Midnight Code Documentation

ThruKeyboardXfer (TKXf) Specification

Classified: COMMERCIAL IN CONFIDENCE
Note that this document will be re-classified and openly published to the public some time after the mobile apps are available in the vendor app stores.

Project Code: “TKXf”

Status: RELEASE
Version: 1.0
Issue Date: 3 April 2014
Document Type: CONTROLLED

Table of Contents

1 How to use this document

This document is a guide to assist people with the understanding and implementation of a Midnight Code Thru-Keyboard Xfer application.

The document has been designed so that it can be read from start to finish, or it can be used as a ready reference to seek out targeted information, by concept.

1.1 Where to find ..

This document consists of five sections;

Chapter 1 – How to use this document
This chapter (the one you're reading now) provides detail on the structure of the document, how each section should be used and what standard mnemonics have been applied.

Chapter 2 – About TKXf
Understand TKXf: Learn about what the Thru-Keyboard Xfer is, what it is intended to be, how it came about and where it is going. Chapter 2 is a good chapter to read if you just want to know what TKXf is meant to do.

Chapter 3 – TKXf Transport Protocol Specification
From the USB HID keyboard packet network (OSI Layer 3) to the TKXf transport (OSI Layer 4), chapter 3 takes you through the TKXf protocol octet by octet.

Chapter 4 – Platform Implementations
With an understanding of the protocol, consider your implementation. What platform options exist and are there any platform-specific implementations that should be considered?

Chapter 5 – Licensing
All of the licensing information about TKXf has been published in chapter 5. This includes the licensing of the reference implementation itself, the copyright of the Midnight Code software, and the licensing of the constituent software packages that the reference implementation is reliant upon.

You will find concepts in each section are also grouped into subsections that further reflect the relationships of that material.

1.2 What does this mean?

This document uses a set of common notations to improve its readability and reference-ability.

1.2.1 Box

Information found in a grey box like this one means that the included text is a literal command or configuration item;

```
type in this command or configuration item
```

1.2.2 List

Numbered lists should be evaluated in the documented order. Bullet-point lists are lists of items that belong to a concept without any particular order. Both types of list can contain nested lists which reflect an ordered or unordered list of derivative concepts (respectively).

1.2.3 Link

Cross referencing (linking from one part of this document to another) will be used to indicate related concept or dependent process without repeating the same text twice in the manual. External links (linking from this document to other documents or Internet resources) are provided for further reading, or to indicate the source of a concept, data or file.

1.3 Is it complete?

This manual is structured to allow you to test its completeness and relevance;

1. Physically, this manual is 42 pages long. If you have less than this number of pages (including the cover page) then you are missing content and should obtain a full copy of the document. All pages, other than the cover, are numbered.

2. Logically this manual is at version number 1.0, or as version 1.0 of the TKXf protocol. If you have a version of the software or protocol that is greater (more recent) than the version of this document, then check to see if there is a newer document.

This document is stored electronically at the Midnight Code web site. It should be accessible via both the Project site (see http://midnightcode.org/projects/TGXf/) and the Papers site (see http://midnightcode.org/papers/).

1.4 Why is it spelt like that?

This manual has been written in English with Australian/British spelling.

2 About TKXf

2.1 Overview

Ever had data on the wrong side of the screen? Annoying, isn't it?

As technology becomes increasingly portable we demand greater access to our data and even greater flexibility when exchanging it between friends, work-mates, platforms and services.

If, like me, you've asked; Why do we have to re-configure a device to connect to a network to transfer a file? Or; Why do I have to connect my phone to that computer to get data into my pocket? Or worse – Why do I need to send a private file to a public cloud service in another country (and jurisdiction) in order to share it with you in the same room?

If, like me, you just want to transfer data from one screen where you can see it, “through the glass” to another so that you can see it there too, then welcome to the Thru-Glass Xfer protocol and applications.

Er, wait a minute, isn't this the *Thru-Keyboard Xfer* specification? Why yes, yes it is. The complement to Thru-Glass is Thru-Keyboard, with the two protocols and their relative applications, you can both receive data from, and send data to, the system on the other side of the screen.

2.2 How does it work?

At a high level, TKXf has a sender and a receiver. The sender encodes a file into a sequence of keyboard key-presses, and then sends them to a “keyboard stuffer” which a hardware device that masquerades as a USB keyboard. The receiver simply receives key-presses (like a text editor does) and then decodes them back into a file, which is now on the receiving device.

If you were an Internet user when dial-up services were common, then you could consider TKXf as a “keyboard modem”.

2.3 What is the “protocol” and the “applications”?

In the next chapter this document includes the specification for the Thru-Keyboard Xfer protocol. This is a description of the processes that the sender and receiver must go through in order to encode and decode the machine-readable images into a common mechanism for data exchange. In the chapter after that there are specifications for implementing the TKXf protocol as an application on various platforms.

In both cases this document has been written for application developers who want to implement the TKXf as a solution in their environment.

2.4 Target Features

The TKXf design features are considered against the following feature objectives;

- **Ease-of-Use;** Simple interface and few configuration options
- **Portable;** It must be easily implemented on new platforms and in many languages
- **Distributed;** Point-to-point transfers between participants (no central service, or registrar)
- **Robust;** The transfer must be resilient to latency and interference
- **Expansible;** It should be able to evolve as users find unexpected use cases (maybe the “killer app” is Email-Merging? If it is, let's tune it to that end)

2.5 About Midnight Code

Midnight Code is a singular resource created by Ian Latter to house and share the most useful open source software that he has developed.

These programs are the publicly publishable, cumulative and structured outputs of the seemingly ceaseless need to create that stems from the author himself. Some of the projects have a long meandering history that is due to their unique evolution, while others have been created simply to fit a niche need. The works that have been developed by the author under contract for commercial organisations are not public, and hence have not been published here. Though public works by the author, as published here, have been used to develop private (commercial) software and appliances.

The main project grouping is "The Planet Series" project set. This series of projects is designed to bring Linux to life, in the Home or Office, to fulfil the complete spectrum of communications and life-style technologies for all non-enterprise consumers. These projects start at Mercury with the development environment required to get you started, and end at Pluto with connectivity from your LAN to the rest of the universe.

Each project is clearly defined, and contains screen shots, documentation, source code, links and activity information, as identified.

3 TKXf Transport Protocol Specification

The TKXf protocol is a transport protocol that allows one way transfer of data, between two peers, typically in the form of binary data bundles (i.e. files, though streams are possible). The protocol supports high latency transfers.

3.1 USB HID as the underlying Packet Network

This simple TKXf transport protocol has been built to consume the USB Human Interface Device (HID) keyboard interface as a packet (datagram) network protocol, but could be equally transferred via any packet protocol.

3.1.1 Requirements of the USB HID Configuration

The protocol has been tested over USB HID keyboard but advantage may be gained by leveraging the USB HID mouse interface (particular in terms of binary native and data volume).

In practice, other than the packet size, this configuration is transparent to the reliant transfer protocol.

3.1.2 USB HID Configuration and Effects on Capacity/Rate

The USB HID keyboard interface relays six octets (i.e. six “key presses”) of data per packet. i.e.

0	1	2	3	4	5

These octets must be recognisable as keyboard scan codes, and will ultimately need to be received by an application without out-of-band interference (such as “Alt-F4” or “Ctrl-C” killing the receiving program).

There are less than 256 scan-codes available (0 is a “clear key” signal, for example), so “key press” octets cannot map 1:1 to octets from the source data. To avoid this limitation, a simple hexadecimal encoding scheme (00-FF) is used in this specification; other encoding schemes may be more efficient. The hexadecimal value is thus conveyed as two “keys” - one nibble each of “0” to “9” or “a” to “f” of the source data. This reduces the effective payload capacity to three source octets per packet.

The USB HID keyboard on the host controller is governed by an interrupt that is triggered once per millisecond. This puts a theoretical upper bound at 1,000 packets per second (6,000 keys per second, or 3,000 source octets per second). However, every second packet is a “key clear” packet, halving the throughput to 500 packets per second (3,000 keys or 1,500 source octets per second). Further, each packet is “debounced” by the host – that is to say that a packet with six “f” keys in it will be deduplicated to dispatch one “f” key on the host.

This implies additional performance impact, though in practice this specification provides protocol specific work-arounds to avoid this (see below).

The current specification therefore has an upper bound of 12Kbps.

3.1.3 Packet Translation

3.1.3.1 Encoding Scheme

Every byte of source data must be encoded before it can be dispatched via the USB HID keyboard interface. For this version of the specification, any given source byte will be encoded as the two scan-codes representing its two four-bit nibbles, in hexadecimal.

For a worked example, let's use the ASCII letter “K” as the source data. In ASCII, “K” is represented by the decimal value 75, which is 4B in hexadecimal. The two constituent nibbles are therefore hexadecimal 4 and hexadecimal B (decimal 11).

Source Nibble Value	Hex Value / Key on Keyboard	Scan-code in USB HID packet
0000 (0)	0	0x27
0001 (1)	1	0x1E
0010 (2)	2	0x1F
0011 (3)	3	0x20
0100 (4)	4	0x21
0101 (5)	5	0x22
0110 (6)	6	0x23
0111 (7)	7	0x24
1000 (8)	8	0x25
1001 (9)	9	0x26
1010 (10)	a	0x04
1011 (11)	b	0x05
1100 (12)	c	0x06
1101 (13)	d	0x07
1110 (14)	e	0x08
1111 (15)	f	0x09

Note that lower-case characters have been used for the keyboard representation of the nibble's hex value.

It therefore follows that the source data of ASCII letter “K”, comprised of nibbles 4 and B (11), will be encoded to the two key-presses of 0x21 followed by 0x05.

3.1.3.2 Decoding Scheme

Be aware that at the infrastructure layer (USB HID keyboard interface) the scan-codes will be received as depicted in the previous table. However, when reading user input at the application layer the Operating System has already interpreted the key-presses and turned them into ASCII before providing them to the requesting application.

Received ASCII Character	ASCII Value in Decimal	Derived Nibble Value
0	48	0000 (0)
1	49	0001 (1)
2	50	0010 (2)
3	51	0011 (3)
4	52	0100 (4)
5	53	0101 (5)
6	54	0110 (6)
7	55	0111 (7)
8	56	1000 (8)
9	57	1001 (9)
a	61	1010 (10)
b	62	1011 (11)
c	63	1100 (12)
d	64	1101 (13)
e	65	1110 (14)
f	66	1111 (15)

Working from the previous example, the key sequence for “K” which was sent as scan-codes 0x21 followed by 0x05 will be received as the ASCII letter “4” followed by the ASCII letter “b” which are shown in the above table corresponding to the derived nibbles 4 and b (11) respectively.

3.1.3.3 Deduplication Algorithm

For one reason or another the USB HID keyboard interface refuses to accept duplicate keystrokes in a single HID packet. The two reasons that come to mind are;

- Pressing and releasing keys (switches or contacts) is not an electrically “clean” process. The closing of the contact as well as its release will actually “bounce”, meaning that will generate a number of on/off sequences in addition to the one the user intended. The technique for cleaning the signal and determining the true “key down” and “key up” event that comes from this noise is called *de-bouncing*.

- It is exceedingly unlikely that a human operator intended to be, or is capable of, pressing the same key twice in less than 1ms.

Regardless of the reason, the effect is that repetitive source data will be corrupted by the deduplicating process within the USB host processing, if it's not mitigated. The solution used in this specification is a unique position-based representation of the duplicated key-presses, according to the following table.

Position Referenced	Hex Value / Key on Keyboard	Scan-code in USB HID packet
0	m	0x10
1	n	0x11
2	o	0x12
3	p	0x13
4	q	0x14

Again, note that lower-case keyboard keys are used.

For a worked example, let's use the ASCII sequence “+OK” as the source data. This sequence encodes in the following way;

- + => 43 decimal or 2B hexadecimal => encoded as key-presses 0x1F,0x05
- O => 79 decimal or 4F hexadecimal => encoded as key-presses 0x21,0x09
- K => 75 decimal or 4B hexadecimal => encoded as key-presses 0x21,0x05

The resulting packet would be sent out as;

0x1F	0x05	0x21	0x09	0x21	0x05

But would be received as;

0x1F	0x05	0x21	0x09		

To ensure all data is received accurately, duplicates are replaced with reference values. Take the previously encoded packet;

0x1F	0x05	0x21	0x09	0x21	0x05

Work right to left and look for duplicates. So in position 5, the value 0x05 is a duplicate of the value in position 1 (which is represented by the scan-code 0x11). Then in position 4, the value 0x21 is a duplicate of the value in position 2 (which is represented by the scan-code 0x12). Substituting the reference values, you will now have the following deduplicated packet for transmission;

0x1F	0x05	0x21	0x09	0x12	0x11

This encoding scheme does work for both multiple and repeating duplicates.

Let's do another worked example, taking the worst case, which is where all nibbles are identical – say the input data of the ASCII string “DDD”. ASCII “D” is hexadecimal 44 which would result in the HID packet of;

0x21	0x21	0x21	0x21	0x21	0x21

Deduplicating this packet from right to left:

- Take position 5, which has value 0x21. The next position to contain 0x21 is position 4;

0x21	0x21	0x21	0x21	0x21	0x14

- Take position 4, which has value 0x21. The next position to contain 0x21 is position 3;

0x21	0x21	0x21	0x21	0x13	0x14

- Take position 3, which has value 0x21. The next position to contain 0x21 is position 2;

0x21	0x21	0x21	0x12	0x13	0x14

- Take position 2, which has value 0x21. The next position to contain 0x21 is position 1;

0x21	0x21	0x11	0x12	0x13	0x14

- Take position 1, which has value 0x21. The next position to contain 0x21 is position 0;

0x21	0x10	0x11	0x12	0x13	0x14

3.1.3.4 Reduplication Algorithm

For receiving and re-duplicating the packet before decoding it, the following table is used;

Received ASCII Character	ASCII Value in Decimal	Position Referenced
m	109	0
n	110	1
o	111	2
p	112	3
q	113	4

Now restoring this packet from the previous worked example, at the receiving end, is straight forward. Recall that the key-presses received are now ASCII characters, which in this example will be the ASCII characters of “4”, “m”, “n”, “o”, “p” and “q”, which would be a buffer with the following decimal values;

52	109	110	111	112	113

Process the packet from left to right;

- Take position 1, which has value 109 and is therefore referencing position 0. The content at position 0 is 52;

52	52	110	111	112	113

- Take position 2, which has value 110 and is therefore referencing position 1. The content at position 1 is 52;

52	52	52	111	112	113

- Take position 3, which has value 111 and is therefore referencing position 2. The content at position 2 is 52;

52	52	52	52	112	113

- Take position 4, which has value 112 and is therefore referencing position 3. The content at position 3 is 52;

52	52	52	52	52	113

- Take position 5, which has value 109 and is therefore referencing position 4. The content at position 4 is 52;

52	52	52	52	52	52

With the original packet restored the program can go on to decode the packet into it nibbles and restore the transmitted octets.

3.1.4 Advantages and Limitations

The only true advantage of USB HID keyboard as a packet protocol is that it is a common protocol, already implemented on many platforms (it is highly portable).

The limitations of USB HID keyboard as a packet protocol are;

- Tiny packet sizes, and;
- No native support for binary payloads, and;
- Low transfer rate (packets per second) due to the low HID interrupt rate, and;
- Unnatural interface (encoding and decoding must take place to make the protocol viable).

3.2 Structure of a TKXf ~~Frame~~ Sequence

For the purposes of this specification the terms *packet* and *frame* are synonymous.

Due to the tiny size of the USB HID keyboard packet and the lack of an imaginative encoding scheme of sufficient efficiency, the TKXf protocol does not subtract a protocol header from the payload capacity of the keyboard packet for each packet in the transmission sequence. Instead, preceding any given data flow, the TKXf protocol will dispatch one single packet as a “control packet” in the sequence.

The result is a sequence of one TKXF Frame (marked as Control or Data), followed by any number of “payload” Frames (each consistent of 1, 2 or 3 source data octets, represented as 2, 4 or 6 key-presses respectively).

It should be noted that none of the advanced control features of the TGXf protocol have been ported to the TKXf protocol in this version. It is intended that this functionality be included, where practical.

3.2.1 Frame Size

The TKXf protocol does not specify a maximum Frame size, but it does require a minimum capacity of 1 octet per Frame. The limitation of the Control Frame size in the TGXf protocol is overcome in the TKXf protocol via the “sequence” use of the underlying packet protocol. Or, from the TGXf perspective, an octet consumed as a header was between ~1% and ~10% overhead, due to the larger packet sizes available from the underlying protocol.

To assume consistency with the TGXf protocol a Control Sequence could be regarded as one Control Frame and three Payload Frames (for a total of nine octets of payload). This has not been implemented in the current reference code.

3.2.2 Structure of a Data Sequence

A Data Sequence consists of one Control Frame followed by any number of Payload Frames.

The Control Frame that signifies a Data Sequence is one octet with the scan-code for the space bar (0x2c) at the transmission end and the ASCII code for the space character (0x20) at the receiving end.

3.2.3 Structure of a Control Sequence

A Control Sequence consists of one Control Frame followed by up to three Payload Frames.

The Control Frame that signifies a Control Sequence is one octet with the scan-code for the comma key (0x36) at the transmission end and the ASCII code for the comma character (0x2C) at the receiving end.

3.2.4 Structure of an Exit Sequence

Until the advanced control features of the TGXf protocol have been ported to the TKXf protocol, an exit/abort sequence has been included.

An Exit Sequence consists of one Control Frame only.

The Control Frame that signifies an Exit Sequence is one octet with the scan-code for the period key (0x37) at the transmission end and the ASCII code for the period character (0x2E) at the receiving end.

4 Platform Implementations

There are multiple platform implementations conceivable for this protocol. The superset of wired and unwired have been considered, with wired being tested successfully and reference code provided herein. Some considerations to the unwired implementations have also been provided.

4.1 Wired USB HID

The simplest implementation is a wired solution with an existing USB HID keyboard device (referred herein as a “Keyboard Stuffer” device).

The following sections will follow the order of the above architecture diagram, beginning with the C transmit code, followed by the USB Serial/Keyboard Stuffer hardware and Arduino transmit code, then the C receive code.

4.1.1 C Reference Implementation (Transmit Only)

The following C reference implementation was used to validate the end-to-end solution that has been documented in this specification.

4.1.1.1 The TKXf Transmit C Source

The following C code is for tkxf-transmit.c, a C implementation of the TKXf protocol (transmit only).

```
/*
                                  _JNJ`
                               .JNMH`
                              JMMF`        `;.
                             .NMM)          `MN.
                             MMM)            (MML
                            (MMM`             MMML
          M  I  D  N  I  G  H  T    C  o  D  E
                            (NMMF             MHNH
                             NMML            .MMM
                              NMML           .NMH
                               4MMNL        .#F
                                `4HNNL_    `
                                   `"""`

                        Copyright (C) 2004-2014
           "Ian (Larry) Latter" <ian dot latter at midnightcode dot org>

    Midnight Code is a registered trademark of Ian Latter.

    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, and mirrored at the Midnight Code web
```

```
    site; as at version 2 of the License only.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    General Public License for more details.

    You should have received a copy of the GNU General Public License
    (version 2) along with this program; if not, write to the Free
    Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
    02111-1307 USA, or see http://midnightcode.org/gplv2.txt

*/

//
// Change the serial port and test file in this program accordingly
//

#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <termios.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>


#define DATA_MODE 0
#define CONTROL_MODE 1

int wire_packets = 0;


// Serial source from wallyk;
//   http://stackoverflow.com/questions/6947413/how-to-open-read-and-write-from-serial-port-
in-c
int
set_interface_attribs(int fd, int speed, int parity) {
  struct termios tty;

  memset (&tty, 0, sizeof tty);
  if(tcgetattr (fd, &tty) != 0) {
    printf("Error: %d from tcgetattr", errno);
    return -1;
  }

  cfsetospeed(&tty, speed);
  cfsetispeed(&tty, speed);

  tty.c_cflag = (tty.c_cflag & ~CSIZE) | CS8;  // 8-bit chars
    // disable IGNBRK for mismatched speed tests; otherwise receive break
    // as \000 chars
  tty.c_iflag &= ~IGNBRK;                 // ignore break signal
  tty.c_lflag = 0;                        // no signaling chars, no echo,
                                          // no canonical processing
  tty.c_oflag = 0;                        // no remapping, no delays
  tty.c_cc[VMIN]  = 0;                    // read doesn't block
  tty.c_cc[VTIME] = 5;                    // 0.5 seconds read timeout

  tty.c_iflag &= ~(IXON | IXOFF | IXANY); // shut off xon/xoff ctrl

  tty.c_cflag |= (CLOCAL | CREAD);        // ignore modem controls,
                                          // enable reading
  tty.c_cflag &= ~(PARENB | PARODD);      // shut off parity
  tty.c_cflag |= parity;
  tty.c_cflag &= ~CSTOPB;
  tty.c_cflag &= ~CRTSCTS;

  if(tcsetattr(fd, TCSANOW, &tty) != 0) {
    printf("Error: %d from tcsetattr", errno);
```

```
    return -1;
  }

  tcflush(fd, TCIOFLUSH);                // flush the port

  return 0;
}


void
set_blocking (int fd, int should_block) {
  struct termios tty;

  memset(&tty, 0, sizeof tty);
  if(tcgetattr (fd, &tty) != 0) {
    printf("Error: %d from tggetattr", errno);
    return;
  }

  tty.c_cc[VMIN]  = should_block ? 1 : 0;
  tty.c_cc[VTIME] = 5;            // 0.5 seconds read timeout

  if(tcsetattr (fd, TCSANOW, &tty) != 0)
    printf("Error: %d setting term attributes", errno);
}


int
setup_port(char * serial_dev) {
  int fd;

  fd = open(serial_dev, O_RDWR|O_NOCTTY|O_SYNC);
  if(fd < 0) {
    printf("Error: %d opening %s: %s", errno, serial_dev, strerror (errno));
    return -1;
  }
  set_interface_attribs(fd, B38400, 0);  // set speed to 115,200 bps, 8n1 (no parity)
  set_blocking(fd, 1);                    // set blocking

  return fd;
}


int
data_byte(int fd) {
  unsigned char octet;

  octet = 32;
  if(write(fd, &octet, 1) < 0) {
    printf("Error: failed to write octet to socket\n");
    return -1;
  }

  return 0;
}


int
stop_byte(int fd) {
  unsigned char octet;

  octet = '.';
  if(write(fd, &octet, 1) < 0) {
    printf("Error: failed to write octet to socket\n");
    return -1;
  }

  return 0;
}


// USB keyboad packet doesn't allow duplicates
```

```
//
// Proposed dedupe scheme is as follows;
//     0x10 = letter m, meaning "as per key[0]"
//     0x11 = letter n, meaning "as per key[1]"
//     0x12 = letter o, meaning "as per key[2]"
//     0x13 = letter p, meaning "as per key[3]"
//     0x14 = letter q, meaning "as per key[4]"
//
// So byte data -> 0x00,0x00,0x00
//   Encoded nibbles => 0x27,0x27,0x27,0x27,0x27,0x27
//   Deduped nibbles => 0x27,0x10,0x11,0x12,0x13,0x14
//
// Encode right to left, decode left to right.
int
dedupe_key_buffer(unsigned char * keys, unsigned int max_offset) {
  int reference_pos;
  int compare_pos;

  for(reference_pos = max_offset; reference_pos > 0; reference_pos--) {
    for(compare_pos = reference_pos - 1;
      compare_pos >= 0; compare_pos--) {
        if(keys[compare_pos] == keys[reference_pos]) {
          keys[reference_pos] = compare_pos + 0x10;
          break;
        }
    }
  }

  return 0;
}


unsigned int
pack_byte(unsigned char * keys, unsigned int offset, unsigned char b) {
  unsigned char nibble;
  // Keyboard codes for 0-9 and a-f
  unsigned char nibbler[] = { 0x27, 0x1e, 0x1f, 0x20,
                              0x21, 0x22, 0x23, 0x24,
                              0x25, 0x26, 0x04, 0x05,
                              0x06, 0x07, 0x08, 0x09 };

  nibble = (b>>4) & 0xf;
  keys[offset] = nibbler[nibble];
  offset++;
  nibble = b & 0xf;
  keys[offset] = nibbler[nibble];
  offset++;

  return offset;
}


int
send_keyboard_packet(int fd, unsigned int mode, unsigned char * keys, unsigned int
max_offset) {
  unsigned char data_buffer;
  unsigned char keyboard_packet[7];
  unsigned int idx;


  // Build 7 byte packet
  if(mode == DATA_MODE)
    keyboard_packet[0] = 0x2c; // space
  if(mode == CONTROL_MODE)
    keyboard_packet[0] = 0x36; // ,
  dedupe_key_buffer(keys, max_offset);
  for(idx = 0; idx <= max_offset; idx++) {
    keyboard_packet[idx + 1] = keys[idx];
  }

  if(write(fd, keyboard_packet, max_offset + 2) < 0) {
    printf("Error: failed to write octet to fd\n");
```

```
    return -1;
  }
  wire_packets++;

  if(wire_packets == 4) {
    data_buffer = 0;
    if(read(fd, &data_buffer, 1) != 1) {
      printf("Error: failed to read octet from fd\n");
      return -2;
    }
    if(data_buffer != 1) {
      printf("Error: non-ack data read from fd [%d]\n", data_buffer);
      return -3;
    }
    wire_packets = 0;
  }

  return 0;
}


void
close_port(int fd) {

  close(fd);

  return;
}


int
main(void) {
  char * serial_dev = "/dev/ttyACM0";
  char filepath[32] = "test.bin";

  int serial_fd;
  unsigned char data_buffer[1];
  unsigned char *data_buffer_p;
  int data_buffer_size = 1;
  unsigned int read_len;
  FILE *fp;
  unsigned int offset;
  unsigned char keys[6];

  int i = 0;

  serial_fd = setup_port(serial_dev);
  if(serial_fd < 0) {
    printf("Error: failed to establish fd\n");
    return -1;
  }

  if((fp = fopen((const char *)filepath, "rb")) == NULL)
    return -1;

  sleep(2);

  data_buffer_p = data_buffer;
  offset = 0;
  while((read_len = fread(data_buffer_p, 1, data_buffer_size, fp)) > 0) {
    // have byte, pack it
    offset = pack_byte(keys, offset, data_buffer[0]);
    if(offset == 6) { // or filepos == file bytes
      // have full packet, send it.
      send_keyboard_packet(serial_fd, DATA_MODE, keys, offset - 1);
      offset = 0;
    }
  }
  if(offset) {
    printf("got %d bytes left\n", offset);
    send_keyboard_packet(serial_fd, DATA_MODE, keys, offset - 1);
  }
```

```
    close_port(serial_fd);

    return 0;
}
```

Note that platform "endian-ness" has not been factored in this code – it has been tested with x86 transmitters/receivers only.

4.1.1.2 Compiling the TKXf Transmit C Source

The following instructions assumes the above source code is saved as "tkxf-transmit.c" and resides in current working directory.

Note that this reference code does not include configurable command-line parameters. In order to configure the program, be sure to change the following entries as needed for your environment;

```
char * serial_dev = "/dev/ttyACM0";
char filepath[32] = "test.bin";
```

The following compiles the software from source;

```
gcc -o tkxf-transmit tkxf-transmit.c
```

4.1.1.3 Using the TKXf Transmit implementation

Once compiled the demonstrator application "tkxf-transmit" is capable of operating as a TKXf client (transmitter). But before you run the program, ensure that:

- The file referenced by "filepath" exists where stated (./test.bin if you've compiled with the defaults), and;
- The USB Serial adapter is connected to the computer, and that the serial device is accessible (/dev/ttyACM0 is read/write to your user ID, if you've retained the defaults).

WARNING: If you've assembled the entire project then running this command will pummel the target computer with keystrokes. Before you run this command, make sure those keystrokes are going to a device and application that you want it to (like a text editor, if not the receive software).

To launch the program, use the following command;

```
./tkxf-transmit
```

4.1.2 Keyboard Stuffer Device (Transmit Only)

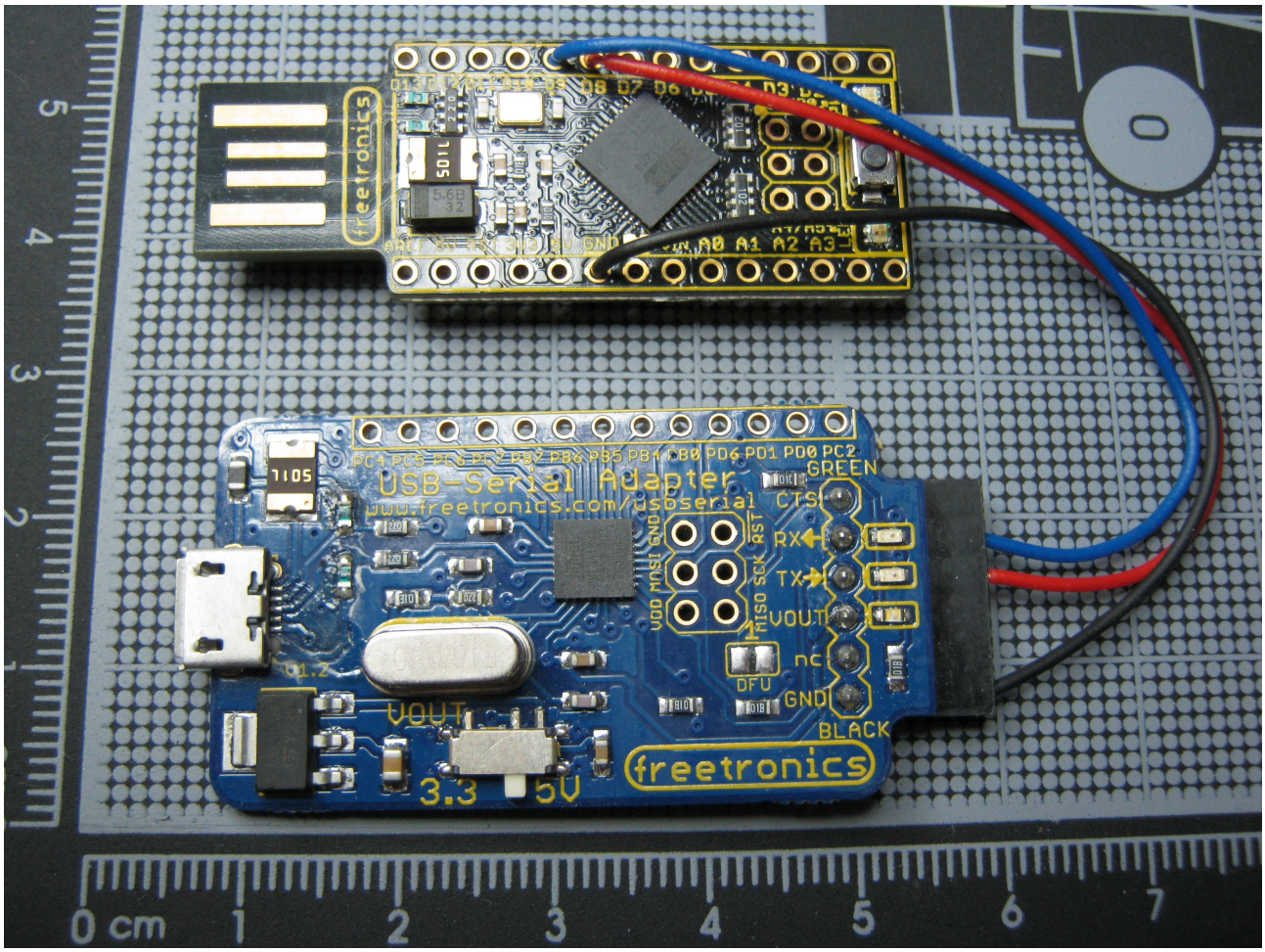
The Arduino Leonardo platform with a USB serial device is a simple example of the USB serial / Keyboard Stuffer solution. This section describes the reference hardware and software used to build the USB Serial and Keyboard Stuffer components.

4.1.2.1 Reference Hardware

The devices depicted are manufactured by Freetronics, and are available in hobby retail stores in a number of countries.

LeoStick (Arduino Compatible)	http://www.freetronics.com/collections/arduino/products/leostick
USB Serial Adapter	http://www.freetronics.com/collections/modules/products/usb-serial-adapter

Digital pin 8 on the Arduino is wired to USB serial transmit (TX). Digital pin 9 on the Arduino is wired to USB serial receive (RX). Ground on the Arduino is wired to USB serial ground (GND).



4.1.2.2 Acquiring the Arduino Software

To acquire the Arduino software for your platform, please visit the following URL and download the Arduino IDE;

```
http://arduino.cc/
```

4.1.2.3 The Arduino TKXf Patch

A change has been made to the Arduino libraries in order to expose the USB HID keyboard interface to the IDE. The following C code is in “patch” format.

```
--- arduino-1.5.6-r2/hardware/arduino/avr/cores/arduino/USBAPI.h-orig   2014-03-29
07:14:34.032627705 +1100
+++ arduino-1.5.6-r2/hardware/arduino/avr/cores/arduino/USBAPI.h        2014-03-29
07:15:00.272626988 +1100
@@ -136,8 +136,8 @@
 {
 private:
        KeyReport _keyReport;
-       void sendReport(KeyReport* keys);
 public:
+       void sendReport(KeyReport* keys);
        Keyboard_(void);
        void begin(void);
        void end(void);
```

Using this private method within the Arduino libraries enables substantially greater performance than using the native “Keyboard.write()” function, as well as exposing the other five key-press variables within the packet structure.

This patch has been made possible through the analysis and documentation published by PJRC and LogicalZero:

PJRC	http://www.pjrc.com/teensy/td_keyboard.html
LogicalZero (Stokes)	http://www.logicalzero.com/blog/?p=627

4.1.2.4 Applying the *Arduino* patch to the Arduino Libraries

Assuming the Arduino patch is called “arduino.patch” and resides in your home directory, and that you are in the “arduino” (i.e. “cd arduino-1.5.6-r2/” after downloading and unpacking the IDE) then the following command should apply the patch to the Arduino library;

```
patch -p1 < ~/arduino.patch
```

If the patch applies successfully, then move on to the reference code for the Arduino (the libraries will compile from the IDE build process for the Arduino project you will create).

4.1.2.5 The TKXf Transmit Arduino Source

The following C code is for tkxf-transmit.ino, an Arduino implementation of the TKXf protocol (transmit only).

```
/*

                                    _JNJ`
                                 .JNMH`
                                JMMF`       `;.
                               .NMM)          `MN.
                               MMM)            (MML
                              (MMM`             MMML
                      M  I  D  N  I  G  H  T    C  o  D  E
                              (NMMF             MHNH
                               NMML            .MMM
                                NMML           .NMH
                                 4MMNL        .#F
                                  `4HNNL_    `
                                     `""`

                           Copyright (C) 2004-2014
          "Ian (Larry) Latter" <ian dot latter at midnightcode dot org>

    Midnight Code is a registered trademark of Ian Latter.

    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, and mirrored at the Midnight Code web
    site; as at version 2 of the License only.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    General Public License for more details.

    You should have received a copy of the GNU General Public License
    (version 2) along with this program; if not, write to the Free
    Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
    02111-1307 USA, or see http://midnightcode.org/gplv2.txt

*/

#include <SoftwareSerial.h>

// User Preferences
int with_light = true;
int with_sound = true;
int start_delay = 5;    // x 1 second
int timeout = 10;       // x 10 ms
int debug = false;
int ack_delay = 1;      // x 1 ms
int end_on_timeout = false;

// Hardware Config
int led = 13;
int piezo = 11;
int PCpin_rxd = 8;
int PCpin_txd = 9;

// Operational Globals
int counter = -1;
unsigned int offset = 0;
unsigned int mode = 99;  // force initial dispatch of data mode
unsigned int processed_octets = 0;
byte keys[6];

SoftwareSerial PCserial(PCpin_rxd, PCpin_txd);
```

```
void
piezo_beep() {
  int frequency = 4000;

  for(int i = 0; i < ((frequency / 1000) * 400); i++) {
    digitalWrite(piezo, HIGH);
    delayMicroseconds(500000 / frequency);
    digitalWrite(piezo, LOW);
    delayMicroseconds(500000 / frequency);
  }

}


void
user_notify(unsigned int count) {

  for(int i = 0; i < count; i++) {
    if(with_light) {
      digitalWrite(led, HIGH);
    }
    if(with_sound) {
      piezo_beep();
    }
    if(with_light) {
      delay(250);
      digitalWrite(led, LOW);
      delay(250);
    }
  }

}


void
sendKeys(byte * keys, unsigned int max_offset) {
  unsigned int idx;

  if(debug == true) {
    char s[32];
    snprintf(s, 32, "[%x %x %x %x %x %x]",
      keys[0], keys[1], keys[2], keys[3], keys[4], keys[5]);
    Serial.println(s);
  }

  KeyReport report_down = { 0 };
  KeyReport report_up = { 0 };
  for(idx = 0; idx < max_offset; idx++) {
    report_down.keys[idx] = keys[idx];
  }
  report_down.modifiers = 0;
  report_down.reserved = 1;
  Keyboard.sendReport(&report_down);
  // delay(2);
  Keyboard.sendReport(&report_up);
  // delay(2);

  return;
}


void
sendControl(unsigned int type) {

  KeyReport report_down = { 0 };
  KeyReport report_up = { 0 };
  report_down.keys[0] = type;
  report_down.modifiers = 0;
  report_down.reserved = 1;
  Keyboard.sendReport(&report_down);
  Keyboard.sendReport(&report_up);
```

```
  return;
}


void setup() {

  // Init Hardware
  pinMode(led, OUTPUT);
  Keyboard.begin();
  PCserial.begin(38400);
  if(debug)
    Serial.begin(9600);

  delay(start_delay * 1000);
  // Flush PC serial port
  while(PCserial.available() > 0)
    PCserial.read();
  user_notify(3);
  delay(750);

}


void loop() {
  unsigned char octet;

  if(!PCserial.available()) {
    if(counter != -1) {
      if(debug)
        delay(1000);
      if(!counter)          // do once, when zero
        PCserial.write(1);
      delay(ack_delay);
      counter += ack_delay;
      if(counter >= timeout * 10) {
        if(offset) {
          sendKeys(keys, offset);
          offset = 0;
        }
        if(end_on_timeout)
          sendControl(0x37);  // . = end transfer
        counter = -1;
      }
    }
  } else {
    counter = 0;
    octet = PCserial.read();
    processed_octets++;
    switch(octet) {
      // Raw Data (keyboard encoded)
      case 0x27: // '0'
      case 0x1e: // '1'
      case 0x1f: // '2'
      case 0x20: // '3'
      case 0x21: // '4'
      case 0x22: // '5'
      case 0x23: // '6'
      case 0x24: // '7'
      case 0x25: // '8'
      case 0x26: // '9'
      case 0x04: // 'a'
      case 0x05: // 'b'
      case 0x06: // 'c'
      case 0x07: // 'd'
      case 0x08: // 'e'
      case 0x09: // 'f'
      // Deduped data (keyboard encoded)
      case 0x10: // 'm':
      case 0x11: // 'n':
      case 0x12: // 'o':
      case 0x13: // 'p':
```

```
        case 0x14: // 'q':
          keys[offset] = octet;
          offset++;
          if(offset == 6) {
            sendKeys(keys, offset);
            offset = 0;
          }
          break;
        case 0x37: // .
          if(offset)
            sendKeys(keys, offset);
          if(mode != 2) {
            sendControl(0x37); // .
            mode = 2;
          }
          offset = 0;
          break;
        case 0x36: // ,
          if(offset)
            sendKeys(keys, offset);
          if(mode != 1) {
            sendControl(0x36); // ,
            mode = 1;
          }
          offset = 0;
          break;
        case 0x2c: // space
          if(offset)
            sendKeys(keys, offset);
          if(mode != 0) {
            sendControl(0x2c); // space
            mode = 0;
          }
          offset = 0;
          break;
        default:
          offset = 0;
          break;
      }
    }
  }
```

Note that platform "endian-ness" has not been factored in this code – it has been tested with x86 transmitters/receivers only.

4.1.2.6 Compiling the TKXf Transmit Arduino Source

Connect the Arduino (LeoStick if you have acquired one) to the computer that will run the Arduino IDE. Then, start the Arduino IDE and ensure that it is configured for the correct Board and Serial Port. Start a new project, and paste the Arudino source code in from the above section.

Before you compile the code, customise it via the provided variables. Particularly note the "sound" option if you have acquired an Arduino Leo without a Piezo device onboard.

```
int with_light = true;
int with_sound = true;
```

Also check the pin assignments for your board to make sure that the software is looking to the right pins for the right devices.

```
// Hardware Config
int led = 13;
int piezo = 11;
int PCpin_rxd = 8;
int PCpin_txd = 9;
```

Press the “Play” button in the IDE to build both the Arduino source from above and the patched library code, and to have it installed on the attached Arduino device.

4.1.2.7 Using the TKXf Transmit implementation

Once compiled and deployed to the Arduino, the device is a fully constructed Keyboard Stuffer.

If you've enabled them, a light and matching beep/buzz will pulse three times before the device is read to operate as designed. This services three purposes;

- You have time to rewrite the device with a new image, if something goes drastically wrong and it spews keystrokes constantly*;
- You have time to remove the device from a target before any activity takes place, should you choose to change your mind*;
- Experiments suggested that the keyboard.write() code was more reliably initialised when given lead time, and that carried through to the sendReport() implementation (whether needed or not). The user notification provides an indication of when the device is ready.

* Note that the Stuffer has been designed to send no keyboard keys until data is received on the USB serial adapter.

The correct procedure for deploying the Keyboard Stuffer is to;

1. Connect the Stuffer into the target computer
2. Connect the USB Serial adapter into the client computer
3. Run the tkxf-receiver on the target computer
4. Run the tkxf-transmitter on the client computer, only when everything is ready.

4.1.3 C Reference Implementation (Receive Only)

The following C reference implementation was used to validate the end-to-end solution that has been documented in this specification.

4.1.3.1 The TKXf Receive C Source

The following C code is for tkxf-receive.c, a C implementation of the TKXf protocol (receive only).

```
/*
                                    _JNJ`
                                 .JNMH`
                                JMMF`        `;.
                               .NMM)          `MN.
                               MMM)            (MML
                              (MMM`             MMML
                      M  I  D  N  I  G  H  T    C  o  D  E
                              (NMMF              MHNH
                               NMML             .MMM
                                NMML           .NMH
                                 4MMNL        .#F
                                  `4HNNL_    `
                                     `""`

                           Copyright (C) 2004-2014
          "Ian (Larry) Latter" <ian dot latter at midnightcode dot org>

    Midnight Code is a registered trademark of Ian Latter.

    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, and mirrored at the Midnight Code web
    site; as at version 2 of the License only.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    General Public License for more details.

    You should have received a copy of the GNU General Public License
    (version 2) along with this program; if not, write to the Free
    Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
    02111-1307 USA, or see http://midnightcode.org/gplv2.txt

*/

#include <termios.h>
#include <string.h>
#include <unistd.h>
#include <stdio.h>


struct termios ubuf_term;
struct termios save_term;


void
setup_terminal() {

  tcgetattr(0, &save_term);
  memcpy(&ubuf_term, &save_term, sizeof(struct termios));
  ubuf_term.c_lflag &= ~(ICANON | ECHO);
  ubuf_term.c_cc[VTIME] = 0;
```

```
  ubuf_term.c_cc[VMIN] = 1;
  tcsetattr(0, TCSANOW, &ubuf_term);

  return;
}


void
restore_terminal() {

  tcsetattr(0, TCSANOW, &save_term);

  return;
}


// USB keyboad packet doesn't allow duplicates
//
// Proposed dedupe scheme is as follows;
//     0x10 = letter m, meaning "as per key[0]"
//     0x11 = letter n, meaning "as per key[1]"
//     0x12 = letter o, meaning "as per key[2]"
//     0x13 = letter p, meaning "as per key[3]"
//     0x14 = letter q, meaning "as per key[4]"
//
// So byte data -> 0x00,0x00,0x00
//   Encoded nibbles => 0x27,0x27,0x27,0x27,0x27,0x27
//   Deduped nibbles => 0x27,0x10,0x11,0x12,0x13,0x14
//
// Encode right to left, decode left to right.
int
redupe_key_buffer(unsigned char * key_buffer, unsigned int max_offset) {
  int reference_pos;
  int compare_pos;

  for(reference_pos = 0; reference_pos <= max_offset; reference_pos++) {
    for(compare_pos = reference_pos + 1;
      compare_pos <= max_offset; compare_pos++) {
        if(key_buffer[compare_pos] - 109 == reference_pos) {
          key_buffer[compare_pos] = key_buffer[reference_pos];
          break;
        }
    }
  }

  return 0;
}


void
dispatch_char(unsigned int mode, unsigned char uchar) {

  if(mode == 0) {
    fprintf(stdout, "%c", uchar);
    fflush(stdout);
  }
  return;
}


int
process_key_buffer(unsigned char * key_buffer, unsigned int max_offset,
  unsigned int mode) {
  unsigned char key_result[2];
  unsigned int offset;
  unsigned int key_state;

  redupe_key_buffer(key_buffer, max_offset);

  key_state = 0;
  for(offset = 0; offset < max_offset; offset++) {
    if(key_buffer[offset] >= '0' && key_buffer[offset] <= '9') {
```

```
      key_result[key_state] = key_buffer[offset] - 48;
    } else {
      if(key_buffer[offset] >= 'a' && key_buffer[offset] <= 'f') {
        key_result[key_state] = 10 + key_buffer[offset] - 97;
      }
    }
    if(key_state)
      dispatch_char(mode, key_result[0] * 16 + key_result[1]);
    key_state = !key_state;
  }

  return 0;
}


int
process_key_stream(int fd) {
  unsigned char user_key;
  unsigned int mode;  // 0 = data, 1 = control, 2 = exit
  unsigned int processed_octets;
  unsigned char key_buffer[6];
  unsigned int offset;

  mode = 0;
  offset = 0;
  processed_octets = 0;
  while(mode != 2) {
    if(read(fd, &user_key, 1) != 1) {
      fprintf(stderr, "Error, failed read on descriptor (STDIN)\n");
      return -1;
    }
    processed_octets++;
    switch(user_key) {
      case '.':
        if(offset) {
          fprintf(stderr, "Last %d bytes\n", offset);
          process_key_buffer(key_buffer, offset, mode);
        }
        mode = 2;
        offset = 0;
        break;
      case ',':
        if(offset)
          process_key_buffer(key_buffer, offset, mode);
        mode = 1;
        offset = 0;
        break;
      case ' ':
        if(offset)
          process_key_buffer(key_buffer, offset, mode);
        mode = 0;
        offset = 0;
        break;
      case '0': // HEX data
      case '1':
      case '2':
      case '3':
      case '4':
      case '5':
      case '6':
      case '7':
      case '8':
      case '9':
      case 'a':
      case 'b':
      case 'c':
      case 'd':
      case 'e':
      case 'f':
      case 'm': // Dedupe data
      case 'n':
      case 'o':
```

```
      case 'p':
      case 'q':
        key_buffer[offset] = user_key;
        offset++;
        if(offset == 6) {
          process_key_buffer(key_buffer, offset, mode);
          offset = 0;
        }
        break;
      default:
        offset = 0;
        break;
    }
  }
  return processed_octets;
}


int
main(void) {

  setup_terminal();
  process_key_stream(0);
  restore_terminal();

  return 0;
}
```

Note that platform "endian-ness" has not been factored in this code – it has been tested with x86 transmitters/receivers only.

4.1.3.2 Compiling the TKXf Receive C Source

Assuming the above source code is saved as “tkxf-receive.c” and resides in current working directory, then the following command should compile the source;

```
gcc -o tkxf-receive -c tkxf-receive.c
```

4.1.3.3 Using the TKXf Receive implementation

Once compiled the demonstrator application “tkxf-receive” is capable of operating as a TKXf server (receiver). There is no command line configuration available on the tool as provided. The following command can be used to receive a file;

```
./tkxf-receive > test.bin
```

Ensure that the receiver is up and running before you run the transmitter.

4.2 Wireless USB HID

Experiments with low-end Bluetooth Serial (rfcomm) devices were not successful. The low-end Bluetooth adapter devices do not support L2CAP and have a tendency to drop data even at short distances.

Two implementations that should work are as follows.

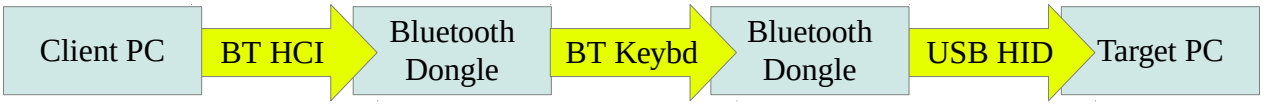
4.2.1 Custom Hardware – Bluetooth Shield on Arduino Leo

This implementation uses an L2CAP capable Bluetooth shield on an Arduino Leo capable board. In this model the USB Serial adapter from the wired implementation is replaced with a USB Bluetooth dongle and the Keyboard Stuffer is given a Bluetooth module that is capable of L2CAP.

One contender for the Keyboard Stuffer hardware platform is the BLEduino (http://bleduino.cc/), yet to be released.

4.2.2 Native Hardware, Custom Software

This implementation uses a Linux HID Keyboard client (software) that communicates via a USB Bluetooth dongle on the client PC to attach to a USB Bluetooth dongle on the target PC that has been configured (via HID2HCI) to boot as USB HID rather than as Bluetooth HCI infrastructure services.



This could be the most portable implementation.

4.3 Implementation Considerations

The following considerations should be evaluated by implementers;

1. **Robustness**
   In this version of the specification the advanced control features from the TGXf protocol have not been ported to the TKXf protocol. This means that there is no error detection and no ability to recover transfers.

2. **Emulating Multiple Devices**
   The Arduino Leo emulates a USB HID keyboard and a USB HID mouse simultaneously. There's no reason why both couldn't be used to increase the overall throughput. Similarly, modern Operating Systems will support multiple keyboards, so a variation based on more than one Keyboard Stuffer may also drastically increase throughput.

3. **Further Manipulation of the USB HID keyboard Interface**
   The USB HID keyboard interface is being under utilised. Half of all packets are being wasted in as an empty “key down” packet. It seems likely that any key not represented in the previous packet will be regarded as “down” by being absent in the subsequent packet. Using an operator (such as the “SHIFT” key) on every alternate USB HID keyboard packet could be sufficient to double the throughput of the protocol.

4. **Improved Encoding of Data**
   The two-byte hexadecimal representation of data is inefficient. There may be sufficient keys to use Base64 encoding, for example which would place three bytes in four key-presses.

5. **Other Future Enhancements**
   Compression and Encryption may be added in future implementations, but there is nothing stopping an integrator from compressing and encrypting source files before they are supplied to a TKXf sender.

5 Licensing

The following sections detail the licensing of the Thru-Keyboard Xfer specification and the components that comprise it.

5.1 Midnight Code Trademark

The term *Midnight Code* and the two half-moon mnemonic are registered trademarks of Ian Latter.

5.2 The Midnight Code Applications and libMidnightCode Library

All Midnight Code source code, including the libMidnightCode library and the Midnight Code applications are released with the following copyright, trademark and licensing terms.

```
/*
                                    _JNJ`
                                 .JNMH`
                                JMMF`        `;.
                               .NMM)          `MN.
                               MMM)            (MML
                              (MMM`             MMML
                      M  I  D  N  I  G  H  T    C  o  D  E
                              (NMMF             MHNH
                               NMML            .MMM
                                NMML           .NMH
                                 4MMNL        .#F
                                  `4HNNL_    `
                                     `"""`

                           Copyright (C) 2004-2014
              "Ian (Larry) Latter" <ian dot latter at midnightcode dot org>

    Midnight Code is a registered trademark of Ian Latter.

    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, and mirrored at the Midnight Code web
    site; as at version 2 of the License only.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    General Public License for more details.

    You should have received a copy of the GNU General Public License
    (version 2) along with this program; if not, write to the Free
    Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
    02111-1307 USA, or see http://midnightcode.org/gplv2.txt

*/
```

5.3 The Reference Code

The reference code is released under the GNU GENERAL PUBLIC LICENSE Version 2, June 1991.

```
                    GNU GENERAL PUBLIC LICENSE
                       Version 2, June 1991

 Copyright (C) 1989, 1991 Free Software Foundation, Inc.
                       51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                            Preamble

  The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change free
software--to make sure the software is free for all its users.  This
General Public License applies to most of the Free Software
Foundation's software and to any other program whose authors commit to
using it.  (Some other Free Software Foundation software is covered by
the GNU Library General Public License instead.)  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
this service if you wish), that you receive source code or can get it
if you want it, that you can change the software or use pieces of it
in new free programs; and that you know you can do these things.

  To protect your rights, we need to make restrictions that forbid
anyone to deny you these rights or to ask you to surrender the rights.
These restrictions translate to certain responsibilities for you if you
distribute copies of the software, or if you modify it.

  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must give the recipients all the rights that
you have.  You must make sure that they, too, receive or can get the
source code.  And you must show them these terms so they know their
rights.

  We protect your rights with two steps: (1) copyright the software, and
(2) offer you this license which gives you legal permission to copy,
distribute and/or modify the software.

  Also, for each author's protection and ours, we want to make certain
that everyone understands that there is no warranty for this free
software.  If the software is modified by someone else and passed on, we
want its recipients to know that what they have is not the original, so
that any problems introduced by others will not reflect on the original
authors' reputations.

  Finally, any free program is threatened constantly by software
patents.  We wish to avoid the danger that redistributors of a free
program will individually obtain patent licenses, in effect making the
program proprietary.  To prevent this, we have made it clear that any
patent must be licensed for everyone's free use or not licensed at all.

  The precise terms and conditions for copying, distribution and
modification follow.

                    GNU GENERAL PUBLIC LICENSE
   TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

  0. This License applies to any program or other work which contains
a notice placed by the copyright holder saying it may be distributed
under the terms of this General Public License.  The "Program", below,
refers to any such program or work, and a "work based on the Program"
means either the Program or any derivative work under copyright law:
that is to say, a work containing the Program or a portion of it,
either verbatim or with modifications and/or translated into another
language.  (Hereinafter, translation is included without limitation in
the term "modification".)  Each licensee is addressed as "you".
```

```
Activities other than copying, distribution and modification are not
covered by this License; they are outside its scope.  The act of
running the Program is not restricted, and the output from the Program
is covered only if its contents constitute a work based on the
Program (independent of having been made by running the Program).
Whether that is true depends on what the Program does.

  1. You may copy and distribute verbatim copies of the Program's
source code as you receive it, in any medium, provided that you
conspicuously and appropriately publish on each copy an appropriate
copyright notice and disclaimer of warranty; keep intact all the
notices that refer to this License and to the absence of any warranty;
and give any other recipients of the Program a copy of this License
along with the Program.

You may charge a fee for the physical act of transferring a copy, and
you may at your option offer warranty protection in exchange for a fee.

  2. You may modify your copy or copies of the Program or any portion
of it, thus forming a work based on the Program, and copy and
distribute such modifications or work under the terms of Section 1
above, provided that you also meet all of these conditions:

    a) You must cause the modified files to carry prominent notices
    stating that you changed the files and the date of any change.

    b) You must cause any work that you distribute or publish, that in
    whole or in part contains or is derived from the Program or any
    part thereof, to be licensed as a whole at no charge to all third
    parties under the terms of this License.

    c) If the modified program normally reads commands interactively
    when run, you must cause it, when started running for such
    interactive use in the most ordinary way, to print or display an
    announcement including an appropriate copyright notice and a
    notice that there is no warranty (or else, saying that you provide
    a warranty) and that users may redistribute the program under
    these conditions, and telling the user how to view a copy of this
    License.  (Exception: if the Program itself is interactive but
    does not normally print such an announcement, your work based on
    the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole.  If
identifiable sections of that work are not derived from the Program,
and can be reasonably considered independent and separate works in
themselves, then this License, and its terms, do not apply to those
sections when you distribute them as separate works.  But when you
distribute the same sections as part of a whole which is a work based
on the Program, the distribution of the whole must be on the terms of
this License, whose permissions for other licensees extend to the
entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest
your rights to work written entirely by you; rather, the intent is to
exercise the right to control the distribution of derivative or
collective works based on the Program.

In addition, mere aggregation of another work not based on the Program
with the Program (or with a work based on the Program) on a volume of
a storage or distribution medium does not bring the other work under
the scope of this License.

  3. You may copy and distribute the Program (or a work based on it,
under Section 2) in object code or executable form under the terms of
Sections 1 and 2 above provided that you also do one of the following:

    a) Accompany it with the complete corresponding machine-readable
    source code, which must be distributed under the terms of Sections
    1 and 2 above on a medium customarily used for software interchange; or,

    b) Accompany it with a written offer, valid for at least three
```

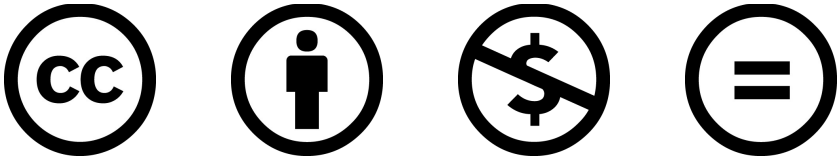
```
    years, to give any third party, for a charge no more than your
    cost of physically performing source distribution, a complete
    machine-readable copy of the corresponding source code, to be
    distributed under the terms of Sections 1 and 2 above on a medium
    customarily used for software interchange; or,

    c) Accompany it with the information you received as to the offer
    to distribute corresponding source code.  (This alternative is
    allowed only for noncommercial distribution and only if you
    received the program in object code or executable form with such
    an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for
making modifications to it.  For an executable work, complete source
code means all the source code for all modules it contains, plus any
associated interface definition files, plus the scripts used to
control compilation and installation of the executable.  However, as a
special exception, the source code distributed need not include
anything that is normally distributed (in either source or binary
form) with the major components (compiler, kernel, and so on) of the
operating system on which the executable runs, unless that component
itself accompanies the executable.

If distribution of executable or object code is made by offering
access to copy from a designated place, then offering equivalent
access to copy the source code from the same place counts as
distribution of the source code, even though third parties are not
compelled to copy the source along with the object code.

  4. You may not copy, modify, sublicense, or distribute the Program
except as expressly provided under this License.  Any attempt
otherwise to copy, modify, sublicense or distribute the Program is
void, and will automatically terminate your rights under this License.
However, parties who have received copies, or rights, from you under
this License will not have their licenses terminated so long as such
parties remain in full compliance.

  5. You are not required to accept this License, since you have not
signed it.  However, nothing else grants you permission to modify or
distribute the Program or its derivative works.  These actions are
prohibited by law if you do not accept this License.  Therefore, by
modifying or distributing the Program (or any work based on the
Program), you indicate your acceptance of this License to do so, and
all its terms and conditions for copying, distributing or modifying
the Program or works based on it.

  6. Each time you redistribute the Program (or any work based on the
Program), the recipient automatically receives a license from the
original licensor to copy, distribute or modify the Program subject to
these terms and conditions.  You may not impose any further
restrictions on the recipients' exercise of the rights granted herein.
You are not responsible for enforcing compliance by third parties to
this License.

  7. If, as a consequence of a court judgment or allegation of patent
infringement or for any other reason (not limited to patent issues),
conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License.  If you cannot
distribute so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you
may not distribute the Program at all.  For example, if a patent
license would not permit royalty-free redistribution of the Program by
all those who receive copies directly or indirectly through you, then
the only way you could satisfy both it and this License would be to
refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under
any particular circumstance, the balance of the section is intended to
apply and the section as a whole is intended to apply in other
circumstances.
```

```
It is not the purpose of this section to induce you to infringe any
patents or other property right claims or to contest validity of any
such claims; this section has the sole purpose of protecting the
integrity of the free software distribution system, which is
implemented by public license practices.  Many people have made
generous contributions to the wide range of software distributed
through that system in reliance on consistent application of that
system; it is up to the author/donor to decide if he or she is willing
to distribute software through any other system and a licensee cannot
impose that choice.

This section is intended to make thoroughly clear what is believed to
be a consequence of the rest of this License.

  8. If the distribution and/or use of the Program is restricted in
certain countries either by patents or by copyrighted interfaces, the
original copyright holder who places the Program under this License
may add an explicit geographical distribution limitation excluding
those countries, so that distribution is permitted only in or among
countries not thus excluded.  In such case, this License incorporates
the limitation as if written in the body of this License.

  9. The Free Software Foundation may publish revised and/or new versions
of the General Public License from time to time.  Such new versions will
be similar in spirit to the present version, but may differ in detail to
address new problems or concerns.

Each version is given a distinguishing version number.  If the Program
specifies a version number of this License which applies to it and "any
later version", you have the option of following the terms and conditions
either of that version or of any later version published by the Free
Software Foundation.  If the Program does not specify a version number of
this License, you may choose any version ever published by the Free Software
Foundation.

  10. If you wish to incorporate parts of the Program into other free
programs whose distribution conditions are different, write to the author
to ask for permission.  For software which is copyrighted by the Free
Software Foundation, write to the Free Software Foundation; we sometimes
make exceptions for this.  Our decision will be guided by the two goals
of preserving the free status of all derivatives of our free software and
of promoting the sharing and reuse of software generally.

                            NO WARRANTY

  11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY
FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW.  EXCEPT WHEN
OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES
PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED
OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.  THE ENTIRE RISK AS
TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU.  SHOULD THE
PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING,
REPAIR OR CORRECTION.

  12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR
REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES,
INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING
OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED
TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY
YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER
PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE
POSSIBILITY OF SUCH DAMAGES.

                     END OF TERMS AND CONDITIONS

            How to Apply These Terms to Your New Programs

  If you develop a new program, and you want it to be of the greatest
possible use to the public, the best way to achieve this is to make it
free software which everyone can redistribute and change under these terms.
```

```
  To do so, attach the following notices to the program.  It is safest
to attach them to the start of each source file to most effectively
convey the exclusion of warranty; and each file should have at least
the "copyright" line and a pointer to where the full notice is found.

    <one line to give the program's name and a brief idea of what it does.>
    Copyright (C) <year>  <name of author>

    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation; either version 2 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program; if not, write to the Free Software
    Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA


Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this
when it starts in an interactive mode:

    Gnomovision version 69, Copyright (C) year name of author
    Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License.  Of course, the commands you use may
be called something other than `show w' and `show c'; they could even be
mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the program, if
necessary.  Here is a sample; alter the names:

  Yoyodyne, Inc., hereby disclaims all copyright interest in the program
  `Gnomovision' (which makes passes at compilers) written by James Hacker.

  <signature of Ty Coon>, 1 April 1989
  Ty Coon, President of Vice

This General Public License does not permit incorporating your program into
proprietary programs.  If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with the
library.  If this is what you want to do, use the GNU Library General
Public License instead of this License.
```

5.4 This Document

When the Commercial-in-Confidence classification has been removed this document will be licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License.



5.5 Constituent Software

In addition to the Midnight Code applications and the libMidnightCode library, the following open source software or information has been referenced in this specification;

Arduino IDE	http://arduino.cc/