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

5 - Low-level Function Reference [U3 Datasheet]

Low-Level Function Reference Overview

This section describes the low level functions of the U3. These are commands sent over USB directly to the processor on the U3.

The majority of Windows users will use the high-level UD driver rather than these low-level functions.

5.1 - General Protocol [U3 Datasheet]

Following is a description of the general U3 low-level communication protocol. There are two types of commands:

Normal: 1 command word plus 0-7 data words.

Extended: 3 command words plus 0-125 data words.

Normal commands have a smaller packet size and can be faster in some situations. Extended commands provide more commands, better error detection, and a larger maximum data payload.

Table 5.1-1. Normal command format

Byte			
0	Checksum8: Includes bytes 1-15		
1	Command Byte: DCCCCWWW		
		Bit 7: Destination Bit:	
			0 = Local,
			1 = Remote.
		Bits 6-3: Normal command number (0-14).	
		Bits 2-0: Number of data words.	
2-15	Data words.		

Table 5.1-2. Extended command format:

Byte			
0	Checksum8: Includes bytes 1-5		
1	Command Byte: D1111WWW		
		Bit 7: Destination Bit:	
			0 = Local,
			1 = Remote.
		Bits 6-3: 1111 specifies that this is an extended command	
		Bits 2-0: Used with some commands.	
2	Number of data words.		
3	Extended command number.		
4	Checksum16 (LSB)		
5	Checksum16 (MSB)		
6-255	Data words.		

Checksum calculations:

All checksums are a “1’s complement checksum”. Both the 8-bit and 16-bit checksum are unsigned. Sum all applicable bytes in an accumulator, 1 at a time. Each time another byte is added, check for overflow (carry bit), and if true add one to the accumulator.

In a high-level language, do the following for the 8-bit normal command checksum:

1. Get the subarray consisting of bytes 1 and up.
2. Convert bytes to U16 and sum into a U16 accumulator.
3. Divide by 2^8 and sum the quotient and remainder.
4. Divide by 2^8 and sum the quotient and remainder.

In a high-level language, do the following for an extended command 16-bit checksum:

1. Get the subarray consisting of bytes 6 and up.
2. Convert bytes to U16 and sum into a U16 accumulator (can't overflow).

Then do the following for the 8-bit extended checksum:

1. Get the subarray consisting of bytes 1 through 5.
2. Convert bytes to U16 and sum into a U16 accumulator.
3. Divide by 2^8 and sum the quotient and remainder.
4. Divide by 2^8 and sum the quotient and remainder.

Destination bit:

This bit specifies whether the command is destined for the local or remote target. This bit is ignored on the U3.

Multi-byte parameters:

In the following function definitions there are various multi-byte parameters. The least significant byte of the parameter will always be found at the lowest byte number. For instance, bytes 10 through 13 of CommConfig are the IP address which is 4 bytes long. Byte 10 is the least significant byte (LSB), and byte 13 is the most significant byte (MSB).

Masks:

Some functions have mask parameters. The WriteMask found in some functions specifies which parameters are to be written. If a bit is 1, that parameter will be updated with the new passed value. If a bit is 0, the parameter is not changed and only a read is performed.

The AINMask found in some functions specifies which analog inputs are acquired. This is a 16-bit parameter where each bit corresponds to AIN0-AIN15. If a bit is 1, that channel will be acquired.

The digital I/O masks, such as FIOMask, specify that the passed value for direction and state are updated if a bit 1. If a bit of the mask is 0 only a read is performed on that bit of I/O.

Binary Encoded Parameters:

Many parameters in the following functions use specific bits within a single integer parameter to write/read specific information. In particular, most digital I/O parameters contain the information for each bit of I/O in one integer, where each bit of I/O corresponds to the same bit in the parameter (e.g. the direction of FIO0 is set in bit 0 of parameter FIODir). For instance, in the function ControlConfig, the parameterFIODir is a single byte (8 bits) that writes/reads the direction of each of the 8 FIO lines:

- if FIODir is 0, all FIO lines are input,
- if FIODir is 1 (2^0), FIO0 is output, FIO1-FIO7 are input,
- if FIODir is 5 ($2^0 + 2^2$), FIO0 and FIO2 are output, all other FIO lines are input,
- if FIODir is 255 ($2^0 + \ldots + 2^7$), FIO0-FIO7 are output.

5.2 - Low-Level Functions [U3 Datasheet]

For usage examples of most low-level functions, seeLabJackPython's u3.py.

5.2.1 - Bad Checksum [U3 Datasheet]

If the processor detects a bad checksum in any command, the following 2-byte normal response will be sent and nothing further will be done.

Response:	
Byte	
0	0xB8
1	0xB8

5.2.2 - ConfigU3 [U3 Datasheet]

Writes and reads various configuration settings. Although this function has many of the same parameters as other functions, most parameters in this

case are affecting the power-up values, not the current values. There is a hardware method to restore bytes 9-20 to the factory default value of 0x00: Power up the U3 with a short from FIO6 to SPC (FIO2 to SCL on U3 1.20/1.21), then remove the jumper and power cycle the device again.

If WriteMask is nonzero, some or all default values are written to flash. The U3 flash has a rated endurance of at least 20000 writes, which is plenty for reasonable operation, but if this function is called in a high-speed loop with a nonzero WriteMask, the flash could eventually be damaged.

Note: If the stream is running, you cannot update any of the values (WriteMask must equal 0).

Table 5.2.2-1. ConfigU3 Command Response

Command:		
Byte		
0	Checksum8	
1	0xF8	
2	0×0A	
3	0×08	
4	Checksum16 (LSB)	
5	Checksum16 (MSB)	
6	WriteMask0	
		Bit 5: CompatibilityOptions Defaults
		Bit 4: TimerClockConfig and Divisor Defaults
		Bit 3: LocalID
		Bit 2: DAC Defaults
		Bit 1: Digital I/O Defaults
		Bit 0: Reserved
7	WriteMask1 (Reserved)	
8	LocalID	
9	TimerCounterConfig	
		Bits 4-7: TimerCounterPinOffset
		Bit 3: Enable Counter1
		Bit 2: Enable Counter0
		Bit 0-1: Number of timers enabled
10	FIOAnalog	
11	FIODirection	
12	FIOState	
13	EIOAnalog	
14	EIODirection	
15	EIOState	
16	CIODirection	
17	CIOState	
18	DAC1Enable	
19	SWDT DAC0 RESPONSE	
20	SWDT DAC1 RESPONSE	
21	TimerClockConfig	
22	TimerClockDivisor (0 = ÷256)	
23	CompatibilityOptions	
24	0×00	
25	0×00	
Response:		
Byte		
0	Checksum8	
1	0xF8	
2	0×10	
3	0×08	
4	Checksum16 (LSB)	
5	Checksum16 (MSB)	
6	Errorcode	
7	Reserved	
8	Reserved	
9-10	FirmwareVersion	
11-12	BootloaderVersion	
13-14	HardwareVersion	
15-18	SerialNumber	
19-20	ProductID	
21	LocalID	
22	TimerCounterMask	
23	FIOAnalog	

24	FIODirection	
25	FIOState	
26	EIOAnalog	
27	EIODirection	
28	EIOState	
29	CIODirection	
30	CIOState	
31	DAC1Enable	
32	DAC0	
33	DAC1	
34	TimerClockConfig	
35	TimerClockDivisor (0 = ÷256)	
36	CompatibilityOptions	
37	VersionInfo	

- **WriteMask**: Has bits that determine which, if any, of the parameters will be written to flash as the reset defaults. If a bit is 1, that parameter will be updated with the new passed value. If a bit is 0, the parameter is not changed and only a read is performed. Note that reads return reset defaults, not necessarily current values (except for LocalID). For instance, the value returned by FIODirection is the directions at reset, not necessarily the current directions.
- **LocalID**: If the WriteMask bit 3 is set, the value passed become the default value, meaning it is written to flash and used at reset. This is a user-configurable ID that can be used to identify a specific LabJack. The return value of this parameter is the current value and the power-up default value.
- **TimerCounterConfig**: If the WriteMask bit 1 is set, the value passed becomes the default value, meaning it is written to flash and used at reset. The return value of this parameter is a read of the power-up default. See Section 5.2.3.
- **FIO/EIO/CIO**: If the WriteMask bit 1 is set, the values passed become the default values, meaning they are written to flash and used at reset. Regardless of the mask bit, this function has no effect on the current settings. The return value of these parameters are a read of the power-up defaults.
- **DAC**: If the WriteMask bit 2 is set, the values passed become the default values, meaning they are written to flash and used at reset. Regardless of the mask bit, this function has no effect on the current settings. The return values of these parameters are a read of the power-up defaults.
- **TimerClockConfig & TimerClockDivisor**: If the WriteMask bit 4 is set, the values passed become the default values, meaning they are written to flash and used at reset. The return values of these parameters are a read of the power-up defaults. See Section 5.2.4.
- **CompatibilityOptions**: If the WriteMask bit 5 is set, the value passed becomes the default value, meaning it is written to flash and used at reset. The return value of this parameter is a read of the power-up default. If bit 0 is set, Timer Counter Pin Offset errors are ignored. If bit 1 is set, all DAC operations will use 8-bit mode rather than 10-bit mode. Once this value has been changed the U3 will need to be restarted before the new setting will take affect.
- **FirmwareVersion**: Fixed parameter specifies the version number of the main firmware. A firmware upgrade will generally cause this parameter to change. The lower byte is the integer portion of the version and the higher byte is the fractional portion of the version.
- **BootloaderVersion**: Fixed parameter specifies the version number of the bootloader. The lower byte is the integer portion of the version and the higher byte is the fractional portion of the version.
- **HardwareVersion**: Fixed parameter specifies the version number of the hardware. The lower byte is the integer portion of the version and the higher byte is the fractional portion of the version.
- **SerialNumber**: Fixed parameter that is unique for every LabJack.
- **ProductID**: (3) Fixed parameter identifies this LabJack as a U3.
- **VersionInfo**: Bit 0 specifies U3B. Bit 1 specifies U3C and if set then bit 4 specifies -HV version.

5.2.3 - ConfigIO [U3 Datasheet]

Writes and reads the current IO configuration.

Table 5.2.3-1. ConfigIO Command Response

Command:		
Byte		
0	Checksum8	
1	0xF8	
2	0×03	
3	0×0B	
4	Checksum16 (LSB)	
5	Checksum16 (MSB)	
6	WriteMask	
		Bit 5: Write UART Related settings
		Bit 4: Reserved, Pass 0
		Bit 3: EIOAnalog
		Bit 2: FIOAnalog
		Bit 1: DAC1Enable
		Bit 0 : TimerCounterConfig

7	Reserved	
8	TimerCounterConfig	
		Bits 4-7: TimerCounterPinOffset
		Bit 3: Enable Counter1
		Bit 2: Enable Counter0
		Bits 0-1: Number of timers enabled
9	DAC1Enable (ignored on hardware rev 1.30+)	
		Bit 2: Assign UART Pins (HW 1.30 only)
		Bit 1: Reserved, Pass 0
		Bit 0: Enable DAC1
10	FIOAnalog	
11	EIOAnalog	
Response:		
Byte		
0	Checksum8	
1	0xF8	
2	0x03	
3	0x0B	
4	Checksum16 (LSB)	
5	Checksum16 (MSB)	
6	Errorcode	
7	Reserved	
8	TimerCounterConfig	
9	DAC1Enable	
10	FIOAnalog	
11	EIOAnalog	

- **WriteMask**: Has bits that determine which, if any, of the parameters will be written.
- **TimerCounterConfig**: Used to enable/disable timers and counters. Timers/counters will be assigned to IO pins starting with FIO0 plus TimerCounterPinOffset (4-8 only starting with hardware revision 1.30). Timer0 takes the first IO pin, then Timer1, Counter0, and Counter1. Whenever this function is called and timers are enabled, the timers are initialized to mode 10, so the desired timer mode must always be specified after every call to this function. Note that Counter0 is not available when using a timer clock base that supports a timer clock divisor (TimerClockBase = 3-6).
- **Assign UART Pins**: On hardware 1.30 setting this bit will assign IO lines to theUART module. This setting will be ignored unless theUART write bit is set in the WriteMask byte.
- **DAC1Enable**: On hardware revisions 1.20/1.21 only, bit 0 enables DAC1. When DAC1 is disabled, it outputs a constant voltage of 1.5 times the internal Vref (~2.44 volts). When DAC1 is enabled, the internal Vref is not available for the analog inputs and Vreg (~3.3 volts) is used as the AIN reference. Starting with hardware revision 1.30, DAC1 is always enabled.
- **FIOAnalog**: Each bit determines whether that bit ofFIO is analog input (=1) or digital I/O (=0).
- **EIOAnalog**: Each bit determines whether that bit ofEIO is analog input (=1) or digital I/O (=0).

LabJackPython Example

```
>>> import u3
>>> d = u3.U3()
>>> d.debug = True
'''Assign timer0 to FIO6, set FIO4, FIO5 as analog input, and set EIO1, EIO2 as analog input'''
>>> d.configIO(TimerCounterPinOffset = 6, NumberOfTimersEnabled = 1, FIOAnalog = 0x30, EIOAnalog = 0x03)
Sent:  [0xa8, 0xf8, 0x3, 0xb, 0xa1, 0x0, 0xd, 0x0, 0x61, 0x0, 0x30, 0x3]
Result:  [0x9b, 0xf8, 0x3, 0xb, 0x94, 0x0, 0x0, 0x0, 0x61, 0x0, 0x30, 0x3]
{'NumberOfTimersEnabled': 1, 'TimerCounterPinOffset': 6, 'DAC1Enable': 0, 'FIOAnalog': 48, 'EIOAnalog': 3, 'TimerCounterConfig': 97, 'EnableCounter1': False, 'EnableCounter0': False}
```

5.2.4 - ConfigTimerClock [U3 Datasheet]

Writes and read the timer clock configuration.

Table 5.2.4-1. ConfigTimerClock Command Response

Command:			
Byte			
0	Checksum8		
1	0xF8		
2	0x02		

3	0x0A	
4	Checksum16 (LSB)	
5	Checksum16 (MSB)	
6	Reserved	
7	Reserved	
8	TimerClockConfig	
		Bit 7: Configure the clock
		Bits 2-0: TimerClockBase
		b000: 4 MHz
		b001: 12 MHz
		b010: 48 MHz (Default)
		b011: 1 MHz/Divisor
		b100: 4 MHz/Divisor
		b101: 12 MHz/Divisor
		b110: 48 MHz/Divisor
9	TimerClockDivisor (0 = ÷256)	
Response:		
Byte		
0	Checksum8	
1	0xF8	
2	0×02	
3	0×0A	
4	Checksum16 (LSB)	
5	Checksum16 (MSB)	
6	Errorcode	
7	Reserved	
8	TimerClockConfig	
9	TimerClockDivisor (0 = ÷256)	

- **TimerClockConfig**: Bit 7 determines whether the new TimerClockBase and TimerClockDivisor are written, or if just a read is performed. Bits 0-2 specify the TimerClockBase. If TimerClockBase is 3-6, then Counter0 is not available.
- **TimerClockDivisor**: The base timer clock is divided by this value, or divided by 256 if this value is 0. Only applies if TimerClockBase is 3-6.

5.2.5 - Feedback [U3 Datasheet]

Feedback Overview

A flexible function that handles all command/response functionality. One or moreIOTypes are used to perform a single write/read or multiple writes/reads.

Note that the general protocol described inSection 5.1 defines byte 2 of an extended command as the number of data words, which is the number of words in a packet beyond the first 3 (a word is 2 bytes). Also note that the overall size of a packet must be an even number of bytes, so in this case an extra 0x00 is added to the end of the command and/or response if needed to accomplish this.

Since this command has a flexible size, byte 2 will vary. For instance, if a singleIOType of PortStateRead (d26) is passed, byte 2 would be equal to 1 for the command and 3 for the response. If a single IOType of LED (d9) is passed, an extra 0 must be added to the command to make the packet have an even number of bytes, and byte 2 would be equal to 2. The response would also need an extra 0 to be even, and byte 2 would be equal to 2.

Table 5.2.5-1. Feedback Command Response

Command:	
Byte	
0	Checksum8
1	0xF8
2	0.5 + Number of Data Words (IOTypes and Data)
3	0x00
4	Checksum16 (LSB)
5	Checksum16 (MSB)

6	Echo
7-63	IOTypes and Data
Response:	
Byte	
0	Checksum8
1	0xF8
2	1.5 + Number of Data Words (If Errorcode = 0)
3	0x00
4	Checksum16 (LSB)
5	Checksum16 (MSB)
6	Errorcode
7	ErrorFrame
8	Echo
9-63	Data

- **IOTypes & Data**: One or more IOTypes can be passed in a single command, up to the maximum packet size. More info about the available IOTypes is below. In the outgoing command each IOType is passed and accompanied by 0 or more data bytes. In the incoming response, only data bytes are returned without the IOTypes.
- **Echo**: This byte is simply echoed back in the response. A host application might pass sequential numbers to ensure the responses are in order and associated with the proper command.
- **ErrorFrame**: If Errorcode is not zero, this parameter indicates which IOType caused the error. For instance, if the 3rd passed IOType caused the error, the ErrorFrame would be equal to 3. Also note that data is only returned for IOTypes before the one that caused the error, so if any IOType causes an error the overall function response will have less bytes than expected.

Table 5.2.5-2.

Name	IOType (dec)	WriteBytes	ReadBytes
AIN	1	3	2
WaitShort	5	2	0
WaitLong	6	2	0
LED	9	2	0
BitStateRead	10	2	1
BitStateWrite	11	2	0
BitDirRead	12	2	1
BitDirWrite	13	2	0
PortStateRead	26	1	3
PortStateWrite	27	7	0
PortDirRead	28	1	3
PortDirWrite	29	7	0
DAC0 (8-bit)	34	2	0
DAC1 (8-bit)	35	2	0
DAC0 (16-bit)	38	3	0
DAC1 (16-bit)	39	3	0
Timer0	42	4	4
Timer0Config	43	4	0
Timer1	44	4	4
Timer1Config	45	4	0
Counter0	54	2	4
Counter1	55	2	4
Buzzer	63	6	0

5.2.5.1 - AIN: IOType = 1 [U3 Datasheet]

Table 5.2.5.1-1.

AIN, 3 Command Bytes:	
0	IOType = 1
1	Bits 4-0: Positive Channel
	Bit 6: LongSettling
	Bit 7: QuickSample
2	Negative Channel
2 Response Bytes:	
0	AIN LSB
1	AIN MSB

This IOType returns a single analog input reading.

Note: Do **not** use this IO type while streaming.

- **Positive Channel**: 0-15 for AIN0-AIN15, 30 for temp sensor, or 31 for Vreg. Note that AIN0-AIN7 appear on FIO0-FIO7, and AIN8-AIN15 appear on EIO0-EIO7.
- **LongSettling**: If this bit is set, additional settling time is added between the multiplexer configuration and the analog to digital conversion.
- **QuickSample**: If this bit is set, a faster analog input conversion is done, at the expense of increased noise. If bits 6 & 7 are both set, then the entire byte is a channel number (for digital I/O, timers, and counters).
- **Negative Channel**: 0-15 for AIN0-AIN15, 30 for Vref, or 31 for single-ended. Note that AIN0-AIN7 appear on FIO0-FIO7, and AIN8-AIN15 appear on EIO0-EIO7.
- **AIN LSB & MSB**: Analog input reading is returned justified as a 16-bit value (always unsigned).

Support - Python U3 Get Feedback Example

LabJackPython example session

Automatically extracted from u3.py. Debugging turned on to show the bytes sent and received.

```
Analog Input Feedback command

  specify the positive and negative channels to use
  (0-16, 30 and 31 are possible)
  also specify whether to turn on longSettle or quick Sample

  returns 16-bit signed int sample

  >>> import u3
  >>> d = u3.U3()
  >>> d.debug = True
  >>> d.getFeedback(u3.AIN(PositiveChannel = 0, NegativeChannel=31, LongSettling=False, QuickSample=False))
  Sent:  [0x1b, 0xf8, 0x2, 0x0, 0x20, 0x0, 0x0, 0x1, 0x0, 0x1f]
  Response:  [0xab, 0xf8, 0x3, 0x0, 0xaf, 0x0, 0x0, 0x0, 0x0, 0x20, 0x8f, 0x0]
  [36640]
```

u3-feedback-AIN.txt hosted with ❤ by **GitHub** **view raw**

5.2.5.2 - WaitShort: IOType=5 [U3 Datasheet]

Table 5.2.5.2-1.

WaitShort, 2 Command Bytes:	
0	IOType = 5
1	Time (*128 us)
0 Response Bytes:	

This IOType provides a way to add a delay during execution of the Feedback function. The typical use would be putting this IOType in between IOTypes that set a digital output line high and low, thus providing a simple way to create a pulse. Note that this IOType uses the same internal timer as stream mode, so cannot be used while streaming.

- **Time**: This value (0-255) is multiplied by 128 microseconds to determine the delay (U3C = 128 us, U3B = 64 us, U3A = 128 us).

Support - U3 Python Wait Short Example

LabJackPython example session

Automatically extracted from u3.py. Debugging turned on to show the bytes sent and received.

```
WaitShort Feedback command

    specify the number of 128us time increments to wait

    >>> import u3
    >>> d = u3.U3()
    >>> d.debug = True
    >>> d.getFeedback(u3.WaitShort(Time = 9))
    Sent:  [0x9, 0xf8, 0x2, 0x0, 0xe, 0x0, 0x0, 0x5, 0x9, 0x0]
    Response:  [0xfa, 0xf8, 0x2, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0]
    [None]
```

u3-feedback-WaitShort.txt hosted with ❤ by **GitHub** **view raw**

5.2.5.3 - WaitLong: IOType=6 [U3 Datasheet]

Log in or register to post comments

Table 5.2.5.3-1.

WaitLong, 2 Command Bytes:	
0	IOType = 6
1	Time (*16 ms)
0 Response Bytes:	

This IOType provides a way to add a delay during execution of the Feedback function. The typical use would be putting this IOType in between IOTypes that set a digital output line high and low, thus providing a simple way to create a pulse. Note that this IOType uses the same internal timer as stream mode, so cannot be used while streaming.

- **Time**: This value (0-255) is multiplied by 16384 microseconds (U3C = 16384 us, U3B = 16384 us, U3A = 32768 us) to determine the delay.

LabJackPython example session

Automatically extracted from u3.py. Debugging turned on to show the bytes sent and received.

```
WaitLong Feedback command

    specify the number of 16ms time increments to wait

    >>> import u3
    >>> d = u3.U3()
    >>> d.debug = True
    >>> d.getFeedback(u3.WaitLong(Time = 70))
    Sent: [0x47, 0xf8, 0x2, 0x0, 0x4c, 0x0, 0x0, 0x6, 0x46, 0x0]
    Response: [0xfa, 0xf8, 0x2, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0]
    [None]
```

5.2.5.4 - LED: IOType=9 [U3 Datasheet]

Log in or register to post comments

Table 5.2.5.4-1.

LED, 2 Command Bytes:	
0	IOType = 9
1	State
0 Response Bytes:	

This IOType simply turns the status LED on or off.

- **State**: 1=On, 0=Off.

u3 - LED code example

LabJackPython example session

Automatically extracted from u3.py. Debugging turned on to show the bytes sent and received.

```
LED Toggle

    specify whether the LED should be on or off by truth value

    1 or True = On, 0 or False = Off

    >>> import u3
    >>> d = u3.U3()
    >>> d.debug = True
    >>> d.getFeedback(u3.LED(State = False))
    Sent:  [0x4, 0xf8, 0x2, 0x0, 0x9, 0x0, 0x0, 0x9, 0x0, 0x0]
    Response:  [0xfa, 0xf8, 0x2, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0]
    [None]
    >>> d.getFeedback(u3.LED(State = True))
    Sent:  [0x5, 0xf8, 0x2, 0x0, 0xa, 0x0, 0x0, 0x9, 0x1, 0x0]
    Response:  [0xfa, 0xf8, 0x2, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0]
    [None]
```

u3-feedback-LED.txt hosted with ❤ by **GitHub** **view raw**

5.2.5.5 - BitStateRead: IOType=10 [U3 Datasheet]

Table 5.2.5.5-1. BitStateRead Command Response

BitStateRead, 2 Command Bytes:	
0	IOType = 10
1	Bits 0-4: IONumber
1 Response Byte:	
0	Bit 0: State

This IOType reads the state of a single bit of digital I/O. The direction setting of the IO will not be changed. When the digital IO is set to output the actual state of the line is read. Only lines configured as digital (not analog) return valid readings.

- **IO Number**: 0-7=FIO, 8-15=EIO, or 16-19=CIO.
- **State**: 1=High, 0=Low.

LabJackPython example session

Automatically extracted from u3.py. Debugging turned on to show the bytes sent and received.

```
BitStateRead Feedback command

    read the state of a single bit of digital I/O.  Only digital
    lines return valid readings.

    IONumber: 0-7=FIO, 8-15=EIO, 16-19=CIO
    return 0 or 1

    >>> import u3
    >>> d = u3.U3()
    >>> d.debug = True
    >>> d.getFeedback(u3.BitStateRead(IONumber = 5))
    Sent:  [0xa, 0xf8, 0x2, 0x0, 0xf, 0x0, 0x0, 0xa, 0x5, 0x0]
    Response:  [0xfb, 0xf8, 0x2, 0x0, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1]
    [1]
```

u3-feedback-BitStateRead.txt hosted with ❤ by **GitHub** **view raw**

5.2.5.6 - BitStateWrite: IOType=11 [U3 Datasheet]

Table 5.2.5.6-1.

BitStateWrite, 2 Command Bytes:	
0	IOType = 11
1	Bits 0-4: IO Number
	Bit 7: State
0 Response Bytes:	

This IOType writes the state of a single bit of digital I/O. The direction of the specified line is forced to output.

- **IO Number**: 0-7=FIO, 8-15=EIO, or 16-19=CIO.
- **State**: 1=High, 0=Low.

u3 - bitstatewrite code example

LabJackPython example session

Automatically extracted from u3.py. Debugging turned on to show the bytes sent and received.

```
BitStateWrite Feedback command

  write a single bit of digital I/O.  The direction of the
  specified line is forced to output.

  IONumber: 0-7=FIO, 8-15=EIO, 16-19=CIO
  State: 0 or 1

  >>> import u3
  >>> d = u3.U3()
  >>> d.debug = True
  >>> d.getFeedback(u3.BitStateWrite(IONumber = 5, State = 0))
  Sent:  [0xb, 0xf8, 0x2, 0x0, 0x10, 0x0, 0x0, 0xb, 0x5, 0x0]
  Response:  [0xfa, 0xf8, 0x2, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0]
  [None]
```

u3-feedback-BitStateWrite.txt hosted with ❤ by **GitHub** **view raw**

5.2.5.7 - BitDirRead: IOType=12 [U3 Datasheet]

Log in or register to post comments

Table 5.2.5.7-1.

BitDirRead, 2 Command Bytes:	
0	IOType = 12
1	Bits 0-4: IO Number
1 Response Byte:	
0	Bit 0: Direction

This IOType reads the direction of a single bit of digital I/O. This is the digital direction only, and does not provide any information as to whether the line is configured as digital or analog.

- **IO Number**: 0-7=FIO, 8-15=EIO, or 16-19=CIO.
- **Direction**: 1=Output, 0=Input.

u3 - bitdirread code example

LabJackPython example session

Automatically extracted from u3.py. Debugging turned on to show the bytes sent and received.

```
Read the digital direction of one I/O

  IONumber: 0-7=FIO, 8-15=EIO, 16-19=CIO
  returns 1 = Output, 0 = Input

  >>> import u3
  >>> d = u3.U3()
  >>> d.debug = True
  >>> d.getFeedback(u3.BitDirRead(IONumber = 5))
  Sent:  [0xc, 0xf8, 0x2, 0x0, 0x11, 0x0, 0x0, 0xc, 0x5, 0x0]
  Response:  [0xfb, 0xf8, 0x2, 0x0, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1]
  [1]
```

u3-feedback-BitDirRead.txt hosted with ❤ by **GitHub** **view raw**

5.2.5.8 - BitDirWrite: IOType=13 [U3 Datasheet]

Log in or register to post comments

Table 5.2.5.8-1. BitDirWrite Command Response

BitDirRead, 2 Command Bytes:	
0	IOType = 13
1	Bits 0-4: IO Number
	Bit 7: Direction
0 Response Bytes:	

This IOType writes the direction of a single bit of digital I/O.

- **IO Number**: 0-7=FIO, 8-15=EIO, or 16-19=CIO.
- **Direction**: 1=Output, 0=Input.

u3 - bitdirwrite code example

LabJackPython example session

Automatically extracted from u3.py. Debugging turned on to show the bytes sent and received.

```
BitDirWrite Feedback command

  Set the digital direction of one I/O

  IONumber: 0-7=FIO, 8-15=EIO, 16-19=CIO
  Direction: 1 = Output, 0 = Input

  >>> import u3
  >>> d = u3.U3()
  >>> d.debug = True
  >>> d.getFeedback(u3.BitDirWrite(IONumber = 5, Direction = 0))
  Sent:  [0xd, 0xf8, 0x2, 0x0, 0x12, 0x0, 0x0, 0xd, 0x5, 0x0]
  Response:  [0xfa, 0xf8, 0x2, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0]
  [None]
```

u3-feedback-BitDirWrite.txt hosted with ❤ by **GitHub** **view raw**

5.2.5.9 - PortStateRead: IOType=26 [U3 Datasheet]

Log in or register to post comments

Table 5.2.5.9-1.

PortStateRead, 1 Command Byte:	
0	IOType = 26
3 Response Bytes:	
0-2	State

This IOType reads the state of all digital I/O, where 0-7=FIO, 8-15=EIO, and 16-19=CIO. Only lines configured as digital (not analog) return valid readings.

- **State**: Each bit of this value corresponds to the specified bit of I/O such that 1=High and 0=Low. If all are low, State=d0. If all 20 standard digital I/O are high, State=d1048575. If FIO0-FIO2 are high, EIO0-EIO2 are high, CIO0 are high, and all other I/O are low (b00000001000001111000001111), State=d67335.

u3 - portstateread code example

LabJackPython example session

Automatically extracted from u3.py. Debugging turned on to show the bytes sent and received.

```
PortStateRead Feedback command
    Reads the state of all digital I/O.

    >>> import u3
    >>> d = u3.U3()
    >>> d.debug = True
    >>> d.getFeedback(u3.PortStateRead())
    Sent:  [0x14, 0xf8, 0x1, 0x0, 0x1a, 0x0, 0x0, 0x1a]
    Response:  [0xeb, 0xf8, 0x3, 0x0, 0xee, 0x1, 0x0, 0x0, 0x0, 0xe0, 0xff, 0xf]
    [{'CIO': 15, 'FIO': 224, 'EIO': 255}]
```

u3-feedback-PortStateRead.txt hosted with ❤ by **GitHub** **view raw**

5.2.5.10 - PortStateWrite: IOType=27 [U3 Datasheet]

Log in or register to post comments

Table 5.2.5.10-1.

PortStateWrite, 7 Command Bytes:	
0	IOType = 27
1-3	WriteMask
4-6	State
0 Response Bytes:	

This IOType writes the state of all digital I/O, where 0-7=FIO, 8-15=EIO, and 16-19=CIO. The direction of the selected lines is forced to output.

The firmware does the actual updates in the following order: FIO4-5, FIO6-7, FIO0-3, EIO0-5, EIO6-7, CIO0, CIO1, CIO2, CIO3. These all happen within 1 microsecond or so, but EIO0-EIO5, for example, are all updated at the exact same time.

- **WriteMask**: Each bit specifies whether to update the corresponding bit of I/O.
- **State**: Each bit of this value corresponds to the specified bit of I/O such that 1=High and 0=Low. To set all low, State=d0. To set all 20 standard digital I/O high, State=d1048575. To set FIO0-FIO2 high, EIO0-EIO2 high, CIO0 high, and all other I/O low (b00000001000001111000001111), State=d67335.

U3 - portstatewrite code example

LabJackPython example session

Automatically extracted from u3.py. Debugging turned on to show the bytes sent and received.

```
PortStateWrite Feedback command

    State: A list of 3 bytes representing FIO, EIO, CIO
    WriteMask: A list of 3 bytes, representing which to update.
            The Default is all ones.

    >>> import u3
    >>> d = u3.U3()
    >>> d.debug = True
    >>> d.getFeedback(u3.PortStateWrite(State = [0xab, 0xcd, 0xef], WriteMask = [0xff, 0xff, 0xff]))
    Sent:  [0x81, 0xf8, 0x4, 0x0, 0x7f, 0x5, 0x0, 0x1b, 0xff, 0xff, 0xff, 0xab, 0xcd, 0xef]
    Response:  [0xfa, 0xf8, 0x2, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0]
    [None]
```

u3-feedback-PortStateWrite.txt hosted with ❤ by **GitHub** **view raw**

5.2.5.11 - PortDirRead: IOType=28 [U3 Datasheet]

Table 5.2.5.11-1. PortDirRead Command Response

PortDirRead, 1 Command Byte:	
0	IOType = 28
3 Response Bytes:	
0-2	Direction

This IOType reads the directions of all digital I/O, where 0-7=FIO, 8-15=EIO, and 16-19=CIO. These are the digital directions only, and do not provide any information as to whether the lines are configured as digital or analog.

- **Direction**: Each bit of this value corresponds to the specified bit of I/O such that 1=Output and 0=Input. If all are input, Direction=d0. If all 20 standard digital I/O are output, Direction=d1048575. If FIO0-FIO2 are output, EIO0-EIO2 are output, CIO0 are output, and all other I/O are input (b00000001000001111000000111), Direction=d67335.

LabJackPython example session

Automatically extracted from u3.py. Debugging turned on to show the bytes sent and received.

```
PortDirRead Feedback command
    Reads the direction of all digital I/O.

    >>> import u3
    >>> d = u3.U3()
    >>> d.debug = True
    >>> d.getFeedback(u3.PortDirRead())
    Sent:  [0x16, 0xf8, 0x1, 0x0, 0x1c, 0x0, 0x0, 0x1c]
    Response:  [0xfb, 0xf8, 0x3, 0x0, 0xfe, 0x1, 0x0, 0x0, 0x0, 0xf0, 0xff, 0xf]
    [{'CIO': 15, 'FIO': 240, 'EIO': 255}]
```

u3-feedback-PortDirRead.txt hosted with ❤ by **GitHub** **view raw**

5.2.5.12 - PortDirWrite: IOType=29 [U3 Datasheet]

Table 5.2.5.12-1. PortDirWrite Command Response

PortStateWrite, 7 Command Bytes:	
0	IOType = 29
1-3	WriteMask
4-6	Direction
0 Response Bytes:	

This IOType writes the direction of all digital I/O, where 0-7=FIO, 8-15=EIO, and 16-19=CIO. Note that the desired lines must be configured as digital (not analog).

- **WriteMask**: Each bit specifies whether to update the corresponding bit of I/O.
- **Direction**: Each bit of this value corresponds to the specified bit of I/O such that 1=Output and 0=Input. To configure all as input, Direction=d0. For all 20 standard digital I/O as output, Direction=d1048575. To configure FIO0-FIO2 as output, EIO0-EIO2 as output, CIO0 as output, and all other I/O as input (b00000001000001110000111), Direction=d67335.

u3 - portdirwrite code example

LabJackPython example session

Automatically extracted from u3.py. Debugging turned on to show the bytes sent and received.

```
PortDirWrite Feedback command

    Direction: A list of 3 bytes representing FIO, EIO, CIO
    WriteMask: A list of 3 bytes, representing which to update. Default is all ones.


    >>> import u3
    >>> d = u3.U3()
    >>> d.debug = True
    >>> d.getFeedback(u3.PortDirWrite(Direction = [0xaa, 0xcc, 0xff], WriteMask = [0xff, 0xff, 0xff]))
    Sent:  [0x91, 0xf8, 0x4, 0x0, 0x8f, 0x5, 0x0, 0x1d, 0xff, 0xff, 0xff, 0xaa, 0xcc, 0xff]
    Response:  [0xfa, 0xf8, 0x2, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0]
    [None]
```

u3-feedback-PortDirWrite.txt hosted with ❤ by **GitHub** **view raw**

5.2.5.13 - DAC# (8-bit): IOType=34,35 [U3 Datasheet]

Table 5.2.5.13-1. DAC 8-bit Command Response

DAC# (8-bit), 2 Command Bytes:	
0	IOType = 34, 35
1	Value
0 Response Bytes:	

This IOType controls a single analog output.

- **Value**: 0=Minimum, 255=Maximum.

U3 - DAC stuff code examples

LabJackPython example session

Automatically extracted from u3.py. Debugging turned on to show the bytes sent and received.

```
8-bit DAC Feedback command

    Controls a single analog output

    Dac: 0 or 1
    Value: 0-255

    >>> import u3
    >>> d = u3.U3()
    >>> d.debug = True
    >>> d.getFeedback(u3.DAC8(Dac = 0, Value = 0x55))
    Sent:  [0x72, 0xf8, 0x2, 0x0, 0x77, 0x0, 0x0, 0x22, 0x55, 0x0]
    Response:  [0xfa, 0xf8, 0x2, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0]
    [None]
```

u3-feedback-DAC8.txt hosted with ❤ by **GitHub** **view raw**

```
8-bit DAC Feedback command for DAC0

    Controls DAC0 in 8-bit mode.

    Value: 0-255

    >>> import u3
    >>> d = u3.U3()
    >>> d.debug = True
    >>> d.getFeedback(u3.DAC0_8(Value = 0x33))
    Sent:  [0x50, 0xf8, 0x2, 0x0, 0x55, 0x0, 0x0, 0x22, 0x33, 0x0]
    Response:  [0xfa, 0xf8, 0x2, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0]
    [None]
```

u3-feedback-DAC0_8.txt hosted with ❤ by **GitHub** **view raw**

```
8-bit DAC Feedback command for DAC1

    Controls DAC1 in 8-bit mode.

    Value: 0-255

    >>> import u3
    >>> d = u3.U3()
    >>> d.debug = True
    >>> d.getFeedback(u3.DAC1_8(Value = 0x22))
    Sent:  [0x40, 0xf8, 0x2, 0x0, 0x45, 0x0, 0x0, 0x23, 0x22, 0x0]
    Response:  [0xfa, 0xf8, 0x2, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0]
    [None]
```

u3-feedback-DAC1_8.txt hosted with ❤ by **GitHub** **view raw**

5.2.5.14 - DAC# (16-bit): IOType=38,39 [U3 Datasheet]

Table 5.2.5.14-1. DAC 16 bit Command Response

DAC# (16-bit), 3 Command Bytes:	
0	IOType = 38, 39
1	Value LSB
2	Value MSB
0 Response Bytes:	

This IOType controls a single analog output.

- **Value**: 0=Minimum, 65535=Maximum.

u3 - dac16 code snippets

LabJackPython example session

Automatically extracted from u3.py. Debugging turned on to show the bytes sent and received.

```
16-bit DAC Feedback command

    Controls a single analog output

    Dac: 0 or 1
    Value: 0-65535

    >>> import u3
    >>> d = u3.U3()
    >>> d.debug = True
    >>> d.getFeedback(u3.DAC16(Dac = 0, Value = 0x5566))
    Sent:  [0xdc, 0xf8, 0x2, 0x0, 0xe1, 0x0, 0x0, 0x26, 0x66, 0x55]
    Response:  [0xfa, 0xf8, 0x2, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0]
    [None]
```

u3-feedback-DAC16.txt hosted with ❤ by **GitHub** **view raw**

```
16-bit DAC Feedback command for DAC0

    Controls DAC0 in 16-bit mode.

    Value: 0-65535

    >>> import u3
    >>> d = u3.U3()
    >>> d.debug = True
    >>> d.getFeedback(u3.DAC0_16(Value = 0x1122))
    Sent:  [0x54, 0xf8, 0x2, 0x0, 0x59, 0x0, 0x0, 0x26, 0x22, 0x11]
    Response:  [0xfa, 0xf8, 0x2, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0]
    [None]
```

u3-feedback-DAC0_16.txt hosted with ❤ by **GitHub** **view raw**

```
16-bit DAC Feedback command for DAC1

    Controls DAC1 in 16-bit mode.

    Value: 0-65535

    >>> import u3
    >>> d = u3.U3()
    >>> d.debug = True
    >>> d.getFeedback(u3.DAC1_16(Value = 0x2233))
    Sent:  [0x77, 0xf8, 0x2, 0x0, 0x7c, 0x0, 0x0, 0x27, 0x33, 0x22]
    Response:  [0xfa, 0xf8, 0x2, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0]
    [None]
```

u3-feedback-DAC1_16.txt hosted with ❤ by **GitHub** **view raw**

5.2.5.15 - Timer#: IOType=42,44 [U3 Datasheet]

Table 5.2.5.15-1. Timer Command Response

Timer#, 4 Command Bytes:	
0	IOType = 42, 44
1	Bit 0: UpdateReset
2	Value LSB
3	Value MSB
4 Response Bytes:	
0	Timer LSB
1	Timer
2	Timer
3	Timer MSB

This IOType provides the ability to update/reset a given timer, and read the timer value.

- **Value**: These values are only updated if the UpdateReset bit is 1. The meaning of this parameter varies with the timer mode.
- **Timer**: Returns the value from the timer module. This is the value before reset (if reset was done).

U3 - timer code examples

LabJackPython example session

Automatically extracted from u3.py. Debugging turned on to show the bytes sent and received.

```
For reading the value of the Timer. It provides the ability to update/reset
  a given timer, and read the timer value.
  (Section 5.2.5.14 of the User's Guide)

  timer: Either 0 or 1 for timer 0 or timer 1

  UpdateReset: Set True if you want to update the value

  Value: Only updated if the UpdateReset bit is 1.  The meaning of this
      parameter varies with the timer mode.

  Mode: Set to the timer mode to handle any special processing. See classes
      QuadratureInputTimer and TimerStopInput1.

  Returns an unsigned integer of the timer value, unless Mode has been
  specified and there are special return values. See Section 2.9.1 for
  expected return values.

  >>> import u3
  >>> d = u3.U3()
  >>> d.debug = True
  >>> d.configIO(NumberOfTimersEnabled = 1)
  Sent:  [0x49, 0xf8, 0x3, 0xb, 0x42, 0x0, 0x1, 0x0, 0x41, 0x0, 0x0, 0x0]
  Response:  [0x57, 0xf8, 0x3, 0xb, 0x50, 0x0, 0x0, 0x0, 0x41, 0x0, 0xf, 0x0]
  {'NumberOfTimersEnabled': 1, 'TimerCounterPinOffset': 4, 'DAC1Enable': 0, 'FIOAnalog': 15, 'EIOAnalog': 0, 'TimerCounterConfig': 65, 'EnableCounter1': False, 'EnableCounter0': False}
  >>> d.getFeedback(u3.Timer(timer = 0, UpdateReset = False, Value = 0, Mode = None))
  Sent:  [0x26, 0xf8, 0x3, 0x0, 0x2a, 0x0, 0x0, 0x2a, 0x0, 0x0, 0x0, 0x0]
  Response:  [0xfc, 0xf8, 0x4, 0x0, 0xfe, 0x1, 0x0, 0x0, 0x0, 0x63, 0xdd, 0x4c, 0x72, 0x0]
  [1917640035]
```

u3-feedback-Timer.txt hosted with ❤ by **GitHub** **view raw**

```
For reading the value of the Timer0. It provides the ability to
    update/reset Timer0, and read the timer value.
    (Section 5.2.5.14 of the User's Guide)

    UpdateReset: Set True if you want to update the value

    Value: Only updated if the UpdateReset bit is 1.  The meaning of this
         parameter varies with the timer mode.

    Mode: Set to the timer mode to handle any special processing. See classes
        QuadratureInputTimer and TimerStopInput1.

    >>> import u3
    >>> d = u3.U3()
    >>> d.debug = True
    >>> d.configIO(NumberOfTimersEnabled = 1)
    Sent:  [0x49, 0xf8, 0x3, 0xb, 0x42, 0x0, 0x1, 0x0, 0x41, 0x0, 0x0, 0x0]
    Response:  [0x57, 0xf8, 0x3, 0xb, 0x50, 0x0, 0x0, 0x0, 0x41, 0x0, 0xf, 0x0]
    {'NumberOfTimersEnabled': 1, 'TimerCounterPinOffset': 4, 'DAC1Enable': 0, 'FIOAnalog': 15, 'EIOAnalog': 0, 'TimerCounterConfig': 65, 'EnableCounter1': False, 'EnableCounter0': False}
    >>> d.getFeedback(u3.Timer0(UpdateReset = False, Value = 0, Mode = None))
    Sent:  [0x26, 0xf8, 0x3, 0x0, 0x2a, 0x0, 0x0, 0x2a, 0x0, 0x0, 0x0, 0x0]
    Response:  [0x51, 0xf8, 0x4, 0x0, 0x52, 0x2, 0x0, 0x0, 0x0, 0xf6, 0x90, 0x46, 0x86, 0x0]
    [2252771574]
```

u3-feedback-Timer0.txt hosted with ❤ by **GitHub** **view raw**

```
For reading the value of the Timer1. It provides the ability to
    update/reset Timer1, and read the timer value.
    (Section 5.2.5.14 of the User's Guide)

    UpdateReset: Set True if you want to update the value

    Value: Only updated if the UpdateReset bit is 1.  The meaning of this
         parameter varies with the timer mode.

    Mode: Set to the timer mode to handle any special processing. See classes
        QuadratureInputTimer and TimerStopInput1.

    >>> import u3
    >>> d = u3.U3()
    >>> d.debug = True
    >>> d.configIO(NumberOfTimersEnabled = 2)
    Sent:  [0x4a, 0xf8, 0x3, 0xb, 0x43, 0x0, 0x1, 0x0, 0x42, 0x0, 0x0, 0x0]
    Response:  [0x58, 0xf8, 0x3, 0xb, 0x51, 0x0, 0x0, 0x0, 0x42, 0x0, 0xf, 0x0]
    {'NumberOfTimersEnabled': 2, 'TimerCounterPinOffset': 4, 'DAC1Enable': 0, 'FIOAnalog': 15, 'EIOAnalog': 0, 'TimerCounterConfig': 66, 'EnableCounter1': False, 'EnableCounter0': False}
    >>> d.getFeedback(u3.Timer1(UpdateReset = False, Value = 0, Mode = None))
    Sent:  [0x28, 0xf8, 0x3, 0x0, 0x2c, 0x0, 0x0, 0x2c, 0x0, 0x0, 0x0, 0x0]
    Response:  [0x8d, 0xf8, 0x4, 0x0, 0x8e, 0x2, 0x0, 0x0, 0x0, 0xf3, 0x31, 0xd0, 0x9a, 0x0]
    [2597335539]
```

u3-feedback-Timer1.txt hosted with ❤ by **GitHub** **view raw**

```
For reading Quadrature input timers. They are special because their values
  are signed.

  (Section 2.9.1.8 of the User's Guide)

  Args:
    UpdateReset: Set True if you want to reset the counter.
    Value: Set to 0, and UpdateReset to True to reset the counter.

  Returns a signed integer.

  >>> import u3
  >>> d = u3.U3()
  >>> d.debug = True
  >>> d.configIO(NumberOfTimersEnabled = 2)
  Sent:  [0x4a, 0xf8, 0x3, 0xb, 0x43, 0x0, 0x1, 0x0, 0x42, 0x0, 0x0, 0x0]
  Response:  [0x58, 0xf8, 0x3, 0xb, 0x51, 0x0, 0x0, 0x0, 0x42, 0x0, 0xf, 0x0]
  {'NumberOfTimersEnabled': 2, 'TimerCounterPinOffset': 4, 'DAC1Enable': 0, 'FIOAnalog': 15, 'EIOAnalog': 0, 'TimerCounterConfig': 66, 'EnableCounter1': False, 'EnableCounter0': False}
  >>> # Setup the two timers to be quadrature
  >>> d.getFeedback(u3.Timer0Config(8), u3.Timer1Config(8))
  Sent:  [0x66, 0xf8, 0x5, 0x0, 0x68, 0x0, 0x0, 0x2b, 0x8, 0x0, 0x0, 0x2d, 0x8, 0x0, 0x0, 0x0]
  Response:  [0xfa, 0xf8, 0x2, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0]
  [None, None]
  >>> # Read the value
  [0]
  >>> d.getFeedback(u3.QuadratureInputTimer())
  Sent:  [0x26, 0xf8, 0x3, 0x0, 0x2a, 0x0, 0x0, 0x2a, 0x0, 0x0, 0x0, 0x0]
  Response:  [0xf5, 0xf8, 0x4, 0x0, 0xf5, 0x3, 0x0, 0x0, 0x0, 0xf8, 0xff, 0xff, 0xff, 0x0]
  [-8]
  >>> d.getFeedback(u3.QuadratureInputTimer())
  Sent:  [0x26, 0xf8, 0x3, 0x0, 0x2a, 0x0, 0x0, 0x2a, 0x0, 0x0, 0x0, 0x0]
  Response:  [0x9, 0xf8, 0x4, 0x0, 0xc, 0x0, 0x0, 0x0, 0x0, 0xc, 0x0, 0x0, 0x0, 0x0]
  [12]
```

u3-feedback-QuadratureInputTimer.txt hosted with ❤ by **GitHub** **view raw**

```
For reading a stop input timer. They are special because the value returns
  the current edge count and the stop value.

  (Section 2.9.1.9 of the User's Guide)

  Args:
    UpdateReset: Set True if you want to update the value.
    Value: The stop value. Only updated if the UpdateReset bit is 1.

  Returns a tuple where the first value is current edge count, and the second
  value is the stop value.

  >>> import u3
  >>> d = u3.U3()
  >>> d.debug = True
  >>> d.configIO(NumberOfTimersEnabled = 2)
  Sent:  [0x4a, 0xf8, 0x3, 0xb, 0x43, 0x0, 0x1, 0x0, 0x42, 0x0, 0x0, 0x0]
  Response:  [0x58, 0xf8, 0x3, 0xb, 0x51, 0x0, 0x0, 0x0, 0x42, 0x0, 0xf, 0x0]
  {'NumberOfTimersEnabled': 2, 'TimerCounterPinOffset': 4, 'DAC1Enable': 0, 'FIOAnalog': 15, 'EIOAnalog': 0, 'TimerCounterConfig': 66, 'EnableCounter1': False, 'EnableCounter0': False}
  >>> # Setup the timer to be Stop Input
  >>> d.getFeedback(u3.Timer1Config(9, Value = 30))
  Sent:  [0x50, 0xf8, 0x3, 0x0, 0x54, 0x0, 0x0, 0x2d, 0x9, 0x1e, 0x0, 0x0]
  Response:  [0xfa, 0xf8, 0x2, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0]
  [None]
  >>> d.getFeedback(u3.TimerStopInput1())
  Sent:  [0x28, 0xf8, 0x3, 0x0, 0x2c, 0x0, 0x0, 0x2c, 0x0, 0x0, 0x0, 0x0]
  Response:  [0x1b, 0xf8, 0x4, 0x0, 0x1e, 0x0, 0x0, 0x0, 0x0, 0x1e, 0x0, 0x0, 0x0, 0x0]
  [(0, 0)]
```

5.2.5.16 - Timer#Config: IOType=43,45 [U3 Datasheet]

Table 5.2.5.16-1. TimerConfig Command Response

Timer#Config, 4 Command Bytes:	
0	IOType = 43, 45
1	TimerMode
2	Value LSB
3	Value MSB
0 Response Bytes:	

This IOType configures a particular timer.

- **TimerMode**: See Section 2.9 for more information about the available modes.
- **Value**: The meaning of this parameter varies with the timer mode.

U3 - timerconfig code examples

LabJackPython example session

Automatically extracted from u3.py. Debugging turned on to show the bytes sent and received.

```
This IOType configures a particular timer.

    timer = # of the timer to configure

    TimerMode = See Section 2.9 for more information about the available modes.

    Value = The meaning of this parameter varies with the timer mode.

    >>> import u3
    >>> d = u3.U3()
    >>> d.debug = True
    >>> d.configIO(NumberOfTimersEnabled = 1)
    Sent:  [0x49, 0xf8, 0x3, 0xb, 0x42, 0x0, 0x1, 0x0, 0x41, 0x0, 0x0, 0x0]
    Response:  [0x57, 0xf8, 0x3, 0xb, 0x50, 0x0, 0x0, 0x0, 0x41, 0x0, 0xf, 0x0]
    {'NumberOfTimersEnabled': 1, 'TimerCounterPinOffset': 4, 'DAC1Enable': 0, 'FIOAnalog': 15, 'EIOAnalog': 0, 'TimerCounterConfig': 65, 'EnableCounter1': False, 'EnableCounter0': False}
    >>> d.getFeedback(u3.TimerConfig(timer = 0, TimerMode = 0, Value = 0))
    Sent:  [0x27, 0xf8, 0x3, 0x0, 0x2b, 0x0, 0x0, 0x2b, 0x0, 0x0, 0x0, 0x0]
    Response:  [0xfa, 0xf8, 0x2, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0]
    [None]
    >>> d.getFeedback(u3.TimerConfig(timer = 0, TimerMode = 0, Value = 65535))
    Sent:  [0x27, 0xf8, 0x3, 0x0, 0x29, 0x2, 0x0, 0x2b, 0x0, 0xff, 0xff, 0x0]
    Response:  [0xfa, 0xf8, 0x2, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0]
    [None]
```

```
This IOType configures Timer0.

  TimerMode = See Section 2.9 for more information about the available modes.

  Value = The meaning of this parameter varies with the timer mode.

  >>> import u3
  >>> d = u3.U3()
  >>> d.debug = True
  >>> d.configIO(NumberOfTimersEnabled = 1)
  Sent:  [0x49, 0xf8, 0x3, 0xb, 0x42, 0x0, 0x1, 0x0, 0x41, 0x0, 0x0, 0x0]
  Response:  [0x57, 0xf8, 0x3, 0xb, 0x50, 0x0, 0x0, 0x0, 0x41, 0x0, 0xf, 0x0]
  {'NumberOfTimersEnabled': 1, 'TimerCounterPinOffset': 4, 'DAC1Enable': 0, 'FIOAnalog': 15, 'EIOAnalog': 0, 'TimerCounterConfig': 65, 'EnableCounter1': False, 'EnableCounter0': False}
  >>> d.getFeedback(u3.Timer0Config(TimerMode = 1, Value = 0))
  Sent:  [0x28, 0xf8, 0x3, 0x0, 0x2c, 0x0, 0x0, 0x2b, 0x1, 0x0, 0x0, 0x0]
  Response:  [0xfa, 0xf8, 0x2, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0]
  [None]
  >>> d.getFeedback(u3.Timer0Config(TimerMode = 1, Value = 65535))
  Sent:  [0x28, 0xf8, 0x3, 0x0, 0x2a, 0x2, 0x0, 0x2b, 0x1, 0xff, 0xff, 0x0]
  Response:  [0xfa, 0xf8, 0x2, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0]
  [None]
```

u3-feedback-Timer0Config.txt hosted with ❤ by **GitHub** **view raw**

```
This IOType configures Timer1.

  TimerMode = See Section 2.9 for more information about the available modes.

  Value = The meaning of this parameter varies with the timer mode.

  >>> import u3
  >>> d = u3.U3()
  >>> d.debug = True
  >>> d.configIO(NumberOfTimersEnabled = 2)
  Sent:  [0x4a, 0xf8, 0x3, 0xb, 0x43, 0x0, 0x1, 0x0, 0x42, 0x0, 0x0, 0x0]
  Response:  [0x58, 0xf8, 0x3, 0xb, 0x51, 0x0, 0x0, 0x0, 0x42, 0x0, 0xf, 0x0]
  {'NumberOfTimersEnabled': 2, 'TimerCounterPinOffset': 4, 'DAC1Enable': 0, 'FIOAnalog': 15, 'EIOAnalog': 0, 'TimerCounterConfig': 66, 'EnableCounter1': False, 'EnableCounter0': False}
  >>> d.getFeedback(u3.Timer1Config(TimerMode = 6, Value = 1))
  Sent:  [0x30, 0xf8, 0x3, 0x0, 0x34, 0x0, 0x0, 0x2d, 0x6, 0x1, 0x0, 0x0]
  Response:  [0xfa, 0xf8, 0x2, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0]
  [None]
```

u3-feedback-Timer1Config.txt hosted with ❤ by **GitHub** **view raw**

5.2.5.17 - Counter#: IOType=54,55 [U3 Datasheet]

Table 5.2.5.17-1. Counter Command Response

Counter#, 2 Command Bytes:	
0	IOType = 54, 55
1	Bit 0: Reset
4 Response Bytes:	
0	Counter LSB
1	Counter
2	Counter
3	Counter MSB

This IOType reads a hardware counter, and optionally can do a reset.

- **Reset**: Setting this bit resets the counter to 0 after reading.
- **Counter**: Returns the current count from the counter if enabled. This is the value before reset (if reset was done).

U3 - counter code examples

LabJackPython example session

Automatically extracted from u3.py. Debugging turned on to show the bytes sent and received.

```
Counter Feedback command

    Reads a hardware counter, optionally resetting it

    counter: 0 or 1
    Reset: True ( or 1 ) = Reset, False ( or 0 ) = Don't Reset

    Returns the current count from the counter if enabled.  If reset,
    this is the value before the reset.
    >>> import u3
    >>> d = u3.U3()
    >>> d.debug = True
    >>> d.configIO(EnableCounter0 = True, FIOAnalog = 15)
    Sent:  [0x5f, 0xf8, 0x3, 0xb, 0x58, 0x0, 0x5, 0x0, 0x44, 0x0, 0xf, 0x0]
    Response:  [0x5a, 0xf8, 0x3, 0xb, 0x53, 0x0, 0x0, 0x0, 0x44, 0x0, 0xf, 0x0]
    {'NumberOfTimersEnabled': 0, 'TimerCounterPinOffset': 4, 'DAC1Enable': 0, 'FIOAnalog': 15, 'EIOAnalog': 0, 'TimerCounterConfig': 68, 'EnableCounter1': False, 'EnableCounter0': True}
    >>> d.getFeedback(u3.Counter(counter = 0, Reset = False))
    Sent:  [0x31, 0xf8, 0x2, 0x0, 0x36, 0x0, 0x0, 0x36, 0x0, 0x0]
    Response:  [0xfc, 0xf8, 0x4, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0]
    [0]
    >>> # Tap a ground wire to counter 0
    >>> d.getFeedback(u3.Counter(counter = 0, Reset = False))
    Sent:  [0x31, 0xf8, 0x2, 0x0, 0x36, 0x0, 0x0, 0x36, 0x0, 0x0]
    Response:  [0xe9, 0xf8, 0x4, 0x0, 0xec, 0x0, 0x0, 0x0, 0x0, 0xe8, 0x4, 0x0, 0x0, 0x0]
    [1256]
```

u3-feedback-Counter.txt hosted with ❤ by **GitHub** **view raw**

```
Counter0 Feedback command

    Reads hardware counter0, optionally resetting it

    Reset: True ( or 1 ) = Reset, False ( or 0 ) = Don't Reset

    Returns the current count from the counter if enabled.  If reset,
    this is the value before the reset.

    >>> import u3
    >>> d = u3.U3()
    >>> d.debug = True
    >>> d.configIO(EnableCounter0 = True, FIOAnalog = 15)
    Sent:  [0x5f, 0xf8, 0x3, 0xb, 0x58, 0x0, 0x5, 0x0, 0x44, 0x0, 0xf, 0x0]
    Response:  [0x5a, 0xf8, 0x3, 0xb, 0x53, 0x0, 0x0, 0x0, 0x44, 0x0, 0xf, 0x0]
    {'NumberOfTimersEnabled': 0, 'TimerCounterPinOffset': 4, 'DAC1Enable': 0, 'FIOAnalog': 15, 'EIOAnalog': 0, 'TimerCounterConfig': 68, 'EnableCounter1': False, 'EnableCounter0': True}
    >>> d.getFeedback(u3.Counter0( Reset = False ) )
    Sent:  [0x31, 0xf8, 0x2, 0x0, 0x36, 0x0, 0x0, 0x36, 0x0, 0x0]
    Response:  [0xfc, 0xf8, 0x4, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0]
    [0]
    >>> # Tap a ground wire to counter 0
    >>> d.getFeedback(u3.Counter0(Reset = False))
    Sent:  [0x31, 0xf8, 0x2, 0x0, 0x36, 0x0, 0x0, 0x36, 0x0, 0x0]
    Response:  [0xe, 0xf8, 0x4, 0x0, 0x11, 0x0, 0x0, 0x0, 0x0, 0x11, 0x0, 0x0, 0x0, 0x0]
    [17]
    >>> # Tap a ground wire to counter 0
    >>> d.getFeedback(u3.Counter0(Reset = False))
    Sent:  [0x31, 0xf8, 0x2, 0x0, 0x36, 0x0, 0x0, 0x36, 0x0, 0x0]
    Response:  [0x19, 0xf8, 0x4, 0x0, 0x1c, 0x0, 0x0, 0x0, 0x0, 0xb, 0x11, 0x0, 0x0, 0x0]
    [4363]
```

u3-feedback-Counter0.txt hosted with ❤ by **GitHub** **view raw**

```
Counter1 Feedback command

    Reads hardware counter1, optionally resetting it

    Reset: True ( or 1 ) = Reset, False ( or 0 ) = Don't Reset

    Returns the current count from the counter if enabled.  If reset,
    this is the value before the reset.

    >>> import u3
    >>> d = u3.U3()
    >>> d.debug = True
    >>> d.configIO(EnableCounter1 = True, FIOAnalog = 15)
    Sent:  [0x63, 0xf8, 0x3, 0xb, 0x5c, 0x0, 0x5, 0x0, 0x48, 0x0, 0xf, 0x0]
    Response:  [0x5e, 0xf8, 0x3, 0xb, 0x57, 0x0, 0x0, 0x0, 0x48, 0x0, 0xf, 0x0]
    {'NumberOfTimersEnabled': 0, 'TimerCounterPinOffset': 4, 'DAC1Enable': 0, 'FIOAnalog': 15, 'EIOAnalog': 0, 'TimerCounterConfig': 72, 'EnableCounter1': True, 'EnableCounter0': False}
    >>> d.getFeedback(u3.Counter1(Reset = False))
    Sent:  [0x32, 0xf8, 0x2, 0x0, 0x37, 0x0, 0x0, 0x37, 0x0, 0x0]
    Response:  [0xfc, 0xf8, 0x4, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0]
    [0]
    >>> # Tap a ground wire to counter 1
    >>> d.getFeedback(u3.Counter1(Reset = False))
    Sent:  [0x32, 0xf8, 0x2, 0x0, 0x37, 0x0, 0x0, 0x37, 0x0, 0x0]
    Response:  [0xfd, 0xf8, 0x4, 0x0, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x0, 0x0, 0x0]
    [1]
    >>> # Tap a ground wire to counter 1
    >>> d.getFeedback(u3.Counter1(Reset = False))
    Sent:  [0x32, 0xf8, 0x2, 0x0, 0x37, 0x0, 0x0, 0x37, 0x0, 0x0]
    Response:  [0xb4, 0xf8, 0x4, 0x0, 0xb7, 0x0, 0x0, 0x0, 0x0, 0x6b, 0x2b, 0x21, 0x0, 0x0]
    [2173803]
```

u3-feedback-Counter1.txt hosted with ❤ by **GitHub** **view raw**

5.2.5.18 - Buzzer: IOType=63 [U3 Datasheet]

Log in or register to post comments

Table 5.2.5.18-1.

Buzzer, 6 Command Bytes:	
0	IOType = 63
1	Bit 0: Continuous
2	Period LSB
3	Period MSB
4	Toggles LSB
5	Toggles MSB
0 Response Bytes:	

This IOType is used to make the buzzer buzz. The buzzer is only available on hardware revisions 1.20 and 1.21, not on 1.30.

- **Continuous**: If this bit is set, the buzzer will toggle continuously.
- **Period**: This value determines how many main firmware loops the processor will execute before toggling the buzzer voltage.
- **Toggles**: If Continuous is false, this value specifies how many times the buzzer will toggle.

5.2.6 - ReadMem (ReadCal) [U3 Datasheet]

Log in or register to post comments

Reads 1 block (32 bytes) from the non-volatile user or calibration memory. The available memory is different for different versions of the U3.

Hardware version 1.30 & 1.21:

Command number 0x2A accesses the user memory area which consists of 512 bytes (block numbers 0-15). Command number 0x2D accesses the calibration memory area consisting of 512 bytes (block numbers 0-15). Do not call this function while streaming. Note that there are 512 Bytes of user memory available, but only 256 Bytes are accessible when using the LJ_chUSER_MEM function in the UD driver.

Hardware version 1.20:
Command number 0x2A accesses the user memory area which consists of 256 bytes (block numbers 0-7). Command number 0x2D accesses the calibration memory area consisting of 256 bytes (block numbers 0-7). Do not call this function while streaming.

Table 5.2.6-1. ReadMem command response

Command:	
Byte	
0	Checksum8
1	0xF8
2	0×01
3	0×2A (0×2D)
4	Checksum16 (LSB)
5	Checksum16 (MSB)
6	0×00
7	BlockNum
Response:	
Byte	
0	Checksum8
1	0xF8
2	0×11
3	0×2A (0×2D)
4	Checksum16 (LSB)
5	Checksum16 (MSB)
6	Errorcode
7	0×00
8-39	32 Bytes of Data

5.2.7 - WriteMem (WriteCal) [U3 Datasheet]

Writes 1 block (32 bytes) to the non-volatile user or calibration memory.

Command number 0x28 accesses the user memory area. Hardware version 1.2 has 256 bytes (block numbers 0-7), hardware versions 1.21 and 1.3 have 512 bytes (block 0-15). The ensure backwards compatibility only 256 Bytes are accessible when using the LJ_chUSER_MEM function in the UD driver.

Command number 0x2B accesses the calibration memory area consisting of 512 bytes (block numbers 0-15), of which the last 8 blocks are not used. Memory must be erased before writing. Do not call this function while streaming.

The U3 flash has a rated endurance of at least 20000 writes, which is plenty for reasonable operation, but if this function is called in a high-speed loop the flash could eventually be damaged. In the case of these functions, that means 20000 writes per 32-byte block.

Table 5.2.7-1. WriteMem Command Response

Command:	
Byte	
0	Checksum8
1	0xF8
2	0×11
3	0×28 (0×2B)
4	Checksum16 (LSB)
5	Checksum16 (MSB)
6	0×00
7	BlockNum
8-39	32 Bytes of Data
Response:	

Byte	
0	Checksum8
1	0xF8
2	0×01
3	0×28 (0×2B)
4	Checksum16 (LSB)
5	Checksum16 (MSB)
6	Errorcode
7	0×00

5.2.8 - EraseMem (EraseCal) [U3 Datasheet]

The U3 uses flash memory that must be erased before writing. Command number 0x29 erases the entire user memory area. Command number 0x2C erases the entire calibration memory area. The EraseCal command has two extra constant bytes, to make it more difficult to call the function accidentally. Do not call this function while streaming.

Table 5.2.8-1. EraseMem Command Response

Command:	
Byte	
0	Checksum8
1	0xF8
2	0x00 (0x01)
3	0x29 (0x2C)
4	Checksum16 (LSB)
5	Checksum16 (MSB)
(6)	(0x4C)
(7)	(0x6C)
Response:	
Byte	
0	Checksum8
1	0xF8
2	0x01
3	0x29 (0x2C)
4	Checksum16 (LSB)
5	Checksum16 (MSB)
6	Errorcode
7	0x00

5.2.9 - Reset [U3 Datasheet]

Causes a soft or hard reset. A soft reset consists of re-initializing the USB buffers without re-enumeration. A hard reset is a reboot of the processor and does cause re-enumeration.

Table 5.2.9-1. Reset Command Response

Command:		
Byte		
0	Checksum8	
1	0x99	
2	ResetOptions	
		Bit 1: Hard Reset
		Bit 0: Soft Reset
3	0x00	
Response:		
Byte		
0	Checksum8	
1	0x99	
2	0x00	

3	Errorcode	

5.2.10 - StreamConfig [U3 Datasheet]

Stream mode operates on a table of channels that are scanned at the specified scan rate. Before starting a stream, you need to call this function to configure the table and scan clock. Requires U3 hardware version 1.21 or higher.

Table 5.2.10-1. StreamConfig Command Response

Command:			
Byte			
0	Checksum8		
1	0xF8		
2	NumChannels + 3		
3	0×11		
4	Checksum16 (LSB)		
5	Checksum16 (MSB)		
6	NumChannels		
7	SamplesPerPacket (1-25)		
8	Reserved		
9	ScanConfig		
		Bit 7: Reserved	
		Bit 6: Reserved	
		Bit 3: Internal stream clock frequency.	
			b0: 4 MHz
			b1: 48 MHz
		Bit 2: Divide Clock by 256	
		Bits 0-1: Resolution	
			b00: 12.8-bit effective
			b01: 11.9-bit effective
			b10: 11.3-bit effective
			b11: 10.5-bit effective
10-11	Scan Interval (1-65535)		
12	PChannel		
13	NChannel		
Repeat 12-13 for each channel			
Response:			
Byte			
0	Checksum8		
1	0xF8		
2	0×01		
3	0×11		
4	Checksum16 (LSB)		
5	Checksum16 (MSB)		
6	Errorcode		
7	0×00		

- **NumChannels**: This is the number of channels you will sample per scan (1-25).
- **SamplesPerPacket**: Specifies how many samples will be pulled out of the U3FIFO buffer and returned per data read packet. For faster stream speeds, 25 samples per packet are required for data transfer efficiency. A small number of samples per packet would be desirable for low-latency data retrieval. Note that this parameter is not necessarily the same as the number of channels per scan. Even if only 1 channel is being scanned, SamplesPerPacket will usually be set to 25, so there are usually multiple scans per packet.
- **ScanConfig**: Has bits to specify the stream base clock and effective resolution.
- **ScanInterval**: (1-65535) This value divided by the clock frequency defined in the ScanConfig parameter, gives the interval (in seconds) between scans.

- **PChannel/NChannel**: For each channel, these two parameters specify the positive and negative voltage measurement point. PChannel is 0-7 for FIO0-FIO7, 8-15 for EIO0-EIO15, 30 for temp sensor, 31 for Vreg, or 193-224 for digital/timer/counter channels. NChannel is 0-7 for FIO0-FIO7, 8-15 for EIO0-EIO15, 30 for Vref, or 31 for single-ended. Note that firmware does not support setting the negative channel 199. Instead, specify 31 for single ended measurements. When using the UD driver's stream functions 199 will be automatically converted to 31 for ease of use.

5.2.11 - StreamStart [U3 Datasheet]

Once the stream settings are configured, this function is called to start the stream. Call the StreamConfig function before every stream start. Requires U3 hardware version 1.21 or higher.

Table 5.2.11-1. StreamStart Command Response

Command:	
Byte	
0	0xA8
1	0xA8
Response:	
Byte	
0	Checksum8
1	0xA9
2	Errorcode
3	0x00

5.2.12 - StreamData [U3 Datasheet]

After starting the stream, the data will be sent as available in the following format. Reads oldest data from buffer. Requires U3 hardware version 1.21 or higher.

Table 5.2.12-1. StreamData Response

Response:	
Byte	
0	Checksum8
1	0xF9
2	4 + SamplesPerPacket
3	0xC0
4	Checksum16 (LSB)
5	Checksum16 (MSB)
6-9	TimeStamp
10	PacketCounter
11	Errorcode
12-13	Sample0
62 (max)	Backlog
63 (max)	0×00

- **SamplesPerPacket**: From StreamConfig function.
- **TimeStamp**: Not currently implemented during normal operation, but after auto-recovery bytes 6-7 reports the number of packets missed (1-65535).
- **PacketCounter**: An 8-bit (0-255) counter that is incremented by one for each packet of data. Useful to make sure packets are in order and no packets are missing.
- **Sample#**: Stream data is placed in a FIFO (first in first out) buffer, so Sample0 is the oldest data read from the buffer. The analog input reading is returned justified as a 16-bit value. Differential readings are signed, while single-ended readings are unsigned.
- **Backlog**: When streaming, the processor acquires data at precise intervals, and transfers it to a FIFO buffer until it can be sent to the host. This value represents how much data is left in the buffer after this read. The value ranges from 0-255, where 256 would equal 100% full.

Stream mode on the U3 uses a feature called auto-recovery. If the stream buffer gets too full, the U3 will go into auto-recovery mode. In this mode, the U3 no longer stores new scans in the buffer, but rather new scans are discarded. Data already in the buffer will be sent until the buffer contains less samples than SamplesPerPacket, and every StreamData packet will have errorcode 59. Once the stream buffer contains less samples than SamplesPerPacket, the U3 will start to buffer new scans again. The next packet returned will have errorcode 60. This packet will have 1 dummy scan where each sample is 0xFFFF, and this scan separates new data from any pre auto-recovery data. Note that the dummy scan could be at the beginning, middle, or end of this packet, and can even extend to following packets. Also, the TimeStamp parameter in this packet contains the number of scans that were discarded, allowing correct time to be calculated. The dummy scan counts as one of the missing scans included in the TimeStamp value.

5.2.13 - StreamStop [U3 Datasheet]

Requires U3 hardware version 1.21 or higher.

Table 5.2.13-1. StreamStop Command Response

Command:	
Byte	
0	0xB0
1	0xB0
Response:	
Byte	
0	Checksum8
1	0xB1
2	Errorcode
3	0x00

5.2.14 - Watchdog [U3 Datasheet]

Requires U3 hardware version 1.21. Controls a firmware based watchdog timer. Unattended systems requiring maximum up-time might use this capability to reset the U3 or the entire system. When any of the options are enabled, an internal timer is enabled which resets on any incoming USB communication. If this timer reaches the defined TimeoutPeriod before being reset, the specified actions will occur. Note that while streaming, data is only going out, so some other command will have to be called periodically to reset the watchdog timer.

If the watchdog is accidentally configured to reset the processor with a very low timeout period (such as 1 second), it could be difficult to establish any communication with the device. In such a case, the reset-to-default jumper can be used to turn off the watchdog (sets bytes 7-10 to 0). Power up the U3 with a short from FIO6 to SPC (FIO2 to SCL on U3 1.20/1.21), then remove the jumper and power cycle the device again. This also affects the parameters in the ConfigU3 function.

The watchdog settings (bytes 7-10) are stored in non-volatile flash memory, so every call to this function where settings are written causes a flash erase/write. The flash has a rated endurance of at least 20000 writes, which is plenty for reasonable operation, but if this function is called in a high-speed loop the flash could be damaged.

Note: Do **not** call this function while streaming.

Table 5.2.14-1. Watchdog Command Response

Command:		
Byte		
0	Checksum8	
1	0xF8	
2	0x05	
3	0x09	
4	Checksum16 (LSB)	
5	Checksum16 (MSB)	
6	WriteMask	
		Bit 0: Write
7	WatchdogOptions	
		Bit 5: Reset on Timeout
		Bit 4: Set DIO State on Timeout
8-9	TimeoutPeriod	
10	DIOConfig	
		Bit 7: State
		Bit 0-4: DIO#
11	Reserved	
12	Reserved	
13	Reserved	
14	Reserved	
15	Reserved	
Response:		
Byte		

0	Checksum8	
1	0xF8	
2	0×05	
3	0×09	
4	Checksum16 (LSB)	
5	Checksum16 (MSB)	
6	Errorcode	
7	WatchdogOptions	
8-9	TimeoutPeriod	
10	DIOConfig	
11	Reserved	
12	Reserved	
13	Reserved	
14	Reserved	
15	Reserved	

- **WatchdogOptions**: The watchdog is enabled when this byte is nonzero. Set the appropriate bits to reset the device and/or update the state of 1 digital output.
- **TimeoutPeriod**: The watchdog timer is reset to zero on any incomingUSB communication. Note that most functions consist of a write and read, but StreamData is outgoing only and does not reset the watchdog. If the watchdog timer is not reset before it counts up to TimeoutPeriod, the actions specified by WatchdogOptions will occur. The watchdog timer has a clock rate of about 1 Hz, so a TimeoutPeriod range of 3-65535 corresponds to about 3 to 65535 seconds.
- **DIOConfig**: Determines which digital I/O is affected by the watchdog, and the state it is set to. The specifiedDIO must have previously been configured for output. DIO# is a value from 0-19 according to the following: 0-7 => FIO0-FIO7, 8-15 => EIO0-EIO7, 16-19 => CIO0-CIO3

5.2.15 - SPI [U3 Datasheet]

Requires U3 hardware version 1.21. Sends and receives serial data usingSPI synchronous communication.

Table 5.2.15-1. SPI Command Response

Command:		
Byte		
0	Checksum8	
1	0xF8	
2	4 + NumSPIWords	
3	0×3A	
4	Checksum16 (LSB)	
5	Checksum16 (MSB)	
6	SPIOptions	
		Bit 7: AutoCS
		Bit 6: DisableDirConfig
		Bits 1-0: SPIMode (0=A, 1=B, 2=C, 3=D)
7	SPIClockFactor	
8	Reserved	
9	CSPinNum	
10	CLKPinNum	
11	MISOPinNum	
12	MOSIPinNum	
13	NumSPIBytesToTransfer	
14	SPIByte0	
…	…	
Response:		
Byte		
0	Checksum8	
1	0xF8	
2	1 + NumSPIWords	
3	0×3A	
4	Checksum16 (LSB)	
5	Checksum16 (MSB)	
6	Errorcode	
7	NumSPIBytesTransferred	
8	SPIByte0	
…	…	

- **NumSPIWords**: This is the number ofSPI bytes divided by 2. If the number ofSPI bytes is odd, round up and add an extra zero to the

packet.

- **SPIOptions**: If AutoCS is true, the CS line is automatically driven low during the SPI communication and brought back high when done. If DisableDirConfig is true, this function does not set the direction of the lines, whereas if it is false the lines are configured as CS=output, CLK=output, MISO=input, and MOSI=output. SPIMode specifies the standard SPI mode as discussed below.
- **SPIClockFactor**: Sets the frequency of the SPI clock. A zero corresponds to the maximum speed of about 80kHz and 255 the minimum speed of about 5.5kHz.
- **CS/CLK/MISO/MOSI -PinNum**: Assigns which digital I/O line is used for each SPI line. Value passed is 0-19 corresponding to the normal digital I/O numbers as specified in Section 2.8.
- **NumSPIBytesToTransfer**: Specifies how many SPI bytes will be transferred (1-50).

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

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

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

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 50 bytes can be written/read. Communication is full duplex so 1 byte is read at the same time each byte is written.

5.2.16 - AsynchConfig [U3 Datasheet]

Requires U3 hardware version 1.21+. Configures the U3UART for asynchronous communication. On hardware version 1.30 the TX (transmit) and RX (receive) lines appear on FIO/EIO after any timers and counters, so with no timers/counters enabled, and pin offset set to 4, TX=FIO4 and RX=FIO5. On hardware version 1.21, the UART uses SDA for TX and SCL for RX. Communication is in the common 8/n/1 format. Similar to RS232, except that the logic is normal CMOS/TTL. Connection to an RS232 device will require a converter chip such as the MAX233, which inverts the logic and shifts the voltage levels.

Table 5.2.16-1. AsynchConfig Command Response

Command:		
Byte		
0	Checksum8	
1	0xF8	
2	0x02	
3	0x14	
4	Checksum16 (LSB)	
5	Checksum16 (MSB)	
6	0x00	
7	AsynchOptions	
		Bit 7: Update
		Bit 6: UARTEnable
		Bit 5: Reserved
8	BaudFactor LSB (1.30 only)	
9	BaudFactor MSB	
Response:		
Byte		
0	Checksum8	
1	0xF8	
2	0x02	
3	0x14	
4	Checksum16 (LSB)	
5	Checksum16 (MSB)	
6	Errorcode	
7	AsynchOptions	
8	BaudFactor LSB (1.30 only)	

9	BaudFactor MSB	

- **AsynchOptions**:
  Bit 7: Update If true, the new parameters are written (otherwise just a read is done).
  Bit 6: UARTEnable If true, the UART module is enabled. Note that no data can be transfered until pins have been assigned to the UART module using the ConfigIO function.
- **BaudFactor16 (BaudFactor8)**: This 16-bit value sets the baud rate according the following formula: BaudFactor16 = 2^{16} – 48000000/(2 x Desired Baud). For example, a BaudFactor16 = 63036 provides a baud rate of 9600 bps. (With hardware revision 1.21, the value is only 8-bit and the formula is BaudFactor8 = 2^{8} – TimerClockBase/(Desired Baud)).

5.2.17 - AsynchTX [U3 Datasheet]

Requires U3 hardware version 1.21. Sends bytes to the U3UART which will be sent asynchronously on the transmit line.

Table 5.2.17-1. AsynchTX Command Response

Command:	
Byte	
0	Checksum8
1	0xF8
2	1 + NumAsynchWords
3	0×15
4	Checksum16 (LSB)
5	Checksum16 (MSB)
6	0×00
7	NumAsynchBytesToSend
8	AsynchByte0
…	…
Response:	
Byte	
0	Checksum8
1	0xF8
2	0×02
3	0×15
4	Checksum16 (LSB)
5	Checksum16 (MSB)
6	Errorcode
7	NumAsynchBytesSent
8	NumAsynchBytesInRXBuffer
9	0×00

- **NumAsynchWords**: This is the number of asynch data bytes divided by 2. If the number of bytes is odd, round up and add an extra zero to the packet.
- **NumAsynchBytesToSend**: Specifies how many bytes will be sent (0-56).
- **NumAsynchBytesInRXBuffer**: Returns how many bytes are currently in the RX buffer.

5.2.18 - AsynchRX [U3 Datasheet]

Requires U3 hardware version 1.21. Reads the oldest 32 bytes from the U3UART RX buffer (received on receive terminal). The buffer holds 256 bytes.

Table 5.2.18-1. AsynchRX Command Response

Command:	
Byte	
0	Checksum8
1	0xF8
2	0×01
3	0×16
4	Checksum16 (LSB)
5	Checksum16 (MSB)
6	0×00
7	Flush
Response:	

Byte	
0	Checksum8
1	0xF8
2	1 + NumAsynchWords
3	0×15
4	Checksum16 (LSB)
5	Checksum16 (MSB)
6	Errorcode
7	NumAsynchBytesInRXBuffer
8	AsynchByte0
…	…
39	AsynchByte31

- **Flush**: If nonzero, the entire 256-byte RX buffer is emptied. If there are more than 32 bytes in the buffer that data is lost.
- **NumAsynchBytesInRXBuffer**: Returns the number of bytes in the buffer before this read.
- **AsynchByte#**: Returns the 32 oldest bytes from the RX buffer.

5.2.19 - I²C [U3 Datasheet]

Log in or register to post comments

Requires U3 hardware version 1.21+. Sends and receives serial data using I²C (I2C) synchronous communication.

Table 5.2.19-1. I2C Command Response

Command:		
Byte		
0	Checksum8	
1	0xF8	
2	4 + NumI2CWordsSend	
3	0x3B	
4	Checksum16 (LSB)	
5	Checksum16 (MSB)	
6	I2COptions	
		Bits 7-4: Reserved
		Bit 3: Enable Clock Stretching
		Bit 2: No Stop when restarting
		Bit 1: ResetAtStart
		Bit 0: Reserved
7	SpeedAdjust	
8	SDAPinNum	
9	SCLPinNum	
10	AddressByte	
11	Reserved	
12	NumI2CBytesToSend	
13	NumI2CBytesToReceive	
14	I2CByte0	
...	...	
Response:		
Byte		
0	Checksum8	
1	0xF8	
2	3 + NumI2CWordsReceive	
3	0x3B	
4	Checksum16 (LSB)	
5	Checksum16 (MSB)	
6	Errorcode	
7	Reserved	
8	AckArray0	
9	AckArray1	
10	AckArray2	
11	AckArray3	
12	I2CByte0	
...	...	

- **NumI2CWordsSend:** This is the number of I²C bytes to send divided by 2. If the number of bytes is odd, round up and add an extra zero to the packet. This parameter is actually just to specify the size of this packet, as the NumI2CbytesToSend parameter below actually specifies how many bytes will be sent.
- **I2COptions:** If ResetAtStart is true, an I²C bus reset will be done before communicating.
- **SpeedAdjust:** Allows the communication frequency to be reduced. 0 is the maximum speed of about 150 kHz. 20 is a speed of about 70 kHz. 255 is the minimum speed of about 10 kHz.
- **SDAP/SCLP -PinNum:** Assigns which digital I/O line is used for each I²C line. Value passed is 0-19 corresponding to the normal digital I/O numbers as specified in Section 2.8. Note that the screw terminals labeled “SDA” and “SCL” on hardware revision 1.20 or 1.21 are not used for I²C. Note that the I²C bus generally requires pull-up resistors of perhaps 4.7 kΩ from SDA to Vs and SCL to Vs.
- **AddressByte:** This is the first byte of data sent on the I²C bus. The upper 7 bits are the address of the slave chip and bit 0 is the read/write bit. Note that the read/write bit is controlled automatically by the LabJack, and thus bit 0 is ignored.
- **NumI2CBytesToSend:** Specifies how many I²C bytes will be sent (0-50).
- **NumI2CBytesToReceive:** Specifies how many I²C bytes will be read (0-52).
- **I2Cbyte#**: In the command, these are the bytes to send. In the response, these are the bytes read.
- **NumI2CWordsReceive:** This is the number of I²C bytes to receive divided by 2. If the number of bytes is odd, the value is rounded up and an extra zero is added to the packet. This parameter is actually just to specify the size of this packet, as the NumI2CbytesToReceive parameter above actually specifies how many bytes to read.
- **AckArray#:** Represents a 32-bit value where bits are set if the corresponding I²C write byte was ACKed. Useful for debugging up to the first 32 write bytes of communication. Bit 0 corresponds to the last data byte, bit 1 corresponds to the second to last data byte, and so on up to the address byte. So if n is the number of data bytes, the ACK value should be (2^(n+1))-1.

5.2.20 - SHT1X [U3 Datasheet]

Requires U3 hardware version 1.21. Reads temperature and humidity from a Sensirion SHT1X sensor (which is used by the EI-1050). For more information, see the EI-1050 datasheet, and the SHT1X datasheet from sensirion.com.

Table 5.2.10-1. SHT1X Command Response

Command:	
Byte	
0	Checksum8
1	0xF8
2	0×02
3	0×39
4	Checksum16 (LSB)
5	Checksum16 (MSB)
6	DataPinNum (0-19)
7	ClockPinNum (0-19)
8	Reserved
9	Options
Response:	
Byte	
0	Checksum8
1	0xF8
2	0×05
3	0×39
4	Checksum16 (LSB)
5	Checksum16 (MSB)
6	Errorcode
7	0×00
8	StatusReg
9	StatusRegCRC
10-11	Temperature
12	TemperatureCRC
13-14	Humidity
15	HumidityCRC

- **Data/Clock -PinNum**: Assigns which digital I/O line is used for each SPI line. Value passed is 0-7 corresponding to FIO0-FIO7. State and direction are controlled automatically for the specified lines.
- **StatusReg**: Returns a read of the SHT1X status register.
- **Temperature**: Returns the raw binary temperature reading.
- **Humidity**: Returns the raw binary humidity reading.
- **#CRC**: Returns the CRC values from the sensor.

- **Options:**
  - Bit 7: Read Humidity
  - Bit 6: Read Temperature
  - Bit 5-3: Reserved, write 0
  - Bit 2: Enable Heater
  - Bit 1: Reserved, write 0
  - Bit 0: Resolution. 1 = 8-bit RH and 12-bit Temp, 0 = 12-bit RH and 14-bit Temp

5.2.21 - SetDefaults (SetToFactoryDefaults) [U3 Datasheet]

Executing this function causes the current or last used values (or the factory defaults) to be stored in flash as the power-up defaults.

The U3 flash has a rated endurance of at least 20000 writes, which is plenty for reasonable operation, but if this function is called in a high-speed loop the flash could eventually be damaged.

Note: Do **not** call this function while streaming.

(The values in parenthesis will set the defaults to factory settings.)

Table 5.2.21-1. SetDefaults Command Response

Command:	
Byte	
0	Checksum8
1	0xF8
2	0x01
3	0x0E
4	Checksum16 (LSB)
5	Checksum16 (MSB)
6	0xBA (0x82)
7	0x26 (0xC7)
Response:	
Byte	
0	Checksum8
1	0xF8
2	0x01
3	0x0E
4	Checksum16 (LSB)
5	Checksum16 (MSB)
6	Errorcode
7	0x00

LabJackPython Example

```
>>> import u3
>>> d = u3.U3()
>>> d.debug = True
'''Make the U3's current configuration the power-up defaults.'''
>>> d.setDefaults()
Sent:  [0xe8, 0xf8, 0x1, 0xe, 0xe0, 0x0, 0xba, 0x26]
Response:  [0x8, 0xf8, 0x1, 0xe, 0x0, 0x0, 0x0, 0x0]
```

5.2.22 - ReadDefaults (ReadCurrent) [U3 Datasheet]

Reads the power-up defaults from flash (Read the current configuration).

Table 5.2.22-1. ReadDefaults Command Response and Defaults map

Command:				Defaults Map		
Byte			Block Number	Byte Offset	Description	Nominal Values

0	Checksum8	0	0-3	Not Used	0x00
1	0xF8	0	4	FIO Directions	0x00
2	0x01	0	5	FIO States	0xFF
3	0x0E	0	6	FIO Analog	0x00
4	Checksum16 (LSB)	0	7	Not Used	0x00
5	Checksum16 (MSB)	0	8	EIO Directions	0x00
6	0x00	0	9	EIO States	0xFF
7	bits[0:6] BlockNum 0-7	0	10	EIO Analog	0x00
	bit 7: 1 = ReadCurrent	0	11	Not Used	0x00
		0	12	CIO Directions	0x00
Response:		0	13	CIO States	0xFF
Byte		0	14-15	Not Used	0x00
0	Checksum8	0	16	Config Write Mask	0x00 (NM)
1	0xF8	0	17	NumOfTimersEnabled	0x00
2	0x01	0	18	Counter Mask	0x00
3	0x0E	0	19	Pin Offset	0x04
4	Checksum16 (LSB)	0	20	Options	0x00
5	Checksum16 (MSB)	0	21-31	Not Used	0x00
6	Errorcode	1	0 (32)	Clock_Source	0x02
7	0x00	1	1 (33)	Divisor	0x00
8-39	Data	1	2-15 (33-47)	Not Used	0x00
		1	16 (48)	TMR0 Mode	0x0A
		1	17 (49)	TMR0 Value L	0x00
		1	18 (50)	TMR0 Value H	0x00
		1	19 (51)	Not Used	0x00
		1	20 (52)	TMR1 Mode	0x0A
		1	21 (53)	TMR1 Value L	0x00
		1	22 (54)	TMR1 Value H	0x00
		1	23-31 (55-63)	Not Used	0x00
		2	0-15 (64-79)	Not Used	0x00
		2	16-17 (80-81)	DAC0 (2 Bytes)	0x0000
		2	18-19 (82-83)	Not Used	0x00
		2	20-21 (84-85)	DAC1 (2 Bytes)	0x0000
		2	22-31 (86-95)	Not Used	0x00
		3	0-15 (96-111)	AIN Neg Channel	0x1F
		3	16-31 (112-127)	Not Used	0x00

5.2.23 - 1-Wire [U3 Datasheet]

This function performs 1-Wire communication.

Because the EIO lines on the U3 can be configured as either digital or analog, ensure that the I/O being used for 1-Wire sensor comm is first configured as either digital input or digital output, then use the function outlined below. For additional information on how to use this function, please see the 1-Wire App Note.

Table 5.2.23-1. 1-Wire Command Response

Command:		
Byte		
0	Csum8	
1	0xF8	
2	0x1D	
3	0x3C	

4	Csum16 L	
5	Csum16 H	
6	Options	
		Bit 0: DPU Control Enable
		Bit 1: DPU Polarity
		Bit 2: DPU Idle
7	Reserved	
8	Sense Pin	
9	DPU Pin	
10	Reserved	
11	ROM Function	
12	ROM0 (LSB)	
13	ROM1	
14	ROM2	
15	ROM3	
16	ROM4	
17	ROM5	
18	ROM6	
19	ROM7 (MSB)	
20	Reserved	
21	Num TX	
22	Reserved	
23	Num RX	
24	TX Byte 0	
…	...	
63	TX Byte 39	
Response:		
Byte		
0	Csum8	
1	0xF8	
2	0x1D	
3	0x3C	
4	Csum16 L	
5	Csum16 H	
6	Error Code	
7	Reserved	
8	Reserved	
9	Warnings	
		Bit 0: No Devices Detected
		Bit 1: Type 1 interrupt (Not Tested)
		Bit 2: Type 2 interrupt (Not Supported)
10	Reserved	
11	Reserved	
16	Data 0	
…	…	
63	Data 47	

Options: This byte provides control of the dynamic pull-up.

Bit 0: enables control of the DPU line.
Bit 1: sets the polarity of the switch. 1 = high on the specified DIO turns the switch on.
Bit 2: sets the idle state. 1 = DPU on while IDLE.
Sense Pin: This is the DIO on the LabJack that is connected to the data line of the 1-Wire bus.
DPU Pin: This is the DIO line that will control the dynamic pull-up if enabled in the options byte.
ROM Function: This byte specifies the function to be performed on the 1-Wire bus.
ROM[0:7]: This is the ROM of the target device or search path.
Num TX: This is the number of data bytes to transmit.
Num RX: This is the number of data bytes to receive.

Depending on the ROM function used the data returned can have different meanings. Refer to the following table for data definitions.

Table 5.2.23-2. ROM Functions

		Parameter	**Data Returned**	
ROM Function:	***Number***	***ROM***	***Bytes 0-7***	***Bytes 8-15***

Search ROM	0xF0	List of branches to take.	Discovered ROM Code	1s indicate detected branches.
Read ROM	0x33	None	ROM read from device	
Match ROM	0x55	The specific ROM		
Skip ROM	0xCC			
Alarm Search	0xEC			

Additional information

U3 firmware v1.31 and hardware v1.30 are required for 1-Wire.

Maxim has a 1-Wire app note which covers Dynamic Pull-Ups, and another on the search algorithm. There are several kinds of 1-wire temperature sensors from Maxim(DS1820, DS1821, DS1822, DS18S20, and DS18B20). The most common part is probably the DS18B20. Note that these temperature sensors require about 750ms of time to resolve a temperature reading.

5.3 - Errorcodes [U3 Datasheet]

Table 5.3-1. Low-level function errorcodes.

Error	***Code (HEX)***	***Code (DEC)***
SCRATCH_WRT_FAIL	0x01	1
SCRATCH_ERASE_FAIL	0x02	2
DATA_BUFFER_OVERFLOW	0x03	3
ADC0_BUFFER_OVERFLOW	0x04	4
FUNCTION_INVALID	0x05	5
SWDT_TIME_INVALID	0x06	6
XBR_CONFIG_ERROR	0x07	7
FLASH_WRITE_FAIL	0x10	16
FLASH_ERASE_FAIL	0x11	17
FLASH_JMP_FAIL	0x12	18
FLASH_PSP_TIMEOUT	0x13	19
FLASH_ABORT_RECIEVED	0x14	20
FLASH_PAGE_MISMATCH	0x15	21
FLASH_BLOCK_MISMATCH	0x16	22
FLASH_PAGE_NOT_IN_CODE_AREA	0x17	23
MEM_ILLEGAL_ADDRESS	0x18	24
FLASH_LOCKED	0x19	25
INVALID_BLOCK	0x1A	26
FLASH_ILLEGAL_PAGE	0x1B	27
FLASH_TOO_MANY_BYTES	0x1C	28
FLASH_INVALID_STRING_NUM	0x1D	29
SMBUS_INQ_OVERFLOW	0x20	32
SMBUS_OUTQ_UNDERFLOW	0x21	33
SMBUS_CRC_FAILED	0x22	34
SHT1x_COMM_TIME_OUT	0x28	40
SHT1x_NO_ACK	0x29	41
SHT1x_CRC_FAILED	0x2A	42
SHT1X_TOO_MANY_W_BYTES	0x2B	43
SHT1X_TOO_MANY_R_BYTES	0x2C	44
SHT1X_INVALID_MODE	0x2D	45
SHT1X_INVALID_LINE	0x2E	46
STREAM_IS_ACTIVE	0x30	48
STREAM_TABLE_INVALID	0x31	49
STREAM_CONFIG_INVALID	0x32	50
STREAM_BAD_TRIGGER_SOURCE	0x33	51
STREAM_NOT_RUNNING	0x34	52
STREAM_INVALID_TRIGGER	0x35	53
STREAM_ADC0_BUFFER_OVERFLOW	0x36	54
STREAM_SCAN_OVERLAP	0x37	55
STREAM_SAMPLE_NUM_INVALID	0x38	56
STREAM_BIPOLAR_GAIN_INVALID	0x39	57
STREAM_SCAN_RATE_INVALID	0x3A	58
STREAM_AUTORECOVER_ACTIVE	0x3B	59

STREAM_AUTORECOVER_REPORT	0x3C	60
STREAM_SOFTPWM_ON	0x3D	61
STREAM_INVALID_RESOLUTION	0x3F	63
PCA_INVALID_MODE	0x40	64
PCA_QUADRATURE_AB_ERROR	0x41	65
PCA_QUAD_PULSE_SEQUENCE	0x42	66
PCA_BAD_CLOCK_SOURCE	0x43	67
PCA_STREAM_ACTIVE	0x44	68
PCA_PWMSTOP_MODULE_ERROR	0x45	69
PCA_SEQUENCE_ERROR	0x46	70
PCA_LINE_SEQUENCE_ERROR	0x47	71
TMR_SHARING_ERROR	0x48	72
EXT_OSC_NOT_STABLE	0x50	80
INVALID_POWER_SETTING	0x51	81
PLL_NOT_LOCKED	0x52	82
INVALID_PIN	0x60	96
PIN_CONFIGURED_FOR_ANALOG	0x61	97
PIN_CONFIGURED_FOR_DIGITAL	0x62	98
IOTYPE_SYNCH_ERROR	0x63	99
INVALID_OFFSET	0x64	100
IOTYPE_NOT_VALID	0x65	101
INVALID_CODE	0x66	102
UART_TIMEOUT	0x70	112
UART_NOTCONNECTED	0x71	113
UART_NOTENALBED	0x72	114
I2C_BUS_BUSY	0x74	116
TOO_MANY_BYTES	0x76	118
TOO_FEW_BYTES	0x77	119
DSP_PERIOD_DETECTION_ERROR	0x80	128
DSP_SIGNAL_OUT_OF_RANGE	0x81	129
MODBUS_RSP_OVERFLOW	0x90	144
MODBUS_CMD_OVERFLOW	0x91	145

5.4 - Calibration Constants [U3 Datasheet]

This information is only needed when using low-level functions and other ways of getting binary readings. Readings in volts already have the calibration constants applied. The UD driver, for example, normally returns voltage readings unless binary readings are specifically requested.

Calibration Constant

The majority of the U3's analog interface functions return or require binary values. Converting between binary and voltages requires the use of calibration constants and formulas.

When using Modbus the U3 will apply calibration automatically, so voltages are sent to and read from the U3, formatted as a float.

Which Constants Should I Use?

The calibration constants stored on the U3 can be categorized as follows:

- Analog Input
- Analog Output
- Internal Temperature

Analog Input: Since the U3 uses multiplexed channels connected to a single analog-to-digital converter (ADC), all low-voltage channels have the same calibration for a given configuration. High-voltage channels have individual scaling circuitry out front, and thus the calibration is unique for each channel. The table below shows where the various calibration values are stored in the Mem area. Generally when communication is initiated with the U3, four calls will be made to the ReadMem function to retrieve the first 4 blocks of memory. This information can then be used to convert all analog input readings to voltages. Again, the high level Windows DLL (LabJackUD) does this automatically.

Analog Output: Only two calibrations are provided, one for DAC0 and one for DAC1.

Internal Temperature: This calibration is applied to the bits of a reading from channel 30 (internal temp).

U3 Input Ranges

The U3 input ranges can be found in section 2.6.2 of the User's Guide. For your convenience, that table has been provided again below.

Table 2.6.2-1. Nominal Analog Input Voltage Ranges for Low-Voltage Channels

	Max V	Min V
Single-Ended	2.44	0
Differential	2.44	-2.44
Special 0-3.6	3.6	0

Table 2.6.2-2. Nominal Analog Input Voltage Ranges for High-Voltage Channels

	Max V	Min V
Single-Ended	10.3	-10.3
Differential	N/A	N/A
Special -10/+20	20.1	-10.3

U3 Calibration Formulas (Analog In)

The readings returned by the analog inputs are raw binary values (low level functions). An approximate voltage conversion can be performed as:

Volts(uncalibrated) = (Bits/65536)*Span (Single-Ended)

Volts(uncalibrated) = (Bits/65536)*Span – Span/2 (Differential)

Where span is the maximum voltage minus the minimum voltage from the table above. For a proper voltage conversion, though, use the calibration values (Slope and Offset) stored in the internal flash on the Control processor.

Volts = (Slope * Bits) + Offset

U3 Calibration Formulas (Analog Out)

Writing to the U3's DAC require that the desired voltage be converted into a binary value. To convert the desired voltage to binary select the Slope and Offset calibration constants for the DAC being used and plug into the following formula.

Bits = (DesiredVolts * Slope) + Offset

U3 Calibration Formulas (Internal Temp)

Internal Temperature can be obtained by reading channel 30 and using the following formula.

Temp (K) = Bits * TemperatureSlope

U3 Calibration Constants

Below are the various calibration values are stored in the Mem area. Generally when communication is initiated with the U3, eight calls will be made to the ReadMem function to retrieve the first 8 blocks of memory. This information can then be used to convert all analog input readings to voltages. Again, the high level Windows DLL (LabJackUD) does this automatically.

Table 5.4-1. Normal Calibration Constant Memory Locations

Block #	Starting Byte		Nominal Value	
0	0	LV AIN SE Slope	3.7231E-05	volts/bit
0	8	LV AIN SE Offset	0.0000E+00	volts
0	16	LV AIN Diff Slope	7.4463E-05	volts/bit
0	24	LV AIN Diff Offset	-2.4400E+00	volts

1	0	DAC0 Slope	5.1717E_01	bits/volt
1	8	DAC0 Offset	0.0000E+00	bits
1	16	DAC1 Slope	5.1717E+1	bits/volt
1	24	DAC1 Offset	0.0000E+00	bits
2	0	Temp Slope	1.3021E-02	degK/bit
2	8	Vref @Cal	2.4400E+00	volts
2	16	Reserved	-	-
2	24	Reserved	-	-

Table 5.4-2. Additional High-Voltage Calibration Constant Memory Locations

Block #	Byte		Nominal Value	
3	0	HV AIN0 Slope	3.1400E-4	volts/bit
3	8	HV AIN1 Slope	3.1400E-4	volts/bit
3	16	HV AIN2 Slope	3.1400E-4	volts/bit
3	24	HV AIN3 Slope	3.1400E-4	volts/bit
4	0	HV AIN0 Offset	-10.3	volts
4	8	HV AIN1 Offset	-10.3	volts
4	16	HV AIN2 Offset	-10.3	volts
4	24	HV AIN3 Offset	-10.3	volts

Format of the Calibration Constants

Each value is stored in 64-bit fixed point format (signed 32.32 little endian, 2's complement). Following are some examples of fixed point arrays and the associated floating point double values.

Table 5.4-3. Fixed Point Conversion Examples

Fixed Point Byte Array	
{LSB, ..., MSB}	Floating Point Double
{0,0,0,0,0,0,0,0}	0.0000000000
{0,0,0,0,1,0,0,0}	1.0000000000
{0,0,0,0,255,255,255,255}	-1.0000000000
{51,51,51,51,0,0,0,0}	0.2000000000
{205,204,204,204,255,255,255,255}	-0.2000000000
{73,20,5,0,0,0,0,0}	0.0000775030
{255,122,20,110,2,0,0,0}	2.4300000000
{102,102,102,38,42,1,0,0}	298.1500000000