0-15 for D.
- **writeD** – If >0, a D line is written as opposed to an IO line.
- **state** – If >0, the line is set, otherwise the line is cleared.

Outputs:

- ***idnum** – Returns the local ID or –1 if no LabJack is found.

LabJackPython Example

```
>>> import u12
```

```
>>> d = u12.U12()
open called
Writing: [0x0, 0x0, 0x0, 0x0, 0x0, 0x57, 0x0, 0x0]
Received: [0x57, 0x0, 0x0, 0x0, 0xff, 0xff, 0x0, 0x0]
>>> d.eDigitalOut(0, 1)
Writing: [0x0, 0x0, 0x0, 0x0, 0x0, 0x57, 0x0, 0x0]
Received: [0x57, 0x0, 0x0, 0x50, 0x0, 0x0, 0x0, 0x0]
Writing: [0x0, 0x0, 0x0, 0x0, 0x1, 0x57, 0x1, 0x0]
Received: [0x57, 0x0, 0x0, 0x10, 0x0, 0x0, 0x0, 0x0]
{'idnum': 12}
```

4.6 - AISample

Reads the voltages from 1,2, or 4 analog inputs. Also controls/reads the 4 IO ports.

Execution time for this function is 20 milliseconds or less (typically 16 milliseconds in Windows).

Declaration:

```
long AISample (    long *idnum,
                   long demo,
                   long *stateIO,
                   long updateIO,
                   long ledOn,
                   long numChannels,
                   long *channels,
                   long *gains,
                   long disableCal,
                   long *overVoltage,
                   float *voltages )
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.
Inputs:

- ***idnum** – Local ID, serial number, or -1 for first found.
- **demo** – Send 0 for normal operation, >0 for demo mode. Demo mode allows this function to be called without a LabJack.
- ***stateIO** – Output states for IO0-IO3. Has no effect if IO are configured as inputs, so a different function must be used to configure as output.
- **updateIO** – If >0, state values will be written. Otherwise, just a read is performed.
- **ledOn** – If >0, the LabJack LED is turned on.
- **numChannels** – Number of analog input channels to read (1,2, or 4).
- ***channels** – Pointer to an array of channel commands with at least numChannels elements. Each channel command is 0-7 for single-ended, or 8-11 for differential.
- ***gains** – Pointer to an array of gain commands with at least numChannels elements. Gain commands are 0=1, 1=2, …, 7=20. This amplification is only available for differential channels.
- **disableCal** – If >0, voltages returned will be raw readings that are not corrected using calibration constants.
- ***voltages** – Pointer to an array where voltage readings are returned. Send a 4-element array of zeros.

Outputs:

- ***idnum** – Returns the local ID or –1 if no LabJack is found.
- ***stateIO** – Returns input states of IO0-IO3.
- ***overVoltage** – If >0, an overvoltage has been detected on one of the selected analog inputs.
- ***voltages** – Pointer to an array where numChannels voltage readings are returned.

ActiveX Function Differences:
The “channels” and “gains” arrays are replaced with “channelsPacked” and “gainsPacked”. The OCX has a function “FourPack” which will convert 4 elements to a packed value. The packed value is determined as: element[0] + (element[1] * 2^8) + (element[2] * 2^{16}) + (element[3] * 2^{24}).

The “voltages” array is replaced with 4 individual parameters.

Declaration (ActiveX):

```
long AISampleX (    long FAR* idnum,
                    long demo,
                    long FAR* stateIO,
                    long updateIO,
                    long ledOn,
                    long numChannels,
```

```
                    long channelsPacked,
                    long gainsPacked,
                    long disableCal,
                    long FAR* overVoltage,
                    float FAR* voltageA,
                    float FAR* voltageB,
                    float FAR* voltageC,
                    float FAR* voltageD )
```

4.7 - AIBurst

Reads a specified number of scans (up to 4096) at a specified scan rate (up to 8192 Hz) from 1,2, or 4 analog inputs. First, data is acquired and stored in the LabJack's 4096 sample RAM buffer. Then, the data is transferred to the PC.

If the LED is enabled (ledOn>0), it will blink at about 4 Hz while waiting for a trigger, turn off during acquisition, blink rapidly while transferring data to the PC, and turn on when done.

The execution time of this function, in milliseconds, depends on transfermode and can be estimated with the below formulas. The actual number of samples collected and transferred by the LabJack is the smallest power of 2 from 64 to 4096 which is at least as big as numScans*numChannels. This is represented below as numSamplesActual.

Normal => 30+(1000*numSamplesActual/sampleRate)+(2.5*numSamplesActual)

Turbo => 30+(1000*numSamplesActual/sampleRate)+(0.4*numSamplesActual)

Declaration:

```
long AIBurst (    long *idnum,
                  long demo,
                  long stateIOin,
                  long updateIO,
                  long ledOn,
                  long numChannels,
                  long *channels,
                  long *gains,
                  float *scanRate,
                  long disableCal,
                  long triggerIO,
                  long triggerState,
                  long numScans,
                  long timeout,
                  float (*voltages)[4],
                  long *stateIOout,
                  long *overVoltage,
                  long transferMode )
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.
Inputs:

- ***idnum** – Local ID, serial number, or -1 for first found.
- **demo** – Send 0 for normal operation, >0 for demo mode. Demo mode allows this function to be called without a LabJack.
- **stateIOin** – Output states for IO0-IO3. Has no effect if IO are configured as inputs, so a different function must be used to configure as output.
- **updateIO** – If >0, state values will be written. Otherwise, just a read is performed.
- **ledOn** – If >0, the LabJack LED is turned on.
- **numChannels** – Number of analog input channels to read (1,2, or 4).
- ***channels** – Pointer to an array of channel commands with at least numChannels elements. Each channel command is 0-7 for single-ended, or 8-11 for differential.
- ***gains** – Pointer to an array of gain commands with at least numChannels elements. Gain commands are 0=1, 1=2, …, 7=20. This amplification is only available for differential channels.
- ***scanRate** – Scans acquired per second. A scan is a reading from every channel (1,2, or 4). The sample rate (scanRate * numChannels) must be between 400 and 8192.
- **disableCal** – If >0, voltages returned will be raw readings that are not corrected using calibration constants.
- **triggerIO** – The IO port to trigger on (0=none, 1=IO0, or 2=IO1).
- **triggerState** – If >0, the acquisition will be triggered when the selected IO port reads high.
- **numScans** – Number of scans which will be returned. Minimum is 1. Maximum numSamples is 4096, where numSamples is numScans * numChannels.
- **timeout** – This function will return immediately with a timeout error if it does not receive a scan within this number of seconds. Timeouts of 3 seconds or less are generally recommended to keep the U12 in turbo transfer mode.
- ***voltages** – Pointer to a 4096 by 4 array where voltage readings are returned. Send filled with zeros.

- ***stateIOout** – Pointer to a 4096 element array where IO states are returned. Send filled with zeros.
- **transferMode** – 0=auto,1=normal, 2=turbo. If auto, turbo mode is used unless timeout is >= 4, or numScans/scanRate >=4.

Outputs:

- ***idnum** – Returns the local ID or –1 if no LabJack is found.
- ***scanRate** – Returns the actual scan rate, which due to clock resolution is not always exactly the same as the desired scan rate.
- ***voltages** – Pointer to a 4096 by 4 array where voltage readings are returned. Unused locations are filled with 9999.0.
- ***stateIOout** – Pointer to a 4096 element array where IO states are returned. Unused locations are filled with 9999.0.
- ***overVoltage** – If >0, an overvoltage has been detected on at least one sample of one of the selected analog inputs.

ActiveX Function Differences:
The “channels” and “gains” arrays are replaced with “channelsPacked” and “gainsPacked”. The OCX has a function “FourPack” (4.39) which will convert 4 elements to a packed value. The packed value is determined as: element[0] + (element[1] * 2^8) + (element[2] * 2^{16}) + (element[3] * 2^{24}).

The parameters “demo”, “ledOn”, “disableCal”, “transferMode”, “updateIO”, and “stateIOin”, are replaced by an “optionBits” parameter. Call the OCX function “BuildOptionBits” (4.38) to determine this parameter.

The “voltages” and “stateIOout” arrays are represented as strings. Floating point data is returned as 13 characters per number (XXXX.XXXXXXXX) and integers are returned as 10 characters per number (XXXXXXXXXX). Zeros are used for padding where necessary. The total number of bytes in the “voltages” string is 13*numSamples. The total number of bytes in the “stateIOout” string is 10*numScans. Note that to avoid a memory leak, these strings should be emptied (set to “”) after each call to AIBurstX.

Declaration (ActiveX):

```
long AIBurstX (    long FAR* idnum,
                   long numChannels,
                   long channelsPacked,
                   long gainsPacked,
                   float FAR* scanRate,
                   long triggerIO,
                   long triggerState,
                   long numScans,
                   long timeout,
                   BSTR FAR* voltages,
                   BSTR FAR* stateIOout,
                   long FAR* overVoltage,
                   long optionBits)
```

4.8 - AIStreamStart

Starts a hardware timed continuous acquisition where data is sampled and stored in the LabJack RAM buffer, and can be simultaneously transferred out of the RAM buffer to the PC application. A call to this function should be followed by periodic calls to AIStreamRead, and eventually a call to AIStreamClear. Note that while streaming the LabJack U12 is too busy to do anything else. If any function besides AIStreamRead is called while a stream is in progress, the stream will be stopped.

Execution time for this function is 30 milliseconds or less (typically 24 milliseconds in Windows).

If the LED is enabled (ledOn>0), it will toggle every 40 samples during acquisition and turn on when the stream operation stops.

Declaration:

```
long AIStreamStart (    long *idnum,
                        long demo,
                        long stateIOin,
                        long updateIO,
                        long ledOn,
                        long numChannels,
                        long *channels,
                        long *gains,
                        float *scanRate,
                        long disableCal,
                        long reserved1,
                        long readCount )
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.
Inputs:

- ***idnum** – Local ID, serial number, or -1 for first found.
- **demo** – Send 0 for normal operation, >0 for demo mode. Demo mode allows this function to be called without a LabJack.
- **stateIOin** – Output states for IO0-IO3.
- **updateIO** – If >0, state values will be written. Otherwise, just a read is performed.
- **ledOn** – If >0, the LabJack LED is turned on.
- **numChannels** – Number of analog input channels to read (1,2, or 4). If readCount is >0, numChannels should be 4.
- ***channels** – Pointer to an array of channel commands with at least numChannels elements. Each channel command is 0-7 for single-ended, or 8-11 for differential.
- ***gains** – Pointer to an array of gain commands with at least numChannels elements. Gain commands are 0=1, 1=2, …, 7=20. This amplification is only available for differential channels.
- ***scanRate** – Scans acquired per second. A scan is a reading from every channel (1,2, or 4). The sample rate (scanRate * numChannels) must be between 200 and 1200.
- **disableCal** – If >0, voltages returned will be raw readings that are not corrected using calibration constants.
  reserved1 – Reserved for future use. Send 0.
- **readCount** – If >0, the current count (CNT) is returned instead of the 2nd, 3rd, and 4th analog input channels. 2nd channel is bits 0-11. 3rd channel is bits 12-23. 4th channel is bits 24-31. This feature was added to the LabJack U12 starting with firmware version 1.03, and this input has no effect with earlier firmware versions.

Outputs:

- ***idnum** – Returns the local ID or –1 if no LabJack is found.
- ***scanRate** – Returns the actual scan rate, which due to clock resolution is not always exactly the same as the desired scan rate.

ActiveX Function Differences:
The “channels” and “gains” arrays are replaced with “channelsPacked” and “gainsPacked”. The OCX has a function “FourPack” (4.39) which will convert 4 elements to a packed value. The packed value is determined as: element[0] + (element[1] * 2^8) + (element[2] * 2^{16}) + (element[3] * 2^{24}).

The parameters “demo”, “ledOn”, “disableCal”, “updateIO”, and “stateIOin”, are replaced by an “optionBits” parameter. Call the OCX function “BuildOptionBits” (4.38) to determine this parameter.

Declaration (ActiveX):

```
long AIStreamStartX (    long FAR* idnum,
                         long numChannels,
                         long channelsPacked,
                         long gainsPacked,
                         float FAR* scanRate,
                         long optionBits,
                         long readCount)
```

4.9 - AIStreamRead

Waits for a specified number of scans to be available and reads them. AIStreamStart should be called before this function and AIStreamClear should be called when finished with the stream. Note that while streaming the LabJack U12 is too busy to do anything else. If any function besides AIStreamRead is called while a stream is in progress, the stream will be stopped.

Note that you must pass the actual local ID to this function, not the idnum parameter used for most functions. Usually you simply pass the value returned by the idnum parameter in AIStreamStart.

Declaration:

```
long AIStreamRead (    long localID,
                       long numScans,
                       long timeout,
                       float (*voltages)[4],
                       long *stateIOout,
                       long *reserved,
                       long *ljScanBacklog,
                       long *overVoltage )
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.
Inputs:

- **localID** – Send the local ID from AIStreamStart.
- **numScans** – Function will wait until this number of scans is available. Minimum is 1. Maximum numSamples is 4096, where numSamples is numScans * numChannels. Internally this function gets data from the LabJack in blocks of 64 samples, so it is recommended that numSamples be at least 64.

- **timeout** – Function timeout value in seconds. 1 is usually a good value.
- ***voltages** – Pointer to a 4096 by 4 array where voltage readings are returned. Send filled with zeros.
- ***stateIOout** – Pointer to a 4096 element array where IO states are returned. Send filled with zeros.

Outputs:

- ***voltages** – Pointer to a 4096 by 4 array where voltage readings are returned. Unused locations are filled with 9999.0.
- ***stateIOout** – Pointer to a 4096 element array where IO states are returned. Unused locations are filled with 9999.0.
- ***reserved** – Reserved for future use. Send a pointer to a 0.
- ***ljScanBacklog** – Returns the scan backlog of the LabJack RAM buffer. This is the number of scans remaining in the U12 buffer after this read. If this value is growing from read to read, data is not being read fast enough and the buffer will eventually overflow. In normal operation this will return 0 almost all the time. The size of the buffer in terms of scans is 4096/numChannels.
- ***overVoltage** – If >0, an overvoltage has been detected on at least one sample of one of the selected analog inputs.

ActiveX Function Differences:
The "voltages" and "stateIOout" arrays are represented as strings. Floating point data is returned as 13 characters per number (XXXX.XXXXXXXX) and integers are returned as 10 characters per number (XXXXXXXXXX). Zeros are used for padding where necessary. . The total number of bytes in the "voltages" string is 13*numSamples. The total number of bytes in the "stateIOout" string is 10*numScans. Note that to avoid a memory leak, these strings should be emptied (set to "") after each call to AIStreamReadX.

Declaration (ActiveX):

```
long AIStreamReadX (    long localID,
                        long numScans,
                        long timeout,
                        BSTR FAR* voltages,
                        BSTR FAR* stateIOout,
                        long FAR* ljScanBacklog,
                        long FAR* overVoltage)
```

4.10 - AIStreamClear

This function stops the continuous acquisition. It should be called once when finished with the stream. The sequence of calls for a typical stream operation is: AIStreamStart, AIStreamRead, AIStreamRead, AIStreamRead, …, AIStreamClear.

Note that you must pass the actual localID to this function, not the idnum parameter used for most functions. Usually you simply pass the value returned by the idnum parameter in AIStreamStart.

Declaration:

```
long AIStreamClear (    long localID )
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.
Input:

- **localID** – Send the local ID from AIStreamStart/Read.

4.11 - AOUpdate

Sets the voltages of the analog outputs. Also controls/reads all 20 digital I/O and the counter.

Execution time for this function is 20 milliseconds or less (typically 16 milliseconds in Windows).

If either passed voltage is less than zero, the DLL uses the last set voltage. This provides a way to update 1 output without changing the other. Note that when the DLL is first loaded, it does not know if the analog outputs have been set, and assumes they are both the default of 0.0 volts. Similarly, there are situations where the LabJack could reset without the knowledge of the DLL, and thus the DLL could think the analog outputs are set to a non-zero voltage when in fact they have been reinitialized to 0.0 volts.

Declaration:

```
long AOUpdate (    long *idnum,
                   long demo,
                   long trisD,
                   long trisIO,
                   long *stateD,
                   long *stateIO,
```

```
                    long updateDigital,
                    long resetCounter,
                    unsigned long *count,
                    float analogOut0,
                    float analogOut1)
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.
Inputs:

- ***idnum** – Local ID, serial number, or -1 for first found.
- **demo** – Send 0 for normal operation, >0 for demo mode. Demo mode allows this function to be called without a LabJack.
- **trisD** – Directions for D0-D15. 0=Input, 1=Output.
- **trisIO** – Directions for IO0-IO3. 0=Input, 1=Output.
- ***stateD** – Output states for D0-D15.
- ***stateIO** – Output states for IO0-IO3.
- **updateDigital** – If >0, tris and state values will be written. Otherwise, just a read is performed.
- **resetCounter** – If >0, the counter is reset to zero after being read.
- **analogOut0** – Voltage from 0.0 to 5.0 for AO0.
- **analogOut1** – Voltage from 0.0 to 5.0 for AO1.

Outputs:

- ***idnum** – Returns the local ID or –1 if no LabJack is found.
- ***stateD** – States of D0-D15.
- ***stateIO** – States of IO0-IO3.
- ***count** – Current value of the 32-bit counter (CNT). This value is read before the counter is reset.

ActiveX Function Differences:
The counter read is returned as a double precision float, instead of an unsigned long.

Declaration (ActiveX):

```
long AOUpdateX (    long FAR* idnum,
                    long demo,
                    long trisD,
                    long trisIO,
                    long FAR* stateD,
                    long FAR* stateIO,
                    long updateDigital,
                    long resetCounter,
                    double FAR* count,
                    float analogOut0,
                    float analogOut1)
```

4.12 - AsynchConfig

Requires firmware V1.1 or higher. This function writes to the asynch registers and sets the direction of the D lines (input/output) as needed.

Execution time for this function is 60 milliseconds or less (typically 48 milliseconds in Windows).

The actual 1-bit time is about 1.833 plus a “full” delay (µs).

The actual 1/2-bit time is about 1.0 plus a “half” delay (µs).

full/half delay = 0.833 + 0.833C + 0.667BC + 0.5ABC (µs)

Common baud rates

Rate	Full A,B,C	Half A,B,C
1	55,153,232	114,255,34
10	63,111,28	34,123,23
100	51,191,2	33,97,3
300	71,23,4	84,39,1
600	183,3,6	236,7,1
1000	33,29,2	123,8,1
1200	23,17,4	14,54,1
2400	21,37,1	44,3,3
4800	10,18,2	1,87,1
7200	134,2,1	6,9,2
9600	200,1,1	48,2,1
10000	63,3,1	46,2,1
19200	96,1,1	22,2,1
38400	3,5,2	9,2,1
57600	3,3,2	11,1,1
100000	3,3,1	1,2,1
115200	9,1,1	2,1,1 or 1,1,1

When using data rates over 38.4 kbps, the following conditions need to be considered:

- When reading the first byte, the start bit is first tested about 11.5 µs after the start of the tx stop bit.
- When reading bytes after the first, the start bit is first tested about “full” + 11 us after the previous bit 8 read, which occurs near the middle of bit 8.

When enabled, STB does the following to aid in debugging asynchronous reads:

- STB is set about 6 µs after the start of the last tx stop bit, or about “full” + 6 us after the previous bit 8 read.
- STB is cleared about 0-2 us after the rx start bit is detected.
- STB is set after about “half”.
- STB is cleared after about “full”.
- Bit 0 is read about 1 µs later.
- STB is set about 1 µs after the bit 0 read.
- STB is cleared after about “full”.
- Bit 1 is read about 1 µs later.
- STB is set about 1 µs after the bit 1 read.
- STB is cleared after about “full”.
- This continues for all 8 data bits and the stop bit, after which STB remains low.

Declaration:

```
long AsynchConfig (    long *idnum,
                       long demo,
                       long timeoutMult,
                       long configA,
                       long configB,
                       long configTE,
                       long fullA,
                       long fullB,
                       long fullC,
                       long halfA,
                       long halfB,
                       long halfC )
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.
Inputs:

- ***idnum** – Local ID, serial number, or -1 for first found.
- **demo** – Send 0 for normal operation, >0 for demo mode. Demo mode allows this function to be called without a LabJack.
- **timeoutMult** – If enabled, read timeout is about 100 milliseconds times this value (0-255).
- **configA** – If >0, D8 is set to output-high and D9 is set to input.
- **configB** – If >0, D10 is set to output-high and D11 is set to input.
- **configTE** – If >0, D12 is set to output-low.
- **fullA/B/C** – Time constants for a “full” delay (1-255).
- **halfA/B/C** – Time constants for a “half” delay (1-255).

Outputs:

- ***idnum** – Returns the local ID or –1 if no LabJack is found.

4.13 - Asynch

Requires firmware V1.1 or higher. This function writes and then reads half-duplex asynchronous data on 1 of two pairs of D lines (8,n,1). Call AsynchConfig to set the baud rate. Similar to RS232, except that logic is normal CMOS/TTL (0=low=GND, 1=high=+5V, idle state of transmit line is high). Connection to a normal RS232 device will probably require a converter chip such as the MAX233.

Execution time for this function is about 20 milliseconds to write and/or read up to 4 bytes, plus about 20 milliseconds for each additional 4 bytes written or read. Slow baud rates can result in longer execution time.

PortA => TX is D8 and RX is D9
PortB => TX is D10 and RX is D11
Transmit Enable is D12

Up to 18 bytes can be written and read. If more than 4 bytes are written or read, this function uses calls to WriteMem/ReadMem to load/read the LabJack's data buffer.

Declaration:

```
long Asynch (    long *idnum,
                 long demo,
                 long portB,
                 long enableTE,
                 long enableTO,
                 long enableDel,
                 long baudrate,
                 long numWrite,
                 long numRead,
                 long *data )
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.
Inputs:

- ***idnum** – Local ID, serial number, or -1 for first found.
- **demo** – Send 0 for normal operation, >0 for demo mode. Demo mode allows this function to be called without a LabJack.
- **portB** – If >0, asynch PortB is used instead of PortA.
- **enableTE** – If >0, D12 (Transmit Enable) is set high during transmit and low during receive.
- **enableTO** – If >0, timeout is enabled for the receive phase (per byte).
- **enableDel** – If >0, a 1 bit delay is inserted between each transmit byte.
- **baudrate** – This is the bps as set by AsynchConfig. Asynch needs this so it has an idea how long the transfer should take.
- **numWrite** – Number of bytes to write (0-18).
- **numRead** – Number of bytes to read (0-18).
- ***data** – Serial data buffer. Send an 18 element array. Fill unused locations with zeros.

Outputs:

- ***idnum** – Returns the local ID or –1 if no LabJack is found.
- ***data** – Serial data buffer. Returns any serial read data. Unused locations are filled with 9999s.

ActiveX Function Differences:
The maximum number of bytes to read and/or write is limited to 6 (numWrite and numRead should be 0-6). The data array is replaced with pointers to 6 individual data bytes.

Declaration (ActiveX):

```
long AsynchX (    long FAR* idnum,
                  long demo,
                  long portB,
                  long enableTE,
                  long enableTO,
                  long enableDel,
                  long baudrate,
                  long numWrite,
                  long numRead,
                  long FAR* data0,
                  long FAR* data1,
                  long FAR* data2,
                  long FAR* data3,
                  long FAR* data4,
```

```
                    long FAR* data5 )
```

4.14 - BitsToVolts

Converts a 12-bit (0-4095) binary value into a LabJack voltage. No hardware communication is involved.

```
Volts=((2*Bits*Vmax/4096)-Vmax)/Gain where Vmax=10 for SE, 20 for Diff.
```

Declaration:

```
long BitsToVolts (    long chnum,
                      long chgain,
                      long bits,
                      float *volts )
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.
Inputs:

- **chnum** – Channel index. 0-7=SE, 8-11=Diff.
- **chgain** – Gain index. 0=1, 1=2, …, 7=20.
- **bits** – Binary value from 0-4095.

Outputs:

- ***volts** – Voltage.

4.15 - VoltsToBits

Converts a voltage to it's 12-bit (0-4095) binary representation. No hardware communication is involved.

```
Bits=(4096*((Volts*Gain)+Vmax))/(2*Vmax) where Vmax=10 for SE, 20 for Diff.
```

Declaration:

```
long VoltsToBits (    long chnum,
                      long chgain,
                      float volts,
                      long *bits )
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.
Inputs:

- **chnum** – Channel index. 0-7=SE, 8-11=Diff.
- **chgain** – Gain index. 0=1, 1=2, …, 7=20.
- **volts** – Voltage.

Outputs:

- ***bits** – Binary value from 0-4095.

4.16 - Counter

Controls and reads the counter. The counter is disabled if the watchdog timer is enabled.

Execution time for this function is 20 milliseconds or less (typically 16 milliseconds in Windows).

Declaration:

```
long Counter (    long *idnum,
                  long demo,
                  long *stateD,
                  long *stateIO,
                  long resetCounter,
                  long enableSTB,
                  unsigned long *count )
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.
Inputs:

- ***idnum** – Local ID, serial number, or -1 for first found.
- **demo** – Send 0 for normal operation, >0 for demo mode. Demo mode allows this function to be called without a LabJack.
- **resetCounter** – If >0, the counter is reset to zero after being read.
- **enableSTB** – If >0, STB is enabled. Used for testing and calibration. (This input has no effect with firmware V1.02 or earlier, in which case STB is always enabled)

Outputs:

- ***idnum** – Returns the local ID or –1 if no LabJack is found.
- ***stateD** – States of D0-D15.
- ***stateIO** – States of IO0-IO3.
- ***count** – Current value of the 32-bit counter (CNT). This value is read before the counter is reset.

ActiveX Function Differences:
The counter read is returned as a double precision float, instead of an unsigned long.

Declaration (ActiveX):

```
long CounterX (    long FAR* idnum,
                   long demo,
                   long FAR* stateD,
                   long FAR* stateIO,
                   long resetCounter,
                   long enableSTB,
                   double FAR* count )
```

4.17 - DigitalIO

Reads and writes to all 20 digital I/O. The order of execution within the U12 is:

1. Set D states
2. Set D directions
3. Set IO states
4. Set IO directions
5. Read D states
6. Read IO states

Even more detail of the execution order with the approximate time between each step:

1. Set D7-D0 states
2. 1 µs
3. Set D12-D8 states
4. 2 µs
5. Set D15-D13 states
6. 0.5 µs
7. Set D7-D0 directions
8. 1 µs
9. Set D12-D8 directions
10. 1 µs
11. Set D15-D13 directions
12. 16 µs
13. Set IO states
14. 16 µs
15. Set IO directions
16. 2 µs
17. Read D7-D0 states
18. 0.3 µs
19. Read D12-D8 states
20. 0.7 µs
21. Read D15-D13 states
22. 10 µs
23. Read IO states

Execution time for this function is 20 milliseconds or less (typically 16 milliseconds in Windows).

Declaration:

```
long DigitalIO (    long *idnum,
                    long demo,
                    long *trisD,
                    long trisIO,
                    long *stateD,
                    long *stateIO,
                    long updateDigital,
                    long *outputD )
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.
Inputs:

- ***idnum** – Local ID, serial number, or -1 for first found.
- **demo** – Send 0 for normal operation, >0 for demo mode. Demo mode allows this function to be called without a LabJack.
- ***trisD** – Directions for D0-D15. 0=Input, 1=Output.
- **trisIO** – Directions for IO0-IO3. 0=Input, 1=Output.
- ***stateD** – Output states for D0-D15.
- ***stateIO** – Output states for IO0-IO3.
- **updateDigital** – If >0, tris and state values will be written. Otherwise, just a read is performed.

Outputs:

- ***idnum** – Returns the local ID or –1 if no LabJack is found.
- ***trisD** – Returns a read of the direction registers for D0-D15.
- ***stateD** – States of D0-D15.
- ***stateIO** – States of IO0-IO3.
- ***outputD** – Returns a read of the output registers for D0-D15.

4.18 - GetDriverVersion

Returns the version number of ljackuw.dll. No hardware communication is involved.

Declaration:

```
float GetDriverVersion ( void )
```

Parameter Description:
Returns: Version number of ljackuw.dll.

ActiveX Function Differences:
Uses parameters to return DLL and OCX version.

Declaration (ActiveX):

```
void GetDriverVersionX (    float FAR* dllVersion,
                            float FAR* ocxVersion )
```

4.19 - GetErrorString

Converts a LabJack errorcode, returned by another function, into a string describing the error. No hardware communication is involved.

Declaration:

```
void GetErrorString (    long errorcode,
                         char *errorString )
```

Parameter Description:
Returns: Nothing.
Inputs:

- **errorcode** – LabJack errorcode.
- ***errorString** – Pointer to a 50 element array of characters.

Outputs:

- ***errorString** – Pointer to a sequence of characters describing the error. Unused locations are filled with 0×00.

4.20 - GetFirmwareVersion

Retrieves the firmware version from the LabJack's processor.

Execution time for this function is 20 milliseconds or less (typically 16 milliseconds in Windows).

Declaration:

```
float GetFirmwareVersion ( long *idnum )
```

Parameter Description:
Returns: Version number of the LabJack firmware or 0 for error.
Inputs:

- ***idnum** – Local ID, serial number, or -1 for first found.

Outputs:

- ***idnum** – Returns the local ID or –1 if no LabJack is found. If error, returns 512 plus a normal LabJack errorcode.

4.21 - GetWinVersion

Uses a Windows API function to get the OS version.

Declaration:

```
long GetWinVersion (    unsigned long *majorVersion,
                        unsigned long *minorVersion,
                        unsigned long *buildNumber,
                        unsigned long *platformID,
                        unsigned long *servicePackMajor,
                        unsigned long *servicePackMinor )
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.

Outputs:

OS Name	Platform	Major	Minor	Build
Windows 3.1	0	-	-	-
Windows 95	1	4	0	950
Windows 95 OSR2	1	4	0	1111
Windows 98	1	4	10	1998
Windows 98 SE	1	4	10	2222
Windows ME	1	4	90	3000
Windows NT 3.15	2	3	51	-
Windows NT 4.0	2	4	0	1381
Windows 2000	2	5	0	2195
Windows XP	2	5	1	-

ActiveX Function Differences:
All unsigned long parameters are changed to double precision float.

4.22 - ListAll

Searches the USB for all LabJacks, and returns the serial number and local ID for each.

Declaration:

```
long ListAll (    long *productIDList,
                  long *serialnumList,
                  long *localIDList,
                  long *powerList,
```

```
                long (*calMatrix)[20],
                long *numberFound,
                long *reserved1,
                long *reserved2 )
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.
Inputs:

- ***productIDList** – Pointer to a 127 element array. Send filled with zeros.
- ***serialnumList** – Pointer to a 127 element array. Send filled with zeros.
- ***localIDList** – Pointer to a 127 element array. Send filled with zeros.
- ***powerList** – Pointer to a 127 element array. Send filled with zeros.
- ***calMatrix** – Pointer to a 127 by 20 element array. Send filled with zeros.

Outputs:

- ***serialnumList** – Pointer to a 127 element array where serial numbers are returned. Unused locations are filled with 9999.0.
- ***localIDList** – Pointer to a 127 element array where local ID numbers are returned. Unused locations are filled with 9999.0.
- ***numberFound** – Number of LabJacks found on the USB.

ActiveX Function Differences:
The arrays are represented as strings with 10 characters per number (XXXXXXXXXX). Zeros are used for padding where necessary.

Declaration (ActiveX):

```
long ListAllX (    BSTR FAR* productIDList,
                   BSTR FAR* serialnumList,
                   BSTR FAR* localIDList,
                   BSTR FAR* powerList,
                   BSTR FAR* calMatrix,
                   long FAR* numberFound,
                   long FAR* reserved1,
                   long FAR* reserved2 )
```

4.23 - LocalID

Changes the local ID of a specified LabJack. Changes will not take effect until the LabJack is re-enumerated or reset, either manually by disconnecting and reconnecting the USB cable or by calling ReEnum or Reset.

Execution time for this function is 20 milliseconds or less (typically 16 milliseconds in Windows).

Declaration:

```
long LocalID (    long *idnum,
                  long localID )
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.
Inputs:

- ***idnum** – Local ID, serial number, or -1 for first found.
- **localID** – New local ID.

Outputs:

- ***idnum** – Returns the local ID or –1 if no LabJack is found.

4.24 - NoThread

This function is needed when interfacing TestPoint to the LabJack DLL on Windows 98/ME (see ljackuw.h for more information). Call this function to disable/enable thread creation for other functions. Normally, thread creation should be enabled, but it must be disabled for LabJack functions to work when called from TestPoint. One other situation where disabling thread creation might be useful, is when running a time-critical application in the Visual C debugger. Slow thread creation is a known problem with the Visual C debugger.

Execution time for this function is about 80 milliseconds.

If the read thread is disabled, the “timeout” specified in AIBurst and AIStreamRead is also disabled.

Declaration:

```
long NoThread (    long *idnum,
                   long noThread )
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.
Inputs:

- ***idnum*** – Local ID, serial number, or -1 for first found.
- **noThread** – If >0, the thread will not be used.

Outputs:

- ***idnum*** – Returns the local ID or –1 if no LabJack is found.

4.25 - PulseOut

Requires firmware V1.1 or higher. This command creates pulses on any/all of D0-D7. The desired D lines must be set to output using another function (DigitalIO or AOUpdate). All selected lines are pulsed at the same time, at the same rate, for the same number of pulses.

Execution time for this function is about 20 milliseconds plus pulse output time.

This function commands the time for the first half cycle of each pulse, and the second half cycle of each pulse. Each time is commanded by sending a value B & C, where the time is,

1st half-cycle microseconds = ~17 + 0.83*C + 20.17*B*C,
2nd half-cycle microseconds = ~12 + 0.83*C + 20.17*B*C,

which can be approximated as,

microseconds = 20*B*C.

For best accuracy when using the approximation, minimize C. B and C must be between 1 and 255, so each half cycle can vary from about 38/33 microseconds to just over 1.3 seconds.

If you have enabled the LabJack Watchdog function, make sure it’s timeout is longer than the time it takes to output all pulses.

The timeout of this function, in milliseconds, is set to: 5000+numPulses*((B1*C1*0.02)+(B2*C2*0.02))

Declaration:

```
long PulseOut (    long *idnum,
                   long demo,
                   long lowFirst,
                   long bitSelect,
                   long numPulses,
                   long timeB1,
                   long timeC1,
                   long timeB2,
                   long timeC2)
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.
Inputs:

- ***idnum*** – Local ID, serial number, or -1 for first found.
- **demo** – Send 0 for normal operation, >0 for demo mode. Demo mode allows this function to be called without a LabJack.
- **lowFirst** – If >0, each line is set low then high, otherwise the lines are set high then low.
- **bitSelect** – Set bits 0 to 7 to enable pulsing on each of D0-D7 (0-255).
- **numPulses** – Number of pulses for all lines (1-32767).
- **timeB1** – B value for first half cycle (1-255).
- **timeC1** – C value for first half cycle (1-255).
- **timeB2** – B value for second half cycle (1-255).
- **timeC2** – C value for second half cycle (1-255).

Outputs:

- ***idnum** – Returns the local ID or –1 if no LabJack is found.

4.26 - PulseOutStart

Requires firmware V1.1 or higher. PulseOutStart and PulseOutFinish are used as an alternative to PulseOut (See PulseOut for more information). PulseOutStart starts the pulse output and returns without waiting for the finish. PulseOutFinish waits for the LabJack's response which signifies the end of the pulse output.

Execution time for this function about 10 milliseconds.

If anything besides PulseOutFinish is called after PulseOutStart, the pulse output will be terminated and the LabJack will execute the new command. Calling PulseOutStart repeatedly, before the previous pulse output has finished, provides a pretty good approximation of continuous pulse output.

Note that due to boot-up tests on the LabJack U12, if PulseOutStart is the first command sent to the LabJack after reset or power-up, there will be no response for PulseOutFinish. In practice, even if no precautions were taken, this would probably never happen, since before calling PulseOutStart, a call is needed to set the desired D lines to output.

Also note that PulseOutFinish must be called before the LabJack completes the pulse output to read the response. If PulseOutFinish is not called until after the LabJack sends it's response, the function will never receive the response and will timeout.

Declaration:

```
long PulseOutStart (    long *idnum,
                        long demo,
                        long lowFirst,
                        long bitSelect,
                        long numPulses,
                        long timeB1,
                        long timeC1,
                        long timeB2,
                        long timeC2)
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.
Inputs:

- ***idnum** – Local ID, serial number, or -1 for first found.
- **demo** – Send 0 for normal operation, >0 for demo mode. Demo mode allows this function to be called without a LabJack.
- **lowFirst** – If >0, each line is set low then high, otherwise the lines are set high then low.
- **bitSelect** – Set bits 0 to 7 to enable pulsing on each of D0-D7 (0-255).
- **numPulses** – Number of pulses for all lines (1-32767).
- **timeB1** – B value for first half cycle (1-255).
- **timeC1** – C value for first half cycle (1-255).
- **timeB2** – B value for second half cycle (1-255).
- **timeC2** – C value for second half cycle (1-255).

Outputs:

- ***idnum** – Returns the local ID or –1 if no LabJack is found.

4.27 - PulseOutFinish

Requires firmware V1.1 or higher. See PulseOutStart for more information.

Declaration:

```
long PulseOutFinish (    long *idnum,
                         long demo,
                         long timeoutMS )
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.
Inputs:

- ***idnum** – Local ID, serial number, or -1 for first found.
- **demo** – Send 0 for normal operation, >0 for demo mode. Demo mode allows this function to be called without a LabJack.
- **timeoutMS** – Amount of time, in milliseconds, that this function will wait for the PulseOutStart response.

Outputs:

- ***idnum** – Returns the local ID or –1 if no LabJack is found.

4.28 - PulseOutCalc

Requires firmware V1.1 or higher. This function can be used to calculate the cycle times for PulseOut or PulseOutStart.

Declaration:

```
long PulseOutCalc (    float *frequency,
                       long *timeB,
                       long *timeC )
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.
Inputs:

- ***frequency** – Desired frequency in Hz.

Outputs:

- ***frequency** – Actual best frequency found in Hz.
- ***timeB** – B value for first and second half cycle.
- ***timeC** – C value for first and second half cycle.

4.29 - ReEnum

Causes the LabJack to electrically detach from and re-attach to the USB so it will re-enumerate. The local ID and calibration constants are updated at this time.

Declaration:

```
long ReEnum ( long *idnum )
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.
Inputs:

- ***idnum** – Local ID, serial number, or -1 for first found.

Outputs:

- ***idnum** – Returns the local ID or –1 if no LabJack is found.

4.30 - Reset (or ResetLJ)

Causes the LabJack to reset after about 2 seconds. After resetting the LabJack will re-enumerate. Reset and ResetLJ are identical.

Declaration:

```
long Reset ( long *idnum )
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.
Inputs:

- ***idnum** – Local ID, serial number, or -1 for first found.

Outputs:

- ***idnum** – Returns the local ID or –1 if no LabJack is found.

4.31 - SHT1X

This function retrieves temperature and/or humidity readings from an SHT1X sensor. Data rate is about 2 kbps with firmware V1.1 or higher (hardware communication). If firmware is less than V1.1, or TRUE is passed for softComm, data rate is about 20 bps.

```
DATA = IO0
SCK = IO1
```

The EI-1050 has an extra enable line that allows multiple probes to be connected at the same time using only the one line for DATA and one line for SCK. This function does not control the enable line.

This function automatically configures IO0 has an input and IO1 as an output.

Note that internally this function operates on the state and direction of IO0 and IO1, and to operate on any of the IO lines the LabJack must operate on all 4. The DLL keeps track of the current direction and output state of all lines, so that this function can operate on IO0 and IO1 without changing IO2 and IO3. When the DLL is first loaded, though, it does not know the direction and state of the lines and assumes all directions are input and output states are low.

Declaration:

```
long SHT1X (    long *idnum,
                long demo,
                long softComm,
                long mode,
                long statusReg,
                float *tempC,
                float *tempF,
                float *rh)
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.
Inputs:

- ***idnum** – Local ID, serial number, or -1 for first found.
- **demo** – Send 0 for normal operation, >0 for demo mode. Demo mode allows this function to be called without a LabJack.
- **softComm** – If >0, forces software based communication. Otherwise software communication is only used if the LabJack U12 firmware version is less than V1.1.
- **mode** – 0=temp and RH,1=temp only,2=RH only. If mode is 2, the current temperature must be passed in for the RH corrections using *tempC.
- **statusReg** – Current value of the SHT1X status register. The value of the status register is 0 unless you have used advanced functions to write to the status register (enabled heater, low resolution, or no reload from OTP).

Outputs:

- ***idnum** – Returns the local ID or –1 if no LabJack is found.
- ***tempC** – Returns temperature in degrees C. If mode is 2, the current temperature must be passed in for the RH corrections.
- ***tempF** – Returns temperature in degrees F.
- ***rh** – Returns RH in percent.

4.32 - SHTComm

Low-level public function to send and receive up to 4 bytes to from an SHT1X sensor. Data rate is about 2 kbps with firmware V1.1 or higher (hardware communication). If firmware is less than V1.1, or TRUE is passed for softComm, data rate is about 20 bps.

```
DATA = IO0
SCK = IO1
```

The EI-1050 has an extra enable line that allows multiple probes to be connected at the same time using only the one line for DATA and one line for SCK. This function does not control the enable line.

This function automatically configures IO0 has an input and IO1 as an output.

Note that internally this function operates on the state and direction of IO0 and IO1, and to operate on any of the IO lines the LabJack must operate on all 4. The DLL keeps track of the current direction and output state of all lines, so that this function can operate on IO0 and IO1 without changing IO2 and IO3. When the DLL is first loaded, though, it does not know the direction and state of the lines and assumes all directions are input and output states are low.

Declaration:

```
long SHTComm (    long *idnum,
                  long softComm,
                  long waitMeas,
                  long serialReset,
                  long dataRate,
                  long numWrite,
                  long numRead,
                  unsigned char *datatx,
                  unsigned char *datarx)
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.
Inputs:

- ***idnum** – Local ID, serial number, or -1 for first found.
- **softComm** – If >0, forces software based communication. Otherwise software communication is only used if the LabJack U12 firmware version is less than V1.1.
- **waitMeas** – If >0, this is a T or RH measurement request.
- **serialReset** – If >0, a serial reset is issued before sending and receiving bytes.
- **dataRate** – 0=no extra delay (default), 1=medium delay, 2=max delay.
- **numWrite** – Number of bytes to write (0-4).
- **numRead** – Number of bytes to read (0-4).
- ***datatx** – Array of 0-4 bytes to send. Make sure you pass at least numWrite number of bytes.

Outputs:

- ***idnum** – Returns the local ID or –1 if no LabJack is found.
- ***datarx** – Returns 0-4 read bytes as determined by numRead.

ActiveX Function Differences:
Transmit and receive data arrays are replaced with 4 individual parameters which each pass 1 transmit byte and return 1 read byte.

Declaration (ActiveX):

```
long SHTCommX (    long FAR* idnum,
                   long softComm,
                   long waitMeas,
                   long serialReset,
                   long dataRate,
                   long numWrite,
                   long numRead,
                   long FAR* data0,
                   long FAR* data1,
                   long FAR* data2,
                   long FAR* data3 )
```

4.33 - SHTCRC

Checks the CRC on an SHT1X communication. Last byte of datarx is the CRC. Returns 0 if CRC is good, or SHT1X_CRC_ERROR_LJ if CRC is bad.

Declaration:

```
long SHTCRC (    long statusReg,
                 long numWrite,
                 long numRead,
                 unsigned char *datatx,
                 unsigned char *datarx)
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.
Inputs:

- **statusReg** – Current value of the SHT1X status register.
- **numWrite** – Number of bytes that were written (0-4).
- **numRead** – Number of bytes that were read (1-4).
- ***datatx** – Array of 0-4 bytes that were sent.
- ***datarx** – Array of 1-4 bytes that were read.

ActiveX Function Differences:
Transmit and receive data arrays are each replaced with 4 individual parameters.

Declaration (ActiveX):

```
long SHTCRCX (    long statusReg,
                  long numWrite,
                  long numRead,
                  long datatx0,
                  long datatx1,
                  long datatx2,
                  long datatx3,
                  long datarx0,
                  long datarx1,
                  long datarx2,
                  long datarx3 )
```

4.34 - Synch

Requires firmware V1.1 or higher. This function performs SPI communication. Data rate is about 160 kbps with no extra delay, although delays of 100 us or 1 ms per bit can be enabled.

Execution time for this function is about 20 milliseconds to write and/or read up to 4 bytes, plus about 20 milliseconds for each additional 4 bytes written or read. Extra 20 milliseconds if configD is true, and extra time if delays are enabled.

Control of CS (chip select) can be enabled in this function for D0-D7 or handled externally via any digital output.

```
MOSI is D13
MISO is D14
SCK  is D15
```

If using the CB25, the protection resistors might need to be shorted on all SPI connections (MOSI, MISO, SCK, CS).

The initial state of SCK is set properly (CPOL), by this function, before !CS is brought low (final state is also set properly before !CS is brought high again). If chip-select is being handled manually, outside of this function, care must be taken to make sure SCK is initially set to CPOL.

All modes supported (A, B, C, and D).

```
Mode A: CPHA=1, CPOL=1
Mode B: CPHA=1, CPOL=0
Mode C: CPHA=0, CPOL=1
Mode D: CPHA=0, CPOL=0
```

If Clock Phase (CPHA) is 1, data is valid on the edge going to CPOL. If CPHA is 0, data is valid on the edge going away from CPOL. Clock Polarity (CPOL) determines the idle state of SCK.

Up to 18 bytes can be written/read. Communication is full duplex so 1 byte is read at the same time each byte is written. If more than 4 bytes are written or read, this function uses calls to WriteMem/ReadMem to load/read the LabJack's data buffer.

This function has the option (configD) to automatically configure default state and direction for MOSI (D13 Output), MISO (D14 Input), SCK (D15 Output CPOL), and CS (D0-D7 Output High for !CS). This function uses a call to DigitalIO to do this. Similar to EDigitalIn and EDigitalOut, the DLL keeps track of the current direction and output state of all lines, so that these 4 D lines can be configured without affecting other digital lines. When the DLL is first loaded, though, it does not know the direction and state of the lines and assumes all directions are input and output states are low.

Declaration:

```
long Synch (    long *idnum,
                long demo,
                long mode,
                long msDelay,
                long husDelay,
                long controlCS,
                long csLine,
                long csState,
                long configD,
                long numWriteRead,
                long *data )
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.

Inputs:

- ***idnum** – Local ID, serial number, or -1 for first found.
- **demo** – Send 0 for normal operation, >0 for demo mode. Demo mode allows this function to be called without a LabJack.
- **mode** – Specify SPI mode as: 0=A,1=B,2=C,3=D (0-3).
- **msDelay** – If >0, a 1 ms delay is added between each bit.
- **husDelay** – If >0, a hundred us delay is added between each bit.
- **controlCS** – If >0, D0-D7 is automatically controlled as CS. The state and direction of CS is only tested if control is enabled.
- **csLine** – D line to use as CS if enabled (0-7).
- **csState** – Active state for CS line. This would be 0 for the normal !CS, or >0 for the less common CS.
- **configD** – If >0, state and tris are configured for D13, D14, D15, and !CS.
- **numWriteRead** – Number of bytes to write and read (1-18).
- ***data** – Serial data buffer. Send an 18 element array of bytes. Fill unused locations with zeros.

Outputs:

- ***idnum** – Returns the local ID or –1 if no LabJack is found.
- ***data** – Serial data buffer. Returns any serial read data. Unused locations are filled with 9999s.

ActiveX Function Differences:
The maximum number of bytes to write/read is limited to 5 (numWriteRead should be 1-5). The data array is replaced with pointers to 5 individual data bytes.

Declaration (ActiveX):

```
long SynchX (    long FAR* idnum,
                 long demo,
                 long mode,
                 long msDelay,
                 long husDelay,
                 long controlCS,
                 long csLine,
                 long csState,
                 long configD,
                 long numWriteRead,
                 long FAR* data0,
                 long FAR* data1,
                 long FAR* data2,
                 long FAR* data3,
                 long FAR* data4,
                 long FAR* data5 )
```

4.35 - Watchdog

Controls the LabJack watchdog function. When activated, the watchdog can change the states of digital I/O if the LabJack does not successfully communicate with the PC within a specified timeout period. This function could be used to reboot the PC allowing for reliable unattended operation. The 32-bit counter (CNT) is disabled when the watchdog is enabled.

Execution time for this function is 20 milliseconds or less (typically 16 milliseconds in Windows).

If you set the watchdog to reset the LabJack, and choose too small of a timeout period, it might be difficult to make the device stop resetting. To disable the watchdog, reset the LabJack with IO0 shorted to STB, and then reset again without the short.

Declaration:

```
long Watchdog (    long *idnum,
                   long demo,
                   long active,
                   long timeout,
                   long reset,
                   long activeD0,
                   long activeD1,
                   long activeD8,
                   long stateD0,
                   long stateD1,
                   long stateD8 )
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.
Inputs:

- ***idnum** – Local ID, serial number, or -1 for first found.

- **demo** – Send 0 for normal operation, >0 for demo mode. Demo mode allows this function to be called without a LabJack.
- **active** – Enables the LabJack watchdog function. If enabled, the 32-bit counter is disabled.
- **timeout** – Timer reset value in seconds (1-715).
- **reset** – If >0, the LabJack will reset on timeout.
- **activeDn** – If >0, Dn will be set to stateDn upon timeout.
- **stateDn** – Timeout state of Dn, 0=low, >0=high.

Outputs:

- ***idnum** – Returns the local ID or –1 if no LabJack is found.

4.36 - ReadMem

Reads 4 bytes from a specified address in the LabJack's nonvolatile memory.

Execution time for this function is 20 milliseconds or less (typically 16 milliseconds in Windows).

Declaration:

```
long ReadMem (    long *idnum,
                  long address,
                  long *data3,
                  long *data2,
                  long *data1,
                  long *data0 )
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.
Inputs:

- ***idnum** – Local ID, serial number, or -1 for first found.
- **address** – Starting address of data to read (0-8188).

Outputs:

- ***idnum** – Returns the local ID or –1 if no LabJack is found.
- ***data3** – Byte at address.
- ***data2** – Byte at address+1.
- ***data1** – Byte at address+2.
- ***data0** – Byte at address+3.

4.37 - WriteMem

Writes 4 bytes to the LabJack's 8,192 byte nonvolatile memory at a specified address. The data is read back and verified after the write. Memory 0-511 is reserved for configuration and calibration data. Memory from 512-1023 is unused by the LabJack and available for the user (this corresponds to starting addresses from 512-1020). Memory 1024-8191 is used as a data buffer in hardware timed AI modes (burst and stream).

Execution time for this function is 20 milliseconds or less (typically 16 milliseconds in Windows).

Declaration:

```
long WriteMem (    long *idnum,
                   long unlocked,
                   long address,
                   long data3,
                   long data2,
                   long data1,
                   long data0 )
```

Parameter Description:
Returns: LabJack errorcodes or 0 for no error.
Inputs:

- ***idnum** – Local ID, serial number, or -1 for first found.
- **unlocked** – If >0, addresses 0-511 are unlocked for writing.
- **address** – Starting address for writing (0-8188).
- **data3** – Byte for address.

- **data2** – Byte for address+1.
- **data1** – Byte for address+2.
- **data0** – Byte for address+3.

Outputs:

- ***idnum** – Returns the local ID or –1 if no LabJack is found.

4.38 - BuildOptionBits (ActiveX only)

This function is only in the OCX, and is used to build the optionBits parameter for AIBurst and AIStreamStart.

The parameter optionBits is made up of the following bits and can often just be set to 2 (normal operation with the LED on):

```
bit 0     => demo
bit 1     => ledOn
bit 2     => disableCal
bits 3, 4 => transferMode
bit 5     => updateIO
bit 6     => stateIOin(0)
bit 7     => stateIOin(1)
bit 8     => stateIOin(2)
bit 9     => stateIOin(3)
```

Declaration:

```
long BuildOptionBits (    long demo,
                          long ledOn,
                          long disableCal,
                          long transferMode,
                          long updateIO,
                          long stateIOin )
```

Parameter Description:
Returns: optionBits
Inputs:

- **demo** – Send 0 for normal operation, >0 for demo mode. Demo mode allows this function to be called without a LabJack.
- **ledOn** – If >0, the LabJack LED is turned on.
- **disableCal** – If >0, voltages returned will be raw readings that are not corrected using calibration constants.
- **transferMode** – Set to 0 (automatic).
- **updateIO** – If >0, state values will be written. Otherwise, just a read is performed.
- ***stateIOin** – Output states for IO0-IO3.

4.39 FourPack (ActiveX only)

This function is only in the OCX, and is used to convert a 4 element array into an integer. The packed value is determined as: valueA + (valueB * 2^8) + (valueC * 2^{16}) + (valueD * 2^{24}).

Declaration:

```
long FourPack (    long valueA,
                   long valueB,
                   long valueC,
                   long valueD )
```

Parameter Description:
Returns: Packed representation of a 4 element array.
Inputs:

- **valueA** – Element 0 of the array to be converted.
- **valueB** – Element 1 of the array to be converted.
- **valueC** – Element 2 of the array to be converted.
- **valueD** – Element 3 of the array to be converted.

4.40 - Description of Errorcodes

It is recommended that the function GetErrorString be used to interpret errorcodes, but this list is provided as a convenience.

Error Number	Description
0	No error.
1	Unknown error.
2	No LabJacks found.
3	LabJack n not found.
4	Set USB buffer error.
5	Open handle error.
6	Close handle error.
7	Invalid ID.
8	Invalid array size or value.
9	Invalid power index.
10	FCDD size too big.
11	HVC size too big.
12	Read error.
13	Read timeout error.
14	Write error.
15	Turbo error.
16	Illegal channel index.
17	Illegal gain index.
18	Illegal AI command.
19	Illegal AO command.
20	Bits out of range.
21	Illegal number of channels.
22	Illegal scan rate.
23	Illegal number of samples.
24	AI response error.
25	LabJack RAM checksum error.
26	AI sequence error.
27	Maximum number of streams.
28	AI stream start error.
29	PC buffer overflow.
30	LabJack buffer overflow.
31	Stream read timeout.
32	Illegal number of scans.
33	No stream was found.
40	Illegal input.
41	Echo error.
42	Data echo error.
43	Response error.
44	Asynch read timeout error.
45	Asynch read start bit error.
46	Asynch read framing error.
47	Asynch DIO config error.
48	Caps error.
49	Caps error.
50	Caps error.
51	HID number caps error.
52	HID get attributes warning.
57	Wrong firmware version error.
58	DIO config error.
64	Could not claim all LabJacks.
65	Error releasing all LabJacks.
66	Could not claim LabJack.
67	Error releasing LabJack.
68	Claimed abandoned LabJack.
69	Local ID –1 thread stopped.
70	Stop thread timeout.
71	Thread termination failed.
72	Feature handle creation error.
73	Create mutex error.
80	Synchronous CS state or direction error.
81	Synchronous SCK direction error.
82	Synchronous MISO direction error.
83	Synchronous MOSI direction error.
89	SHT1X CRC error.
90	SHT1X measurement ready error.
91	SHT1X ack error.
92	SHT1X serial reset error.

If bit 8 is set, the error occurred in the stream thread. Bit 10 is set for Windows API errors.

5 - Low-Level Function Reference

This section describes the bytes that are sent to the U12 over USB. Most users are going to use the functions provided by the UW driver (Section 4).

This section is a valuable resource for anyone looking to interact directly with the U12.

Note that the LabJack U12 does not respond to the first command.

On the description of the commands, an 'X' means that the bit is ignored and can be set to either 0 or 1.

5.1 - AISample

This function reads from 4 analog inputs. It can also toggle the status LED and update the state of the IOs.

Command	
Byte #	Description
0	Bit 7: X
	Bits 6-4: PGA for 1st Channel
	Bits 3-0: MUX command for 1st Channel.
1	Bit 7: X
	Bits 6-4: PGA for 2nd Channel
	Bits 3-0: MUX command for 2nd Channel.
2	Bit 7: X
	Bits 6-4: PGA for 3rd Channel
	Bits 3-0: MUX command for 3rd Channel.
3	Bit 7: X
	Bits 6-4: PGA for 4th Channel
	Bits 3-0: MUX command for 4th Channel.
4	Bits 7-2: XXXXXX
	Bit 1: Update IO
	Bit 0: LED State
5	Bit 7-4: 1100 (Command/Response)
	Bit 3-0: Bits for IO3 through IO0 States
6	XXXXXXXX
7	Echo Value
Response	
Byte #	Description
0	Bit 7: 1
	Bit 6: 0
	Bit 5: X
	Bit 4: PGA Overvoltage
	Bits 3-0: Bits for IO3 through IO0 States
1	Echo byte 7 from Command
2	Bits 7-4: Most Significant Bits from 1st Channel
	Bits 3-0: Most Significant Bits from 2nd Chanel
3	Least Significant Byte from 1st Channel
4	Least Significant Byte from 2nd Channel
5	Bits 7-4: Most Significant Bits from 3rd Channel
	Bits 3-0: Most Significant Bits from 4th Chanel
6	Least Significant Byte from 3rd Channel
7	Least Significant Byte from 4th Channel

- **PGA Gain Setting** – (Differential Only) 0b000 = 1, 0b001 = 2, 0b010 = 4, 0b100 = 8, 0b101 = 10, 0b110 = 16, 0b111 = 20
- **Mux Settings** – 0b0000 = 0-1 (Differential), 0b0001 = 2-3 (Differential), 0b0010 = 4-5 (Differential), 0b0011 = 6-7 (Differential). Single-Ended readings = 0b1000 + AI Number.

LabJackPython Example

```
>>> import u12
```

```
>>> d = u12.U12()
open called
Writing: [0x0, 0x0, 0x0, 0x0, 0x0, 0x57, 0x0, 0x0]
Received: [0x57, 0x0, 0x0, 0x0, 0xff, 0xff, 0x0, 0x0]
>>> d.rawAISample()
Writing: [0x8, 0x9, 0xa, 0xb, 0x1, 0xc0, 0x0, 0x0]
Received: [0x80, 0x0, 0x99, 0xb, 0x28, 0x99, 0x2c, 0x5]
{
  'IO3toIO0States':
    <BitField object: [ IO3 = Low (0), IO2 = Low (0),
                        IO1 = Low (0), IO0 = Low (0) ] >,
  'Channel1': 1.4453125,
  'Channel3': 1.2744140625,
  'Channel2': 1.46484375,
  'PGAOvervoltage': False,
  'Channel0': 1.3037109375,
  'EchoValue': 0
}
```

5.2 - DIO

This commands reads the direction and state of all the digital I/O. If Update Digital = 1, this function also sets the directions and states before reading.

Command	
Byte #	Description
0	Bits for D15 through D8 Direction (0 = Output, 1 = Input)
1	Bits for D7 through D0 Direction (0 = Output, 1 = Input)
2	Bits for D15 through D8 State (0 = Low, 1 = High)
3	Bits for D7 through D0 State (0 = Low, 1 = High)
4	Bits 7-4: Bits for IO3 through IO0 Direction
	Bits 3-0: Bits for IO3 through IO0 State
5	01X10111 (DIO)
6	Bits 7-1: XXXXXXX
	Bit 0: Update Digital
7	XXXXXXXX
Response	
Byte #	Description
0	01X10111 (DIO Response)
1	Bits for D15 through D8 State (0 = Low, 1 = High)
2	Bits for D7 through D0 State (0 = Low, 1 = High)
3	Bits 7-4: Bits for IO3 through IO0 State
	Bits 3-0: XXXX
4	Bits for D15 through D8 Direction (0 = Output, 1 = Input)
5	Bits for D7 through D0 Direction (0 = Output, 1 = Input)
6	Bits for D15 through D8 Output Latch States
7	Bits for D7 through D0 Output Latch States

LabJackPython Example

```
>>> import u12
>>> d = u12.U12()
open called
Writing: [0x0, 0x0, 0x0, 0x0, 0x0, 0x57, 0x0, 0x0]
Received: [0x57, 0x0, 0x0, 0x0, 0xff, 0xff, 0x0, 0x0]
>>> d.rawDIO()
Writing: [0x0, 0x0, 0x0, 0x0, 0x0, 0x57, 0x0, 0x0]
Received: [0x57, 0x0, 0x0, 0x0, 0xff, 0xff, 0x0, 0x0]
{
  'D7toD0States':
    <BitField object: [ D7 = Low (0), D6 = Low (0),
                        D5 = Low (0), D4 = Low (0),
                        D3 = Low (0), D2 = Low (0),
                        D1 = Low (0), D0 = Low (0) ] >,
  'IO3toIO0States':
    <BitField object: [ IO3 = Low (0), IO2 = Low (0),
                        IO1 = Low (0), IO0 = Low (0) ] >,
```

```
  'D15toD8Directions':
    <BitField object: [ D15 = Input (1), D14 = Input (1),
                        D13 = Input (1), D12 = Input (1),
                        D11 = Input (1), D10 = Input (1),
                        D9 = Input (1), D8 = Input (1) ] >,
  'D7toD0OutputLatchStates':
    <BitField object: [ D7 = 0 (0), D6 = 0 (0),
                        D5 = 0 (0), D4 = 0 (0),
                        D3 = 0 (0), D2 = 0 (0),
                        D1 = 0 (0), D0 = 0 (0) ] >,
  'D15toD8OutputLatchStates':
    <BitField object: [ D15 = 0 (0), D14 = 0 (0),
                        D13 = 0 (0), D12 = 0 (0),
                        D11 = 0 (0), D10 = 0 (0),
                        D9 = 0 (0), D8 = 0 (0) ] >,
  'D15toD8States':
    <BitField object: [ D15 = Low (0), D14 = Low (0),
                        D13 = Low (0), D12 = Low (0),
                        D11 = Low (0), D10 = Low (0),
                        D9 = Low (0), D8 = Low (0) ] >,
  'D7toD0Directions':
    <BitField object: [ D7 = Input (1), D6 = Input (1),
                        D5 = Input (1), D4 = Input (1),
                        D3 = Input (1), D2 = Input (1),
                        D1 = Input (1), D0 = Input (1) ] >
}
```

5.3 - Counter

This command controls and reads the 32-bit counter.

Command	
Byte #	Description
0	Bits 7-2: XXXXXX
	Bit 1: Strobe Enabled
	Bit 0: Reset Counter
1	XXXXXXXX
2	XXXXXXXX
3	XXXXXXXX
4	XXXXXXXX
5	01X1XX10 (Counter)
6	XXXXXXXX
7	XXXXXXXX
Response	
Byte #	Description
0	01X1XX10 (Counter)
1	Bits for D15 though D8 State (0 = Low, 1 = High)
2	Bits for D7 though D0 State (0 = Low, 1 = High)
3	Bits 7-4 = IO3 through IO0 State
	Bits 3-0 = XXXX
4	Most Significant Byte of Counter
5	Bits 23-16 of Counter
6	Bits 15-8 of Counter
7	Least Significant Byte of Counter

[textile]
This command controls and reads the 32-bit counter.

Command	
Byte #	Description
0	Bits 7-2: XXXXXX
	Bit 1: Strobe Enabled
	Bit 0: Reset Counter
1	XXXXXXXX
2	XXXXXXXX
3	XXXXXXXX
4	XXXXXXXX
5	01X1XX10 (Counter)
6	XXXXXXXX
7	XXXXXXXX
Response	
Byte #	Description
0	01X1XX10 (Counter)
1	Bits for D15 though D8 State (0 = Low, 1 = High)
2	Bits for D7 though D0 State (0 = Low, 1 = High)
3	Bits 7-4 = IO3 through IO0 State
	Bits 3-0 = XXXX
4	Most Significant Byte of Counter
5	Bits 23-16 of Counter
6	Bits 15-8 of Counter
7	Least Significant Byte of Counter

LabJackPython Example

```
>>> import u12
>>> d = u12.U12()
open called
Writing: [0x0, 0x0, 0x0, 0x0, 0x0, 0x57, 0x0, 0x0]
Received: [0x57, 0x0, 0x0, 0x0, 0xff, 0xff, 0x0, 0x0]
>>> d.rawCounter()
Writing: [0x0, 0x0, 0x0, 0x0, 0x0, 0x52, 0x0, 0x0]
Received: [0x52, 0x0, 0x0, 0x0, 0xbb, 0x10, 0x0, 0xef]
{
  'IO3toIO0States':
    <BitField object: [ IO3 = Low (0), IO2 = Low (0),
                        IO1 = Low (0), IO0 = Low (0) ] >,
  'Counter': 3138388207,
  'D7toD0States':
    <BitField object: [ D7 = Low (0), D6 = Low (0),
                        D5 = Low (0), D4 = Low (0),
                        D3 = Low (0), D2 = Low (0),
                        D1 = Low (0), D0 = Low (0) ] >,
  'D15toD8States':
    <BitField object: [ D15 = Low (0), D14 = Low (0),
                        D13 = Low (0), D12 = Low (0),
                        D11 = Low (0), D10 = Low (0),
                        D9 = Low (0), D8 = Low (0) ] >}
```

[/textile]

5.4 - Counter/AO/DIO

This command controls all 20 digital I/O, and the 2 analog outputs (AO0 & AO1). The response provides the state of all I/O and the current counter value. If Reset Counter = 1, the counter is reset after reading.

Command	
Byte #	Description
0	D15 though D8 Direction (0 = Output, 1 = Input)
1	Bits for D7 though D0 Direction (0 = Output, 1 = Input)
2	Bits for D15 though D8 State (0 = Low, 1 = High)
3	Bits for D7 though D0 State (0 = Low, 1 = High)
4	Bits 7-4 = Bits for IO3 through IO0 Direction
	Bits 3-0 = Bits for IO3 through IO0 State
5	Bits 7-6 = 00 (Counter/AO/DIO)
	Bit 5 = Reset Counter
	Bit 4 = Update Digital
	Bits 3-2 = Least Significant Bits of AO0 duty cycle
	Bits 1-0 = Least Significant Bits of AO1 duty cycle
6	Most Significant bits of AO0 duty cycle (0 = 0 V, 0x3ff = 5.0 V)
7	Most Significant bits of AO1 duty cycle (0 = 0 V, 0x3ff = 5.0 V)
Response	
Byte #	Description
0	00XXXXXX (counter/pwm response)
1	Bits for D15 though D8 State (0 = Low, 1 = High)
2	Bits for D7 though D0 State (0 = Low, 1 = High)
3	Bits 7-4 = Bits for IO3 through IO0 State
	Bits 3-0 = XXXX
4	Most Significant Byte of Counter
5	Bits 23-16 of Counter
6	Bits 15-8 of Counter
7	Least Significant Byte of Counter

LabJackPython Example

```
>>> import u12
>>> d = u12.U12()
open called
Writing: [0x0, 0x0, 0x0, 0x0, 0x0, 0x57, 0x0, 0x0]
Received: [0x57, 0x0, 0x0, 0x0, 0xff, 0xff, 0x0, 0x0]
>>> d.rawCounterPWMDIO()
Writing: [0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0]
Received: [0x0, 0x0, 0x0, 0x0, 0xbb, 0x10, 0x0, 0xef]
{
  'IO3toIO0States':
    <BitField object: [ IO3 = Low (0), IO2 = Low (0),
                        IO1 = Low (0), IO0 = Low (0) ] >,
  'Counter': 3138388207,
  'D7toD0States':
    <BitField object: [ D7 = Low (0), D6 = Low (0),
                        D5 = Low (0), D4 = Low (0),
                        D3 = Low (0), D2 = Low (0),
                        D1 = Low (0), D0 = Low (0) ] >,
  'D15toD8States':
    <BitField object: [ D15 = Low (0), D14 = Low (0),
                        D13 = Low (0), D12 = Low (0),
                        D11 = Low (0), D10 = Low (0),
                        D9 = Low (0), D8 = Low (0) ] >
}
```

5.5 - AIBurst

After receiving a AIBurst command, the LabJack collects 4 channels at the specified data rate, and puts data in the buffer. This continues until the buffer is full, at which time the LabJack starts sending the data to the host. Data is sent to the host 1 scan at a time while checking for a command from the host. If a command is received the burst operation is canceled and the command is executed normally. If the LED is enabled, it blinks at 4 Hz while waiting for a trigger, is off during acquisition, blinks at about 8 Hz during data delivery, and is set on when done or stopped.

Command	
Byte #	Description
0	Bit 7: X
	Bits 6-4: PGA for 1st Channel
	Bits 3-0: MUX command for 1st Channel.
1	Bit 7: X
	Bits 6-4: PGA for 2nd Channel
	Bits 3-0: MUX command for 2nd Channel.
2	Bit 7: X
	Bits 6-4: PGA for 3rd Channel
	Bits 3-0: MUX command for 3rd Channel.
3	Bit 7: X
	Bits 6-4: PGA for 4th Channel
	Bits 3-0: MUX command for 4th Channel.
4	Bits 7-5: Number of scans (000 = 1024, 110 = 16)
	Bits 4-3: IO to trigger burst on
	Bit 2: State to trigger on
	Bit 1: Update IO
	Bit 0: LED State
5	Bits 7-4: 1010 (Start Burst)
	Bits 3-0: Bits for IO3 through IO0 States
6	Bit 7: Feature Reports
	Bit 6: Trigger On
	Bits 5-0: AIINTMSB (High 6 bits of sample interval)
7	AIINTLSB (733 = min => ~8192 Hz)
Response	
Byte #	Description
0	Bit 7-6: 10 (binary)
	Bit 5: Buffer Overflow if Backlog = 11111, Checksum Error if Backlog = 0
	Bit 4: PGA Overvoltage
	Bits 3-0: Bits for IO3 through IO0 State
1	Bits 7-5: Iteration Counter
	Bits 4-0: Backlog/256
2	Bits 7-4: Most Significant Bits from 1st Channel
	Bits 3-0: Most Significant Bits from 2nd Chanel
3	Least Significant Byte from 1st Channel
4	Least Significant Byte from 2nd Channel
5	Bits 7-4: Most Significant Bits from 3rd Channel
	Bits 3-0: Most Significant Bits from 4th Chanel
6	Least Significant Byte from 3rd Channel
7	Least Significant Byte from 4th Channel

- **PGA Gain Setting** – (Differential Only) 0b000 = 1, 0b001 = 2, 0b010 = 4, 0b100 = 8, 0b101 = 10, 0b110 = 16, 0b111 = 20
- **Mux Settings** – 0b0000 = 0-1 (Differential), 0b0001 = 2-3 (Differential), 0b0010 = 4-5 (Differential), 0b0011 = 6-7 (Differential). Single-Ended readings = 0b1000 + AI Number.

LabJackPython Example

```
>>> import u12
>>> d = u12.U12()
open called
Writing: [0x0, 0x0, 0x0, 0x0, 0x0, 0x57, 0x0, 0x0]
Received: [0x57, 0x0, 0x0, 0x0, 0xff, 0xff, 0x0, 0x0]
>>> d.rawAIBurst()
Writing: [0x8, 0x9, 0xa, 0xb, 0xe1, 0xa0, 0xa, 0x98]
Received: [0x80, 0x0, 0x99, 0x8, 0x2a, 0x99, 0x2c, 0x6]
Received: [0x80, 0x20, 0x99, 0xc, 0x2a, 0x99, 0x2c, 0x4]
Received: [0x80, 0x40, 0x99, 0xc, 0x2c, 0x99, 0x2a, 0x6]
Received: [0x80, 0x60, 0x99, 0xc, 0x2a, 0x99, 0x2c, 0x4]
Received: [0x80, 0x80, 0x99, 0xc, 0x2c, 0x99, 0x2c, 0x6]
Received: [0x80, 0xa0, 0x99, 0x0, 0x2a, 0x99, 0x2c, 0x4]
Received: [0x80, 0xc0, 0x99, 0xc, 0x2a, 0x99, 0x2c, 0x6]
Received: [0x80, 0x0, 0x99, 0xc, 0x2a, 0x99, 0x2c, 0x6]
{
  'Channel0': [1.2890625, 1.30859375, ..., 1.30859375],
  'Channel3': [1.279296875, 1.26953125, ..., 1.279296875],
```

```
  'Channel2': [1.46484375, 1.46484375, ..., 1.46484375],
  'PGAOvervoltages': [False, False, ..., False],
  'IO3toIO0States':
    [
      <BitField object: [ IO3 = Low (0), IO2 = Low (0),
                          IO1 = Low (0), IO0 = Low (0) ] >,
      <BitField object: [ IO3 = Low (0), IO2 = Low (0),
                          IO1 = Low (0), IO0 = Low (0) ] >,
    ...,
      <BitField object: [ IO3 = Low (0), IO2 = Low (0),
                          IO1 = Low (0), IO0 = Low (0) ] >],
  'BufferOverflowOrChecksumErrors': [False, False, ..., False],
  'Channel1': [1.455078125, 1...., 1.455078125],
  'IterationCounters': [0, 1, 2, 3, 4, 5, 6, 0],
  'Backlogs': [0, 0, 0, 0, 0, 0, 0, 0]
}
```

5.6 - AIContinuous

After receiving a continuous AI command, the LabJack collects 4 channels at a sample rate determined by AIINT (Analog Input Interval) which is the interval between samples in cycles. The data is put in the buffer as it it is sampled. Simultaneously, the LabJack pulls data out of the buffer and tries to send it to the host. This continues until the LabJack receives any command from the host. If a command is received the continuous operation is canceled and the command is executed normally. If the LED is enabled, it toggles every 40 samples during acquisition and is set on when stopped.

Command	
Byte #	Description
0	Bit 7: X
	Bits 6-4: PGA for 1st Channel
	Bits 3-0: MUX command for 1st Channel.
1	Bit 7: X
	Bits 6-4: PGA for 2nd Channel
	Bits 3-0: MUX command for 2nd Channel.
2	Bit 7: X
	Bits 6-4: PGA for 3rd Channel
	Bits 3-0: MUX command for 3rd Channel.
3	Bit 7: X
	Bits 6-4: PGA for 4th Channel
	Bits 3-0: MUX command for 4th Channel.
4	Bit 7: Feature Reports
	Bit 6: Counter Read
	Bits 5-2: XXXX
	Bit 1: Update IO
	Bit 0: LED State
5	Bits 7-4: 1001 (Start Continuous)
	Bits 3-0: Bits for IO3 through IO0 States
6	AIINTMSB
7	AIINTLSB
Response	
Byte #	Description
0	Bit 7-6: 11
	Bit 5: Buffer Overflow if Backlog = 11111, Checksum Error if Backlog = 0
	Bit 4: PGA Overvoltage
	Bits 3-0: Bits for IO3 through IO0 States
1	Bits 7-5: Iteration Counter
	Bits 4-0: Backlog/256
2	Bits 7-4: Most Significant Bits from 1st Channel
	Bits 3-0: Most Significant Bits from 2nd Chanel
3	Least Significant Byte from 1st Channel
4	Least Significant Byte from 2nd Channel
5	Bits 7-4: Most Significant Bits from 3rd Channel
	Bits 3-0: Most Significant Bits from 4th Chanel
6	Least Significant Byte from 3rd Channel
7	Least Significant Byte from 4th Channel

- **PGA Gain Setting** – (Differential Only) 0b000 = 1, 0b001 = 2, 0b010 = 4, 0b100 = 8, 0b101 = 10, 0b110 = 16, 0b111 = 20

- **Mux Settings** – 0b0000 = 0-1 (Differential), 0b0001 = 2-3 (Differential), 0b0010 = 4-5 (Differential), 0b0011 = 6-7 (Differential). Single-Ended readings = 0b1000 + AI Number.

5.7 - Pulseout

This command creates pulses on any to all of D0-D7. The desired D lines must be set to output with some other function. For best accuracy, when using the BC*20us approximation, keep C as small as possible. B and C must be >= 1, and must do at least 1 pulse (LSB >= 1).

Command	
Byte #	Description
0	B1
1	C1
2	B2
3	C2
4	Bits for D7 through D0 (1 = Pulse the D line)
5	011XX100 (Pulseout)
6	Bit 7: Clear First?
	Bits 6-0: MSB Number of Pulses
7	LSB Number of Pulses
Response	
Byte #	Description
0	Echo of byte 0 of Command
1	Echo of byte 1 of Command
2	Echo of byte 2 of Command
3	Echo of byte 3 of Command
4	Bits of direction errors for D7 through D0
5	011XX100 (Echo of byte 5)
6	Echo of byte 6 of Command
7	Echo of byte 7 of Command

- **B1/C1** – Period of first half cycle is ~100+5C+121BC or about BC*20µs
- **B2/C2** – Period of second half cycle is ~70+5C+121BC or about BC*20µs

LabJackPython Example

Connect a jumper wire from D0 to CNT.

```
>>> import u12
>>> d = u12.U12()
open called
Writing: [0x0, 0x0, 0x0, 0x0, 0x0, 0x57, 0x0, 0x0]
Received: [0x57, 0x0, 0x0, 0x0, 0xff, 0xff, 0x0, 0x0]
>>> d.rawDIO(D7toD0Directions = 0, UpdateDigital = True)
>>> d.rawCounter(ResetCounter = True)
>>> d.rawPulseout(ClearFirst = True)
>>> print d.rawCounter()
{
  'IO3toIO0States': ... ,
  'Counter': 5,
  'D7toD0States': ... ,
  'D15toD8States': ...
}
```

5.8 - Reset

Sits in an infinite loop until micro watchdog timeout after about 2 seconds.

Command	
Byte #	Description
0	XXXXXXXX
1	XXXXXXXX
2	XXXXXXXX
3	XXXXXXXX
4	XXXXXXXX
5	01X11111 (Reset)
6	XXXXXXXX
7	XXXXXXXX

LabJackPython Example

```
>>> import u12
>>> d = u12.U12()
open called
Writing: [0x0, 0x0, 0x0, 0x0, 0x0, 0x57, 0x0, 0x0]
Received: [0x57, 0x0, 0x0, 0x0, 0xff, 0xff, 0x0, 0x0]
>>> d.rawReset()
Writing: [0x0, 0x0, 0x0, 0x0, 0x0, 0x5f, 0x0, 0x0]
```

5.9 - Re-Enumerate

Detaches from the USB, reloads config parameters, and then reattaches so the device can be re-enumerated. Config parameters include local ID, power allowance, and calibration data.

Command	
Byte #	Description
0	XXXXXXXX
1	XXXXXXXX
2	XXXXXXXX
3	XXXXXXXX
4	XXXXXXXX
5	0100XXXX (Re-Enumerate)
6	XXXXXXXX
7	XXXXXXXX

LabJackPython Example

```
>>> import u12
>>> d = u12.U12()
open called
Writing: [0x0, 0x0, 0x0, 0x0, 0x0, 0x57, 0x0, 0x0]
Received: [0x57, 0x0, 0x0, 0x0, 0xff, 0xff, 0x0, 0x0]
>>> d.rawReenumerate()
Writing: [0x0, 0x0, 0x0, 0x0, 0x0, 0x40, 0x0, 0x0]
```

5.10 - Watchdog

Controls a watchdog function, which is separate from the micro watchdog. If this watchdog function is set active, D0, D1, & D8 can be set or cleared upon timeout. Timeout range is about 0.01 to 715 seconds. The counter is reset upon any successful communication to or from the USB host. If Reset on Timeout = 1, the Reset function will be executed upon timeout. User will need to configure the digital channels as outputs. When the watchdog function is active, the external 32-bit counter doesn't work since timer1 counts program cycles.

Command	
Byte #	Description
0	Bits 7-1: XXXXXXX
	Bit 0: Ignore Commands
1	XXXXXXXX
2	XXXXXXXX
3	XXXXXXXX
4	Bit 7: D0 Active
	Bit 6: D0 State
	Bit 5: D1 Active
	Bit 4: D1 State
	Bit 3: D8 Active
	Bit 2: D8 State
	Bit 1: Reset on Timeout
	Bit 0: Watchdog Function Active?
5	01X1X011 (Watchdog)
6	Most Significant Timeout Byte
7	Least Significant Timeout Byte
Results	
Byte #	Description
0	Left side of Firmware Version (X.00)
1	Right side of Firmware Version (0.XX)
2	Echo of Byte 2 of Command
3	Echo of Byte 3 of Command
4	Echo of Byte 4 of Command
5	Echo of Byte 5 of Command
6	Echo of Byte 6 of Command
7	Echo of Byte 7 of Command

LabJackPython Example

```
>>> import u12
>>> d = u12.U12()
open called
Writing: [0x0, 0x0, 0x0, 0x0, 0x0, 0x57, 0x0, 0x0]
Received: [0x57, 0x0, 0x0, 0x0, 0xff, 0xff, 0x0, 0x0]
>>> d.rawWatchdog()
Writing: [0x0, 0x0, 0x0, 0x0, 0x0, 0x53, 0x17, 0x70]
Received: [0x1, 0xa, 0x0, 0x0, 0x0, 0x53, 0x17, 0x70]
{'FirmwareVersion': '1.10'}
```

5.11 - Read RAM

Reads 4 bytes out of the U12's internal memory.

Note: Bytes are read down from the starting address. For example, if you read at address 0x203, then byte 0 comes from 0x203, byte 1 comes from 0x202, byte 2 comes from 0x201, and byte 0 comes from 0x200.

Command	
Byte #	Description
0	XXXXXXXX
1	XXXXXXXX
2	XXXXXXXX
3	XXXXXXXX
4	XXXXXXXX
5	01X1XX00 (Read RAM)
6	Most Significant Address Byte
7	Least Significant Address Byte

Response	
Byte #	Description
0	01010000
1	Readback of data byte 3
2	Readback of data byte 2
3	Readback of data byte 1
4	Readback of data byte 0
5	XXXXXXXX
6	Most Significant Address Byte
7	Least Significant Address Byte

Non-Volatile RAM Map	
Address Range	Description
0x00	A copy of the serial # which is stored in ROM
0x08 - 0x0A	Always 0
0x0B	LocalID
0x0C	wdoglj (Watchdog Variable)
0x0D	wdogtoh (Watchdog Variable)
0x0E	wdogtol (Watchdog Variable)
0x070	fullA (Asynch Variable)
0x071	fullB (Asynch Variable)
0x072	fullC (Asynch Variable)
0x073	halfA (Asynch Variable)
0x074	halfB (Asynch Variable)
0x075	halfC (Asynch Variable)
0x076	tomult (Asynch Variable)
0x080 - 0x0BF	Serial data send buffer (64 bytes, 28 used)
0x0C0 - 0x0FF	Serial data receive buffer (64 bytes, 28 used)
0x1C0 - 0x1DF	Serial data buffer (32 bytes)
0x200 - 0x3FF	User area
0x400 - 0x1FFF	Circular RAM buffer.

LabJackPython Example

```
>>> import u12
>>> d = u12.U12()
open called
Writing: [0x0, 0x0, 0x0, 0x0, 0x0, 0x57, 0x0, 0x0]
Received: [0x57, 0x0, 0x0, 0x0, 0xff, 0xff, 0x0, 0x0]
>>> r = d.rawReadRAM()
Writing: [0x0, 0x0, 0x0, 0x0, 0x0, 0x50, 0x0, 0x0]
Received: [0x50, 0x5, 0xf6, 0x8b, 0xaa, 0x0, 0x0, 0x0]
>>> print r
{'DataByte3': 5, 'DataByte2': 246, 'DataByte1': 139, 'DataByte0': 170}
>>> bytes = [ r['DataByte3'], r['DataByte2'], r['DataByte1'], r['DataByte0'] ]
>>> import struct
>>> print struct.unpack(">I", struct.pack("BBBB", *bytes))[0]
100043690
```

5.12 - Write RAM

Writes 4 bytes to the U12's internal memory.

Note: Bytes are written down from the starting address. For example, if you write starting at address 0x203, then byte 0 is written to 0x203, byte 1 is written to 0x202, byte 2 is written to 0x201, and byte 0 is written to 0x200.

Command	
Byte #	Description
0	Data Byte 3
1	Data Byte 2
2	Data Byte 1
3	Data Byte 0
4	XXXXXXXX
5	01X1XX01 (Write RAM)
6	Most Significant Address Byte
7	Least Significant Address Byte
Response	
Byte #	Description
0	01010001
1	Readback of data byte 3
2	Readback of data byte 2
3	Readback of data byte 1
4	Readback of data byte 0
5	XXXXXXXX
6	Most Significant Address Byte
7	Least Significant Address Byte

Non-Volatile RAM Map	
Address Range	Description
0x00	A copy of the serial # which is stored in ROM
0x08 - 0x0A	Always 0
0x0B	LocalID
0x0C	wdoglj (Watchdog Variable)
0x0D	wdogtoh (Watchdog Variable)
0x0E	wdogtol (Watchdog Variable)
0x070	fullA (Asynch Variable)
0x071	fullB (Asynch Variable)
0x072	fullC (Asynch Variable)
0x073	halfA (Asynch Variable)
0x074	halfB (Asynch Variable)
0x075	halfC (Asynch Variable)
0x076	tomult (Asynch Variable)
0x080 - 0x0BF	Serial data send buffer (64 bytes, 28 used)
0x0C0 - 0x0FF	Serial data receive buffer (64 bytes, 28 used)
0x1C0 - 0x1DF	Serial data buffer (32 bytes)
0x200 - 0x3FF	User area
0x400 - 0x1FFF	Circular RAM buffer.

LabJackPython Example

```
>>> import u12
>>> d = u12.U12()
open called
Writing: [0x0, 0x0, 0x0, 0x0, 0x0, 0x57, 0x0, 0x0]
Received: [0x57, 0x0, 0x0, 0x0, 0xff, 0xff, 0x0, 0x0]
>>> d.rawWriteRAM([1, 2, 3, 4], 0x200)
Writing: [0x4, 0x3, 0x2, 0x1, 0x0, 0x51, 0x2, 0x0]
Received: [0x51, 0x4, 0x3, 0x2, 0x1, 0x0, 0x2, 0x0]
{'DataByte3': 4, 'DataByte2': 3, 'DataByte1': 2, 'DataByte0': 1}
```

5.13 - Asynch

Sends data asynchronously, then receives data. First four bytes sent or received are passed in this call, and copied to the first 4 bytes of RAM buffer (first 4 bytes of RAM transmit buffer are never used). Any additional bytes must be written/read with the RAM.

Baud rate and timeout period must be set by writing to the proper RAM locations. Timeout in milliseconds is about 100*tomult.

Command	
Byte #	Description
0	Data Byte 3
1	Data Byte 2
2	Data Byte 1
3	Data Byte 0
4	Bits 7-4: XXXX
	Bit 3: 1 bit delay between each tx byte
	Bit 2: Timeout active?
	Bit 1: Set transmit enable?
	Bit 0: port B?
5	011XXXX1 (Asynch)
6	Number of bytes to write (0-18)
7	Number of bytes to read (0-18)
Response	
Byte #	Description
0	Data Byte 3
1	Data Byte 2
2	Data Byte 1
3	Data Byte 0
4	Bits 7-6: XX
	Bit 5: Timeout Error Flag
	Bit 4: STRT Error Flag
	Bit 3: FRM Error Flag
	Bit 2: RXTris Error Flag
	Bit 1: TETris Error Flag
	Bit 0: TXTris Error Flag
5	011XXXX1
6	Number of bytes to write (0-18)
7	Number of bytes to read (0-18)

LabJackPython Example

```
>>> import u12
>>> d = u12.U12()
open called
Writing: [0x0, 0x0, 0x0, 0x0, 0x0, 0x57, 0x0, 0x0]
Received: [0x57, 0x0, 0x0, 0x0, 0xff, 0xff, 0x0, 0x0]
>>> # Set the full and half A,B,C to 9600
>>> d.rawWriteRAM([0, 1, 1, 200], 0x073)
Writing: [0xc8, 0x1, 0x1, 0x0, 0x0, 0x51, 0x0, 0x73]
Received: [0x51, 0xc8, 0x1, 0x1, 0x0, 0x0, 0x0, 0x73]
{'DataByte3': 200, 'DataByte2': 1, 'DataByte1': 1, 'DataByte0': 0}
>>> d.rawWriteRAM([5, 1, 2, 48], 0x076)
Writing: [0x30, 0x2, 0x1, 0x5, 0x0, 0x51, 0x0, 0x76]
Received: [0x51, 0x30, 0x2, 0x1, 0x5, 0x0, 0x0, 0x76]
{'DataByte3': 48, 'DataByte2': 2, 'DataByte1': 1, 'DataByte0': 5}
>>> d.rawAsynch([1, 2, 3, 4], NumberOfBytesToWrite = 4, NumberOfBytesToRead = 4)
Writing: [0x4, 0x3, 0x2, 0x1, 0x0, 0x61, 0x4, 0x4]
Received: [0x4, 0x3, 0x2, 0x1, 0x1, 0x61, 0x4, 0x4]
{
  'DataByte3': 4, 'DataByte2': 3, 'DataByte1': 2, 'DataByte0': 1,
  'ErrorFlags':
    <BitField object: [ Timeout Error Flag = 0 (0),
                        STRT Error Flag = 0 (0),
                        FRM Error Flag = 0 (0),
                        RXTris Error Flag = 0 (0),
                        TETris Error Flag = 0 (0),
                        TXTris Error Flag = 1 (1) ] >
}
```

5.14 - SPI

Sends and receives data synchronously (SPI). First four bytes sent/received are passed in this call, and copied to the first 4 bytes

of RAM buffer (first 4 bytes of RAM transmit buffer are never used). Any additional bytes must be written/read with the RAM. Baud rate with no delay is about 160 kpbs.

Command	
Byte #	Description
0	Data Byte 3
1	Data Byte 2
2	Data Byte 1
3	Data Byte 0
4	Bit 7: ms Delay
	Bit 6: Hundred µs Delay
	Bits 5-4: XX
	Bit 3: SPI Mode D
	Bit 2: SPI Mode C
	Bit 1: SPI Mode B
	Bit 0: SPI Mode A
5	011XXX10 (SPI)
6	Number of bytes to write/read (1-18)
7	Bit 7: Control CS
	Bit 6: State of Active CS
	Bits 5-3: XXX
	Bits 2-0: D line to use as CS (0-7)
Response	
Byte #	Description
0	Data Byte 3
1	Data Byte 2
2	Data Byte 1
3	Data Byte 0
4	Bits 7-4: XXXX
	Bit 3: CSStateTris Error Flag
	Bit 2: SCKTris Error Flag
	Bit 1: MISOTris Error Flag
	Bit 0: MOSITris Error Flag
5	011XXX10 (Echo of byte 5)
6	Echo of byte 6 from Command
7	Echo of byte 7 from Command

LabJackPython Example

```
>>> import u12
>>> d = u12.U12()
open called
Writing: [0x0, 0x0, 0x0, 0x0, 0x0, 0x57, 0x0, 0x0]
Received: [0x57, 0x0, 0x0, 0x0, 0xff, 0xff, 0x0, 0x0]
>>> d.rawSPI([1,2,3,4], NumberOfBytesToWriteRead = 4)
Writing: [0x4, 0x3, 0x2, 0x1, 0x1, 0x62, 0x4, 0x0]
Received: [0x4, 0x3, 0x2, 0x1, 0x1, 0x62, 0x4, 0x0]
{
  'DataByte3': 4, 'DataByte2': 3, 'DataByte1': 2, 'DataByte0': 1,
  'ErrorFlags':
    <BitField object: [ CSStateTris Error Flag = 0 (0),
                        SCKTris Error Flag = 0 (0),
                        MISOTris Error Flag = 0 (0),
                        MOSITris Error Flag = 1 (1) ] >
}
```

5.15 - SHT1X

Sends and receives data from a SHT1X T/RH sensor from Sensirion.

Baud rate without delay is about 2 kpbs.

DATA is IO0, SCK is IO1

Enable is **not** handled by this function.

Command	
Byte #	Description
0	Data Byte 3
1	Data Byte 2
2	Data Byte 1
3	Data Byte 0
4	Bit 7: Wait for measurement ready signal
	Bit 6: Issue serial reset
	Bit 5: 1ms Delay
	Bit 4: 300µs Delay
	Bit 3: IO3 State
	Bit 2: IO2 State
	Bit 1: IO3 Tris (Direction?)
	Bit 0: IO2 Tris (Direction?)
5	011X1000 (SHT1X)
6	Number of bytes to write (0-4)
7	Number of bytes to read (0-4)
Response	
Byte #	Description
0	Data Byte 3
1	Data Byte 2
2	Data Byte 1
3	Data Byte 0
4	Bit 2: Serial reset error
	Bit 1: Measurement ready error
	Bit 0: Ack error
5	011X1000 (Echo of byte 5)
6	Echo of byte 6 from Command
7	Echo of byte 7 from Command

LabJackPython Example

```
>>> import u12
>>> d = u12.U12()
open called
Writing: [0x0, 0x0, 0x0, 0x0, 0x0, 0x57, 0x0, 0x0]
Received: [0x57, 0x0, 0x0, 0x0, 0xff, 0xff, 0x0, 0x0]
>>> results = d.rawSHT1X()
Writing: [0x0, 0x0, 0x0, 0x3, 0x8f, 0x68, 0x1, 0x3]
Received: [0x0, 0x66, 0x34, 0x19, 0x0, 0x68, 0x1, 0x3]
>>> print results
{
  'DataByte3': 0, 'DataByte2': 102, 'DataByte1': 52, 'DataByte0': 25,
  'ErrorFlags':
    <BitField object: [ Serial Reset Error Flag = 0 (0),
                        Measurement Ready Error Flag = 0 (0),
                        Ack Error Flag = 0 (0) ] >}
>>> tempC = (results['DataByte0'] * 256 ) + results['DataByte1']
>>> tempC = (tempC * 0.01) - 40
>>> print tempC
24.52
>>> results = d.rawSHT1X(Data = [5,0,0,0])
Writing: [0x0, 0x0, 0x0, 0x5, 0x8f, 0x68, 0x1, 0x3]
Received: [0x0, 0xaa, 0xae, 0x2, 0x0, 0x68, 0x1, 0x3]
>>> print results
{
  'DataByte3': 0, 'DataByte2': 170, 'DataByte1': 174, 'DataByte0': 2,
  'ErrorFlags':
    <BitField object: [ Serial Reset Error Flag = 0 (0),
                        Measurement Ready Error Flag = 0 (0),
                        Ack Error Flag = 0 (0) ] >}
>>> sorh = (results['DataByte0'] * 256 ) + results['DataByte1']
>>> rhlinear = (-0.0000028*sorh*sorh)+(0.0405*sorh)-4.0
>>> rh = ((tempC-25.0)*(0.01+(0.00008*sorh)))+rhlinear
>>> print rh
22.4341888
```

Appendix A - Specifications

Parameter	Conditions	Min	Typical	Max	Units
General					
USB Cable Length				3	meters
User Connection(s) Length	CE compliance			3	meters
Supply Current (1)			20		mA
Operating Temperature		-40		85	°C
Clock Error	~ 25 °C			±30	ppm
	0 to 70 °C			±50	ppm
	-40 to 85 °C			±100	ppm
+5 Volt Power Supply (+5V)					
Voltage (Vs) (2)	Self-Powered	4.5		5.25	volts
	Bus-Powered	4.1		5.25	volts
Output Current (2) (3)	Self-Powered	450		500	mA
	Bus-Powered	50		100	mA
Analog Inputs (AI0 - AI7)					
Input Range For Linear Operation	AIx to GND, SE	-10		10	volts
	AIx to GND, Diff.	-10		30	volts
Maximum Input Range	AIx to GND	-40		40	volts
Input Current (4)	Vin = +10 volts		70.1		µA
	Vin = 0 volts		-11.7		µA
	Vin = -10 volts		-93.5		µA
Resolution (No Missing Code)	C/R and Stream		12		bits
	Burst Diff. (5)		12		bits
	Burst SE (5)		11		bits
Offset	G = 1 to 20		±1 * G		bits
Absolute Accuracy	SE		±0.2	±0.4	% FS
	Diff.		±1		% FS
Noise	C/R and Stream		±1		bits
Integral Linearity Error			±1		bits
Differential Linearity Error			±0.5		bits
Repeatability			±1		bits
CAL Accuracy	CAL = 2.5 volts		±0.05	±0.25	%
CAL Current	Source			1	mA
	Sink	20	100		µA
Trigger Latency	Burst	25		50	µs
Trigger Pulse Width	Burst	40			µs
Analog Outputs (AO0 & AO1)					
Maximum Voltage (6)	No Load		Vs		volts
	At 1 mA		0.99 * Vs		volts
	At 5 mA		0.96 * Vs		volts
Source Impedance			20		Ω
Output Current	Each AO			30	mA
IO					
Low Level Input Voltage				0.8	volts
Hight Level Input Voltage		3		15	volts
Input Leakage Current (7)			±1		µA
Output Short-Circuit Current (8)	Output High		3.3		mA
Output Voltage (8)	No Load	Vs - 0.4	Vs		volts
	At 1 mA		Vs - 1.5		volts
D					
Low Level Input Voltage (9)	D0 - D12			0.8	volts
	D13 - D15			1	volts
Hight Level Input Voltage (9)	D0 - D12	2		Vs + 0.3	volts
	D13 - D15	4		Vs + 0.3	volts
Input Leakage Current (7)			±1		µA
Output Current (9)	Per Line			25	mA
	Total D0-D15			200	mA
Output Low Voltage				0.6	volts
Output Hight Voltage		Vs - 0.7			volts
CNT					
Low Voltage (10)		GND		1	volts
High Voltage (10)		3		15	volts
Schmitt Trigger Hysteresis			20-100		mV
Input Leakage Current (7)			±1		µA
Minimum High Time				500	ns
Minimum Low Time				500	ns
Maximum Input Frequency		1			MHz

(1) Current drawn by the LabJack through the USB. The status LED is responsible for 4-5 mA of this current.
(2) Self-powered would apply to USB hubs with a power supply, all known desktop computer USB hosts, and some notebook computer USB hosts. Bus-powered would apply to USB hubs without a power supply and some notebook computer USB hosts.
(3) This is the total current that can be sourced by +5V, analog outputs, and digital outputs.
(4) The input current at each analog input is a function of the voltage at that input (Vin) with respect to ground and can be calculated as: (8.181*Vin - 11.67) µA.
(5) Single-ended burst mode only returns even binary codes, and thus has a net resolution of 11 bits. In addition, extra noise in burst mode can reduce the effective resolution.
(6) Maximum analog output voltage is equal to the supply voltage at no load.
(7) Must also consider current due to 1 MΩ resistor to ground.
(8) The IO lines each have a 1500 ohm series resistor.
(9) These lines have no series resistor. It is up to the user to make sure the maximum voltages and currents are not exceeded.
(10) CNT is a Schmitt Trigger input.