# DI-245 Communication Protocol

Although DATAQ Instruments provides ready-to-run WinDaq software with its DI-245 Data Acquisition Starter Kits, programmers will want the flexibility to integrate the DI-245 in the context of their own application. To do so they want complete control over DI-245 hardware, which can be accomplished by using the device at the protocol level. This white paper describes how protocol-level programming of the DI-245 is implemented across the Windows and Linux operating systems. First, we'll describe the virtual COM operation of the DI-245's interface and how communicating with the DI-245 is accomplished via a COM port hooked by the operating system. Then we'll define the DI-245's command set and scan list architecture and finish with a description of the DI-245's binary and ASCII response formats. **Note that all of the commands and their arguments described in this protocol are lower case unless otherwise stated.**

## Virtual COM Port Operation

Installing the DI-245 driver package and connecting DI-245 hardware to the host computer's USB port results in a COM port being hooked by the operating system and assigned to the DI-245 device. COM port configured should be:

| COM Port Communication Settings | |
|---|---|
| **Parameter** | **Value** |
| Baud rate | 115200 |
| Data bits | 8 |
| Stop bits | 1 |
| Parity | none |

Multiple DI-245 devices may be connected to the same PC without additional driver installations, which results in a unique COM port number assignment to each by the operating system. Hooking a COM port in this manner facilitates ease of programming from any operating system and programming language by simply writing commands to and reading responses from the port, but before any meaningful communication with a connected DI-245 can begin the controlling program must determine the COM port number assigned to the device. The method used for this varies as a function of the host operating system.

## Virtual COM Driver (Windows)

DATAQ Instruments provides a minimum installation for Windows that you can download and use at no charge, even for OEM applications. This is a scaled-down version of the standard installation that omits WinDaq software and other utilities that are extraneous in a pure programming environment. The download provides a Microsoft-signed INF file that ensures trouble-free operation with both 32- and 64-

bit Windows 7, Windows 8, and Windows 8.1. The installation depends upon drivers ftdibus.sys and ftser2k.sys usbser.sys, located in path *%SystemDrive%\Windows\System32\Drivers*.

**COM Port Number Discovery (Windows)**

Using the DI-245's vendor and product IDs, Windows' registry can be accessed programmatically to determine the COM port number that the operating system assigned to one or more connected DI-245s. The Vendor and Product ID combination for the DI-245 is: *VID_0683* and *PID_2450* respectively. With this information and at least one connected DI-245, determining assigned COM port numbers is a two-step process:

1.The registry tree *HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\ftser2k\Enum\* will contain one Device Instance ID for each DI-245 connected to the PC. The Device Instance ID assumes the following typical data for string name 0:
*FTDIBUS\VID_0683+PID_2450+5&18B6E64F&0&8\0000*.

The first two sections of this string *(FTDIBUS\VID_0683+PID_2450+)* are constant for all DI-245s. The second section *(5&18B6E64F&0&8)* will vary depending upon where in the USB port hierarchy the DI-245 is physically connected. Since more than one DI-245 cannot be connected to the same USB port this value will be unique for each concurrently connected DI-245. The entire string value is required for the second step.

2.Registry tree *HKEY_LOCAL_MACHINE \System\CurrentControlSet\Enum\FTDIBUS\VID_0683+PID_2450+5&18B6E64F&0&8\0000* (continuing with the above example) shows a variable called *FriendlyName* set to string value *DATAQ DI-245 (COMXX)*, where *XX* is the COM port number assigned to the specified DI-245. This string may be parsed to extract the port number assigned to the DI-245. The process may be repeated using other Device Instance IDs determined from step (1) for other connected DI-245 instruments.

COM port number assignments may also be determined manually from Windows' Device Manager in its *Ports (COM & LPT)* section. However, the assigned value will change depending upon the physical USB port connected to the DI-245, any other devices that may hook COM ports, and the apparently arbitrary whims of the Windows operating system

**Virtual COM Driver (Linux)**

To use FTDI-based DI-245 using a virtual COM port interface on Linux, the following commands must be issued on each reboot or unloading of the drivers (ftdi_sio & usbserial):

```
# sudo modprobe usbserial vendor=0x0683 product=0x2450
# echo "0683 2450" | sudo tee -a /sys/bus/usb-serial/drivers/ftdi_sio/new_id
```

If the DI-245 wasn't plugged in, plug it in, otherwise, unplug and re-plug for the changes to take effect. A new device file (treated as the COM port to talk to the DI-245) will show up in the "/dev/" directory using the next available increment for "ttyUSB0". For example, if no other "ttyUSB" files are present under "/dev/", plugging in the DI-245 after issuing the previous commands will result in the following file (COM port) being created: /dev/ttyUSB0

## DI-245 Command Set Overview

The DI-245 employs a simple ASCII character command set that allows complete control of the instrument.

Long commands and arguments (longer than two characters) are separated by a space character (0x20), and each long command string must be terminated with a carriage return character (x0D). Long commands do not echo until the 0x0D character is received.

Short commands (2 characters or less) are preceded with a null character (0x00), which is not echoed, but each command character is echoed as it is sent.

Basic command structure is as follows:

<0x00>**command**<(0x20)<**argument1**>(0x20)<**agrument2**>(0x0D)>

| DI-245 Command Set | |
|---|---|
| **ASCII command** | **Action** |
| Basic Information Commands | |
| Various 2-character commands | Returns ASCII information related to the hardware as a function of the argument value. |
| Scanning Commands | |
| `chn arg0 arg1` | Populates the DI-245 scan list |
| `S1` | Start scanning |
| `S0` | Stop scanning |
| CJC Commands | |
| `cjcdelta arg0 <arg1>` | Adjusts CJC sensor offsets |
| Digital Input Commands | |
| `dchn arg0` | Enable or disable the digital channel |

## Basic Information Commands

The DI-245 command set supports a number of basic command/response items that provide a simple means to ensure the integrity of the communication link of the program to the device. These commands produce simple, yet useful responses from the instrument and should be employed as the programmer's first DI-245 communication attempt. If these commands don't work with a functioning DI-245 then a problem exists in the communication chain and further programming efforts will be futile until resolved. Basic information commands are preceded with a Null character that is not echoed, but each command character is echoed by the DI-245 as it is sent. Responses to this set of commands include echoing each command character as it is received, followed by the response. For example, the command "`(0x00)A1`" generates the following response: `A12450`

| DI-245 Command Set | |
|---|---|
| **ASCII command** | **Action** |
| `A1` | Returns device name: "2450" |
| `A2` | Returns firmware version as two hex bytes (e.g. 0x65 = $101_{10}$ for firmware revision 1.01) |
| `NZ` | Returns the DI-245's serial number (of the ten digits returned, the left-most eight are the device's s/n) |
| `A7` | Returns last calibration date as the number of elapsed seconds since Jan. 1, 1970 in hex |

## *chn* Command

The DI-245 employs a scan list approach to data acquisition. A scan list is an internal schedule (or list) of channels to be sampled in a defined order. It is important to note that a scan list defines only the type and order in which data is to be sampled, not the sampled data itself. The DI-245's scan list supports only analog inputs. Analog input channels may be further defined in the scan list for input type or range. Scan list members are populated using the following general syntax (arguments are in ASCII and separated with a space):

***chn(0x20)member(0x20)value*(0x0D)**

where:

0 ≤ *member* ≤ 3 and indicates the position in the scan list.

0 ≤ *value* ≤ 65535 and indicates the value assigned to the defined scan list member to define channel number and measurement function. Refer to the following **Scan List Configuration and Measurement Tables.**

### Scan List Configuration Table

| Function | Scan List Bit Position | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Analog in, Channel 0 | All unused bits = 0 | | | **Mode** (see Measurement table) | **Range** (see Measurement table) | **Full Scale Range** (see Measurement table) | | | All unused bits = 0 | | | | 0 | 0 | 0 | 0 |
| Analog in, Channel 1 | | | | | | | | | | | | | 0 | 0 | 0 | 1 |
| Analog in, Channel 2 | | | | | | | | | | | | | 0 | 0 | 1 | 0 |
| Analog in, Channel 3 | | | | | | | | | | | | | 0 | 0 | 1 | 1 |

### Measurement Table

| Full Scale Range | | | Measurement | | |
|---|---|---|---|---|---|
| Scan List Bit Position | | | Range = 0 | Range =1 | Range = Don't care |
| 10 | 9 | 8 | Mode = 0 | Mode = 0 | Mode = 1 |
| 0 | 0 | 0 | ±500 mV | ±50 V | B thermocouple |
| 0 | 0 | 1 | ±250 mV | ±25 V | E thermocouple |
| 0 | 1 | 0 | ±100 mV | ±10 V | J thermocouple |
| 0 | 1 | 1 | ±50 mV | ±5 V | K thermocouple |
| 1 | 0 | 0 | ±25 mV | ±2.5 V | N thermocouple |
| 1 | 0 | 1 | ±10 mV | ±1 V | R thermocouple |
| 1 | 1 | 0 | n/a | n/a | S thermocouple |
| 1 | 1 | 1 | n/a | n/a | T thermocouple |

### Example chn Commands*

| Command(s) | Action |
|---|---|
| `"chn 0 0(0x0D)"` | Enable analog channel 0 for a ±50 V range as the first scan list member |
| `"chn 0 2(0x0D)"` | Enable analog channel 2 for a ±50 V range as the first scan list member |
| `"chn 0 5120(0x0D)"`<br>`"chn 1 514(0x0D)"`<br>`"chn 2 3331(0x0D)"` | Enable analog channel 0 to measure an N type TC as the first scan list member<br>Enable analog channel 2 to measure ±100 mV as the second scan list member<br>Enable analog channel 3 to measure ±1 V as the third scan list member |

* Scan list parameters are in decimal.

Observe the following when populating the DI-245 scan list:

1. Insert analog channels in the scan list in order of lowest to highest channel number.
2. Do not duplicate channel numbers in the scan list.
3. The scan list is not terminated.

## *xrate* Command

Use the *xrate* command to define the burst sample rate (Hz) of the DI-245. A single enabled channel is sampled at the rate defined by *xrate*. If multiple channels are enabled, the sample rate per channel is defined as the value defined by *xrate*, divided by ten and divided again by the number of enabled channels. For example, if *xrate* = 2000 Hz and there are two channels enabled, the sample rate per channel is 100 Hz.

*xrate* definition is a function of two arguments:

```
xrate arg0 arg1
```

where:

*arg0* is a 16-bit unsigned integer (see arg0 Configuration)
*arg1* is a 16-bit unsigned integer representing the integer result of the calculated burst rate

| arg0 Configuration | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit Position | | | | | | | | | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | Sinc4 | AF value | | | | 0 | SF value | | | | | | |

Sample rates are constructed through use of integer values AF, SF, and bit Sinc4 where:

0 ≤ AF ≤ 15
0 ≤ SF ≤ 123
IF (*burst rate in Hz)* ≥ 500 then Sinc4 = 1 ELSE Sinc4 = 0

Any given burst sampling rate (B) is calculated using at least SF and possibly AF values and the following two equations:

if *AF* = 0

$$SF = \frac{8000}{B} - 1 \qquad (1)$$

Solving for *Burst rate*

$$B = \frac{8000}{(SF + 1)} \qquad (1a)$$

if *AF* > 0

$$AF = \frac{8000}{B(SF + 1)} - 3 \qquad (2)$$

Solving for *Burst rate*

$$B = \frac{8000}{(SF + 1)(3 + AF)} \qquad (2a)$$

The choice of one equation over another is a matter of which arrives closest to a given desired burst sampling rate in Hz. However, if the desired burst rate is less than or equal to 64 Hz, Equations (1) and (1a) should be applied, since no AF values greater that zero contribute to a solution. For burst rates greater than 64, the determination of SF and AF factors is an iterative process with as many as 1,845 factor value combinations to arrive at a burst rate that is nearest the desired value. In cases where two or more combinations of AF and SF values arrive at the same burst rate, the combination with the highest SF value should be chosen.

**Example 1:**
Our desired burst rate is 128 Hz. Using (1) we determine that an integer value of 61 or 62 arrives closest to the ideal 128 Hz value. Substituting either into (2) yields values for AF that are outside its defined range. Therefore, (1a) is used with the values 61 and 62 to determine which results in a value closest to 128 Hz. SF value 62 wins, with a sampling throughput rate of 126.9841 Hz. Set *arg0* and *arg1* as follows:

| arg0 = 62 | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit Position | | | | | | | | | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | **0** | **0** | **0** | **0** | **0** | 0 | **0** | **1** | **1** | **1** | **1** | **1** | **0** |

arg1 = 127

**Example 2:**
Our desired burst rate is 10 Hz. Using (2a) values SF=79 and AF=7 produce a burst rate value precisely equal to 10 Hz. Set *arg0* and *arg1* as follows:

| arg0 = 1871 | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit Position | | | | | | | | | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | **0** | **0** | **1** | **1** | **1** | 0 | **1** | **0** | **0** | **1** | **1** | **1** | **1** |

arg1 = 10

**Example 3:**
Our desired burst rate is 750 Hz. Using (1) we determine that the ideal SF factor is 9.67, setting the integer value to either 9 or 10. Either SF value renders values for AF that are out of range. Therefore, use of equation (1a) shows that an SF value of 10 yielding an actual burst rate of 727.27 Hz falls closest to the 750 Hz ideal. Set *arg0* and *arg1* as follows:

| arg0 = 4106 | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit Position | | | | | | | | | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | **1** | **0** | **0** | **0** | **0** | 0 | **0** | **0** | **0** | **1** | **0** | **1** | **0** |

arg1 = 727

**Table of typical burst rates versus AF and SF values:**

| Burst Rate (Hz) | SF | AF | Actual Burst Rate (Hz) | Burst Rate (Hz) | SF | AF | Actual Burst Rate (Hz) |
|---|---|---|---|---|---|---|---|
| 1 | 123 | 15 | 3.58 | 70 | 113 | 0 | 70.175 |
| 2 | 123 | 15 | 3.58 | 80 | 99 | 0 | 80 |
| 3 | 123 | 15 | 3.58 | 90 | 88 | 0 | 89.89 |
| 4 | 110 | 15 | 4.004 | 100 | 79 | 0 | 100 |
| 5 | 99 | 13 | 5 | 200 | 39 | 0 | 200 |
| 6 | 110 | 9 | 6.006 | 300 | 26 | 0 | 296.3 |
| 7 | 103 | 8 | 6.99 | 400 | 19 | 0 | 400 |
| 8 | 99 | 7 | 8 | 500 | 15 | 0 | 500 |
| 9 | 110 | 5 | 9.009 | 600 | 12 | 0 | 615.38 |
| 10 | 99 | 5 | 10 | 700 | 10 | 0 | 727.27 |
| 20 | 99 | 1 | 20 | 800 | 9 | 0 | 800 |
| 30 | 37 | 4 | 30.08 | 900 | 8 | 0 | 888.89 |
| 40 | 49 | 1 | 40 | 1000 | 7 | 0 | 1000 |
| 50 | 39 | 1 | 50 | 1500 | 4 | 0 | 1600 |
| 60 | 18 | 4 | 60.15 | 2000 | 3 | 0 | 2000 |

## *cjcdelta* Command

The *cjcdelta* command applies CJC (cold junction compensation) offsets per channel to ensure measurement accuracy when using thermocouples. Since accurate thermocouple measurements depend upon an equally accurate measurement of junction temperature (where the thermocouple connects to the DI-245), the *cjcdelta* command exists to ensure accurate junction temperature readings. Adjustments using *cjcdelta* should be applied only on a channel with a connected thermocouple whose junction is held in an ice bath. This allows *cjcdelta* to adjust the measured temperature to 0°C, ± 0.0625°C.

The command syntax of *cjcdelta* consists of the command with at least one, but no more than two ASCII integer arguments that are separated by a space character (0x20), and terminated with a carriage return character (0x0D). General forms of the command follow:

```
cjcdelta(0x20)-1(0x0D)
```

Reads CJC offsets from all channels and returns four values separated by a space in the order of channel 0 to channel 3. Each value will fall in the range of -100 to +100. An offset may be calculated by multiplying the value returned per channel by 0.0625°C.

```
cjcdelta(0x20)i(0x20)j(0x0D)
```

where:
0 ≤ i ≤ 3 and represents the channel number
-100 ≤ j ≤ 100 and represents the offset multiplier (j * 0.0625°C)

```
cjcdelta(0x20)-2(0x0D)
```

Writes CJC offsets to the DI-245's flash memory.

Note that CJC offsets are inversely proportional to temperature. Higher offset values decrease temperature readings, and lower values increase temperature readings.

## *dchn* Command

The dchn command is used to enable or disable the digital input channel, which conveys information about the state of the DI-245's remote stop/start and event inputs. The command allows one argument in the form of a single ASCII character, as follows:

```
dchn(0x20)i(0x0D)
```

where:
i = 1 enables the digital input channel
i = 0 disables the digital input channel

## Scanning Commands

The DI-245 accepts two commands that control its scanning process, one to start and another to stop. These two-character ASCII commands consist of an null character (0x00), followed by an upper-case S character (0x53), followed by either an ASCII "0" or "1" (0x30 and 0x31 respectively). All but the null character are echoed by the DI-245.

```
(0x00)Si
```

where:
i = 1 starts the scanning processes
i = 0 stops the scanning process

## DI-245 Binary Output Format

Issuing the command to start scanning causes the DI-245 to respond with a continuous binary stream of one 16-bit words per enabled measurement. The least significant bit of the first byte in this stream is always cleared and set in all other response bytes to allow the host program to synchronize with the data stream. The stream sequence repeats until data acquisition is halted by the stop command. Assuming that all four analog channels and the digital channel are enabled in order, data stream composition is as follows:

| DI-245 Binary Data Stream Example<br>(all channels enabled in order) | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Scan list position | Word Count | Byte Count | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0<br>(sync) |
| 0<br>(Analog in 0) | 1 | 1 | A6 | A5 | A4 | A3 | A2 | A1 | A0 | 0 |
| | | 2 | A13 | A12 | A11 | A10 | A9 | A8 | A7 | 1 |
| 1<br>(Analog in 0) | 2 | 3 | A6 | A5 | A4 | A3 | A2 | A1 | A0 | 1 |
| | | 4 | A13 | A12 | A11 | A10 | A9 | A8 | A7 | 1 |
| 2<br>(Analog in 0) | 3 | 5 | A6 | A5 | A4 | A3 | A2 | A1 | A0 | 1 |
| | | 6 | A13 | A12 | A11 | A10 | A9 | A8 | A7 | 1 |
| 3<br>(Analog in 0) | 4 | 7 | A6 | A5 | A4 | A3 | A2 | A1 | A0 | 1 |
| | | 8 | A13 | A12 | A11 | A10 | A9 | A8 | A7 | 1 |
| 4<br>(Digital in) | 5 | 9 | D0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| | | 10 | 0 | 0 | 0 | 0 | 0 | 0 | D1 | 1 |
| 0<br>(Analog in 0) | 6 | 11 | A6 | A5 | A4 | A3 | A2 | A1 | A0 | 0 |
| | | 12 | A13 | A12 | A11 | A10 | A9 | A8 | A7 | 1 |
| ••• | ••• | ••• | ••• | ••• | ••• | ••• | ••• | ••• | ••• | ••• |

## Binary Analog Channel Coding

The DI-245 transmits a 14-bit binary number for every analog channel conversion. Meaningful information is extracted from these readings by inverting the most significant bit, and treating the result as a two’s complement number:

| AD13 | AD12 | AD11 | AD10 | AD9 | AD8 | AD7 | AD6 | AD5 | AD4 | AD3 | AD2 | AD1 | AD0 | ADC Counts |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 8191 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 8190 |
| ••• | ••• | ••• | ••• | ••• | ••• | ••• | ••• | ••• | ••• | ••• | ••• | ••• | ••• | ••• |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 2 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | -1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | -2 |
| ••• | ••• | ••• | ••• | ••• | ••• | ••• | ••• | ••• | ••• | ••• | ••• | ••• | ••• | ••• |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | -8191 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -8192 |

Channels configured as a voltage input use the following equation to derive volts as a function of FSR (full scale range) and reported ADC counts:

$$V = FSR * \frac{counts}{8192}$$

If FSR = 25 mV and measured ADC counts = 2587:

$$.025 * \frac{2587}{8192} = 0.0079\ V$$

If FSR = 2.5 V and measured ADC counts = -1279:

$$2.50 * \frac{-1279}{8192} = -0.39\ V$$

Channels configured as a  thermocouple (TC) input  borrow two ADC counts from the measurement range to indicate error conditions.

ADC counts = +8191 indicates an unrecoverable CJC error. The DI-245's processor cannot communicate with the CJC temperature sensor, or the reading is outside the CJC sensor's measurement range.

ADC counts = -8192 indicates a TC burnout condition.

An applied temperature is derived from ADC counts (A) according to the following equation, where m and b are determined by TC type:

$$°C = mA + b$$

| TC Type | m | b |
|---|---|---|
| J | 0.08606 | 495 |
| K | 0.095947 | 586 |
| T | 0.036621 | 100 |
| B | 0.095825 | 1035 |
| R/S | 0.110962 | 859 |
| E | 0.073242 | 400 |
| N | 0.091553 | 550 |

# Control

| Revision | Date | Description |
|---|---|---|
| 1.0 | Apr 22, 2014 | Original release level |
| 1.01 | Apr 29, 2014 | Changed he CJCDELTA command to "-2" from "4" to write CJC offsets to the DI-245's flash memory. |
| 1.02 | May 15, 2014 | Reduced burst rate equations; Added a comment regarding burst rates less than 64 Hz |
| 1.03 | May 16, 2014 | Corrected a typo in the "Example chn Commands" table where an argument was in hex instead of decimal; Corrected a typo in CJC scale factor to 0.0625 from 0.0624°C; Added a footnote to the "Example chn Command" table stating that all parameters are in decimal; Corrected a calculation error in the Speced vs. Actual temperature measurements table that used 8191 as full scale ADC counts vs the correct 8190. |
| 1.04 | May 16, 2014 | Returned values from the CJCDELTA command range from ±100, not 0-99. Added note that CJC offsets are inversely proportional to temperature readings. |
| 1.05 | May 19, 2014 | Various corrections to "COM Port Number Discovery (Windows)" |
| 1.06 | May 20, 2014 | Changed nomenclature of arg0 and arg1 for the xrate command to decimal from hex. |
| 1.07 | Jun 20, 2014 | Removed the reference to specified vs. measured TC ranges as misleading and unnecessary. |
| 1.08 | Aug 18, 2015 | Added COM port settings table and minor clean up. |
| 1.09 | Sep 30, 2015 | Added details about how to hook a COM port under Linux |