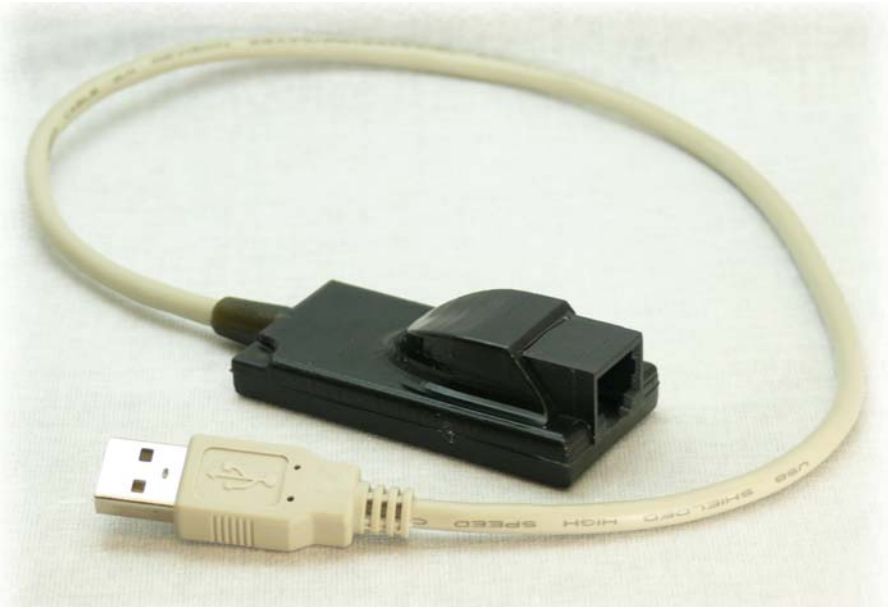
LinkUSB™
Users Guide

An Advanced, Intelligent USB Interface for the iButton™, 1-Wire™ and FxB™ Bus Protocols



Firmware Version 1.3
March 15, 2009

Table of Contents

Introduction

The LinkUSB™ is an advanced, intelligent USB 2.0 full speed interface for use with Dallas Semiconductor 1-Wire™ and iButton™ components. It uses superior digital and analog methods to accomplish more-reliable operation on a wide variety of network topologies. It also offers many functions that simplify network communications and some diagnostic tools as well.

The LinkUSB™ has been designed to operate reliably on long and short 1-Wire busses. It has a special analog interface design that uses matched impedances and slew rate controls, as well as smart cable pre-charge. In addition, the firmware (programming) is adaptive and makes adjustments for a variety of network parameters automatically.

The LinkUSB™ is based on The Link, which was designed in cooperation with Dallas Semiconductor Corp., Dallas, Texas, manufacturer of 1-Wire and iButton devices and various accessories. Some terms used in this document may be trademarks of Dallas Semiconductor Corporation, and are used with their permission. Application note 244 from the Maxim-Dallas web site describes the improvements implemented by the LinkUSB™ in greater detail.

Questions about the LinkUSB™ should be directed to info@iButtonLink.com.

USB to Serial bridge

The LinkUSB™ is based on the proven iButtonLink Link45 product, with an FTDI FT232R USB UART chip bridging the USB to the internal microprocessor's serial port. The FT232R is USB 2.0 Full Speed compatible. Drivers should already be present in recent Windows versions. The latest driver versions for Windows, Linux and Macintosh may be downloaded from http://www.ftdichip.com/.

The FTDI driver creates a virtual COM port on the host computer to communicate with the LinkUSB™. If you change the virtual COM port baud rate it changes the baud rate in the FT232R chip that connects to the LinkUSB™ microprocessor. This will make the LinkUSB™ appear unresponsive until the baud rate is returned to its previous setting. The proper sequence to enable higher baud rates is to first change the LinkUSB™ baud rate first the ASCII '^', '`', or ',' commands, then change the virtual COM port speed.

DS2480B/9097U Emulation

The LinkUSB™ is programmed to closely emulate a Dallas DS2480B serial port to 1-Wire line driver so that existing software can use it without modification. (The DS2480B is the basis for the Dallas DS9097 series of 1-Wire adapters.) This allows the LinkUSB™ to be used in place of the Dallas DS2480B in most cases where increased performance or reliability is required. Some DS2480B functions are not emulated by the LinkUSB™, as described below.

Calibration:

Unlike the DS2480B, the LinkUSB™ does not require calibration from the serial port because it has a crystal-controlled time base. This means that drift in the interface data rate and waveform timing is not an issue. The LinkUSB™ expects a calibration byte for DS2480B software compatibility, but it is ignored.

Overdrive Speed:

The LinkUSB™ does not support Overdrive speed bus communication. Because the analog components in the bus interface are carefully tuned for optimum long- and short-line performance at standard speed, the higher speed mode cannot be supported. However, for the vast majority of applications, this is not an issue.

Flex Timing Modes:

The LinkUSB™ 1-Wire waveform timing is carefully and dynamically controlled by adaptive control algorithms. Flex mode timing changes supported by the DS2480B are unnecessary and are not supported in the LinkUSB™. However, the LinkUSB™ emulates the Flex Mode registers, and so will appear to behave like the DS2480B for software compatibility.

EPROM Programming:

The LinkUSB™ does not support +12V pulses for programming EPROM 1-Wire or iButton devices. Programming pulses for 5V devices (EEPROM) are supported.

Device Version:

The LinkUSB™ reports device version 7 when queried using the normal DS2480B method. This allows the software to distinguish the LinkUSB™ from a standard Dallas DS2480B interface, which (at the time of this writing) returns a 3. Since The LINK and LinkHub-E also return device version 7, the response to an ASCII " " (space) command can be used to determine which model is present.

LED Indicator

The LinkUSB™ includes an LED indicator located on a corner near the USB cable attachment. The LED illuminates when there is traffic on the 1-Wire bus.

ASCII Control Functions:

In addition to DS2480B emulation, the LinkUSB™ also provides many 1-Wire bus functions in response to ASCII commands. These commands consist of every command code that is not used in the DS2480B emulation, and are coded for normal ASCII keys so that all basic 1-Wire functions can be performed manually on a terminal program keyboard or with programming languages that are limited to ASCII serial port I/O.

The LinkUSB™ ASCII functions are letters and symbols as follows:

Key (**spacebar**) – Displays the ID string and version number.

Key **h** – Displays a list of available LinkUSB™ ASCII commands. (These are in addition to the available standard DS2480B binary command codes.)
Similar command: **\h (Help for commands beginning with \)**
Example: (output from the h command, each line is terminated with <CR><LF>)

```
LinkUSB™ Single Character Commands

f=First         \=Expanded Command Set
n=Next
r=Reset         l=Level
(=Extend
b=Byt(NN+)      ^=57600
j=Bit(N+)       `=38400
p=BytW/P        ,=19200
~=BitW/P        x=BusLo
d=Aux+          $=Scan
z=Aux-          *=List
&=Aux?          h=Help
t=Search Type
```

Key **r** – Performs a 1-Wire bus reset and returns the status of the bus.

Normal Mode:
The **r** returns either a “P” representing the presence of one or more devices on the bus, an “N” representing no devices on the bus, or an “S” representing a shorted bus.
Example: P<CR><LF>

Extended Mode:
The **r** returns either a “P” representing the presence of one or more devices on the bus, an “N” representing no devices on the bus, or an “S” representing a shorted bus. Following the status character, the status byte is displayed as a two character hexadecimal value.
Example: PFE <CR><LF>

Key **b** – Places the LinkUSB™ into byte mode. The next two characters entered will be taken as a hexadecimal byte value, which is then issued onto the bus. The response byte is then displayed in hexadecimal. Subsequent pairs of hexadecimal characters will also generate bytes, allowing for streaming of bytes without intervening commands. Hitting ENTER (Carriage Return) will end the Byte mode.
The following subcommands are recognized in byte mode instead of a 2 character hex pair:

P... Issue 64 byte reads to the OW bus and return results. This subcommand is used to read a memory page from devices that have a page size of 64 bytes. Returns 128 characters followed by <CR><LF>

p ... Issue 66 byte reads to the OW bus and return results. This subcommand is used to read a memory page WITH CRC from devices that have a page size of 64 bytes. Returns 132 characters followed by <CR><LF>

M ... Issue 32 byte reads to the OW bus and return results. This subcommand is used to read a memory page from devices that have a page size of 32 bytes. Returns 64 characters followed by <CR><LF>

m ... Issue 34 byte reads to the OW bus and return results. This subcommand is used to read a memory page WITH CRC from devices that have a page size of 32 bytes. Returns 68 characters followed by <CR><LF>

Key **p** – Performs the same type of operation as the "**b**" key above, except that power (strong pull-up) is applied to the bus after the last bit of the first byte is issued. Subsequent bytes generated will not be followed by strong pull-up. Hitting ENTER (Carriage Return) will end the Byte mode.

Key **j** – Performs the same type of operation as the "**b**" key above, except with single bits. Only single ASCII digits of 0 or 1 value are allowed, and single ASCII digits are returned. Hitting ENTER (Carriage Return) will end Bit mode.

Key **~ (tilde)** – Performs the same type of operation as the "**j**" key above except that power (strong pull-up) is applied to the bus after the first bit is issued. (Subsequent bits generated will not be followed by strong pull-up.)

Key **f** – Performs a 1-Wire bus "first" operation. This operation searches the bus and finds the first 1-Wire or iButton device and displays the device serial number prefixed with a "+" or "-" character to indicate if there are more ("+") or no more ("-") parts remaining to be found. Similar commands: **"n (next)", \f (Family search first), $ (Scanning mode)**

If locator reporting is turned on, the device serial number is followed by a comma and the serial number of the LinkLocator associated with the 1-Wire device.

Key **n** – Performs a 1-Wire bus "next" operation. This operation searches the bus and returns the next 1-Wire or iButton device, displaying the device serial number prefixed with a "+" or "-" character to indicate if there are more ("+") or no more ("-") parts remaining to be found. (If used again after a "-" response is received, this function will find the first part again.) Similar commands**: "f (first), \f (Family search first), $ (Scanning mode)**

If locator reporting is turned on, the device serial number is followed by a comma and the serial number of the LinkLocator associated with the 1-Wire device.

Key **t** – When followed by a two-character hexadecimal value, will change the search type to this function command value. Using "tF0" will make the search normal. Using "tEC" will make the search conditional and will discover only devices for which the search conditions are satisfied. (See the data sheets for each individual Dallas iButton or 1-Wire device for specific search type command codes.)

Key **l (lower case L)** – This will test the level of the 1-Wire bus and report a "0" if the bus is low or "1" if the bus is high, followed by a carriage return.

Key **x** – Forces the 1-Wire bus to a low level. Cancelled by a reset ("r") command, or any bit or byte command, to return the 1-Wire bus to functionality. This function is used to cause a bus-wide reset by robbing power from all the bus devices for a few seconds.

Key **z** – Set the Auxiliary line to the LOW level and low impedance. (See *Auxiliary I/O* below.)

Key **d** – Set the Auxiliary line to the HIGH (default) level and low impedance. (See *Auxiliary I/O* below.)

Key **&** – Set the Auxiliary line to high impedance and report the current intended input level. (See *Auxiliary I/O* below.)

Key **$** – Start arrival/departure scanning (See *Scanning* below.)
Similar commands**: f (first), n (next), \f (Family search first)**

Key ***** – Report scan list. If scanning is NOT active the response is a single carriage return character. (See *Scanning* below.)

Key **^ (hat or shift-6)** - Switches the device to the 57,600 baud serial port data rate. The host terminal will be required to switch to 57,600 baud before it can communicate with the LinkUSB™ further. When a "break" condition is detected, the LinkUSB™ resets and returns to the 9600 baud data rate, so sending the "^" followed by more 9600 baud data will often find the device resetting and the speed returning to 9600 baud. ***See note (1) below.***

Key ` **(single quote under ~)** - Switches the device to the 38,400 baud serial port data rate. The host terminal will be required to switch to 38,400 baud before it can communicate with the LinkUSB™ further. When a "break" condition is detected, the LinkUSB™ resets and returns to the 9600 baud data rate, so sending the "^" followed by more 9600 baud data will often find the device resetting and the speed returning to 9600 baud. ***See note (1) below.***

Key **, (comma)** - Switches the device to the 19,200 baud serial port data rate. The host terminal will be required to switch to 19,200 baud before it can communicate with the LinkUSB™ further. When a "break" condition is detected, the device resets and returns to the 9600 baud data rate, so sending the "^" followed by more 9600 baud data will often find the device resetting and the speed returning to 9600 baud. ***See note (1) below.***

Key **|** (**vertical bar**) – This will cause the LinkUSB™ to enter a pass-thru mode. In this mode, all activity on the serial port is passed through (inverted) to the 1-Wire bus, and all activity on the 1-Wire bus is passed-through (inverted) to the serial port input line. This mode can be used to bypass the LinkUSB™ and allow the serial port direct access to the 1-Wire bus. Because the device is no longer able to interpret serial data in this mode, the only way to get out of the pass thru mode is by a power-on-reset of the LinkUSB™.

Note 1: The 1-Wire bus with relaxed timing suitable for long lines can only process bits at a rate of about 14,000 per second. Streaming bytes using the (b) command will fail if the baud rate is set to more than 19,200 because the host will overrun the 1-Wire bus. When the baud rate is set to any value greater than 19,200 the host commands must be paced to assure that 1-Wire bus overrun does not occur.

Key **"** (**double quote**) – This will cause the Auxiliary line to enter the "Iso" mode and to become an image of the 1-Wire bus transmitter drive waveform. This signal can be useful when trying to accomplish an isolated 1-Wire connection or whenever a two-wire connection is preferred. The original 1-Wire bus remains unaffected. Setting the Auxiliary line level using the "z" or "d" keys will turn this feature off, as will any reset of the device. (See Appendix A for more information.)

Key "**.**" (**Period**) – This will turn OFF the dynamic pull-up (DPU) driver in the 1-Wire bus interface. The DPU helps extend the useable length of the 1-Wire bus by increasing the charge current at the appropriate times in the 1-Wire waveform. In the rare event that the action of the DPU causes a problem on shorter networks, this command allows it to be turned off. The LinkUSB™ responds with a carriage return/line feed. The DPU is turned back ON by any reset of the LinkUSB™ (break or power cycle).

Key **"(" (Left parenthesis)** – Relax 1-Wire timings.

The Extended Wire "(" command relaxes the 1-Wire timings to accommodate long bus lengths. Invoke this timing if you are experiencing bus errors due to reflections from busses over 300 feet in length.

Key \ \ (**backslash backslash**) – Causes the LinkUSB™ to enter the 1-Wire sniffer mode and switch to the 57,600 baud serial port rate. The 1-Wire sniffer function listens to the 1-Wire bus and decodes data on the bus. The LinkUSB™ ceases serving as the bus master in this mode. When data is detected on the bus, it is converted into hexadecimal bytes and displayed. Each time a 1-Wire bus reset is detected, a carriage return and line feed (CR/LF) are sent. This mode is used to debug 1-Wire master programs by capturing the actual data bytes that are observed on the bus. Overdrive speed is not supported by the sniffer mode.

The Dallas DS1410 series of parallel port 1-Wire interfaces generate narrow pulses (actually out-of-spec) that may not make it through the filtering that is an integral part of The LinkUSB™'s analog front-end, and so this bus master may not work with the sniffer function. DS2480B and DS9097U adaptors, or other LINKs, will work well as "sniff-able" bus masters.

In sniffer mode, only whole bytes are reported, so any partial bytes that occur prior to a reset will not be reported.

The switch to 57,600 baud is necessary to keep up with standard speed 1-Wire bus data.

Sending a "break" condition will cause the device to be reset to default settings, and to the default 9600 baud serial port data rate.

Key **\f** (**backslash f**) – Family search first.

The \f requires two hex characters as data.

Similar to the standard first command, the family search first expects the next two hex characters to specify the 1-Wire family code to be included in the search. This causes the search to start with a particular family code. The standard "n" (next) command is used to retrieve the next 1-Wire serial number from the bus search. Since the family search may not terminate on the last device on the bus, care should be taken to check the family code returned for each serial number. In the case of a family search mismatch, a "?" is returned instead of the normal "+" or "-". **Similar commands: "f" (first) "n" (next)**

Example: **\f26**

```
A first is performed beginning with serial number
0000000000000026.
```

Key **\h (backslash h) –** Display help for commands which begin with \ (backslash).
Similar commands: **h (Help for single character commands)**

Output from the \h command (All lines terminated with <CR><LF>

```
LinkUSB™ Expanded Commands

\\ = Sniff Mode
\h = help
\L = Locator reporting on
\l = Locator reporting off
\f = Family Search first
\O = Normal slave mode
\P = Set Page read settings
\p = Report Page read settings
\X = Extended slave mode
\s = Report LinkUSB serial number
```

Key **\L (backslash uppercase L)** – Locator reporting on

Arms the reporting of LinkLocator data for the following commands:

f (first)
\f (Family code search first)
n (next)
$ (scanning mode arrivals)

The LinkLocator serial number associated with the 1-Wire address is appended to the reporting line. If there is not LinkLocator associated with the 1-Wire address, a serial number of FFFFFFFFFFFFFFFF is reported.

Key **\l (backslash lowercase L)** – Locator reporting off

Disables LinkLocator reporting.

Key **\s (backslash lowercase S)** – Display the hardware serial number of the LinkUSB™

The serial number is displayed followed by <CR><LF>
The serial number format is the same as all devices. The LinkUSB™ has a family code of 0xFE.

NOTE: The LinkUSB™ serial number is for registration and software validation purposes. At the present time, The LinkUSB™ serial number does not participate in any searches. The LinkUSB™ serial number cannot be changed after manufacturing and is guaranteed unique.

Scanning:

It is often useful to scan the 1-Wire bus and report the ***arrival*** of a new 1-Wire device or the ***departure*** of a device from the bus. The LinkUSB™ includes a bus scanning function for this purpose. When turned ON using the '$' character, the scanning system continually performs First and Next device discovery operations and builds a table of up to ten (10) device serial numbers. As a new device appears on the 1-Wire bus, and after the same device serial number (with correct CRC8 and a non-zero family code) has been observed on two subsequent full discovery passes, an Arrival is reported as a string with a "!" (exclamation-point) character, a comma, and then the arriving device serial number. When a device has been present on the bus and then is not found in ten (10) successive complete discovery cycles, it is reported as a Departure with a "?" (question mark) followed by a comma and then the serial number of the device that departed from the bus. (This de-bounces the departure.) Scanning is terminated by a Reset, First, Next or any number of other operations.

The scanning system will not work properly if more than 10 iButton or 1-Wire devices are present on the bus. This is due to the limited memory available in the LinkUSB™.

Auxiliary I/O:

The LinkUSB™ supports the standard RJ45 type 1-Wire bus connection wherein the center two contacts of the RJ connector are the data and ground connections to the 1-Wire bus. However, another pin in the RJ connector is also brought into play. This line is, by default, driven to the high impedance state and left un-powered. It can be an output supplying a low level (0 VDC) or a high level (5 VDC), or it can be an input sensing a 0-5VDC logic level. When set to the 5V level by the "**d**" command, this line can be used to provide power for DS2409 and DS2406/7 type 1-Wire switches. However, the current available from this output is limited. The amount of current that the Aux line can provide is limited to approximately 75 mA. ASCII commands used by the host can cause the Auxiliary I/O pin to change its behavior as needed.

Extended Mode Slaves:

General:

Extended mode slaves are slaves which are based upon a microprocessor instead of a 1-Wire part from Maxim-Dallas. They coexist on the same bus with Maxim-Dallas 1-Wire parts but do NOT participate in Normal Mode communications. When the host selects Extended Mode, the Maxim-Dallas parts disappear from the bus, and the Extended Mode slaves appear.

Figure 1 … Example bus. **Standard Mode Slaves** **Extended Mode slaves**.

Differences between Extended and Normal Mode:

Extended Mode is a proprietary extension to the 1-Wire protocol by iButtonLink LLC. The LinkUSB™ uses ASCII commands **(\O , \X)** to select which kind of slave is active on the bus. At present, there is no support for Extended Mode slaves using the 9097U emulation command set.

Extended mode is very similar to Normal mode with several important differences.

1. Extended mode slaves only respond to command sequences that begin with a Extended mode reset (<Reset120>).

2. Extended mode slaves do NOT emit a presence pulse. Instead, an Extended Mode slave will return a Status byte after a <Reset120>, bit 0 of which is analogous to the presence bit in Normal Mode.

3. Serial Numbers of Normal Mode slaves are allocated and managed by Maxim-Dallas. The serial numbers of Extended Mode slaves are allocated and managed by iButtonLink LLC.

4. When entering Extended Mode from Normal Mode, the LinkUSB™ issues the following command sequence to exit Normal Mode:

<Reset480>	A Normal Mode Reset
0x55	Match Rom
0x00	A family code of 0x00. This family code is invalid for all Maxim-Dallas parts.

Note: Entering Extended Mode takes advantage of the fact that family code 0x00 is invalid. If Maxim-Dallas ever deploys a Normal Mode part for family code 0x00, that part can NEVER be mixed with Extended Mode slaves.

Extended mode status byte definitions:

Bit 0 … Clear … An Extended mode slave is present
Bit 1 … Clear … An Extended mode slave is capable of Resume
Bit 2 … Clear … An Extended mode slave is in Alarm
Bit 3 … Undefined (Reserved)
Bit 4 … Undefined (Reserved)
Bit 5 … Undefined (Reserved)
Bit 6 … Undefined (Reserved)
Bit 7 … Undefined (Reserved)

Disclaimer:

iButtonLink LLC will ONLY support Extended Mode Slaves manufactured by iButtonLink LLC. Extended mode is a proprietary extension of the 1-Wire protocol by iButtonLink LLC. Agreements between iButtonLink LLC and Maxim-Dallas to protect their intellectual property DO NOT extend to other manufacturers.

FxB Protocol:

The FxB protocol greatly improves the reliability of 1-Wire slave based networks. The FxB protocol has 100 times the noise immunity of 1-Wire and 20 times the parasitic power available to the slave.

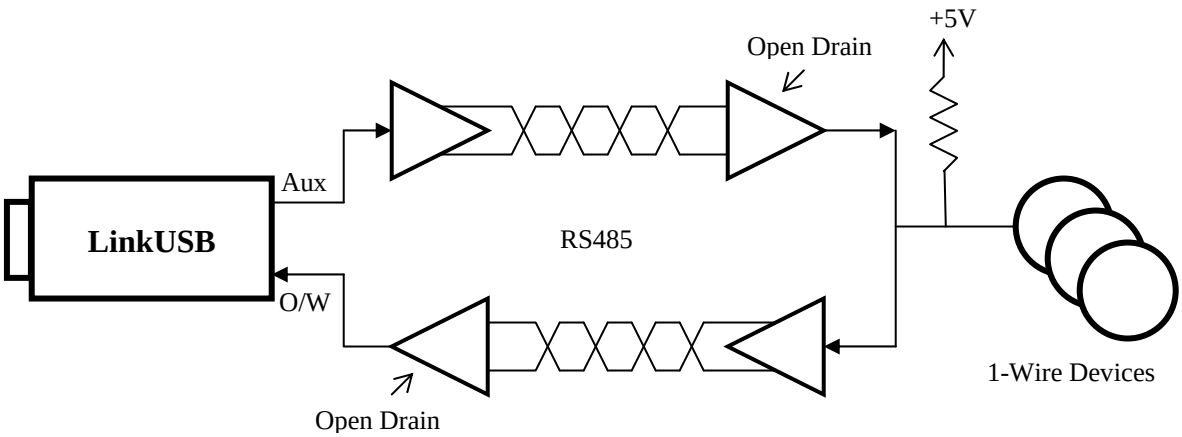
A network must consist entirely of FxB slaves, however, each 1-Wire slave can be used with the addition of the FxB translator. The FxB translator chip takes the FxB waveforms from the main communications bus and translates them into compatible 1-Wire waveforms. The 1-Wire bus behind the translator can be a maximum of 4 inches long. Translators are available either in packaged (for retro-fitting existing slaves) or chip level for inclusion new packages.

The LinkUSB™ automatically recognizes a bus based upon FxB protocol after the first reset is issued.

FxB is licensed from Alicit Engineering (Chris Fox) by iButtonLink LLC. Alicit currently has a patent pending on the FxB protocol.

Appendix A – Isolation mode

When in the "Iso" (Isolation) mode, the LinkUSB™ outputs the 1-Wire driving waveform on the Aux pin. This can be used to extend the 1-Wire bus using other signaling protocols, or to isolate the bus using optical or galvanic isolation devices. The following diagram shows how the Isolation mode could be used to extend the 1-Wire port on the LinkUSB™.



The LinkUSB™ also has a mode in which the timing of the 1-Wire waveforms is slowed to accommodate the longer turn-around times of extended host connections like that shown above. In the normal mode, network lengths of up to 3000 feet can be used (depending on the medium). In the extended mode, network lengths of between 2000 and 6000 feet can be achieved.

Appendix B – 1-Wire Communications Examples

The sequence for reading a DS18B20 temp sensor via the LinkUSB™ is straightforward:

1. Issue a 1-wire reset (`r`).
2. Enter byte mode (b) and address the ROM by sending 0x55 followed by the ROM address in reverse byte order (that is, if the discovered id is `E60000003DA0E128` you would address it as `28E1A03D000000E6`).
3. Send the convert command `44`.
4. Exit byte mode <CR>.
5. Wait at least 900ms for the conversion to complete.
6. Issue a 1-Wire reset (`r`).
7. Enter byte mode (`b`) and address the ROM as before.
8. Send the read command 0xBE.
9. Send two read commands as `FFFF`. The returned data will contain the temperature reading in Intel (little-endian) order as a 16-bit signed int.
10. Exit byte mode (CR).

The returned value will be in 1/16 deg C increments. So a return value of `5701` represents 0x0157, or 343 in decimal. Dividing by 16 gives us 21.4 degrees C, or 70.6 degrees F.

Mind the sign bit, or values below freezing will appear to be unusually warm by several thousand degrees.

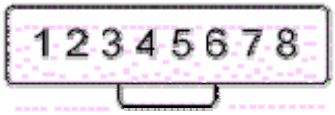
Debugging hint:

The power up default of the 18B20 is 85 degrees C (185 degrees F). Look carefully if you receive this value. This might indicate that a convert has never been executed by this device. It is possible to address a 18B20 and NOT have enough power available for it to execute a convert.

Appendix C – Wiring Diagrams

Pin outs

Viewed looking into the
LinkUSB™ 1-Wire socket



Pin	Signal
1	Ground(Orange/White)
2	+5V DC Power out (Orange) (limited to 50mA.)
3	Ground (Green/White)
4	1-Wire Data (Blue)
5	Ground (Blue/White)
6	Auxiliary (Green)
7	No connection
8	No connection

Wire colors are for “standard” Cat5 cable.

Glossary of terms used in this manual:

<CR> … a single ASCII carriage return character

<LF> … a single ASCII line feed character

<Reset480> … a normal mode bus reset as defined by Maxim-Dallas. A **<Reset480>** consists of a bus low for 480us, followed by a presence pulse issued by all normal mode slaves present on the bus (60-240us)

<Reset120> … an Extended Mode bus reset. A **<Reset120>** consists of a bus low for 120us. Extended Mode slaves will respond with a status byte after a **<Reset120>**