

ThruConsoleXfer (TCXf) White Paper

*Piano thieving for experts; That bathroom window **is** big enough.*



Classified: PUBLIC

Note that this document was re-classified and openly published to the public after the mobile apps were available in the vendor app stores and the COSAC/SABSA presentation in Ireland.

Project Code: "TGXf"

Status: RELEASE

Version: 1.1

Issue Date: 24 September 2014

Document Type: CONTROLLED



Table of Contents

Executive Summary.....	5
Abstract.....	6
Acknowledgements.....	7
1 How to use this document.....	8
1.1 Where to find	8
1.2 What does this mean?.....	9
1.2.1 Box.....	9
1.2.2 List.....	9
1.2.3 Link.....	9
1.3 Is it complete?.....	9
1.4 Why is it spelt like that?.....	9
2 Problem Space.....	10
2.1 Working Hypothesis.....	10
2.2 Testing the Supposition.....	10
3 Technology Solution.....	11
3.1 Data Extraction Through the Screen.....	11
3.1.1 X, Y and ZMODEM File Transfers (1977-1990s).....	11
3.1.2 Terminal Printing (1984).....	11
3.1.3 VHS Tape Backup (1992-1996).....	12
3.1.4 Timex Datalink (1994).....	12
3.1.5 Quick Response Codes (1994-2000-2006).....	13
3.1.6 QR Code as an Optical Packet Network (OSI Layer 3).....	14
3.1.7 Practical Considerations – Flow Control, Moving in the Ether.....	14
3.2 Through-Glass Transfer Transport Protocol Specification (TGXf).....	15
3.2.1 QR code (Denso Wave) as the underlying Packet Network.....	15
3.2.2 TGXf – Transport Protocol (OSI Layer 4).....	17
3.2.3 Structure of a Sample TGXf Transmission Sequence.....	22
3.2.4 Structure of a Sample TGXf Receipt Sequence.....	25
3.3 Through-Glass Transfer Reference Implementations (TGXf).....	26
3.3.1 PHP Reference Implementation – Transmit.....	26
3.3.2 C Reference Implementation – Receive.....	36
3.3.3 C Reference Implementation – Transmit.....	51
3.3.4 Reference Implementation Transmission Examples.....	65
3.3.5 Implementation Considerations.....	66
3.4 Uploading the Transmit Software (Reusing the First Principle is the Key).....	67
3.4.1 Digital Programmable Keyboard – Arduino Leonardo.....	67
3.4.2 What to Type?.....	68
3.4.3 BASH Script that Generates Arduino Keyboard Upload “.ino”	69
3.4.4 Generating and Compiling the Arduino Keyboard Upload Code.....	73
3.4.5 Performing the Upload.....	73
3.5 Milestone Achievement with TGXf and Keyboard Uploader.....	75
3.6 Through-Keyboard Transfer Protocol Specification (TKXf).....	76
3.6.1 USB HID as the underlying Packet Network.....	76
3.6.2 Structure of a TKXf Frame Sequence.....	82

3.7 Through-Keyboard Transfer Reference Implementations.....	84
3.7.1 Wired versus Wireless USB HID.....	84
3.7.2 C Reference Implementation – Transmit.....	86
3.7.3 Hardware Reference Implementation – Keyboard Stuffer Device – Transmit.....	92
3.7.4 C Reference Implementation – Receive.....	99
3.7.5 Implementation Considerations.....	103
3.8 Milestone Achievement with TKXf and TGXf.....	104
3.9 Data, take the Con (TCXf).....	105
3.9.1 Integration – TCXf Application Architecture.....	105
3.9.2 Protocol Specification Deltas for TGXf and TKXf.....	105
3.9.3 C Reference Implementation – Data Centre End (DCE).....	106
3.9.4 C Reference Implementation – Personal Computer End (PCE).....	121
3.9.5 Example Build Script.....	134
3.10 End-to-End Build and Test Process for TCXf.....	135
3.10.1 Establish the Test Bench.....	135
3.10.2 Building tcxf-pce.....	136
3.10.3 Building tcxf-dce.....	138
3.10.4 Building the Keyboard Stuffer.....	139
3.10.5 Preparing for Testing.....	140
3.10.6 Running the Tests – Reset Point.....	142
3.10.7 Test Case #1 – PCE to DCE Connectivity.....	143
3.10.8 Test Case #2 – DCE to PCE Connectivity.....	144
3.10.9 Test Case #3 – PCE to DCE File Transfer.....	145
3.10.10 Test Case #4 – DCE to PCE File Transfer.....	146
3.11 Milestone Achievement with TCXf.....	147
3.11.1 But I could send files before; Evolution to IP Networking.....	147
3.12 Enterprise Implications of this Technology.....	149
3.12.1 The Enterprise Context.....	149
3.12.2 Binary Data over Screen and Keyboard, Where's the Danger?.....	151
3.12.3 Abstraction of the Screen and Keyboard in Enterprise Architectures.....	152
3.12.4 Executing the Attack.....	153
3.12.5 Why wouldn't existing Enterprise Security Controls detect/prevent this?.....	154
3.12.6 So what's the Quick-Fix?.....	155
4 Architectural Analysis.....	156
4.1 Human versus Machine.....	156
4.1.1 Volume and Accuracy.....	156
4.1.2 Codify, Encipher versus Structure.....	157
4.1.3 Utility.....	158
4.2 Mitigation Strategies.....	159
4.2.1 Counter Culture.....	159
4.2.2 Forever War.....	160
4.3 Skipping to the Punch Line.....	161
4.4 Historical Considerations and Support.....	162
4.4.1 Anderson, 1972.....	162
4.4.2 Lampson, 1973.....	164
4.4.3 Vleck, 1976.....	165

PUBLIC
ThruConsoleXfer (TCXf) White Paper

4.4.4 US DoD Standard, 1983/1985.....	167
4.4.5 Further Reading.....	168
4.4.6 Impact of Historical Considerations.....	169
5 Framework Outcomes.....	170
5.1 Australian Legal Context.....	170
5.1.1 Australian Privacy Principles.....	170
5.1.2 Implications from this Solution.....	175
5.2 NIST Publications.....	176
5.2.1 Special Publication 800-46 Revision 1, Guide to Enterprise Telework and Remote Access Security.....	176
5.2.2 Special Publication 800-114, User's Guide to Securing External Devices for Telework and Remote Access.....	178
5.2.3 Special Special Publication 800-53 Revision 4, Security and Privacy Controls for Federal Information Systems and Organizations.....	179
5.2.4 Implications from this Solution.....	184
6 Other.....	186
6.1 Why did you publish this?.....	186
6.2 But why now?.....	186
6.3 About Midnight Code.....	187
7 Licensing.....	188
7.1 Midnight Code Trademark.....	188
7.2 The Midnight Code Applications and libMidnightCode Library.....	188
7.3 The Reference Code.....	188
7.4 This Document.....	193
7.5 Constituent Software.....	194

Executive Summary

This paper explores one example of how user controlled bits (keyboard input and screen output) can be re-imagined as a network interface.

The paper includes:

- Sufficient technical specifications and diagrams to explain the underlying technical principles, and;
- Source code to the software, and assembly instructions for the hardware, required to reproduce the outcomes described for assessment of the proposed technology solution in your own environment.

The paper goes on to evaluate:

- The architectural implications of human versus machine interfacing, and;
- One example of the legal impact of misunderstanding the subtle differences involved.

Abstract

ThruGlassXfer (TGXf) is a new and exciting technique to steal files from a computer through the screen with just a phone.

- No networking configuration needed.
- No portable storage devices needed.
- If you can see it, you can steal it.

ThruGlassXfer introduces major security implications to enterprises globally since it bypasses all major security controls including best practices Data Center and Perimeter deployments and even Data Loss Prevention and End Point security deployments. For enterprises off-shoring their IT: if you're working under a governance mandate of data sovereignty with data staying on-shore, then be aware that as of today, your off-shore users now have full access to copy files off your systems without a trace. The security model for protecting your data is incomplete and it is broken, right now.

Any user that has screen and keyboard access to a shell (CLI, GUI or even a Web Management shell) in your environment has the ability to transfer data, code and executables in and out of your environment without your knowledge, right now. This includes your staff, your partners and your suppliers, on and off-shore. And your implementation of best practice Data Center, Perimeter and End Point Security architectures have no effect on the outcome.

In this session I will take you from first principles to a full exploitation framework. You, as an audience member, will be able to actively participate in a live data theft (of simulated data) in complete anonymity using only your smart phone and an app from either the iPhone or Android app stores. I will be releasing the full specification for the ThruGlassXfer (TGXf) protocol and the reference C and PHP source code under the GPL for both CLI and Web platforms.

At the end of the session you'll learn how build on this unidirectional file transfer and augment the solution into a full duplex communications channel and then a native PPP link, from a user controlled device, through the remote enterprise-controlled screen and keyboard, to the most sensitive infrastructure in the enterprise.

This paper has been presented at the COSAC/SABSA World Congress (Ireland) conference in 2014, as "Piano Thieving for Experts, that bathroom window is big enough".

Acknowledgements

It is with much gratitude that I thank those who've contributed to this material, including:

- My wife and daughter who've forgone so much family time in Q1&2 CY14 while I've buried myself in this project.
- Disconnectionist (<http://disconnectionist.com/>) – James Hudson – the freelance software developer who ported the reference implementation to the Android and Apple platforms.
- Those who provided peer review, guidance and support;
 - Ian Krieger and Christopher Neal
- The management team of my current employer who have supported my personal bid to present the paper;
 - Chris and Mike
- Those whose debates on philosophy, electronics, technology and programming over café and pub tables in the past 20 years have contributed to this material;
 - Ben, Brian, Chris(2), Christoph, Gus, Ian, Marcus, Michael, Peter(2), Ray, Scott and Ty
- And to all of those on whose tall shoulders I have ridden to produce this material;
 - Denso Wave and Masahiro Hara (QR Code)
 - Kentaro Fukuchi (libqrencode)
 - Dominik Dzienia (PHP QR Code)
 - Jeff Brown (Zbar)
 - Stephen Hutchings (Typicons)
 - The rest of the community, particularly comments from;
 - PJRC and LogicalZero (keyboard)
 - Wallyk (serial)
 - GNU and the Free Software Foundation (compiler and supporting libraries)
 - Arduino (platforms and libraries)
 - IETF (RFC1952)
 - Freetronics (Arduino platforms and components)
- As well as anyone who's ever shared information.

1 How to use this document

This paper is a guide to assist people with the understanding and implementation – from first principles to example legal implications – of the Midnight Code Thru-Console Xfer solution. The document has been designed so that it can be read from start to finish like a story, or it can be used as a ready reference (via the expanded Table-of-Contents) to seek out targeted information, by concept.

1.1 Where to find ..

This document consists of seven 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 – Problem Space

Defining the problem from first principles.

Chapter 3 – Technology Solution

Build on historical examples and leverage existing technologies, this chapter defines new ethers and establishes protocols for communicating in them. It goes on to evolve a solution through a number of implementations and ends with a test bench that you can use to validate the assertions yourself with the reference implementations.

Chapter 4 – Architectural Analysis

This chapter abstracts the technology solution and looks to define the framework that would describe its affect. A review of previous research provides a foundation in the existing threat taxonomy.

Chapter 5 – Framework Outcomes

Example of the potential legal, regulatory and security framework impacts from changing the perception of the user controlled bit stream.

Chapter 6 – Other Information

Content that reviewers asked for which did not fit in another chapter.

Chapter 7 – Licensing

All of the licensing information from the technology solution to the paper itself has been published in this chapter.

If you're looking for a specific concept, you will find that this paper has been constructed to ensure that 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. Physically, this manual is 195 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.

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 Problem Space

The problem space is very simple, though I am going to ask you to think a little differently to see it.

2.1 Working Hypothesis

Think about the number of bits you change each day – starting with the snooze button on your alarm clock when you wake up, all the way through to turning the night light off before you go to sleep.

It seems odd, but then consider the following examples where you have already partaken in information transfer, such as a letter in an envelope, or a billboard beside a street, or a phone number on a toilet wall. And then consider some of the following digital examples, like watermarks in text white-space¹, covert transfers via unused bits in packet headers² and even simple channels like packing hidden data beyond the protocol defined end of file markers in JPG images and PDF files³. The abstraction of these examples is that if you can define content you can define a message.

Exploring that further, what if you tap your message out in Morse on a metal pipe, and what if that pipe tapping was the blinking light on an Arduino⁴? Each one of these could be used as a communications channel.

It is my assertion, therefore, that:

Any user controlled bit is a communications channel

Now consider that despite a massive enterprise security apparatus the user is considered the core of the machine – its all here to serve you – to enable you to safely change bits.

2.2 Testing the Supposition

Testing this supposition should be straight-forward. If any user controlled bit is truly a communications channel waiting to be developed, then let's pick off some universal human-machine interfaces and re-imagine them in ways that should horrify security practitioners.

We will start with the most obvious one. As a user I have a good deal of control over the screen and where there are many bits that the user has direct control.

The screen transmits large volumes of user controlled bits

Can we transform the screen into an uncontrolled binary file transfer medium?

1 <https://www.cs.auckland.ac.nz/courses/compsci725s2c/archive/termpapers/725ou.pdf>

2 <http://ojphi.org/ojs/index.php/fm/article/view/528/449>

3 <http://www.online-tech-tips.com/computer-tips/hidden-file-in-picture/>

4 <http://www.youtube.com/watch?v=ODm-b4R5X7E>

3 Technology Solution

In this chapter we will work through some relevant technical history, develop a technical solution and evolve that solution using the first principles (described in the previous chapter). At the end of this chapter you will be able to reproduce the solution and test it in your own enterprise.

3.1 Data Extraction Through the Screen

How are we going to get data out through the screen? You'd be surprised at how much has been done before – decades before.

3.1.1 X, Y and ZMODEM File Transfers (1977-1990s)

Anyone using Bulletin Board Systems to transfer files would be familiar with the XMODEM, YMODEM and/or ZMODEM transfer protocols⁵. XMODEM came first, created by Ward Christensen in 1977.

XMODEM allowed the virtual serial link that was the terminal session to temporarily facilitate a file transfer. In the most integrated case, the BBS host would emit a string that was recognised by the terminal client to initiate the transfer. Upon completion of the transfer, the host would display the BBS user menu content and the terminal client would return to displaying received data to the screen.

The X/Y/ZMODEM protocols are among the first that turn the “screen” into a multi-use data device. Although in this case the data was never displayed on the screen (it was intercepted by the client-side terminal software), this implementation demonstrates machine data transfer through the viewing channel (virtual terminal).

3.1.2 Terminal Printing (1984)

In the 1990's I was working in a data services provider which had half of its user population on local databases and half dialling in with 2400baud to 9600baud modems via multiplex serial boards terminated in the back of SCO servers at the provider side. The modem users were running ANSI capable terminal emulation software on both DOS and Windows PCs. This was an era of terminal access only – where users were able to access advanced features with server side software, such as Pine for email. But users could also print to a local (to the user's side/site) printer without a server-bound/managed print queue, through the terminal. The technique is simple and inherent to the terminal specification.

When a user elected to print a report, the server side software would send the terminal control/escape sequence to turn on print mode. All subsequently “displayed” content was sent to the local user-configured printer/print queue on the PC side. When the server side software sent the last of the report it would turn off print mode by sending the appropriate terminal control/escape

5 <http://www.techfest.com/hardware/modem/xymodem.htm>

sequence. If you're interested in the detail, please see chapter *4.14 Printing* in the DEC VT220 Programmer Reference Manual⁶ (DEC Document number EK-VT220-RM-002) from August 1984⁷.

Like the BBS file transfer protocols, terminal printing creates a multi-use data device from the virtual terminal.

3.1.3 VHS Tape Backup (1992-1996)

A friend and I were working in an electronics store part-time where he managed to find a dusty, abandoned, and unboxed VHS tape backup system based on an ISA expansion board. It was so old that it wasn't even a stock item for it on the microfiche, so the boss sold it to him for a random price to get it out of the store. I don't recall the make or model, but I've sourced some examples⁸ online⁹. The ISA board “played video” via a component (AV-out) video port that was connected to the VCR's AV-in port. It worked, we could backup data and restore it, so we took a closer look. When you connected it to a TV (or if you played the tape back to a TV instead of the ISA board) you could see a grey-scaled picture that contained two rows of eight blocks, where each of those eight blocks was comprised of another subset of blocks of various shades. It was obviously a bit encoding scheme designed to work at a lower resolution than the display space (i.e. not one bit per pixel but something substantially lower and presumably substantially easier to recognise in software).

This is the first example that I know of, where binary data has been transferred via video. I don't claim that this is the historically first instance, it was just the first one that us hobbyists could get for less than fifty bucks. Proof too that the binary file transfer concept is viable.

3.1.4 Timex Datalink (1994)

In 1994 the Timex Datalink watch was released with a wireless transfer mode¹⁰. I was blown away when a friend showed me how his Dad's watch could receive data by holding it up against the screen while a pattern of bars would swish up and down in front of it. In technical terms, a small optical sensor on the face of the watch allowed the watch to interpret the visible light as a data stream which it used to write to the flash memory inside the watch.

Although only officially supported by Timex and Microsoft on Windows 95 and Windows 98, open source code was later developed for the transmit side software¹¹. The author of that software (dfries) notes that this transfer method only works with CRT and not LCD displays due the inherent display methods of the two technologies. He now shows users how to build an LED driven from a Serial/USB port in order to program the original watches.

6 <http://vt100.net/docs/vt220-rm/chapter4.html#S4.14>

7 <http://manx.classiccmp.org/details.php/1,2960>

8 <http://en.wikipedia.org/wiki/ArVid>

9 <http://www.hugolypens.com/VBS.html>

10 <http://www.youtube.com/watch?v=fJiZXLiuAnU>

11 <http://datalink.fries.net/>

I don't recall the transfer rate from my youth but the current Wikipedia entry claims¹² it took 20 seconds to transfer 70 phone numbers, so it would hardly pass as exciting today, but it was a memorable experience none-the-less.

The Timex Datalink demonstrates that a display full of pixels are user defined bits that could send data that is either human readable or machine readable - a unidirectional communications channel through the screen.

3.1.5 Quick Response Codes (1994-2000-2006)

Also birthed to the world in 1994 was the QR (or Quick Response) code. In the 1960's Japanese cashiers were manually entering data at cash registers, resulting in many cashiers suffering from carpal tunnel syndrome. In response, Denso Wave developed 2D bar-codes that could convey machine-readable, Kanji-encoded, data¹³. I.e. they had higher storage capacity than existing single dimension codes.

Denso Wave looked to improve on the technology by increasing the recognition and scanning rate of their 2D bar-codes. Their exhaustive study of printed business materials led them to the iconic QR code structure that we recognise today which was both unique (distinguished from existing markings) and highly machine recognisable (permitting 360 degree scanning). QR Codes were soon adopted by the Japanese auto industry looking for a high-speed scanning technique that would allow them to improve efficiencies from production through to shipping process.

The QR Code was captured in the International Standard "ISO/IEC 18004:2000"¹⁴ and was revised in 2006 ("ISO/IEC 18004:2006"¹⁵) in both model 1 and model 2 formats. The model 2 format is the icing on the cake – it provides a method for reading deformed/distorted codes and provides a huge capacity improvement (up to ~4k octets) over the original, which already featured:

- rapid scanning capability, and;
- automatic re-orientation of the image, and;
- inherent error correction capability, and;
- native support for binary data.

In all, this combination establishes an interesting basis for a machine readable communications channel.

Further, if you recognise the heavily tried and tested libraries and the massive deployment (in industrial applications and cell phones globally, at least) then you see that we're starting from a very mature base also.

12 http://en.wikipedia.org/wiki/Timex_Datalink

13 <http://www.qrcode.com/en/history/>

14 http://raidennii.net/files/datasheets/misc/qr_code.pdf

15 http://www.iso.org/iso/catalogue_detail.htm?csnumber=43655

3.1.6 QR Code as an Optical Packet Network (OSI Layer 3)

What if we were to re-imagine the QR code, not as a single static encoding of text (or more typically a URL) but instead we consider the QR code as an optical packet captured within the ether of the display device? In this model, what we now have is a packet or datagram network protocol (layer 3 of the OSI Model¹⁶), where each displayed QR code is another packet transmitted from the screen.

3.1.7 Practical Considerations – Flow Control, Moving in the Ether

Recall that it is a design feature of the Quick Response Codes to be recognised and decoded rapidly by machine. Imagine then if the transmitter simply replaced one QR code for another, once per second.

Then imagine that a receiver, instead of using a photo to capture the image, used video and instead of stopping and exiting the program when it found one QR code, it just continued finding and decoding them indefinitely. Now we have the means to send a "flow of packets" from the screen sender, out to you the receiver.

Here we create a new problem. There is no synchronisation (flow control) between the sender and the receiver, so how does the sender in a unidirectional protocol know when the receiver has successfully recognised and decoded the packet? It can't. The only way to be confident that the receiver has got the packet is to ensure that the receiver over-samples the transmitter. That is to say that if the transmitter shows a QR code for 1 second, then the receiver must be capable of recognising and decoding that packet at least twice - or once per half second.

In solving one problem we create another. If we ensure that the receiver is over-sampling the transmission, then we are guaranteed to receive the same packet twice or more, and more often than not. How then do we reliably deduplicate the packet flow? Or worse; how do we stop existing QR code libraries from automatically deduplicating the flow, thinking that they've seen the same QR code image, when we're expecting them to be repeating data in transferred content (as seen in the internal structure of JPG or PDF files for example)?

We can't (won't) change the QR code specification because it's an international standard. What we really need is a complementary transport protocol.

16 http://en.wikipedia.org/wiki/OSI_model#Description_of_OSI_layers

3.2 Through-Glass Transfer Transport Protocol Specification (TGXf)

The TGXf 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, interrupted transfers and error detection.

3.2.1 QR code (Denso Wave) as the underlying Packet Network

This simple TGXf transport protocol has been built to consume the Denso Wave's Quick Response Code (QR code) as an optical packet (datagram) network protocol, but could be equally transferred via any packet protocol. The optical nature of the QR code and the unidirectional data flow means that one sender can communicate to one or more receivers without interference.

3.2.1.1 Requirements of the QR code Configuration

In technical terms, the QR code configuration required for implementation consistency is:

- QR code versions 1 (21x21), 2 (25x25), 8 (49x49) and 15 (77x77), and;
- Binary (or 8 bit) encoding, and;
- 15% error correction (M).

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

3.2.1.2 QR code Configuration and Effects on Capacity

Despite being defined as a percentage, in practice the error correction method (ECC) in the QR code protocol is not a strict percentage of the payload size in Binary (8 bit) mode, and will fluctuate dependant upon the data encoded. This creates a problem where some QR code encoder implementations will automatically size the QR code to fit the data, allowing the ECC rounding to produce a larger QR code by resolution, should the requested version be insufficient. Given the need to size the QR code by screen resolution (rather than data capacity) a rounding factor has been used to under-size the QR code rather than risk a variation in display size.

Thus, in order to maximise protocol independence (between the packet/datagram QR code layer and the transmission TGXf layer) four bytes have been subtracted from each QR code version's capacity, per the following table;

Version	Mode	ECC	Specified Capacity per Frame	Reliable Capacity per Frame
1	Binary	M (15%)	14 bytes per frame	10 bytes per frame
2	Binary	M (15%)	26 bytes per frame	22 bytes per frame
8	Binary	M (15%)	152 bytes per frame	148 bytes per frame
15	Binary	M (15%)	412 bytes per frame	408 bytes per frame

Table 1: TGXf QR code Configuration Matrix

3.2.1.3 Advantages and Limitations

The advantages of the QR code as a packet protocol are:

- Native error correction, and;
- Support for binary payloads, and;
- as a common protocol, it is already implemented on many platforms (highly portable).

The limitations of the QR code as a packet protocol are;

- Small packet sizes borne out of low resolution displays and cameras, and;
- Low transfer rate (packets per second) due to the low frame rate of cameras, and;
- High overhead of using an optical (image based) carriage.

3.2.1.4 Packet Rate (Frames or Packets per Second)

The practical limitation to the number of packets per second that a QR code packet network will support is the governed primarily by two factors;

- Display rate of the transmission device
- Capture rate of the receiving device

The rendering of a frame on the transmission device may be governed by drawing artefacts, rendering method and latency between the computation and the display of the image. On the other hand, the capture rate must allow for these issues in the display process and should permit multiple passes (read attempts) of the packet.

Generally, displays now exceed 30fps, while this is the upper limit of consumer camera performance. A Packet rate of 10pps would allow for at least one but an average of two reads per Packet (see Nyquist Rate¹⁷), balancing throughput with forgiveness in display accuracy.

3.2.1.5 Scale

It is recommend that transmit implementations render QR codes at a scale of at least 1:3. That is, for every pixel in the raw/native QR code it should be rendered at least 3 pixels wide and 3 pixels tall.

This will vary based on the DPI scale of the transmission platform. Practical implementations will scale the rendered QR codes to the maximum display space.

17 http://en.wikipedia.org/wiki/Nyquist_rate#Nyquist_rate_relative_to_sampling

3.2.2 TGXf – Transport Protocol (OSI Layer 4)

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

Frame Structure

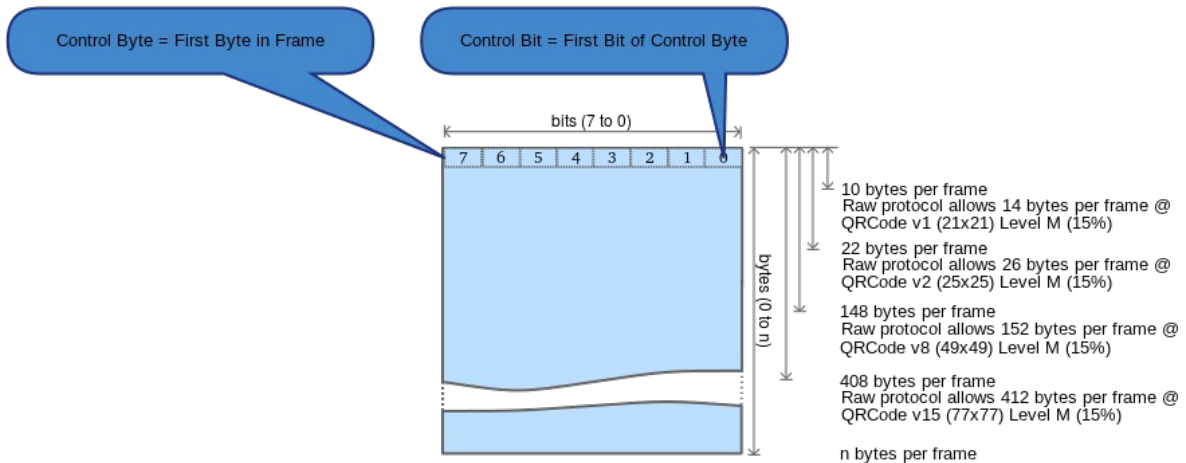


Illustration 1: TGXf Frame Structure

As can be seen in the diagram above, frame sizes vary dependent on the underlying packet network.

The TGXf Frame contains a header that describes what kind of frame it is.

The first byte (octet) of the TGXf frame is the Control Byte. The first bit – the least significant bit (LSB) – is the Control Bit. The Control Bit determines whether the Frame is a Control Frame or a Data Frame.

3.2.2.1 Frame Size

The TGXf protocol does not specify a maximum Frame size, but it does require a minimum capacity of 10 octets per Frame (a limitation of the Control Frame). This consists of 1 octet for the Control Byte and 9 octets for Control payload.

Also, although the TGXf supports an almost unlimited number of Frame sizes, any given TGXf transfer must use a fixed Frame size. i.e. the bytes per Frame in each Data Frame must be identical throughout the transfer.

3.2.2.2 Structure of a Data Frame

A Data Frame consists of one Control Byte (one octet) followed by a Data Payload.

3.2.2.2.1 Control Byte

The Control Byte (one octet) in the Data Frame is comprised as follows;

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

- Bit 0: Control Bit is always 0 (This Frame is a Data Frame)
- Bits 1-4: Counter (Cycles through Frames 0 to 15 and repeats)
- Bits 5-7: Reserved (Must be set to 0)

3.2.2.2.2 Significance of the Counter

The counter is required to provide the means to:

- Over-sample the optical packet flow in the receiver (detecting and rejecting duplicate packets), and;
- Enable binary data flows to be application independent by allowing the transmission of repeating data packets without disregarding them as duplicates, and;
- Allow the receiver to detect lost/dropped Data Frames.

Where multiple reads of the same Data Frame have been identified, the specification does not prescribe which which is preferred (first, last, or an instance between).

3.2.2.2.3 Data Payload

Except for the last Data Frame in a given transmission, the Data Payload will consume the remainder of the packet, regardless of the packet size.

For the last Data Frame in a given transmission, the Data Payload will contain the remainder of the source data octets and be padded with binary 0's. That is to say that the last Data Frame's Data Payload, whilst sized to the underlying packet size, will contain precisely:

“Source Bytes” modulo “Data Payload Capacity Bytes”

3.2.2.3 Structure of a Control Frame

A Control Frame consists of a Control Byte (one octet) and a Control Payload (twenty octets).

3.2.2.3.1 Control Byte

The Control Byte (one octet) in the Control Frame is comprised as follows;

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

- Bit 0: Control Bit is always 1 (This Frame is a Control Frame)
- Bits 1-3: Control Type (see below)
- Bits 4-7: Control Sub-Type (see below)

3.2.2.3.2 Control Types and Sub-Types

The following table provides an overview of the Control Types and Control Sub-Types set out in the TGXf protocol:

Control Type (Bits 1-3)	Control Sub-Type (Bits 4-7)	Label	Function
000 (0)	(any)	Reserved	Unused / Future Use
001 (1)	0001 (1)	START/FILENAME	Name of source data
	0010 (2)	START/FILESIZE	Length of source data (octets)
	0011 (3)	START/QRCODE_VERSION	QR code version
	0100 (4)	START/QRCODE_FPS	QR code frames per second
	0101 (5)	START/QRCODE_BYTES	QR code octets per frame
010 (2)	0001 (1)	STOP/PAUSE	Transmission paused
	0010 (2)	STOP/COMPLETE	Transmission completed
	0011 (3)	STOP/CANCEL	Transmission cancelled
011 (3)	0001 (1)	STATUS/SINCE	Status since last status
100 (4)	(any)	Reserved	Unused / Future Use
101 (5)	(any)	Reserved	Unused / Future Use
110 (6)	(any)	Reserved	Unused / Future Use
111 (7)	(any)	Reserved	Unused / Future Use

Table 2: TGXf Control Types and Sub-Types

The TGXf control detail has been defined by Control Types and the corresponding Control Sub-Types as follows:

- **001 (1) START**

The Control Type START is used to define the file or data that is about to be transferred. START Control Frames must only appear before the first Data Frame. Where a receiver observes multiple START Control Frames with the same Sub-Type (i.e. FILENAME) then most recently observed value will be regarded as the correct value for this transmission.

- 0001 (1) FILENAME

This is the name of the file or data that is to be transferred. The Control Payload will be a NULL padded ASCII string containing the (potentially shortened) file name (maximum of 9 octets). The file name is provided for both session management and seamless (sender to receiver) user interaction.

- 0010 (2) FILESIZE

This is the size (in bytes/octetets) of the file or data that is to be transferred. The Control Payload will be a 16bit (two octet) value representing the size of the source data/file. The file size is provided to allow the receiver to store the correct number of octets from the data stream (recall that the final Data Frame contains a NULL-padded Data Payload). The file size can also be used to provide a “progress bar” or other transfer time estimate.

- 0011 (3) QRCODE_VERSION

This is the version of the QR code that the sender has used to encode the transmission. The only values currently accepted are 1, 2, 8 and 15. The Control Payload will be a 16bit (two octet) value representing the QR code version number. The QR code version number can be used to provide a “progress bar” or other transfer time estimate.

- 0100 (4) QRCODE_FPS

This is the average number of QR code frames that the sender will display per second for the duration of the transmission. The only values currently accepted are 1, 2, 5, 8 and 10. The Control Payload will be a 16bit (two octet) value representing the QR code frames per second number. The QR code FPS number can be used to provide a “progress bar” or other transfer time estimate, as well as selecting the appropriate camera frame rate and reasonable buffer sizes.

- 0101 (5) QRCODE_BYTES

This is the maximum number of bytes (octets) that the sender will encode in each/any QR code frame in this transmission. The Control Payload will be a 16bit (two octet) value representing this maximum number (in octets). The QR code BYTES number can be used to provide a “progress bar” or other transfer time estimate, as well as selecting reasonable buffer sizes.

- **010 (2) STOP**

The Control Type STOP is used to mark the end of the transmission. Where a receiver observes a non-terminating STOP Sub-Type (i.e. PAUSE) the receiver must allow the user to resume the transfer from its current position. STOP Control Frames may appear at any time in the transmission, after the first START Control Frame.

- 0001 (1) PAUSE

This indicates that the transfer has momentarily stopped. There is no payload for this Control Sub-Type, and there is no minimum or maximum duration for the existence of the PAUSE state. This is a non-terminating Sub-Type. The sender can continue a paused transfer with any Data Frame or any non-PAUSE Control Frame.

- 0010 (2) COMPLETE

This Control Sub-Type provides two functions. First it indicates that the transfer has been completed successfully from the sender's perspective, and that the transmission has been terminated. Second, it provides a 32bit (four octet) CRC32 calculation in the Control Payload, for the complete source data. The receiver must use this Control Frame for both functions; terminating the transfer and validating the transfer by independently calculating the CRC32 for the received data and comparing that value to the transmitted CRC32.

- 0011 (3) CANCEL

This indicates that a transmission has been terminated without all of the source data being transmitted. The Control Payload will be a NULL padded ASCII string containing the reason (maximum of 9 octets) for the early termination. The detail has been provided for informative user interaction.

- **011 (3) STATUS**

The Control Type STATUS is used to provide a checkpoint in the progress of the transmission. STATUS Control Frames may appear at any time in the transmission, after the first START Control Frame.

- 0001 (1) SINCE

This marks the end of an arbitrary section (block) of source data (in terms of a number of Data Frames) since either the START of the transmission or the last STATUS/SINCE Control Frame. It provides a 32bit (four octet) CRC32 calculation in the Control Payload, for the last "block" of source data. The receiver must use this Control Frame to validate the transfer by independently calculating the CRC32 for the received data and comparing that value to the transmitted CRC32. The receiver inform the user of the number of failed blocks and provide the user with a list of block numbers (beginning at zero) which have failed.

Note that any Control Type or Sub-Type not explicitly defined in this specification is to be regarded as *Reserved for future* use; Reserved values must not be used in transmission and must be ignored in reception.

3.2.2.3.3 Control Payload

The Control Payload will consume a minimum of zero octets and not exceed 9 octets.

The contents of the Control Payload is defined by the Control Type and Control Sub-Type.

3.2.3 Structure of a Sample TGXf Transmission Sequence

At the conceptual level the pseudo-code for the Transmission Sequence is as follows;

1. Open the file
2. Send the Start Control Frames
 - (a) Encode the file name and render the QR code
 - (b) Encode the file size and render the QR code
 - (c) Encode the bytes per frame and render the QR code
 - (d) Encode the frames per second and render the QR code
3. Send the Data Frames
 - (a) For each bytes-per-frame (for the length of the file);
 - i. Read bytes-per-frame bytes from the file
 - ii. Encode the data and render the QR code
 - iii. Update the Frame counter display
 - iv. Update the Time Estimate display
 - v. Update the Progress Bar
4. Send the Stop Control Frame
 - (a) Calculate the checksum for the file
 - (b) Encode the checksum and render the QR code
5. Close the file, and allow the user to exit the program or run it again

3.2.3.1 Transmission Sequence Worked Example

Lets look at a basic “Hello World!” example.

In this example, Start Control Frames have been rendered in Green on White, Data Frames have been rendered in Black on White and Stop Control Frames have been rendered in Red on White. In production, all Frames would be preferably Black on White.

3.2.3.1.1 START Control Frames

First, the sender will define the transfer for the receiver. This is a 13 byte file called “./helloworld.txt” being sent in a version 8 QR code at 5 frames per second.

START/FILENAME

- Set the Control Bit to Control (1)
- Set the Control Type to START (1)
- Set the Control Sub-Type to FILENAME (1)
- Set the Control Payload to “helloworld”
- Encode the Frame as a QR code datagram



START/FILESIZE

- Set the Control Bit to Control (1)
- Set the Control Type to START (1)
- Set the Control Sub-Type to FILESIZE (2)
- Set the Control Payload to 13 octets
- Encode the Frame as a QR code datagram



START/QRCODE_BYTES

- Set the Control Bit to Control (1)
- Set the Control Type to START (1)
- Set the Control Sub-Type to QRCODE_BYTES (5)
- Set the Control Payload to 148 octets
- Encode the Frame as a QR code datagram



START/QRCODE_FPS

- Set the Control Bit to Control (1)
- Set the Control Type to START (1)
- Set the Control Sub-Type to QRCODE_FPS (4)
- Set the Control Payload to 5 octets
- Encode the Frame as a QR code datagram



3.2.3.1.2 Data Frames

Now the sender can transmit the Data Frames, which for this example is one Data Frame, which also makes it the last Data Frame.

- Set the Control Bit to Data (0)
- Set the Counter to the first frame (0)
- Set the Data Payload to “Hello World!”
- Encode the Frame as a QR code datagram



3.2.3.1.3 STOP Control Frames

The data transmission is now complete, so the sender can send out a completion message.

STOP/COMPLETE

- Set the Control Bit to Control (1)
- Set the Control Type to STOP (2)
- Set the Control Sub-Type to COMPLETE (2)
- Set the Control Payload to the CRC32 of the file contents
- Encode the Frame as a QR code datagram



3.2.4 Structure of a Sample TGXf Receipt Sequence

At the conceptual level the pseudo-code for the Receiving Sequence is as follows;

1. Open the video camera
2. Search each video frame for a QR code and decode it
3. Get the Control Bit
 - (a) If it is a Control Frame
 - i. If it is a Start Control Frame
 - A. If Data Frames have been received then ignore
 - B. Manage the meta data or validate the received data
 - C. Continue to next frame; per 2.
 - ii. If it is a Stop Control Frame
 - A. If it is a Complete/Cancel code, stop searching the video data
 - B. If it is a checksum, validate the file accordingly
 - C. Close the video camera, continue per 5.
 - (b) If it is a Data Frame
 - i. Ensure sufficient meta data exists to begin the transfer
 - ii. Extract the frame counter
 - A. Ignore the frame if its a duplicate
 - B. If frames are missing, add to the error counter (once per missed frame)
 - C. If the error count is too high, close the video camera, and continue per 4.
 - iii. Write data to file at correct offset
 - iv. Continue to next frame; per 4.
4. Update the display;
 - (a) Update the camera picture display
 - (b) Update the Frame counter display
 - (c) Update the Time Estimate display
 - (d) Update the Progress Bar
 - (e) Update the Error counter display
 - (f) Continue to next frame; per 2.
5. If there were noted failed blocks, report the user the block numbers
 - (a) Otherwise, save the file where the user can access it
6. Allow the user to exit the program or run it again

3.3 Through-Glass Transfer Reference Implementations (TGXf)

The transmit side of Through-Glass Transfer is able to be implemented on any compute platform with a display. The receive side of Through-Glass Transfer requires a camera to perceive the images being dispatched by the transmit side.

This section includes reference implementations that, while far from pretty, are functional demonstrators of the TGXf protocol.

3.3.1 PHP Reference Implementation – Transmit

The following PHP reference implementation was used to generate the test cases that have been supplied with this specification.

```
<?php
/*
    _JNJ`
  .JNMH`
 JMMF`  `;.
.NMM)   `MN.
MMM)    (MML
(MMM`   MMLL
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

*/

// TGXf Protocol Constants
$txgf_control_type = array();
$txgf_control_type["START"]["value"] = 1;
$txgf_control_type["START"]["FILENAME"]["value"] = 1;
$txgf_control_type["START"]["FILESIZE"]["value"] = 2;
$txgf_control_type["START"]["QRCODE_VERSION"]["value"] = 3;
$txgf_control_type["START"]["QRCODE_FPS"]["value"] = 4;
$txgf_control_type["START"]["QRCODE_BYTES"]["value"] = 5;
$txgf_control_type[1]["name"] = "START";
$txgf_control_type[1]["sub_type"][1] = "FILENAME";
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
$tgxf_control_type[1]["sub_type"][2] = "FILESIZE";
$tgxf_control_type[1]["sub_type"][3] = "QRCODE_VERSION";
$tgxf_control_type[1]["sub_type"][4] = "QRCODE_FPS";
$tgxf_control_type[1]["sub_type"][5] = "QRCODE_BYTES";
$tgxf_control_type["STOP"]["value"] = 2;
$tgxf_control_type["STOP"]["PAUSE"]["value"] = 1;
$tgxf_control_type["STOP"]["COMPLETE"]["value"] = 2;
$tgxf_control_type["STOP"]["CANCEL"]["value"] = 3;
$tgxf_control_type[2]["name"] = "STOP";
$tgxf_control_type[2]["sub_type"][1] = "PAUSE";
$tgxf_control_type[2]["sub_type"][2] = "COMPLETE";
$tgxf_control_type[2]["sub_type"][3] = "CANCEL";
$tgxf_control_type["STATUS"]["value"] = 3;
$tgxf_control_type["STATUS"]["SINCE"]["value"] = 1;
$tgxf_control_type[3]["name"] = "STATUS";
$tgxf_control_type[3]["sub_type"][1] = "SINCE";

// TGXf Session Parameters (TGXf Global Options)
$tgxf_session_filepath;
$tgxf_session_file_size = 0;
$tgxf_session_qrcode_version = 0;
$tgxf_session_qrcode_fps = 0;
$tgxf_session_qrcode_bytes = 0;

// Include the supporting libraries;
// PHP QRcode: http://phpqrcode.sourceforge.net/
// Gif Creator: https://github.com/Sybio/GifCreator
include('phpqrcode/qrlib.php');
include('GifCreator.php');

function get_basename_for_file($filepath) {
    return pathinfo($filepath, PATHINFO_BASENAME);
}

function get_size_for_file($filepath) {
    return filesize($filepath);
}

function get_crc_for_file($filepath) {
    $file_content = file_get_contents($filepath);
    $crc = crc32($file_content);

    return $crc;
}

function debug_control_byte($var, $lbl="DEBUG", $num=false) {
    if($num) {
        $b = decbin($var);
    } else {
        $b = decbin(ord($var));
    }
    print(" " . $lbl . " :[" . substr("00000000",0,8 - strlen($b)) . $b . " ] ");
    return;
}

function get_control_bit($data) {
    $control_byte = ord($data[0]);
    $control_bit = $control_byte & 1;

    return $control_bit;
}

function get_control_type($data) {
    $control_byte = ord($data[0]);
    $control_type = ($control_byte & 14) >> 1;
}
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
    return $control_type;
}

function pack_control_payload_16bit_number($data) {
    $payload = pack('C2', ($data >> 8) & 0xFF, ($data >> 0) & 0xFF);

    return $payload;
}

function pack_control_payload_32bit_number($data) {
    $payload = pack('C4',
        ($data >> 24) & 0xFF,
        ($data >> 16) & 0xFF,
        ($data >> 8) & 0xFF,
        ($data >> 0) & 0xFF);

    return $payload;
}

function pack_control_payload_string($data) {
    $payload = NULL;
    $string_len = strlen($data);
    if($string_len > 9)
        $string_len = 9;
    for($idx = 0; $idx < 9; $idx++) {
        $payload .= "\0";
    }
    for($idx = 0; $idx < $string_len; $idx++) {
        $payload{$idx} = $data{$idx};
    }

    return $payload;
}

function pack_data_payload_bin($data, $len) {
    $payload = NULL;
    for($idx = 0; $idx < $len; $idx++) {
        $payload .= $data[$idx];
    }

    return $payload;
}

function build_control_frame($control_type, $control_subtype, $data=NULL) {
    global $tgxf_control_type;

    // Zero Control Byte
    $control_byte = 0;

    // Control Frame, Control Bit = 1 (bit 0);
    $control_bit = 1;
    $control_byte = $control_byte | $control_bit;

    // Control Byte has Control Type (bits 3,2,1)
    if($control_type < 0 || $control_type >= 7)
        $control_type = 0;
    $control_type = $control_type << 1;
    $control_byte = $control_byte | $control_type;

    // Control Byte has Sub-Control Type (bits 7,6,5,4)
    if($control_subtype < 0 || $control_subtype >= 15)
        $control_subtype = 0;
    $control_subtype = $control_subtype << 4;
    $control_byte = $control_byte | $control_subtype;

    $payload = NULL;
    switch(($control_type & 14) >> 1) {
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
case $tgxf_control_type["START"]["value"]:
  switch(($control_subtype & 240) >> 4) {
    case $tgxf_control_type["START"]["FILENAME"]["value"]: // FILENAME
      $payload = pack_control_payload_string($data);
      break;
    case $tgxf_control_type["START"]["FILESIZE"]["value"]: // FILESIZE
      $payload = pack_control_payload_16bit_number($data);
      break;
    case $tgxf_control_type["START"]["QR코드_VERSION"]["value"]: // QR코드_VERSION
      $payload = pack_control_payload_16bit_number($data);
      break;
    case $tgxf_control_type["START"]["QR코드_FPS"]["value"]: // QR코드_FPS
      $payload = pack_control_payload_16bit_number($data);
      break;
    case $tgxf_control_type["START"]["QR코드_BYTES"]["value"]: // QR코드_BYTES
      $payload = pack_control_payload_16bit_number($data);
      break;
  }
  break;

case $tgxf_control_type["STOP"]["value"]:
  switch(($control_subtype & 240) >> 4) {
    case $tgxf_control_type["STOP"]["PAUSE"]["value"]: // PAUSE
      break;
    case $tgxf_control_type["STOP"]["COMPLETE"]["value"]: // COMPLETE
      $payload = pack_control_payload_32bit_number($data);
      break;
    case $tgxf_control_type["STOP"]["CANCEL"]["value"]: // CANCEL
      $payload = pack_control_payload_string($data);
      break;
  }
  break;

case $tgxf_control_type["STATUS"]["value"]:
  switch(($control_subtype & 240) >> 4) {
    case $tgxf_control_type["STATUS"]["SINCE"]["value"]: // SINCE
      $payload = pack_control_payload_32bit_number($data);
      break;
  }
  break;

default:
  break;
}

// Frame consists of Control Byte and Control Payload
$buffer = pack('C', ($control_byte) & 0xFF) . $payload;

return $buffer;
}

function build_data_frame($counter, $data, $payload_size) {

  // Zero Control Byte
  $control_byte = 0;

  // Data Frame, Control Bit = 0 (bit 0);
  $control_bit = 0;
  $control_byte = $control_byte | $control_bit;

  // Control Byte has incremented counter (bits 4,3,2,1)
  if($counter < 0 || $counter > 15)
    $counter = 0;
  $counter = $counter << 1;
  $control_byte = $control_byte | $counter;

  // Frame consists of Control Byte and Data Payload
  $payload = pack_data_payload_bin($data, $payload_size);
  $buffer = pack('C', ($control_byte) & 0xFF) . $payload;

  return $buffer;
}
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
}

function render_frame($data, $qr_ver, $qr_ecc, $qr_scale, $qr_margin,
    $in_ascii=false) {
    global $tgxf_control_type;

    // Generate QRCode as an array of 1 and 0 values, from 8bit data
    $code = new QRcode();
    $code->encodeString8bit($data, $qr_ver, $qr_ecc);
    // $code->encodeString($data, $qr_ver, $qr_ecc, QR_MODE_8, false);
    QRtools::markTime('after_encode');
    $frame = QRtools::binarize($code->data);

    // Define image dimensions as 1:1 to QRCode size
    $h = count($frame);
    $w = strlen($frame[0]);
    $imgW = $w + (2 * $qr_scale * $qr_margin);
    $imgH = $h + (2 * $qr_scale * $qr_margin);

    if($in_ascii) {
        // ASCII Output
        $target_image = "";
        $padding = " ";
        $double_x = 0;
        switch("squarecolor") {

            case "compressed":
                // 0.5 row + 1 col per bit
                $target_image .= "\n";
                for($y=0; $y<$h; $y+=2) {
                    $target_image .= $padding;
                    for($x=0; $x<$w; $x++) {
                        if($frame[$y][$x] == '1') {
                            if(!isset($frame[$y+1])) {
                                // Block characters in the IBM850 Character Encoding
                                $target_image .= chr(223); // 1,0
                            } else {
                                if($frame[$y + 1][$x] == '1') { // 1,1
                                    $target_image .= chr(219);
                                } else { // 1,0
                                    $target_image .= chr(223);
                                }
                            }
                        } else {
                            if(!isset($frame[$y+1])) {
                                $target_image .= " "; // 0,0
                            } else {
                                if($frame[$y + 1][$x] == '1') { // 0,1
                                    $target_image .= chr(220);
                                } else { // 0,0
                                    $target_image .= " ";
                                }
                            }
                        }
                    }
                }
                $target_image .= " \n";
            }
            break;

            case "squarecolor":
                $double_x = 1;
            case "color":
                // 1 row + 1 col per bit
                for($y=0; $y<$h; $y++) {
                    $target_image .= "\033[0;30;47m" . "\n"; // White on Black
                    $target_image .= "\033[0;30;47m" . $padding;
                    for($x=0; $x<$w; $x++) {
                        if($frame[$y][$x] == '1') {
                            $target_image .= "\033[0;37;0m "; // Black on White
                        } else {
                            $target_image .= "\033[0;30;47m ";
                        }
                    }
                }
            }
        }
    }
}
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
        }
        if($double_x)
            $target_image .= " ";
    }
    $target_image .= "\033[0;30;47m" . $padding;
}
break;

case "simple":
    // 1 row + 1 col per bit
    for($y=0; $y<$h; $y++) {
        $target_image .= $padding;
        for($x=0; $x<$w; $x++) {
            if($frame[$y][$x] == '1') {
                $target_image .= "#";
            } else {
                $target_image .= " ";
            }
        }
        $target_image .= " \n";
    }
    break;
}

} else {
// GRAPHIC Output

// Create GD image resource
$base_image = imagecreate($imgW, $imgH);
$col[0] = imagecolorallocate($base_image,255,255,255); // BG, white
// Colors for demonstration only
if(get_control_bit($data)) {
    switch(get_control_type($data)) {
        case $tgxf_control_type["START"]["value"]:
            $col[1] = imagecolorallocate($base_image,0,64,0); // FG, START = Green
            break;
        case $tgxf_control_type["STATUS"]["value"]:
            $col[1] = imagecolorallocate($base_image,0,0,64); // FG, STATUS = Blue
            break;
        case $tgxf_control_type["STOP"]["value"]:
            $col[1] = imagecolorallocate($base_image,64,0,0); // FG, STOP = Red
            break;
    }
} else {
    $col[1] = imagecolorallocate($base_image,0,0,0); // FG, black for Data
}
imagefill($base_image, 0, 0, $col[0]);

// Mark pixels in GD image per QRCode array
for($y=0; $y<$h; $y++) {
    for($x=0; $x<$w; $x++) {
        if($frame[$y][$x] == '1') {
            imagesetpixel($base_image,
                $x + ($qr_scale * $qr_margin),
                $y + ($qr_scale * $qr_margin),
                $col[1]);
        }
    }
}

// Resize the GD image according to the requested image dimensions
$target_image = imagecreate($imgW * $qr_scale, $imgH * $qr_scale);
imagecopyresized(
    $target_image,
    $base_image,
    0, 0, 0, 0,
    $imgW * $qr_scale, $imgH * $qr_scale, $imgW, $imgH
);
imagedestroy($base_image);
}

// Return the GD image resource of the final (full scale) image
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
    return $target_image;
}

// Main Program Begins Here
set_time_limit(600);

// Get User Options for Encoding
// CLI (text):  php -q script.php ascii <ver> <fps>
// CLI (graphic): php -q script.php graphic <ver> <fps> > TGXf.gif
// Web (graphic): http://<server>/script.php?ver=<ver>&fps=<fps>
unset($ui_ver);
unset($ui_fps);
if(PHP_SAPI_NAME() == "cli") {
    // error_reporting(E_ALL ^ E_WARNING);
    if($argc == 4) {
        $ui_txt = $argv[1];
        $ui_ver = $argv[2];
        $ui_fps = $argv[3];
    }
} else {
    if(array_key_exists("ver", $_REQUEST)) {
        $ui_ver = $_REQUEST["ver"];
    }
    if(array_key_exists("fps", $_REQUEST)) {
        $ui_fps = $_REQUEST["fps"];
    }
}

// Validate user input or Default it
if($ui_ver != 1 && $ui_ver != 2 && $ui_ver != 8 && $ui_ver != 15)
    $ui_ver = 8; // 1 (21x21), 2 (25x25), 8 (49x49), 15 (77x77)
if($ui_fps != 1 && $ui_fps != 2 && $ui_fps != 5 && $ui_fps != 8 && $ui_fps != 10)
    $ui_fps = 5; // 1, 2, 5, 8, 10
if($ui_txt != "ascii" && $ui_txt != "graphic")
    $ui_txt = "graphic";

// $tgxf_session_filepath = ...
$tgxf_session_qrcode_version = $ui_ver;
$tgxf_session_qrcode_fps = $ui_fps;
$in_ascii = 0;
if($ui_txt == "ascii") // Text mode output
    $in_ascii = 1;

// Static tables
$frame_rounding_correction = 4; // Allow for variable ECC encoding
$frame_bytes_version[1] = 14 - $frame_rounding_correction;
$frame_bytes_version[2] = 26 - $frame_rounding_correction;
$frame_bytes_version[8] = 152 - $frame_rounding_correction;
$frame_bytes_version[15] = 412 - $frame_rounding_correction;

// Customisable parameters
$mode_ecc_level = QR_ECLEVEL_M; // _L, _M, _Q, _H
$mode_pixel_size = 3; // Size according to your display
$mode_margin_size = 1;

// File names
// $tgxf_session_filepath = "./tgxfv2.php";
$tgxf_session_filepath = "./helloworld.txt";
// $tgxf_session_filepath = "test.jpg";
$user_output_file = "TGXf-v" . $tgxf_session_qrcode_version . "-" .
    $tgxf_session_qrcode_fps . "fps-" . $mode_pixel_size . "px.gif";
$basename = get_basename_for_file($tgxf_session_filepath);

// Calculations based on custom parameters
$tgxf_session_qrcode_bytes = $frame_bytes_version[$tgxf_session_qrcode_version];
$read_bytes = $tgxf_session_qrcode_bytes - 1; // Subtract the Control Byte
$duration = 100 / $tgxf_session_qrcode_fps;
$tgxf_session_file_size = get_size_for_file($tgxf_session_filepath);
$tgxf_session_total_crc32 = get_crc_for_file($tgxf_session_filepath);

// Initialisation
```


PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
$output_frames = array();
$output_times = array();
$tgxf_session_frame_count = 0;

// TGXf CONTROL -> START -> FILENAME
$frame_data = build_control_frame(
    $tgxf_control_type["START"]["value"],
    $tgxf_control_type["START"]["FILENAME"]["value"],
    $basename);
$image_data = render_frame($frame_data, $tgxf_session_qrcode_version,
    $mode_ecc_level, $mode_pixel_size, $mode_margin_size, $in_ascii);
$output_frames[] = $image_data;
$output_times[] = $duration;
// First frame seems to get lost in GifCreator
$output_frames[] = $image_data;
$output_times[] = $duration;

// TGXf CONTROL -> START -> FILESIZE
$frame_data = build_control_frame(
    $tgxf_control_type["START"]["value"],
    $tgxf_control_type["START"]["FILESIZE"]["value"],
    $tgxf_session_file_size);
$image_data = render_frame($frame_data, $tgxf_session_qrcode_version,
    $mode_ecc_level, $mode_pixel_size, $mode_margin_size, $in_ascii);
$output_frames[] = $image_data;
$output_times[] = $duration;

// TGXf CONTROL -> START -> QRCODE_BYTES
$frame_data = build_control_frame(
    $tgxf_control_type["START"]["value"],
    $tgxf_control_type["START"]["QRCODE_BYTES"]["value"],
    $read_bytes);
$image_data = render_frame($frame_data, $tgxf_session_qrcode_version,
    $mode_ecc_level, $mode_pixel_size, $mode_margin_size, $in_ascii);
$output_frames[] = $image_data;
$output_times[] = $duration;

// TGXf CONTROL -> START -> QRCODE_FPS
$frame_data = build_control_frame(
    $tgxf_control_type["START"]["value"],
    $tgxf_control_type["START"]["QRCODE_FPS"]["value"],
    $tgxf_session_qrcode_fps);
$image_data = render_frame($frame_data, $tgxf_session_qrcode_version,
    $mode_ecc_level, $mode_pixel_size, $mode_margin_size, $in_ascii);
$output_frames[] = $image_data;
$output_times[] = $duration;

$fp = fopen($tgxf_session_filepath, 'r');
if($fp != false) {
    // TGXf DATA (one data frame per loop iteration)
    $max_blocks = ceil($tgxf_session_file_size / $read_bytes);
    for($block_idx = 0; $block_idx < $max_blocks; $block_idx++) {
        // last block an odd size?
        $read_delta = $tgxf_session_file_size - ($block_idx * $read_bytes);
        if($read_delta > 0 &&
            $read_delta < $read_bytes &&
            $block_idx == ($max_blocks - 1)) {
            $read_bytes = $read_delta;
        }
        $raw_data = fread($fp, $read_bytes);
        $frame_data = build_data_frame($tgxf_session_frame_count, $raw_data, $read_bytes);
        $image_data = render_frame($frame_data, $tgxf_session_qrcode_version,
            $mode_ecc_level, $mode_pixel_size, $mode_margin_size, $in_ascii);
        $output_frames[] = $image_data;
        $output_times[] = $duration;
        $tgxf_session_frame_count++;
        if($tgxf_session_frame_count > 15)
            $tgxf_session_frame_count = 0;
    }
    fclose($fp);
}
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
// TGXf CONTROL -> STOP -> COMPLETE
$frame_data = build_control_frame(
    $tgxf_control_type["STOP"]["value"],
    $tgxf_control_type["STOP"]["COMPLETE"]["value"],
    $tgxf_session_total_crc32);
$image_data = render_frame($frame_data, $tgxf_session_qrcode_version,
    $mode_ecc_level, $mode_pixel_size, $mode_margin_size, $in_ascii);
$output_frames[] = $image_data;
$output_times[] = $duration;

reset($output_frames);
reset($output_times);
if($in_ascii) {
    // ASCII Output
    // http://ascii-table.com/ansi-escape-sequences-vt-100.php
    // See also graphical modes;
    // http://ascii-table.com/ansi-escape-sequences.php
    print "\033[0;30;47m"; // White on Black
    print "\033[2J"; // Clear screen
    sleep(2); // Let camera contrast settle
    for($idx = 0; $idx < count($output_frames); $idx++) {
        print "\033[2J"; // Clear screen
        print "\033[0;0H"; // Home top left
        printf('%s', $output_frames[$idx]);
        usleep($output_times[$idx] * 10000);
    }
    sleep(1);
    print "\033[0;37;0m"; // Black on White
    print "\033[2J";
    print "\033[0;0H";
} else {
    // GRAPHIC Output

    // Initialize and create the final animated GIF
    $gc = new GifCreator();
    $gc->create($output_frames, $output_times, 1);
    $gifBinary = $gc->getGif();

    // Output GIF image
    if(PHP_SAPI_NAME() != "cli") {
        header('Content-type: image/gif');
        header('Content-Disposition: filename="' . $user_output_file . '"');
    }
    echo $gifBinary;
}
?>
```

Note that platform “endian-ness” has not been factored in this code – it has been tested on x86 only.

3.3.1.1 Using the TGXf Transmit implementation

To execute the reference implementation from the command line in graphics mode (i.e. output is a GIF image), then please use the following command;

```
php -q tgif.php graphic <ver> <fps> > TGXf.gif
```

To execute the reference implementation from the command line in ASCII (ANSI) mode, then please use the following command;

```
php -q tgif.php ascii <ver> <fps>
```

3.3.2 C Reference Implementation – Receive

The following C reference implementation was used to validate the test cases that have been supplied with this specification. It is designed as a stand-alone application and includes a patch to the cross-platform QR code library – Zbar.

3.3.2.1 Acquiring Zbar

The following process will get a Linux Debian-based Ubuntu/Mint machine to a working Zbar build;

1. Remove/Install the dependencies first;

```
sudo apt-get remove libzbar0  
sudo apt-get install mercurial libjpeg62-dev libmagickwand-dev python-gtk2-dev libqt4-core qt4-dev-tools
```

2. Acquire the current Zbar source (this build does not work on the zbar-0.10 tarball – the latest available at the time of writing);

```
hg clone http://zbar.hg.sourceforge.net:8000/hgroot/zbar
```

3. Build Zbar;

```
cd zbar  
autoreconf --install  
./configure  
make
```

This is a checkpoint: If the Zbar code does not compile then do not proceed. Check your development environment for missing dependencies and mitigate any new issues that may be in the current Zbar development code-base.

3.3.2.2 The Zbar TGXf Patch

The following C code is in “patch” format. A change has been made to the Zbar decoder library in order to prevent the 8 bit data stream from being interpreted as UTF-8 language data.

```
--- zbar/zbar/qrcode/qrdectxt.c.orig 2014-02-02 18:05:18.811784715 +1100
+++ zbar/zbar/qrcode/qrdectxt.c      2014-02-02 18:05:50.143783857 +1100
@@ -254,7 +254,14 @@
     Does such a thing occur?
     Is it allowed?
     It requires copying buffers around to handle correctly.*/
-   case QR_MODE_BYTE:
+   case QR_MODE_BYTE:{
+       if(sa_ctext-sa_ntext>=(size_t)entry->payload.data.len){
+           memcpy(sa_text+sa_ntext,entry->payload.data.buf,
+                 entry->payload.data.len*sizeof(*sa_text));
+           sa_ntext+=entry->payload.data.len;
+       }
+       else err=1;
+   }break;
+   case QR_MODE_KANJI:{
+       in=(char *)entry->payload.data.buf;
+       inleft=entry->payload.data.len;
```

3.3.2.3 Applying the *Binary Clear* patch to Zbar

Assuming the Zbar patch is called “zbar-binaryclear.patch” and resides in your home directory, and that you are in the “zbar” directory per step 3 above, then the following command should apply the patch to the Zbar library;

```
patch -p1 < ~/zbar-binaryclear.patch
```

If the patch applies successfully, then re-compile Zbar;

```
make
```

And install Zbar;

```
sudo make install
```

You may also wish to update the share library cache;

```
sudo ldconfig
```

3.3.2.4 The TGXf Receive C Source

The following C code is for `tgxf-receive.c`, a C implementation of the TGXf protocol (receive only).

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <getopt.h>
#include <sys/stat.h>
#ifdef _WIN32
# include <io.h>
# include <fcntl.h>
#endif
#include <signal.h>
#include <sys/time.h>
#include <zbar.h>

// The following CRC code comes from RFC1952, Section 8
// http://tools.ietf.org/html/rfc1952#section-8

/* Table of CRCs of all 8-bit messages. */
unsigned long crc_table[256];

/* Flag: has the table been computed? Initially false. */
int crc_table_computed = 0;

/* Make the table for a fast CRC. */
void make_crc_table(void)
{
    unsigned long c;
    int n, k;
    for (n = 0; n < 256; n++) {
        c = (unsigned long) n;
        for (k = 0; k < 8; k++) {
            if (c & 1) {
                c = 0xedb88320L ^ (c >> 1);
            } else {
                c = c >> 1;
            }
        }
        crc_table[n] = c;
    }
    crc_table_computed = 1;
}

/*
    Update a running crc with the bytes buf[0..len-1] and return
    the updated crc. The crc should be initialized to zero. Pre- and
    post-conditioning (one's complement) is performed within this
    function so it shouldn't be done by the caller. Usage example:

    unsigned long crc = 0L;

    while (read_buffer(buffer, length) != EOF) {
        crc = update_crc(crc, buffer, length);
    }
    if (crc != original_crc) error();
*/
unsigned long update_crc(unsigned long crc,
    unsigned char *buf, int len)
{
    unsigned long c = crc ^ 0xffffffffL;
    int n;

    if (!crc_table_computed)
        make_crc_table();
}
```


PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
unsigned int tgif_session_frame_count = 0;
unsigned int tgif_session_error_count = 0;
unsigned long tgif_session_recent_crc32 = 0L;
unsigned long tgif_session_total_crc32 = 0L;
unsigned int tgif_session_max_errors = 5;
unsigned int tgif_session_errors = 0;
char * tgif_session_error_str = NULL;
unsigned int tgif_session_error_size = 0;
unsigned int tgif_session_error_len = 0;
unsigned int tgif_session_aborted = 0;
unsigned int tgif_session_total_bytes = 0;
unsigned int tgif_session_complete = 0;
unsigned int tgif_session_max_timeout = 5;
unsigned int tgif_session_timeout_ticks = 0;
unsigned int tgif_session_paused = 0;

unsigned int tgif_session_debug_image_counter = 0;

// TGXf Session Parameters (TGXf Global Options)
unsigned char tgif_session_file_name[20];
char * tgif_session_filepath;
unsigned int tgif_session_file_size = 0;
unsigned int tgif_session_qrcode_version = 0;
unsigned int tgif_session_qrcode_fps = 0;
unsigned int tgif_session_qrcode_bytes = 0;

unsigned int get_size_for_file(char *filepath) {
    struct stat st;
    unsigned int size;

    stat(filepath, &st);
    size = st.st_size;

    return size;
}

unsigned long get_crc_for_file(char *filepath) {
    unsigned char data_buffer[1024];
    unsigned char *data_buffer_p;
    int data_buffer_size = 1024;
    unsigned int read_len;
    unsigned long crc;
    FILE *fp;

    crc = 0L;
    if(!filepath)
        return crc;

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

    data_buffer_p = data_buffer;
    while((read_len = fread(data_buffer, 1, data_buffer_size, fp)) > 0) {
        crc = update_crc(crc, data_buffer_p, read_len);
    }

    return crc;
}

unsigned int unpack_control_payload_16bit_number(unsigned char *frame_payload, int
payload_size)
{
    if(payload_size >= 2)
        return (unsigned int)frame_payload[0] << 8 | (unsigned int)frame_payload[1];

    return 0;
}
```


PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
unsigned long unpack_control_payload_32bit_number(unsigned char *frame_payload, int
payload_size)
{
    if(payload_size >= 4) {
        return (unsigned long)frame_payload[0] << 24 | (unsigned long)frame_payload[1] << 16 |
            (unsigned long)frame_payload[2] << 8 | (unsigned long)frame_payload[3];
    }

    return 0L;
}
```

```
unsigned int unpack_control_payload_clean_string(unsigned char *string_dest, unsigned char
*frame_payload, int payload_size)
{
    unsigned char *chr_p;

    memset(string_dest, 0, 10);
    if(payload_size > 0) {
        // Store String
        if(memcpy(string_dest, frame_payload, 9) != string_dest) {
            fprintf(stderr, "ERROR(TGXf): badly formatted control string.\n");
            return -1;
        }
        string_dest[10] = '\0';
        // Clean String
        chr_p = string_dest;
        while(*chr_p) {
            // Unsavory characters are replaced
            if(*chr_p < 32 || ( *chr_p > 32 && *chr_p < 45 ) ||
                ( *chr_p > 46 && *chr_p < 48 ) || ( *chr_p > 57 && *chr_p < 65 ) ||
                ( *chr_p > 90 && *chr_p < 97 ) || *chr_p > 122) {
                *chr_p = '_';
            }
            // No leading dot
            if(*chr_p == 46 && chr_p == string_dest) {
                *chr_p = '_';
            }
            chr_p++;
        }
    }

    return 0;
}
```

```
void tgxf_timeout_watchdog(int sig) {
    if(tgxf_session_complete) {
        tgxf_session_timeout_ticks = 0;
        if(TGXF_DEBUG)
            printf("DEBUG(TGXf): Session complete ..\n");
        return;
    }
    if(tgxf_session_paused) {
        tgxf_session_timeout_ticks = 0;
        if(TGXF_DEBUG)
            printf("DEBUG(TGXf): Session paused ..\n");
        return;
    }
    if(tgxf_session_timeout_ticks >= tgxf_session_max_timeout) {
        if(!tgxf_session_aborted) {
            printf("(TGXf): Session receive timeout exceeded, aborting xfer.\n");
            tgxf_session_aborted = 1;
        }
    }
    tgxf_session_timeout_ticks++;

    return;
}
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
static int decode_tgxf_frame(const zbar_symbol_t *sym) {
    struct timeval tbuf;
    struct sigaction action;
    unsigned char *frame_buffer;
    unsigned char *frame_payload;
    char error_buffer[1024];
    int frame_size;
    int payload_size;
    int error_buffer_size = 1024;

    unsigned char control_byte;
    unsigned int control_bit;
    unsigned int control_type;
    unsigned int control_subtype;
    unsigned int data_counter;
    unsigned int expected_counter;
    unsigned int lost_frames;
    unsigned int transfer_bytes_remaining;
    unsigned int frames_remaining;
    unsigned int minutes_remaining;
    unsigned int seconds_remaining;
    unsigned int bytes_to_use;
    unsigned int lost_frame_idx;
    FILE *fp;
    int i;
    float progress_state;

    frame_size = zbar_symbol_get_data_length(sym);
    if(frame_size <= 0) {
        fprintf(stderr, "ERROR(TGXf): invalid frame size: %d\n", frame_size);
        return -1;
    }
    if((frame_buffer = (unsigned char *)malloc(frame_size)) == NULL) {
        fprintf(stderr, "ERROR(TGXf): unable to allocate frame buffer\n");
        return -1;
    }
    if(memcpy(frame_buffer, zbar_symbol_get_data(sym), frame_size) != frame_buffer) {
        fprintf(stderr, "ERROR(TGXf): failed to establish frame buffer\n");
        return -1;
    }
    }

    // Update watchdog timer
    tgxf_session_timeout_ticks = 0;

    // Point to Payload
    frame_payload = frame_buffer;
    frame_payload++;
    payload_size = frame_size - 1;

    // Get Control Byte and Bit
    control_byte = *frame_buffer;
    control_bit = control_byte & 1;

    switch(control_bit) {

        // CONTROL FRAME
        case 1:
            if(TGXF_DEBUG)
                fprintf(stdout, "DEBUG(TGXf): Control Frame\n");
            control_type = (control_byte & 14) >> 1;
            // Only accept STOP if session has been aborted by client
            if(tgxf_session_aborted && control_type != TGXF_CONTROL_TYPE_STOP) {
                if(TGXF_DEBUG)
                    fprintf(stdout, "DEBUG(TGXf): ignoring non-STOP control message in aborted
transfer.\n");
                break;
            }
            control_subtype = (control_byte & 240) >> 4;
            if(TGXF_DEBUG)
                fprintf(stdout, "DEBUG(TGXf): Control Type=[%d], SubType=[%d]\n", control_type,
control_subtype);
            switch(control_type) {
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
case TGXF_CONTROL_TYPE_START:
    if(TGXF_DEBUG)
        fprintf(stdout, "DEBUG(TGXf): Control Type START\n");
    // Don't accept START if data frames have been received
    if(tgxf_session_first_data_frame) {
        if(TGXF_DEBUG)
            fprintf(stdout, "DEBUG(TGXf): ignoring late START control message in active
transfer.\n");
    }
    switch(control_subtype) {
        case TGXF_CONTROL_SUBTYPE_START_FILENAME:
            if(TGXF_DEBUG)
                fprintf(stdout, "DEBUG(TGXf): Control SubType FILENAME\n");
            unpack_control_payload_clean_string(tgxf_session_file_name, frame_payload,
payload_size);
            if(TGXF_DEBUG)
                fprintf(stdout, "DEBUG(TGXf): START/FILENAME [%s].\n",
tgxf_session_file_name);
            break;
        case TGXF_CONTROL_SUBTYPE_START_FILESIZE:
            tgxf_session_file_size = unpack_control_payload_16bit_number(frame_payload,
payload_size);
            if(TGXF_DEBUG)
                fprintf(stdout, "DEBUG(TGXf): START/FILESIZE [%d].\n",
tgxf_session_file_size);
            break;
        case TGXF_CONTROL_SUBTYPE_START_QRCODE_VERSION:
            tgxf_session_qrcode_version =
unpack_control_payload_16bit_number(frame_payload, payload_size);
            if(TGXF_DEBUG)
                fprintf(stdout, "DEBUG(TGXf): START/QRCODE_VERSION [%d].\n",
tgxf_session_qrcode_version);
            break;
        case TGXF_CONTROL_SUBTYPE_START_QRCODE_FPS:
            tgxf_session_qrcode_fps = unpack_control_payload_16bit_number(frame_payload,
payload_size);
            if(TGXF_DEBUG)
                fprintf(stdout, "DEBUG(TGXf): START/QRCODE_FPS [%d].\n",
tgxf_session_qrcode_fps);
            break;
        case TGXF_CONTROL_SUBTYPE_START_QRCODE_BYTES:
            tgxf_session_qrcode_bytes = unpack_control_payload_16bit_number(frame_payload,
payload_size);
            if(TGXF_DEBUG)
                fprintf(stdout, "DEBUG(TGXf): START/QRCODE_BYTES [%d].\n",
tgxf_session_qrcode_bytes);
            break;
        default:
            fprintf(stdout, "WARNING(TGXf): reserved control sub-type used, frame
ignored.\n");
            free(frame_buffer);
            return -1;
    }
    break;

case TGXF_CONTROL_TYPE_STOP:
    if(TGXF_DEBUG)
        fprintf(stdout, "DEBUG(TGXf): Control Type STOP\n");
    switch(control_subtype) {
        case TGXF_CONTROL_SUBTYPE_STOP_PAUSE:
            tgxf_session_paused = !tgxf_session_paused;
            if(TGXF_DEBUG)
                fprintf(stdout, "DEBUG(TGXf): STOP/PAUSE.\n");
            break;
        case TGXF_CONTROL_SUBTYPE_STOP_COMPLETE:
            tgxf_session_total_crc32 = unpack_control_payload_32bit_number(frame_payload,
payload_size);
            if(!tgxf_session_good_start) {
                if(TGXF_DEBUG)
                    fprintf(stdout, "WARNING(TGXf): failed to get required start data,
ignoring out of state STOP.\n");
                break;
            }
    }
}
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
    }
    txf_session_recent_crc32 = get_crc_for_file((char *)txf_session_file_name);
    if(txf_session_total_crc32 != txf_session_recent_crc32) {
        fprintf(stdout, "ERROR(TGXf): CRC doesn't match, calculated %lu, received
%lu.\n",
            txf_session_recent_crc32, txf_session_total_crc32);
    } else {
        txf_session_complete = 1;
        fprintf(stdout, "(TGXf): CRC validated, transfer successful.\n");
    }
    // if(TGXf_DEBUG)
    // fprintf(stdout, "DEBUG(TGXf): STOP/COMPLETE [%u].\n",
txf_session_total_crc32);
    // SESSION RESET
    break;
    case TGXF_CONTROL_SUBTYPE_STOP_CANCEL:
        if(TGXf_DEBUG)
            fprintf(stdout, "DEBUG(TGXf): STOP/CANCEL.\n");
        // SESSION RESET
        break;
    default:
        fprintf(stdout, "WARNING(TGXf): reserved control sub-type used, frame
ignored.\n");
        free(frame_buffer);
        return -1;
    }
    break;

    case TGXF_CONTROL_TYPE_STATUS:
        if(TGXf_DEBUG)
            fprintf(stdout, "DEBUG(TGXf): Control Type STATUS\n");
        switch(control_subtype) {
            case TGXF_CONTROL_SUBTYPE_STATUS_SINCE:
                txf_session_recent_crc32 = unpack_control_payload_32bit_number(frame_payload,
payload_size);
                // if(TGXf_DEBUG)
                // fprintf(stdout, "DEBUG(TGXf): STATUS/SINCE [%u].\n",
txf_session_recent_crc32);
                break;
            default:
                fprintf(stdout, "WARNING(TGXf): reserved control sub-type used, frame
ignored.\n");
                free(frame_buffer);
                return -1;
        }
        break;

    default:
        fprintf(stdout, "WARNING(TGXf): reserved control type used, frame ignored.\n");
        free(frame_buffer);
        return -1;
    }
    break;

// DATA FRAME
case 0:
    if(TGXf_DEBUG)
        fprintf(stdout, "DEBUG(TGXf): Data Frame -----.\n");

    // Ignore data frames if session aborted
    if(txf_session_aborted) {
        if(TGXf_DEBUG)
            fprintf(stdout, "WARNING(TGXf): session aborted, ignoring data frames.\n");
        break;
    }

    // If this is the first Data Frame then do we have the right START info to continue?
    if(!txf_session_first_data_frame) {
        txf_session_first_data_frame = 1;
        fprintf(stdout, ".= TGXf =-\n");
        fprintf(stdout, "  Filename:           %s\n", txf_session_file_name);
        fprintf(stdout, "  File size:           %d\n", txf_session_file_size);
    }
}
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
fprintf(stdout, " QRcode Version:          %d\n", tgif_session_qrcode_version);
fprintf(stdout, " QRcode Frames per Second: %d\n", tgif_session_qrcode_fps);
fprintf(stdout, " QRcode Bytes per Frame:   %d\n", tgif_session_qrcode_bytes);
fprintf(stdout, "\n");
// Do we have sufficient START information to perform the data transfer?
if(strlen((const char *)tgif_session_file_name) > 0 &&
    tgif_session_file_size > 0 &&
    tgif_session_qrcode_fps > 0 &&
    tgif_session_qrcode_bytes > 0) {

    // Establish session state
    tgif_session_good_start = 1;

    // Setup session calculations
    tgif_session_total_frames = tgif_session_file_size / tgif_session_qrcode_bytes;
    if(tgif_session_total_frames * tgif_session_qrcode_bytes <
tgif_session_file_size)
        tgif_session_total_frames++;

    // Setup the TGXf watchdog timer
    action.sa_handler = tgif_timeout_watchdog;
    sigemptyset(&action.sa_mask);
    action.sa_flags = 0;
    if(sigaction(SIGALRM, &action, NULL) < 0) {
timeout.\n");
        fprintf(stderr, "ERROR(TGXf): Watchdog timer setup failed, session won't
    } else {
        tbuf.it_interval.tv_sec = 1;
        tbuf.it_interval.tv_usec = 0;
        tbuf.it_value.tv_sec = 1;
        tbuf.it_value.tv_usec = 0;
        if(setitimer(ITIMER_REAL, &tbuf, NULL) == -1)
timeout.\n");
            fprintf(stderr, "ERROR(TGXf): Watchdog timer setup failed, session won't
        }
    }
}
if(!tgif_session_good_start) {
    if(TGXf_DEBUG)
        fprintf(stdout, "WARNING(TGXf): failed to get required start data, session
aborted.\n");
    tgif_session_aborted = 1;
    break;
}

// Duplicate Data Frame Detection - Ignore
data_counter = (control_byte & 30) >> 1;
if(data_counter == tgif_session_last_data_counter) {
    if(TGXf_DEBUG)
        fprintf(stdout, "WARNING(TGXf): duplicate data frame, frame ignored.\n");
    break;
}

// Missing Data Frame Detection - Accumulate Errors
expected_counter = tgif_session_last_data_counter + 1;
if(expected_counter == 16)
    expected_counter = 0;
if(TGXf_DEBUG)
    fprintf(stdout, "DEBUG(TGXf): data_counter=%02d expected_counter=%02d; frame %d.\n",
        data_counter, expected_counter, tgif_session_frame_count);
if(data_counter != expected_counter) {
    lost_frames = data_counter - expected_counter;
    if(expected_counter > data_counter) {
        lost_frames = data_counter + 16 - expected_counter;
    }
    // Error Accumulation
    tgif_session_errors += lost_frames;
    if(tgif_session_errors >= tgif_session_max_errors) {
        tgif_session_aborted = 1;
        fprintf(stdout, "ERROR(TGXf): maximum errors for session reached, transfer
aborted.\n");
    }
}
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
// Accumulate lost frame numbers to report to user
if(!tgxf_session_error_str) {
    tgxf_session_error_size = 1024;
    if((tgxf_session_error_str = (char *)malloc(tgxf_session_error_size)) == NULL) {
        fprintf(stderr, "ERROR(TGXf): unable to allocate error buffer\n");
        free(frame_buffer);
        return -1;
    }
    memset(tgxf_session_error_str, 0, tgxf_session_error_size);
}
for(lost_frame_idx = 0; lost_frame_idx < lost_frames; lost_frame_idx++) {
    memset(error_buffer, 0, error_buffer_size);
    if(strlen(tgxf_session_error_str) > 0) {
        snprintf(error_buffer, error_buffer_size, "%d",
            tgxf_session_frame_count + lost_frame_idx + 1);
    } else {
        snprintf(error_buffer, error_buffer_size, "%d",
            tgxf_session_frame_count + lost_frame_idx + 1);
    }
    if(strlen(error_buffer) > tgxf_session_error_size - strlen(tgxf_session_error_str)
- 2) {
        tgxf_session_error_size += 1024;
        if((tgxf_session_error_str =
            (char *)realloc(tgxf_session_error_str, tgxf_session_error_size)) == NULL) {
            fprintf(stderr, "ERROR(TGXf): unable to allocate error buffer\n");
            free(frame_buffer);
            return -1;
        }
    }
    strcat(tgxf_session_error_str, error_buffer);
    if(tgxf_session_aborted) {
        strcat(tgxf_session_error_str, "+");
        break;
    }
}
if(lost_frames == 1) {
    fprintf(stdout, "WARNING(TGXf): lost 1 data frame, error recorded.\n");
} else {
    fprintf(stdout, "WARNING(TGXf): lost %d data frames, error recorded.\n",
lost_frames);
}
// Correct the session frame count
tgxf_session_frame_count += lost_frames;
}

// Do last-packet calculation and under-run detection
transfer_bytes_remaining =
    tgxf_session_file_size - (tgxf_session_qrcode_bytes * tgxf_session_frame_count);
bytes_to_use = tgxf_session_qrcode_bytes;
if(transfer_bytes_remaining < tgxf_session_qrcode_bytes)
    bytes_to_use = transfer_bytes_remaining;
tgxf_session_total_bytes += bytes_to_use;
if(TGXF_DEBUG) {
    fprintf(stdout, "DEBUG(TGXf): payload_size=%d bytes_to_use=%d qrcode_bytes=%d; frame
%d.\n", payload_size, bytes_to_use, tgxf_session_qrcode_bytes, tgxf_session_frame_count);
    fprintf(stdout, "DEBUG(TGXf): session_total_bytes after this frame is written = %d/
%d.\n", tgxf_session_total_bytes, tgxf_session_file_size);
}
if(payload_size < bytes_to_use) {
    fprintf(stdout, "ERROR(TGXf): undersized payload on frame %d.\n",
tgxf_session_frame_count);
    break;
}

// Update Display
if(tgxf_session_errors) {
    if(tgxf_session_error_len < strlen(tgxf_session_error_str)) {
        tgxf_session_error_len = strlen(tgxf_session_error_str);
        fprintf(stdout, "\n");
        fprintf(stdout, "(TGXf): Frame errors: %s\n", tgxf_session_error_str);
    }
}
}
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
frames_remaining = tgfx_session_total_frames - tgfx_session_frame_count;
seconds_remaining = frames_remaining / tgfx_session_qrcode_fps;
minutes_remaining = seconds_remaining / 60;
seconds_remaining = seconds_remaining % 60;
progress_state = (float)20 - ((float)frames_remaining /
(float)tgxf_session_total_frames * (float)20);
fprintf(stdout, "(TGXf): Frame: %4d/%d ",
    tgfx_session_frame_count + 1, tgfx_session_total_frames);
fprintf(stdout, " Errors: %2d/%d ",
    tgfx_session_errors, tgfx_session_max_errors);
fprintf(stdout, " [");
for(i = 0; i < 20; i++ ) {
    if(i >= progress_state) {
        fprintf(stdout, "%c", ' ' );
    } else {
        fprintf(stdout, "%c", '=' );
    }
}
fprintf(stdout, "] %02d min %02d sec.\r",
    minutes_remaining, seconds_remaining);
if(TGXF_DEBUG || frames_remaining == 1)
    fprintf(stdout, "\n");

// Write Data to File
if(access((const char *)tgxf_session_file_name, F_OK) == -1) {
    if((fp = fopen((const char *)tgxf_session_file_name, "w+b")) == NULL) {
        fprintf(stdout, "ERROR(TGXf): unable to create/open file for reading/writing, at
%s.\n", tgfx_session_file_name);
        break;
    }
} else {
    if((fp = fopen((const char *)tgxf_session_file_name, "r+b")) == NULL) {
        fprintf(stdout, "ERROR(TGXf): unable to open file for reading/writing, at %s.\n",
tgxf_session_file_name);
        break;
    }
    if(fseek(fp, (tgxf_session_qrcode_bytes * tgfx_session_frame_count), SEEK_SET) < 0)
{
        fprintf(stdout, "ERROR(TGXf): unable to seek file, at %s.\n",
tgxf_session_file_name);
        fclose(fp);
        break;
    }
}
if(fwrite((const void *)frame_payload, (size_t)bytes_to_use, (size_t)1, fp) !=
(size_t)1) {
    fprintf(stdout, "ERROR(TGXf): unable to write to file, at %s.\n",
tgxf_session_file_name);
    fclose(fp);
    break;
}
fclose(fp);

// Increment tally
tgxf_session_last_data_counter = data_counter;
tgxf_session_frame_count++;
break;

}

fflush(stdout);
free(frame_buffer);
return(1);
}

static void barcode_handler(zbar_image_t *img, const void *userdata) {
    const zbar_symbol_t *barsym;
    zbar_symbol_type_t type;

    barsym = zbar_image_first_symbol(img);
    for(; barsym; barsym = zbar_symbol_next(barsym)) {
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
        if(zbar_symbol_get_count(barsym))
            continue;
        if((type = zbar_symbol_get_type(barsym)) == ZBAR_PARTIAL)
            continue;
        decode_tgxf_frame(barsym);
    }
}

static void usage(int help) {
    fprintf(stderr, "ThruGlassXfer Linux Receive Reference Code\n\n");

    if(help) {
        fprintf(stderr,
            "Usage: tgxf-receive [OPTIONS]...\n"
            "  -h, --help      Display this help message.\n"
            "  -d DEVICE, --device=DEVICE\n"
            "                  Video device to use.\n"
            "  -n, --nodisplay\n"
            "                  Do not display video window.\n"
            "  -v, --verbose   Display additional decoder information.\n"
            "\n"
            "Press P for Pause and Q for Quit while the program is running.\n"
        );
    }
}

// Main Program Begins Here
int main(int argc, char **argv) {
    static const struct option options[] = {
        {"help",      no_argument, NULL, 'h'},
        {"device",    required_argument, NULL, 'd'},
        {"nodisplay", no_argument, NULL, 'n'},
        {"verbose",   no_argument, NULL, 'v'},
        {NULL, 0, NULL, 0}
    };
    static char *optstring = "hd:nqv";
    int opt;
    static zbar_processor_t *zbar_proc;
    const char *video_device;
    unsigned int display;
    unsigned int verbose;
    unsigned int active;
    int keypress;

    verbose = 0;
    display = 1;
    video_device = "";

    while((opt = getopt_long(argc, argv, optstring, options, NULL)) != -1) {
        switch(opt) {
            case 'h':
                usage(1);
                exit(0);
                break;
            case 'n':
                display = 0;
                break;
            case 'v':
                verbose++;
                break;
            case 'd':
                video_device = optarg;
                break;
            default:
                fprintf(stderr, "Try `%%s --help' for more information.\n", argv[0]);
                exit(-1);
                break;
        }
    }
}
```


PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
// Validate user input or Default it
// -- no options are mandatory

// Initialise the Zbar library
if(!(zbar_proc = zbar_processor_create(1))) {
    fprintf(stderr, "ERROR: unable to allocate memory?\n");
    return 1;
}
zbar_processor_set_data_handler(zbar_proc, barcode_handler, NULL);

// Optionally increase verbosity
if(verbose) {
    for(; verbose != 0; verbose--)
        zbar_increase_verbosity();
}

// Process QR codes exclusively
zbar_processor_set_config(zbar_proc, 0, ZBAR_CFG_ENABLE, 0);
zbar_processor_set_config(zbar_proc, ZBAR_QRCODE, ZBAR_CFG_ENABLE, 1);

// Open video device
if(zbar_processor_init(zbar_proc, video_device, display))
    return zbar_processor_error_spew(zbar_proc, 0);

// Optionally show window
if(display) {
    if(zbar_processor_set_visible(zbar_proc, 1))
        return zbar_processor_error_spew(zbar_proc, 0);
}

// Start processing video
active = 1;
if(zbar_processor_set_active(zbar_proc, active))
    return zbar_processor_error_spew(zbar_proc, 0);

// Wait for keypress
while((keypress = zbar_processor_user_wait(zbar_proc, -1)) >= 0) {
    if(tgxf_session_complete) {
        printf("ThruGlassXfer session completed successfully.\n");
        break;
    }
    if(tgxf_session_aborted) {
        printf("ThruGlassXfer session aborted.\n");
        break;
    }
    if(keypress == 'q' || keypress == 'Q')
        break;
    if(keypress == 'p' || keypress == 'P') {
        active = !active;
        if(zbar_processor_set_active(zbar_proc, active))
            return zbar_processor_error_spew(zbar_proc, 0);
    }
}

// Report Zbar library exit errors
if(keypress && keypress != 'q' && keypress != 'Q') {
    if(zbar_processor_get_error_code(zbar_proc) != ZBAR_ERR_CLOSED)
        return zbar_processor_error_spew(zbar_proc, 0);
}

// Shutdown the Zbar library
zbar_processor_destroy(zbar_proc);

return 0;
}
```

Note that platform “endian-ness” has not been factored in this code – it has been tested on x86 only.

3.3.2.5 Compiling the TGXf Receive C Source

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

```
gcc -Wall -c tgxf-receive.c
```

And the following should link tgxf-receive against the Zbar shared library;

```
gcc tgxf-receive.o -lzbar -o tgxf-receive
```

3.3.2.6 Using the TGXf Receive implementation

Once compiled and linked, the demonstrator application “tgxf-receive” is capable of operating as a TGXf client (receiver). To get help from the program, use the following command;

```
./tgxf-receive -h
```

You should see the following output;

```
ThruGlassXfer Linux Receive Reference Code

Usage: tgxf-receive [OPTIONS]...
-h, --help    Display this help message.
-d DEVICE, --device=DEVICE
               Video device to use.
-n, --nodisplay
               Do not display video window.
-v, --verbose Display additional decoder information.

Press P for Pause and Q for Quit while the program is running.
```

The program should work with any V4L2 (Video for Linux v2) video camera of sufficient performance and resolution. It was tested/validated against a Microsoft LifeCam HD-3000 (uvccvideo version 1.1.0) in 640x480 YUYV mode.

The debug code has been left in the patch so that you can see the protocol as it is received and interpreted.

Note that this a demonstration only; the working directory is used as the “download” directory, so malicious transfers could be used to overwrite zbar library code/binaries in this configuration.

3.3.3 C Reference Implementation – Transmit

The following C reference implementation was used to validate the test cases that have been supplied with this specification. It has been designed to work with QR code library – libqrencode.

3.3.3.1 Acquiring libqrencode

The following process will get a Linux Debian-based Ubuntu/Mint machine to a working libqrencode build;

1. Acquire the current libqrencode source (this build was tested successfully against qrencode-3.4.3.tar.gz – the latest available at the time of writing);

```
wget http://fukuchi.org/works/qrencode/qrencode-3.4.3.tar.gz
```

2. Build libqrencode;

```
tar xvfz qrencode-3.4.3.tar.gz
cd qrencode-3.4.3
./configure
make
```

3. Install libqrencode;

```
sudo make install
```

4. Update the library shared library cache;

```
sudo ldconfig
```

This is a checkpoint: If the libqrencode code does not compile and install then do not proceed.

Check your development environment for missing dependencies and mitigate any new issues that may be in the current libqrencode code-base.

3.3.3.2 The TGXf Transmit C Source

The following C code is for `tgxf-transmit.c`, a C implementation of the TGXf protocol (transmit only).

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <png.h>
#include <getopt.h>
#include <sys/stat.h>
#include <math.h>
#include <gd.h>
#include <qrencode.h>

// The following CRC code comes from RFC1952, Section 8
// http://tools.ietf.org/html/rfc1952#section-8

/* Table of CRCs of all 8-bit messages. */
unsigned long crc_table[256];

/* Flag: has the table been computed? Initially false. */
int crc_table_computed = 0;

/* Make the table for a fast CRC. */
void make_crc_table(void)
{
    unsigned long c;
    int n, k;
    for (n = 0; n < 256; n++) {
        c = (unsigned long) n;
        for (k = 0; k < 8; k++) {
            if (c & 1) {
                c = 0xedb88320L ^ (c >> 1);
            } else {
                c = c >> 1;
            }
        }
        crc_table[n] = c;
    }
    crc_table_computed = 1;
}

/*
    Update a running crc with the bytes buf[0..len-1] and return
    the updated crc. The crc should be initialized to zero. Pre- and
    post-conditioning (one's complement) is performed within this
    function so it shouldn't be done by the caller. Usage example:

    unsigned long crc = 0L;

    while (read_buffer(buffer, length) != EOF) {
        crc = update_crc(crc, buffer, length);
    }
    if (crc != original_crc) error();
*/
unsigned long update_crc(unsigned long crc,
    unsigned char *buf, int len)
{
    unsigned long c = crc ^ 0xffffffffL;
    int n;

    if (!crc_table_computed)
        make_crc_table();
    for (n = 0; n < len; n++) {
        c = crc_table[(c ^ buf[n]) & 0xff] ^ (c >> 8);
    }
}
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
    return c ^ 0xffffffffL;
}

/* Return the CRC of the bytes buf[0..len-1]. */
unsigned long crc(unsigned char *buf, int len)
{
    return update_crc(0L, buf, len);
}

/*
```

```

                _JNJ`
            .JNMH`
        JMMF`      `;.
    .NMM)          `MN.
    MMM)           (MML
(MMM`             MMLL
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) Letter" <ian dot letter at midnightcode dot org>

Midnight Code is a registered trademark of Ian Letter.

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>

```
*/

#define TGXF_DEBUG 0

#define TGXF_CONTROL_TYPE_START 1
#define TGXF_CONTROL_TYPE_STOP 2
#define TGXF_CONTROL_TYPE_STATUS 3
#define TGXF_CONTROL_SUBTYPE_START_FILENAME 1
#define TGXF_CONTROL_SUBTYPE_START_FILESIZE 2
#define TGXF_CONTROL_SUBTYPE_START_QRCODE_VERSION 3
#define TGXF_CONTROL_SUBTYPE_START_QRCODE_FPS 4
#define TGXF_CONTROL_SUBTYPE_START_QRCODE_BYTES 5
#define TGXF_CONTROL_SUBTYPE_STOP_PAUSE 1
#define TGXF_CONTROL_SUBTYPE_STOP_COMPLETE 2
#define TGXF_CONTROL_SUBTYPE_STOP_CANCEL 3
#define TGXF_CONTROL_SUBTYPE_STATUS_SINCE 1
#define TGXF_DISPLAY_ASCII_SIMPLE 1
#define TGXF_DISPLAY_ASCII_SIMPLE_SQUARE 2
#define TGXF_DISPLAY_ASCII_COMPRESSED 3
#define TGXF_DISPLAY_ANSI_SIMPLE 4
#define TGXF_DISPLAY_ANSI_SIMPLE_SQUARE 5

// TGXf Session Parameters (TGXf State Machine)
unsigned int tgxf_session_total_frames = 0;
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
unsigned int tgxf_session_first_data_frame = 0;
unsigned int tgxf_session_good_start = 0;
unsigned int tgxf_session_last_data_counter = 15;
unsigned int tgxf_session_frame_count = 0;
unsigned int tgxf_session_error_count = 0;
unsigned long tgxf_session_recent_crc32 = 0L;
unsigned long tgxf_session_total_crc32 = 0L;
unsigned int tgxf_session_max_errors = 5;
unsigned int tgxf_session_errors = 0;
unsigned int tgxf_session_aborted = 0;
unsigned int tgxf_session_total_bytes = 0;
unsigned int tgxf_session_complete = 0;

unsigned int tgxf_session_debug_image_counter = 0;

// TGXf Session Parameters (TGXf Global Options)
unsigned char tgxf_session_file_name[20];
char * tgxf_session_filepath;
unsigned int tgxf_session_file_size = 0;
unsigned int tgxf_session_qrcode_version = 0;
unsigned int tgxf_session_qrcode_fps = 0;
unsigned int tgxf_session_qrcode_bytes = 0;

char * get_basename_for_file(char *filepath) {
    char *basename;

    basename = filepath;
    while(*filepath) {
        if(*filepath++ == '/')
            basename = filepath;
    }

    return basename;
}

unsigned int get_size_for_file(char *filepath) {
    struct stat st;
    unsigned int size;

    stat(filepath, &st);
    size = st.st_size;

    return size;
}

unsigned long get_crc_for_file(char *filepath) {
    unsigned char data_buffer[1024];
    unsigned char *data_buffer_p;
    int data_buffer_size = 1024;
    unsigned int read_len;
    unsigned long crc;
    FILE *fp;

    crc = 0L;
    if(!filepath)
        return crc;

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

    data_buffer_p = data_buffer;
    while((read_len = fread(data_buffer, 1, data_buffer_size, fp)) > 0) {
        crc = update_crc(crc, data_buffer_p, read_len);
    }

    return crc;
}
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
unsigned int get_control_bit(unsigned char *data) {
    unsigned char control_byte;
    unsigned int control_bit;

    control_byte = *data;
    control_bit = control_byte & 1;

    return control_bit;
}

unsigned int get_control_type(unsigned char *data) {
    unsigned char control_byte;
    unsigned int control_type;

    control_byte = *data;
    control_type = (control_byte & 14) >> 1;

    return control_type;
}

unsigned int pack_control_payload_16bit_number(unsigned char *payload, void *data) {
    unsigned int value;
    unsigned int *value_p;

    if(!payload || !data)
        return 0;

    value_p = (unsigned int *)data;
    value = *value_p;
    *payload = (value >> 8) & 0xFF;
    payload++;
    *payload = (value >> 0) & 0xFF;

    return value;
}

unsigned long pack_control_payload_32bit_number(unsigned char *payload, void *data) {
    unsigned long value;
    unsigned long *value_p;

    if(!payload || !data)
        return 0;

    value_p = (unsigned long *)data;
    value = *value_p;
    *payload = (value >> 24) & 0xFF;
    payload++;
    *payload = (value >> 16) & 0xFF;
    payload++;
    *payload = (value >> 8) & 0xFF;
    payload++;
    *payload = (value >> 0) & 0xFF;

    return value;
}

unsigned int pack_control_payload_string(unsigned char *payload, void *data, unsigned int
data_len) {
    if(!payload || !data || !data_len)
        return 0;

    memset(payload, 0, 9);
    if(data_len > 9)
        data_len = 9;
    memcpy(payload, data, data_len);

    return data_len;
}
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
}

unsigned int pack_data_payload_bin(unsigned char *payload, void *data, unsigned int
data_len) {

    if(!payload || !data || !data_len)
        return 0;

    memcpy(payload, data, data_len);

    return data_len;
}

unsigned char * build_control_frame(unsigned char * packet, unsigned int control_type,
unsigned int control_subtype, void * data, unsigned int length) {
    unsigned char control_byte;
    unsigned int control_bit;
    unsigned char * payload;

    // Zero Control Byte
    control_byte = 0;

    // Control Frame, Control Bit = 1 (bit 0);
    control_bit = 1;
    control_byte = control_byte | control_bit;

    // Control Byte has Control Type (bits 3,2,1)
    if(control_type < 0 || control_type >= 7)
        control_type = 0;
    control_type = control_type << 1;
    control_byte = control_byte | control_type;

    // Control Byte has Sub-Control Type (bits 7,6,5,4)
    if(control_subtype < 0 || control_subtype >= 15)
        control_subtype = 0;
    control_subtype = control_subtype << 4;
    control_byte = control_byte | control_subtype;

    payload = packet + 1;

    switch((control_type & 14) >> 1) {
    case TGXF_CONTROL_TYPE_START:
        switch((control_subtype & 240) >> 4) {
        case TGXF_CONTROL_SUBTYPE_START_FILENAME: // FILENAME
            pack_control_payload_string(payload, data, length);
            break;
        case TGXF_CONTROL_SUBTYPE_START_FILESIZE: // FILESIZE
            pack_control_payload_16bit_number(payload, data);
            break;
        case TGXF_CONTROL_SUBTYPE_START_QRCODE_VERSION: // QRCODE_VERSION
            pack_control_payload_16bit_number(payload, data);
            break;
        case TGXF_CONTROL_SUBTYPE_START_QRCODE_FPS: // QRCODE_FPS
            pack_control_payload_16bit_number(payload, data);
            break;
        case TGXF_CONTROL_SUBTYPE_START_QRCODE_BYTES: // QRCODE_BYTES
            pack_control_payload_16bit_number(payload, data);
            break;
        }
        break;

    case TGXF_CONTROL_TYPE_STOP:
        switch((control_subtype & 240) >> 4) {
        case TGXF_CONTROL_SUBTYPE_STOP_PAUSE: // PAUSE
            break;
        case TGXF_CONTROL_SUBTYPE_STOP_COMPLETE: // COMPLETE
            pack_control_payload_32bit_number(payload, data);
            break;
        case TGXF_CONTROL_SUBTYPE_STOP_CANCEL: // CANCEL
            pack_control_payload_string(payload, data, length);
        }
    }
}
```


PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
        break;
    }
    break;

    case TGXF_CONTROL_TYPE_STATUS:
        switch((control_subtype & 240) >> 4) {
            case TGXF_CONTROL_SUBTYPE_STATUS_SINCE: // SINCE
                pack_control_payload_32bit_number(payload, data);
                break;
            }
        break;

    default:
        break;
}

// Frame consists of Control Byte and Control Payload
*packet = (control_byte) & 0xFF;

return packet;
}

unsigned char * build_data_frame(unsigned char * packet, unsigned int counter, void * data,
unsigned int payload_size) {
    unsigned char control_byte;
    unsigned int control_bit;
    unsigned char * payload;

    // Zero Control Byte
    control_byte = 0;

    // Data Frame, Control Bit = 0 (bit 0);
    control_bit = 0;
    control_byte = control_byte | control_bit;

    // Control Byte has incremented counter (bits 4,3,2,1)
    if(counter < 0 || counter > 15)
        counter = 0;
    counter = counter << 1;
    control_byte = control_byte | counter;

    // Frame consists of Control Byte and Data Payload
    payload = packet + 1;
    *packet = (control_byte) & 0xFF;
    if(memcpy(payload, data, payload_size) != payload) {
        // fprintf(stderr, "ERROR(TGXf): failed to build frame\n");
        // What else can we do here?
        return packet;
    }

    return packet;
}

unsigned char * render_frame(unsigned char * data, int length, int qr_ver, int qr_ecc, int
qr_scale, int qr_margin, int in_ascii) {
    unsigned int w;
    unsigned int h;
    unsigned int imgW;
    unsigned int imgH;
    unsigned int x;
    unsigned int y;
    unsigned int double_x;
    unsigned char * bit_p;
    const char padding[5] = "   ";
    QRcode *code;
    gdImagePtr base_image;
    gdImagePtr target_image;
    int col[2];
    char debug_filename[32];
    FILE *fp;
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
// Generate QRCode as an array of 1 and 0 values, from 8bit data
code = QRcode_encodeData(length, data, qr_ver, qr_ecc);

// Define image dimensions as 1:1 to QRCode size
w = code->width;
h = w;
imgW = w + (2 * qr_scale * qr_margin);
imgH = h + (2 * qr_scale * qr_margin);

if(in_ascii) {
// ASCII Output
double x = 0;
switch(TGXF_DISPLAY_ANSI_SIMPLE) {
/*
    case TGXF_DISPLAY_ANSI_COMPRESSED:
        // 0.5 row + 1 col per bit
        $target_image .= "\n";
        for($y=0; $y<$h; $y+=2) {
            $target_image .= $padding;
            for($x=0; $x<$w; $x++) {
                if($frame[$y][$x] == '1') {
                    if(!isset($frame[$y+1])) {
                        // Block characters in the IBM850 Character Encoding
                        $target_image .= chr(223); // 1,0
                    } else {
                        if($frame[$y + 1][$x] == '1') { // 1,1
                            $target_image .= chr(219);
                        } else { // 1,0
                            $target_image .= chr(223);
                        }
                    }
                } else {
                    if(!isset($frame[$y+1])) {
                        $target_image .= " "; // 0,0
                    } else {
                        if($frame[$y + 1][$x] == '1') { // 0,1
                            $target_image .= chr(220);
                        } else { // 0,0
                            $target_image .= " ";
                        }
                    }
                }
            }
            $target_image .= " \n";
        }
*/
        break;

    case TGXF_DISPLAY_ANSI_SIMPLE_SQUARE:
        // 1 row + 2 col per bit
        double_x = 1;
    case TGXF_DISPLAY_ANSI_SIMPLE:
        // 1 row + 1 col per bit
        printf("\033[0;30;47m ");
        printf("\033[2J"); // Clear screen
        printf("\033[0;0H"); // Home top left
        printf("\033[0;30;47m\n");
        for($y=0; $y<$h; $y++) {
            printf("\033[0;30;47m%s", $padding);
            for($x=0; $x<$w; $x++) {
                bit_p = code->data+(y*w)+x;
                if(*bit_p & 0x1) {
                    printf("\033[0;37;0m ");
                    if(double_x)
                        printf(" ");
                } else {
                    printf("\033[0;30;47m ");
                    if(double_x)
                        printf(" ");
                }
            }
        }
}
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
    }
    printf("\033[0;30;47m%s", padding);
    printf("\n");
}
printf("\033[0;30;47m\n");
break;

case TGXF_DISPLAY_ASCII_SIMPLE_SQUARE:
    // 1 row + 2 col per bit
    double_x = 1;
case TGXF_DISPLAY_ASCII_SIMPLE:
    // 1 row + 1 col per bit
    printf("\n");
    printf("\n");
    for(y=0; y<h; y++) {
        printf("%s", padding);
        for(x=0; x<w; x++) {
            bit_p = code->data+(y*w)+x;
            if(*bit_p & 0x1) {
                // Block character in the IBM850 Character Encoding = chr(219);
                printf("#");
                if(double_x)
                    printf("#");
            } else {
                printf(" ");
                if(double_x)
                    printf(" ");
            }
        }
        printf("\n");
    }
    printf("\n");
    printf("\n");
    break;
}

} else {
// GRAPHIC Output

// Create GD image resource
base_image = gdImageCreate(imgW, imgH);
col[0] = gdImageColorAllocate(base_image,255,255,255); // BG, white
// Colors for demonstration only
if(get_control_bit(data)) {
    switch(get_control_type(data)) {
        case TGXF_CONTROL_TYPE_START:
            col[1] = gdImageColorAllocate(base_image,0,64,0); // FG, START = Green
            break;
        case TGXF_CONTROL_TYPE_STATUS:
            col[1] = gdImageColorAllocate(base_image,0,0,64); // FG, STATUS = Blue
            break;
        case TGXF_CONTROL_TYPE_STOP:
            col[1] = gdImageColorAllocate(base_image,64,0,0); // FG, STOP = Red
            break;
    }
} else {
    col[1] = gdImageColorAllocate(base_image,0,0,0); // FG, black for Data
}
gdImageFill(base_image, 0, 0, col[0]);

// Mark pixels in GD image per QRCode array
for(y=0; y<h; y++) {
    for(x=0; x<w; x++) {
        bit_p = code->data+(y*w)+x;
        if(*bit_p & 0x1) {
            gdImageSetPixel(base_image,
                x + (qr_scale * qr_margin),
                y + (qr_scale * qr_margin),
                col[1]);
        }
    }
}
}
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
    }

    // Resize the GD image according to the requested image dimensions
    target_image = gdImageCreate(imgW * qr_scale, imgH * qr_scale);
    gdImageCopyResized(
        target_image,
        base_image,
        0, 0, 0, 0,
        imgW * qr_scale, imgH * qr_scale, imgW, imgH
    );
    gdImageDestroy(base_image);

    // Return/Write the GD image resource of the final (full scale) image
    sprintf(debug_filename, 32, "TGXf-output-%04d.png", tgxf_session_debug_image_counter);
    fp = fopen(debug_filename, "wb");
    gdImagePng(target_image, fp);
    fclose(fp);
    gdImageDestroy(target_image);
    tgxf_session_debug_image_counter++;
}

if(code != NULL)
    QRcode_free(code);

return data;
}

static void usage(int help) {
    fprintf(stderr, "ThruGlassXfer Linux Transmit Reference Code\n\n");

    if(help) {
        fprintf(stderr,
            "Usage: tgxf-transmit [OPTIONS]...\n"
            "  -h, --help      Display this help message.\n"
            "  -i FILENAME, --input=FILENAME\n"
            "                  File to encode and transmit.\n"
            "  -a, --ascii     Encode to ASCII output.\n"
            "  -g, --graphic  Encode to Graphic output (default).\n"
            "  -v {1,2,8,15}, --version={1,2,8,15}\n"
            "                  QRcode symbol version to use. (default=8)\n"
            "  -f {1,2,5,8,10}, --fps={1,2,5,8,10}\n"
            "                  QRcode symbols to display per second. (default=5)\n\n"
        );
    }
}

// Main Program Begins Here
int main(int argc, char **argv) {
    static const struct option options[] = {
        {"help", no_argument, NULL, 'h'},
        {"ascii", no_argument, NULL, 'a'},
        {"graphic", no_argument, NULL, 'g'},
        {"input", required_argument, NULL, 'i'},
        {"version", required_argument, NULL, 'v'},
        {"fps", required_argument, NULL, 'f'},
        {NULL, 0, NULL, 0}
    };
    static char *optstring = "hag:i:v:f:";
    int opt;
    int val;
    int in_ascii;
    unsigned int frame_rounding_correction;
    unsigned int frame_bytes_version[64];
    unsigned int read_bytes;
    int mode_ecc_level;
    int mode_pixel_size;
    int mode_margin_size;
    unsigned int max_blocks;
    unsigned int block_idx;
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
int read_delta;
FILE *fp;
unsigned char * tgif_packet;
unsigned int duration;
char * basename;
unsigned char * raw_data;

in_ascii = 0;
tgif_session_qrcode_version = 8;
tgif_session_qrcode_fps = 5;
tgif_session_filepath = NULL;

while((opt = getopt_long(argc, argv, optstring, options, NULL)) != -1) {
    switch(opt) {
        case 'h':
            usage(1);
            exit(0);
            break;
        case 'a':
            // Text mode output
            in_ascii = 1;
            break;
        case 'g':
            in_ascii = 0;
            break;
        case 'i':
            tgif_session_filepath = optarg;
            break;
        case 'v':
            // 1 (21x21), 2 (25x25), 8 (49x49), 15 (77x77)
            val = atoi(optarg);
            if(val != 1 && val != 2 && val != 8 && val != 15)
                val = 8;
            tgif_session_qrcode_version = val;
            break;
        case 'f':
            // 1, 2, 5, 8, 10
            val = atoi(optarg);
            if(val != 1 && val != 2 && val != 5 && val != 8 && val != 10)
                val = 5;
            tgif_session_qrcode_fps = val;
            break;
        default:
            fprintf(stderr, "Try `%s --help' for more information.\n", argv[0]);
            exit(-1);
            break;
    }
}

if(argc == 1) {
    usage(1);
    exit(0);
}

// Validate user input or Default it
if(!tgif_session_filepath ||
    (tgif_session_qrcode_version != 1 && tgif_session_qrcode_version != 2 &&
    tgif_session_qrcode_version != 8 && tgif_session_qrcode_version != 15) ||
    (tgif_session_qrcode_fps != 1 && tgif_session_qrcode_fps != 2 &&
    tgif_session_qrcode_fps != 5 && tgif_session_qrcode_fps != 8 &&
    tgif_session_qrcode_fps != 10)
) {
    fprintf(stderr, "No input filepath provided, or version or FPS is incorrect.\n");
    exit(-1);
}

// Static tables
frame_rounding_correction = 4; // Allow for variable ECC encoding
frame_bytes_version[1] = 14 - frame_rounding_correction;
frame_bytes_version[2] = 26 - frame_rounding_correction;
frame_bytes_version[8] = 152 - frame_rounding_correction;
frame_bytes_version[15] = 412 - frame_rounding_correction;

// Customisable parameters
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
mode_ecc_level = QR_ECLEVEL_M;           // _L, _M, _Q, _H
mode_pixel_size = 3;                     // Size according to your display
mode_margin_size = 1;

// File names
/*
$user_output_file = "TGXf-v" . $tgxf_session_qrcode_version . "-" .
  $tgxf_session_qrcode_fps . "fps-" . $mode_pixel_size . "px.gif";
*/
basename = get_basename_for_file(tgxf_session_filepath);

// Calculations based on custom parameters
tgxf_session_qrcode_bytes = frame_bytes_version[tgxf_session_qrcode_version];
read_bytes = tgxf_session_qrcode_bytes - 1; // Subtract the Control Byte
duration = 100 / tgxf_session_qrcode_fps;
tgxf_session_file_size = get_size_for_file(tgxf_session_filepath);
tgxf_session_total_crc32 = get_crc_for_file(tgxf_session_filepath);

// Initialisation
tgxf_session_frame_count = 0;
if((tgxf_packet = (unsigned char *)malloc(tgxf_session_qrcode_bytes)) == NULL) {
    fprintf(stderr, "ERROR(TGXf): unable to allocate frame buffer\n");
    return -1;
}
if((raw_data = (unsigned char *)malloc(read_bytes)) == NULL) {
    fprintf(stderr, "ERROR(TGXf): unable to allocate raw data buffer\n");
    return -1;
}

// Text Setup
if(in_ascii) {
    printf("\033[0;30;47m"); // White on Black
    printf("\033[2J"); // Clear screen
    printf("\033[0;0H");
    printf("\n");
    sleep(2); // Let camera contrast settle
}

// TGXf CONTROL -> START -> FILENAME
memset(tgxf_packet, 0, tgxf_session_qrcode_bytes);
build_control_frame(
    tgxf_packet,
    TGXF_CONTROL_TYPE_START,
    TGXF_CONTROL_SUBTYPE_START_FILENAME,
    basename, strlen(basename));
render_frame(tgxf_packet, tgxf_session_qrcode_bytes, tgxf_session_qrcode_version,
    mode_ecc_level, mode_pixel_size, mode_margin_size, in_ascii);
if(in_ascii) // Remove if graphic images are displayed real-time
    usleep(duration * 10000);

// TGXf CONTROL -> START -> FILESIZE
memset(tgxf_packet, 0, tgxf_session_qrcode_bytes);
build_control_frame(
    tgxf_packet,
    TGXF_CONTROL_TYPE_START,
    TGXF_CONTROL_SUBTYPE_START_FILESIZE,
    &tgxf_session_file_size, 3);
render_frame(tgxf_packet, tgxf_session_qrcode_bytes, tgxf_session_qrcode_version,
    mode_ecc_level, mode_pixel_size, mode_margin_size, in_ascii);
if(in_ascii) // Remove if graphic images are displayed real-time
    usleep(duration * 10000);

// TGXf CONTROL -> START -> QR_CODE_BYTES
memset(tgxf_packet, 0, tgxf_session_qrcode_bytes);
build_control_frame(
    tgxf_packet,
    TGXF_CONTROL_TYPE_START,
    TGXF_CONTROL_SUBTYPE_START_QR_CODE_BYTES,
    &read_bytes, 3);
render_frame(tgxf_packet, tgxf_session_qrcode_bytes, tgxf_session_qrcode_version,
    mode_ecc_level, mode_pixel_size, mode_margin_size, in_ascii);
if(in_ascii) // Remove if graphic images are displayed real-time
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
    usleep(duration * 10000);

// TGXf CONTROL -> START -> QRCODE_FPS
memset(tgxf_packet, 0, tgxf_session_qrcode_bytes);
build_control_frame(
    tgxf_packet,
    TGXF_CONTROL_TYPE_START,
    TGXF_CONTROL_SUBTYPE_START_QRCODE_FPS,
    &tgxf_session_qrcode_fps, 3);
render_frame(tgxf_packet, tgxf_session_qrcode_bytes, tgxf_session_qrcode_version,
    mode_ecc_level, mode_pixel_size, mode_margin_size, in_ascii);
if(in_ascii) // Remove if graphic images are displayed real-time
    usleep(duration * 10000);

if((fp = fopen(tgxf_session_filepath, "rb")) != NULL) {
    // TGXf DATA (one data frame per loop iteration)
    max_blocks = ceil((float)tgxf_session_file_size / (float)read_bytes);
    for(block_idx = 0; block_idx < max_blocks; block_idx++) {
        // last block an odd size?
        read_delta = tgxf_session_file_size - (block_idx * read_bytes);
        if(read_delta > 0 &&
            read_delta < read_bytes &&
            block_idx == (max_blocks - 1)) {
            read_bytes = read_delta;
        }
        if(fread(raw_data, read_bytes, 1, fp)) {
            memset(tgxf_packet, 0, tgxf_session_qrcode_bytes);
            build_data_frame(tgxf_packet, tgxf_session_frame_count, raw_data, read_bytes);
            render_frame(tgxf_packet, tgxf_session_qrcode_bytes, tgxf_session_qrcode_version,
                mode_ecc_level, mode_pixel_size, mode_margin_size, in_ascii);
            if(in_ascii) // Remove if graphic images are displayed real-time
                usleep(duration * 10000);
        }
        tgxf_session_frame_count++;
        if(tgxf_session_frame_count > 15)
            tgxf_session_frame_count = 0;
    }
    fclose(fp);
}

// TGXf CONTROL -> STOP -> COMPLETE
memset(tgxf_packet, 0, tgxf_session_qrcode_bytes);
build_control_frame(
    tgxf_packet,
    TGXF_CONTROL_TYPE_STOP,
    TGXF_CONTROL_SUBTYPE_STOP_COMPLETE,
    &tgxf_session_total_crc32, 5);
render_frame(tgxf_packet, tgxf_session_qrcode_bytes, tgxf_session_qrcode_version,
    mode_ecc_level, mode_pixel_size, mode_margin_size, in_ascii);
if(in_ascii) // Remove if graphic images are displayed real-time
    usleep(duration * 10000);

// Text Cleanup
if(in_ascii) {
    sleep(1);
    printf("\033[0;37;0m"); // Black on White
    printf("\033[2J");
    printf("\033[0;0H");
}

free(raw_data);
free(tgxf_packet);

return 0;
}
```

Note that platform “endian-ness” has not been factored in this code – it has been tested on x86 only.

3.3.3.3 Compiling the TGXf Transmit C Source

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

Install the dependencies first;

```
sudo apt-get install libgd2-xpm-dev
```

The following command should compile the source;

```
gcc -Wall -c tgxf-transmit.c
```

And the following should link tgxf-transmit against the math, GD and QREncode shared libraries;

```
gcc tgxf-transmit.o -lqrencode -lm -lgd -o tgxf-transmit
```

3.3.3.4 Using the TGXf Transmit implementation

Once compiled and linked, the demonstrator application “tgxf-transmit” is capable of operating as a TGXf server (transmitter). To get help from the program, use the following command;

```
./tgxf-transmit -h
```

You should see the following output;

```
ThruGlassXfer Linux Transmit Reference Code

Usage: tgxf-transmit [OPTIONS]...
-h, --help    Display this help message.
-i FILENAME, --input=FILENAME
               File to encode and transmit.
-a, --ascii   Encode to ASCII output.
-g, --graphic Encode to Graphic output (default).
-v {1,2,8,15}, --version={1,2,8,15}
               QRcode symbol version to use. (default=8)
-f {1,2,5,8,10}, --fps={1,2,5,8,10}
               QRcode symbols to display per second. (default=5)
```

Unlike the PHP Transmit example, the graphical implementation in this program will not produce an animated GIF. Instead this program will write one PNG per TGXf frame to the current working directory.

Note that this a demonstration only; implementation specific considerations – such as pixel scale to screen size – should be evaluated.

3.3.4 Reference Implementation Transmission Examples

The following examples transmit a 40,123 byte (octet) payload (a JPG graphic “test.jpg”, md5 hash of 836b53e26f0f3e6cd99148837da2cd80) in various configurations, though all with an output scale of 3 pixels per QR code pixel.

VER	FPS	Link to Example (HTTP)	Size (Bytes)
1	1	http://midnightcode.org/projects/TGXf/screen/TGXf-v1-1fps-3px.gif	2,291,865
1	2	http://midnightcode.org/projects/TGXf/screen/TGXf-v1-2fps-3px.gif	2,291,873
1	5	http://midnightcode.org/projects/TGXf/screen/TGXf-v1-5fps-3px.gif	2,291,879
1	8	http://midnightcode.org/projects/TGXf/screen/TGXf-v1-8fps-3px.gif	2,291,886
1	10	http://midnightcode.org/projects/TGXf/screen/TGXf-v1-10fps-3px.gif	2,291,872
2	1	http://midnightcode.org/projects/TGXf/screen/TGXf-v2-1fps-3px.gif	1,309,669
2	2	http://midnightcode.org/projects/TGXf/screen/TGXf-v2-2fps-3px.gif	1,309,667
2	5	http://midnightcode.org/projects/TGXf/screen/TGXf-v2-5fps-3px.gif	1,309,679
2	8	http://midnightcode.org/projects/TGXf/screen/TGXf-v2-8fps-3px.gif	1,309,670
2	10	http://midnightcode.org/projects/TGXf/screen/TGXf-v2-10fps-3px.gif	1,309,674
8	1	http://midnightcode.org/projects/TGXf/screen/TGXf-v8-1fps-3px.gif	577,247
8	2	http://midnightcode.org/projects/TGXf/screen/TGXf-v8-2fps-3px.gif	577,230
8	5	http://midnightcode.org/projects/TGXf/screen/TGXf-v8-5fps-3px.gif	577,249
8	8	http://midnightcode.org/projects/TGXf/screen/TGXf-v8-8fps-3px.gif	577,257
8	10	http://midnightcode.org/projects/TGXf/screen/TGXf-v8-10fps-3px.gif	577,238
15	1	http://midnightcode.org/projects/TGXf/screen/TGXf-v15-1fps-3px.gif	474,497
15	2	http://midnightcode.org/projects/TGXf/screen/TGXf-v15-2fps-3px.gif	474,506
15	5	http://midnightcode.org/projects/TGXf/screen/TGXf-v15-5fps-3px.gif	474,510
15	8	http://midnightcode.org/projects/TGXf/screen/TGXf-v15-8fps-3px.gif	474,507
15	10	http://midnightcode.org/projects/TGXf/screen/TGXf-v15-10fps-3px.gif	474,495

Table 3: TGXf Reference Implmentation Transmission Examples

VER = START/QRCODE_VERSION

FPS = START/QRCODE_FPS

3.3.5 Implementation Considerations

The following considerations should be evaluated by implementers;

1. Client Errors

No effort has gone into defining an in-band mechanism to signal transmission side issues such as a “File not found” or other configuration issues. This should be alerted through native platform controls (such as output to “standard error” on the command line or an “alert box” in a graphical environment).

2. Frame Size Selection

The larger the frame size the more efficient the “data” transfer (more data per packet, same over-head per packet) but the less efficient the “control” channel (same control information in increased frame volume).

3. Image Colour Selection

Use of colour to distinguish Control Frames from Data Frames in this specification is for (human) reader clarity only. The stronger the contrast the greater the reliability – Black on White is recommended in production use.

4. Input Validation

The reference code does not filter the file name before encoding. This is deliberate. Receiving implementations must not blindly trust incoming data. Paths and inappropriate characters must be removed from the inbound data stream. Likewise, strings must not be trusted as zero terminated. Directory traversal and buffer overflow exploits will plague implementations that do not validate incoming data.

5. Performance Considerations

Implementations that are slow to process transmission sequences may be better off generating the entire sequence as a batch and “playing it back” rather than generating it real-time. This could be an animated graphic, such as the reference code, or a video or even an a file of ANSI content, for example. A similar consideration should be given to slow receivers (where the problem will be much more acute). If Frames are likely to be missed then consider recording the video and allowing the user to process it at a more convenient time, rather than insisting on a lower frame rate from the sender.

6. Transmit Only versus Transmit and Receive

It is worth considering that some platforms without cameras may be able to receive TGXf transfers. One example might be an animated image received as an email attachment.

7. 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 TGXf sender.

3.4 Uploading the Transmit Software (Reusing the First Principle is the Key)

An interesting consideration at this point is that we have created the ultimate in portable data exporting software but it requires “transmit” software to be run from the computer with the data on it. So how do we upload the TGXf transmit client in the first place?

We can leverage the first principle: the keyboard is another user-controlled bit machine, meaning that we've got an inbound communications channel. But the keyboard is a mechanical interface, so let's now replace the keyboard with a more useful digital consumer device.

3.4.1 Digital Programmable Keyboard – Arduino Leonardo

The Arduino (hobby microprocessor) community have developed a platform called the "Leonardo".

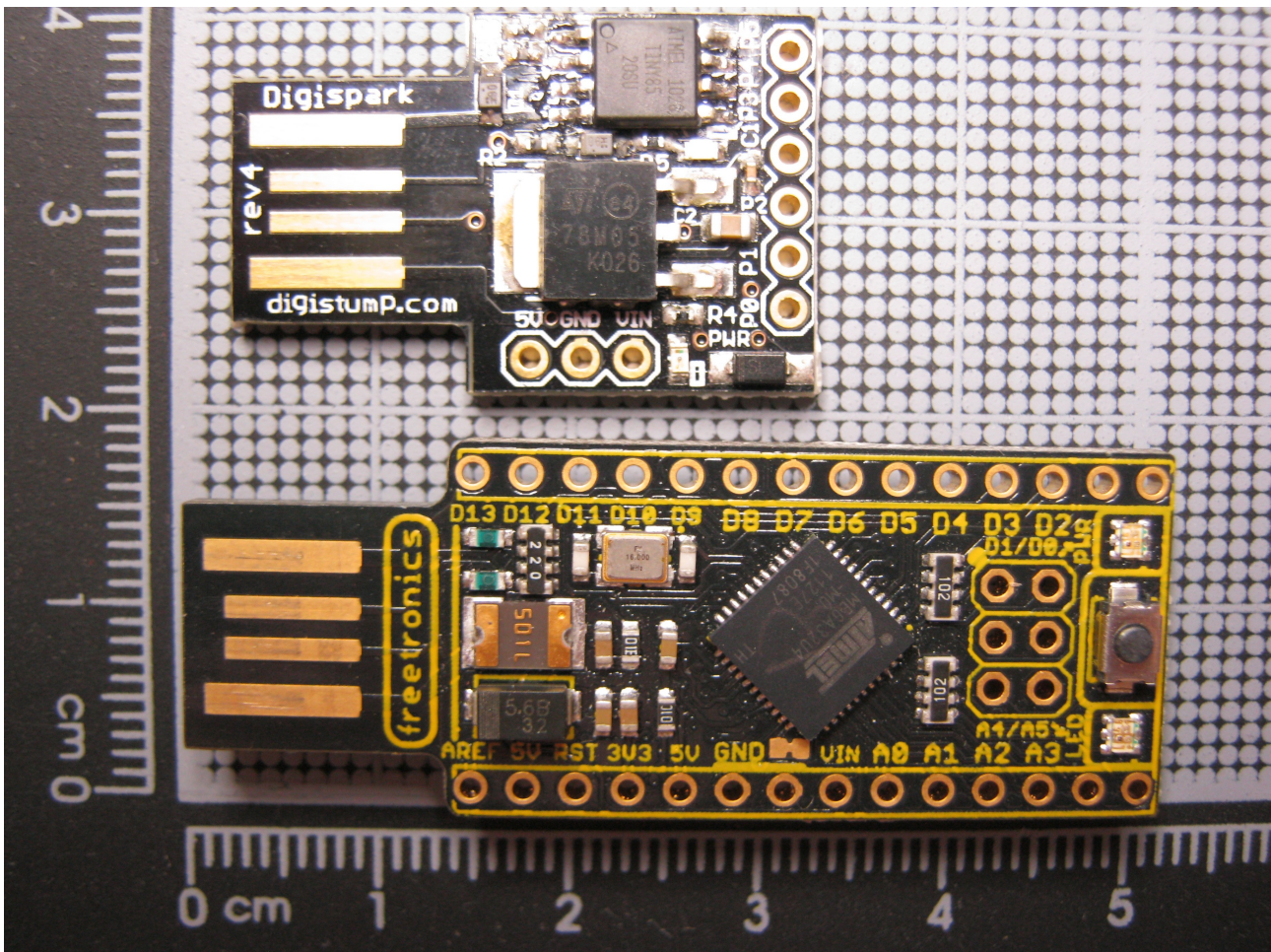


Illustration 2: Arduino Leonardo Compatible Platforms from Digistump and Freetronics

Unlike its predecessors, implementations of the Leonardo platform typically come with a male USB port that is software configurable. I.e. it can present itself as a USB HID (Human Interface Device), meaning that will look electronically like a keyboard to the host Operating System.

This is perfect for our purposes as it becomes a neat digital replacement for the mechanical keyboard, only. It will not require modification of the target computer from either a hardware or a software/driver perspective (other than to physically unplug the existing USB keyboard device and to plug in the Leonardo).

A Leonardo board can be bought off-the-shelf for a few dollars. You want to select a board with enough flash memory to store your upload so that you don't have to add an SD (portable flash memory) module (though you can if want to). I originally set my heart on the minuscule crowd-funded Digispark development board but it has only 6k of flash, where-as I've chosen the larger and more expensive Freetronics Leostick (from my local JayCar store) because it has 32k of flash;

Digistump Digispark	http://digistump.com/products/1
Freetronics Leostick	http://www.freetronics.com/products/leostick

From a software perspective, the Arduino open source libraries include “Keyboard.write()” which allows the Arduino Leonardo to "type" anything that you want into the connected/host computer¹⁸. The result is electronically and digitally identical to a generic USB keyboard.

3.4.2 What to Type?

If you're really keen you could send source code. This has the advantage of being both text based and portable. But for the sake of argument we will assume that your target computer system doesn't have a compiler, so we will send the pre-compiled txf-transmit binary program.

Sending binary data requires us to encode that data into a representation that can be "typed" in by the Arduino "keyboard". The simplest encoding scheme is hexadecimal where each source nibble (four bits) is encoded as the keyboard keys 0-9 and a-f, and where two key presses represent one whole source byte. In itself this isn't particularly useful, as the target computer will simply receive a huge pile of text that will need to be decoded to restore the source binary program.

Instead, we will instruct our little Leonardo to type in a BASH shell script or Perl script that will include the encoded data. When that script is executed natively on the target computer it will decode and restore the source binary program itself.

For larger or more complicated packages, the same process can be used to encode, send, decode and restore a tar ball or other binary bundle. In 2007 the Linux Journal showed how to make a self extracting shell script¹⁹ which is a script that has a tar-ball concatenated on to it. This isn't new (I believe that the Nessus and the Nvidia drivers have been doing this for much longer), but it is a concept that can be leveraged if you have a complex payload. For this proof of concept, the additional level of complexity isn't required.

¹⁸ <http://arduino.cc/en/Reference/KeyboardWrite>

¹⁹ <http://www.linuxjournal.com/node/1005818>

3.4.3 BASH Script that Generates Arduino Keyboard Upload “.ino”

To complete the proof of concept, the following BASH shell script is for encodeBinKey.sh. It will take a binary file (up to 25KB in size), encode it as a variable that is stored in flash memory in Arduino “ino” (source code) with a program that “types” that data in to a host via a native keyboard HID, writing out either a Perl or BASH shell script.

```
#!/bin/bash
export LC_ALL=C

if [ "${#}" != "2" ]
then
  echo "Usage: ${0} [perl|bash] file.bin"
  echo
  exit 1
fi

if [ ! "${1}" == "perl" -a ! "${1}" == "bash" ]
then
  echo "First parameter must be either \"perl\" or \"bash\"."
  echo
  exit 1
fi

if [ ! -f "${2}" ]
then
  echo "File \"${1}\" not found."
  echo
  exit 1
fi

if [ ! -s "${2}" ]
then
  echo "File \"${1}\" is empty."
  echo
  exit 1
fi

size=$(stat -c%s "${2}")
size=$(( $size + 0 ))
if [ $size -gt 25000 ]
then
  echo "File \"${2}\" is too large ($size > 25,000)."
  echo
  exit 1
fi

write_first () {
  cat << EOFFIRST
/*
```

```

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

PUBLIC
ThruConsoleXfer (TCXf) White Paper

\`""`\`

Copyright (C) 2004-2014

"Ian (Larry) Letter" <ian dot letter at midnightcode dot org>

Midnight Code is a registered trademark of Ian Letter.

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 <avr/pgmspace.h>

EOFIRST
}

write_next () {
    size=$(stat -c%s "${2}")
    printf "int for_perl = "
    if [ "${1}" == "perl" ]
    then
        printf "1;\n"
    else
        printf "0;\n"
    fi
    printf "int with_light = 1;\n"
    printf "int with_sound = 1;\n"
    printf "int start_delay = 10;\n"
    printf "\n"
    printf "PROGMEM prog_uchar file_bytes[%d]={\n" "${size}"
}

write_last () {
    cat << EOLAST
};

int led = 13;
int piezo = 11;
int in_row = 0;
int run_count = 0;

void type_interpreter() {
    if(for_perl) {
        Keyboard.println("#!/usr/bin/perl");
    } else {
        Keyboard.println("#!/bin/bash");
    }
    Keyboard.write(KEY_RETURN);
}

void type_nibble(unsigned char b) {
    unsigned char result;
    if(b < 10) {
        result = 48 + b;
    } else {
        result = 65 + b - 10;
    }
}
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
    }
    Keyboard.write(result);
}

void type_byte(unsigned char b) {
    unsigned char nibble;
    Keyboard.write(92);
    Keyboard.write(120);
    nibble = (b>>4) & 0xf;
    type_nibble(nibble);
    nibble = b & 0xf;
    type_nibble(nibble);
}

void row_end() {
    if(in_row) {
        if(for_perl) {
            Keyboard.write(34);
            Keyboard.write(59);
        } else {
            Keyboard.write(39);
        }
        Keyboard.write(KEY_RETURN);
    }
    in_row = 0;
    if(with_light) {
        digitalWrite(led, LOW);
    }
}

void row_start() {
    if(with_light)
        digitalWrite(led, HIGH);
    if(!in_row) {
        if(for_perl) {
            Keyboard.print("print ");
            Keyboard.write(34);
        } else {
            Keyboard.print("printf ");
            Keyboard.write(39);
        }
    }
    in_row = 1;
}

void new_row() {
    row_end();
    row_start();
}

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 setup() {
  pinMode(led, OUTPUT);
  Keyboard.begin();
  delay(start_delay * 1000);
  user_notify(3);
  delay(750);
  type_interpreter();
  row_start();
  for(int i = 0; i < sizeof(file_bytes); i++) {
    if(i > 0 && i % 16 == 0)
      new_row();
    type_byte(pgm_read_byte_near(file_bytes + i));
  }
  row_end();
  delay(500);
  user_notify(1);
  delay(1000);
}

void loop() {

}
EOLAST
}

write_first
write_next "${1}" "${2}"

counter=0
first=0
cat "${2}" | hexdump -ve '1/1 "%.2x"' | while read -n2 char
do
  if [ ${first} != 0 ]
  then
    printf ', '
  fi
  first=1
  if [ $counter -eq 8 ]
  then
    printf '\n'
    counter=0
  fi
  counter=$(( $counter + 1 ))
  if [ $counter -eq 1 ]
  then
    printf '    '
  fi
  printf '0x%s' "$char"
done
printf '\n'

write_last
```


3.4.4 Generating and Compiling the Arduino Keyboard Upload Code

Save the above shell code to encodeBinKey.sh and change it's permissions to make it executable.

```
chmod 755 encodeBinKey.sh
```

Generate the keyboard upload source code for the tgif-transmit binary program (the following command assumes that both programs are in the working directory, and that you want a BASH script typed into the target computer).

```
./encodeBinKey.sh bash ./tgif-transmit > keyup.ino
```

If all went well you should have an Arduino keyboard program complete with a hex encoded version of the binary file, in keyup.ino. Open that file up in a text editor, and copy the content.

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 Arduino 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;
```

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

3.4.5 Performing the Upload

On the target computer, open up a text editor. If you're using a graphical user interface (GUI) make sure that the “focus” is on the text editor. If you're using “vi” then make sure you are in “input” mode.

```
vi /tmp/dump.sh  
i
```

Unplug the USB keyboard and connect the Arduino to the target computer. After three beeps a

stream of characters will appear in the text editor. This data should be recognisable as script for the appropriate language (per the previous section).

When the entire script has been entered, unplug the Arduino and reconnect the USB keyboard. Save the script and change its permissions to make it executable.

```
<esc>  
:wq  
chmod 755 /tmp/dump.sh
```

Now execute the script and retrieve the original binary program – remembering to change its permissions to make it executable.

```
/tmp/dump.sh > /tmp/tgxf-transmit  
chmod 755 /tmp/tgxf-transmit
```

3.5 Milestone Achievement with TGXf and Keyboard Uploader

For implementation simplicity the TGXf protocol has been defined as supporting the following QR code versions and their respective usable capacities;

- v1; 21x21 pixels, 15% ECC, 14 bytes per frame, 10 usable bytes
- v2; 25x25 pixels, 15% ECC, 26 bytes per frame, 22 usable bytes
- v8; 49x49 pixels, 15% ECC, 152 bytes per frame, 148 usable bytes
- v15; 77x77 pixels, 15% ECC, 412 bytes per frame, 408 usable bytes

The TGXf protocol also allows frame rates of 1, 2, 5, 8 and a maximum of 10 frames per second (to allow at least a 2x over-sample on 30fps cameras).

Applying the matrix you get a data transfer rate of between 10 bytes per second (80bps) and 4,080 bytes per second (32kbps). Which isn't shabby for those of us who started out at 1200/75 baud, and considering that in all cases (ECC, resolution and FPS) I've chosen conservative values, and that this is a binary clean communications channel.



By combining the TGXf reference code with the Arduino based uploading mechanism described above you now have unfettered access to export and disseminate binary content from the target computer on a file-by-file basis.

In an Enterprise Security Architecture context, this has significant implications.

3.6 Through-Keyboard Transfer Protocol Specification (TKXf)

In the previous section we walked through sending binary data to the target computer - an inbound communications channel in its own right. That channel could equally be used to complement the TGXf to produce a full duplex console transfer mechanism. So let's define the TKXf and see what's possible.

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.

3.6.1 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.

3.6.1.1 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.

3.6.1.2 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.

3.6.1.3 Packet Translation

3.6.1.3.1 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

Table 4: TKXf transmit – source nibble to USB HID packet value mapping

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.

3.6.1.3.2 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)

Table 5: TKXf receive – received ASCII character to nibble value mapping

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.

3.6.1.3.3 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

Table 6: TKXf transmit – USB HID key deduplication scheme

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		
------	------	------	------	--	--

PUBLIC
ThruConsoleXfer (TCXf) White Paper

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
------	------	------	------	------	------

3.6.1.3.4 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

Table 7: TKXF receive – ASCII data reduplication scheme

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.

3.6.1.4 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).

3.6.2 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.

3.6.2.1 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.

3.6.2.2 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.

3.6.2.3 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.

3.6.2.4 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.

3.7 Through-Keyboard Transfer Reference Implementations

The receive side of Through-Keyboard Transfer is able to be implemented on any compute platform with a terminal (or user text input, presumably including web browsers). The transmit side of Through-Glass Transfer requires a serial device (Keyboard Stuffer) to act as a digital keyboard for the target computer.

This section includes reference implementations that, while far from pretty, are functional demonstrators of the TKXf protocol.

3.7.1 Wired versus Wireless USB HID

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.

3.7.1.1 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.

3.7.1.2 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.

3.7.1.3 Custom Hardware – Bluetooth Shield on Arduino Leo

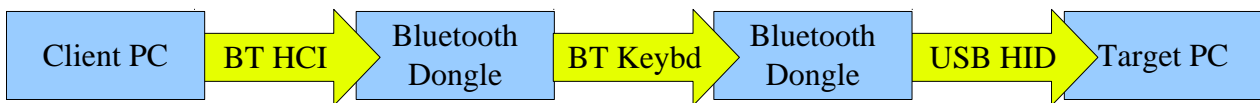
This implementation uses an L2CAP capable Bluetooth shield on an Arduino Leo capable board. In this model the USB Serial adapter from the wired implementation is replaced with a USB Bluetooth dongle and the Keyboard Stuffer is given a Bluetooth module that is capable of L2CAP.



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

3.7.1.4 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.

3.7.2 C Reference Implementation – Transmit

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

3.7.2.1.1 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)
    MMM)
    (MMM`
M I D N I G H T C o D E
    (NMMF
    NMML
    NMML
    4MMNL
    `4HNNL
    `;
    MN.
    (MML
    MMML
    MHNH
    .MMM
    .NMH
    `.#F
    `..

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

    Midnight Code is a registered trademark of Ian Letter.

    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
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
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;
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
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 keyboard 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;
}
```


PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
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.

3.7.2.1.2 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
```

3.7.2.1.3 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
```

3.7.3 Hardware Reference Implementation – Keyboard Stuffer Device – Transmit

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.

3.7.3.1 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).

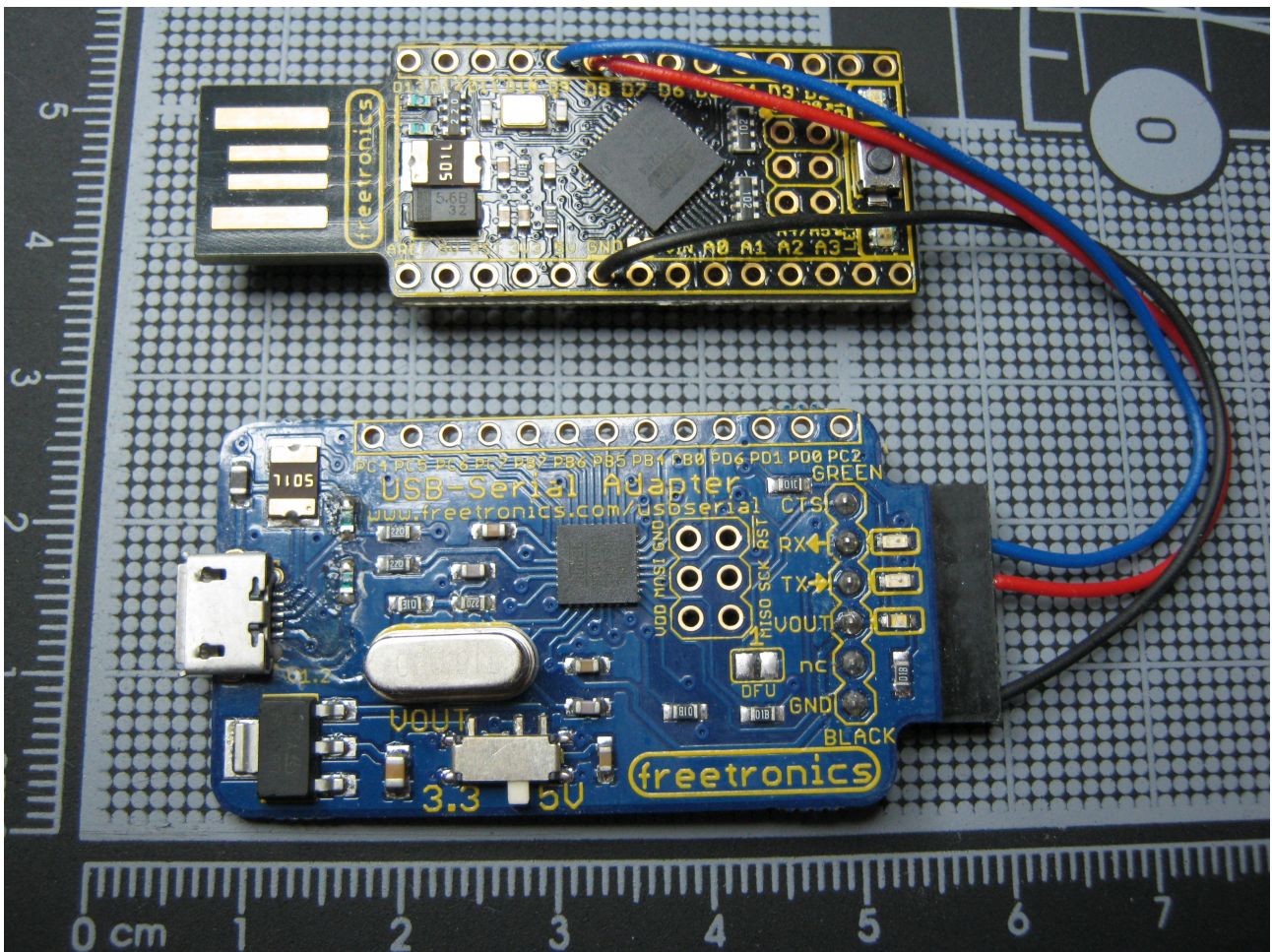


Illustration 3: TKXf Keyboard Stuffer – Wired USB HID Reference Implementation

3.7.3.2 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/
```

3.7.3.3 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

3.7.3.4 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).

3.7.3.5 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_`
                                   `u"u"
                                     `

                                     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);
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
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.

3.7.3.6 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 Arduino 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.

3.7.3.7 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.

3.7.4 C Reference Implementation – Receive

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

3.7.4.1 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_   `.#F
    `.#F

    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;
}
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

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

    return;
}

void
restore_terminal() {

    tcsetattr(0, TCSANOW, &save_term);

    return;
}

// USB keyboard 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') {
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
    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.

3.7.4.2 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
```

3.7.4.3 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.

3.7.5 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.

3.8 Milestone Achievement with TKXf and TGXf

On its own the TKXf reference implementation did not produce a substantial improvement over the Leonardo file uploader. In practical terms the real improvement has been the removal of a file size limit (you can go back to using the much neater Digispark, for example).



Using the TKXf reference code described above you now have unfettered access to import binary content to the target computer on a file-by-file basis.

In an Enterprise Security Architecture context, this has significant implications.

But at this point we have also amassed enough technology to change the interface. i.e.:

- The TKXf reference implementation has a maximum of 12kbps²⁰.
- The TGXf reference implementation has a maximum of 32kbps.

With a small amount of imagination these two protocols combined would yield an asymmetric digital communications link (32kbps down, 12kbps up, from the consumer's perspective), capable of relaying full-duplex (bi-directional) data.

²⁰ Note that it is believed (though untested) that the “key clear” packet can be used for data so long as a unique hex encoding scheme is used in each alternate HID packet, which alone would increase the performance of TKXf to 24kbps. If an appropriate keyboard encoding scheme could be found for it, base64 encoding of the source data would increase this throughput to 32kbps.

3.9 Data, take the Con (TCXf)

In this section we will integrate the TGXf and TKXf reference programs (transmit and receive implementation for each) to create a full-duplex (or bi-directional) binary clear socket, through the console (screen and keyboard).

3.9.1 Integration – TCXf Application Architecture

Admittedly I had to refer to my own ASCII-art software architecture diagram a number of times while integrating the reference applications – mostly because of the added complexity of working “behind” the socket (socket send is programmer's receive, which connects to integrated program's transmit). So for clarity through-out this chapter and future discussion, please find my original ASCII-art diagram reproduced here in graphical glory:

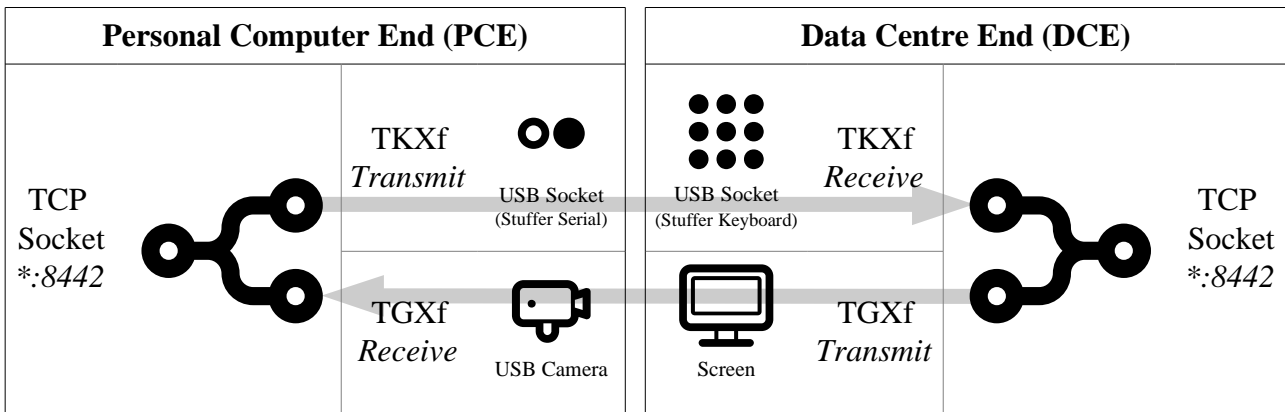


Illustration 4: TCXf Application Architecture (TKXf and TGXf Integration Architecture)

Our client and server model (or transmit and receive) model is no longer applicable when both ends are bi-directional peers. In this new model the local instance is described as the PCE (Personal Computer End), while the remote (or target) instance is described as the DCE (Data Centre End)²¹.

Note from the above diagram that all of the hardware specific interfacing requirements (the camera and serial side of the keyboard stuffer) are bound to the PCE. And at both ends of the service the software provides a TCP socket interface (rather than file interface) for data input and output.

3.9.2 Protocol Specification Deltas for TGXf and TKXf

The change in paradigm from a file based interface to a stream based interface means that all of the file related constructs can be removed from both protocols. In both cases this means stripping out any control channel requirements and simply retaining the data channels. All other aspects of both protocol specifications remain and can be seen in the TCXf implementation.

²¹ The DCE terminology has been used to make the subsequent discussion simpler, it is not of consequence in this chapter.

3.9.3 C Reference Implementation – Data Centre End (DCE)

The following C source code is for the Through Console Xfer Data Centre End (DCE) Reference Code.

```
/*
    .JNJ`
    .JNMH`
    JMMF`
    .NMM)
    MMM)
    (MMM`
    (NMMF
    NMML
    NMML
    4MMNL
    `4HNNL`
    `..
    ;.
    MN.
    (MML
    MMML
    MHNH
    .MMM
    .NMH
    .#F
    `..
    `..

M I D N I G H T C o D E

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

*/
/* tcxf-dce.c :: ThruConsoleXfer - Data Centre End */
/*
Software Architecture

TCXF PCE                                TCXF DCE
socket -> r => keyb xmt [~TKXF~] keyb rcv => w -> socket
<- w <= scrn rcv [~TGXF~] scrn xmt <= r <-

*/
// #define ANSIONLY

#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>
#include <termios.h>
#include <pthread.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <errno.h>
#include <sys/ioctl.h>
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
#include <linux/tcp.h>
#include <signal.h>
// #include <qrencode.h>
#ifdef ANSIONLY
#include <gtk/gtk.h>
#endif

int terminate = 0;
int listener_socket_fd;

typedef struct {
    int socket_fd;
    int ttyinp_fd;
} tkxf_rcv_parms;

typedef struct {
    int socket_fd;
    int in_ascii;
} tgxf_xmt_parms;

// ----- TKXF RCV --

#define DATA_MODE 0
#define CONTROL_MODE 1
#define EXIT_MODE 2

unsigned int
tkxf_setup_terminal(struct termios * save_term) {
    struct termios ubuf_term;

    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 0;
}

void
tkxf_restore_terminal(struct termios * save_term) {
    tcsetattr(0, TCSANOW, save_term);

    return;
}

// USB keyboard 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]"
// 0x15 = letter r, meaning "as per key[5]"
//
// 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.
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
int
tkxf_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;
}
```

```
int
tkxf_dispatch_char(int fd, unsigned int mode, unsigned char octet) {

    if(mode == DATA_MODE) {
        if(write(fd, &octet, 1) < 0) {
            fprintf(stderr, "TCXF Error: tkxf_data_byte/write failed: %s\n",
                strerror(errno));
            return -1;
        }
    }

    return 0;
}
```

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

    tkxf_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)
            tkxf_dispatch_char(fd, mode, key_result[0] * 16 + key_result[1]);
        key_state = !key_state;
    }

    return 0;
}
```

```
unsigned int
tkxf_octets_available(int fd) {
    int available_octets;

    // Win ioctlsocket(fd, FIONREAD, &available_octets)
    ioctl(fd, FIONREAD, &available_octets);

    if(available_octets < 0)
        return 0;

    return available_octets;
}
```

```
}

void *
tkxf_rcv(void * thread_param) {
    tkxf_rcv_parms * param;
    unsigned char user_key;
    unsigned int mode;
    unsigned int processed_octets;
    unsigned char key_buffer[6];
    unsigned int offset;
    unsigned int octets;

    param = (tkxf_rcv_parms *)thread_param;

    mode = DATA_MODE;
    offset = 0;
    processed_octets = 0;
    while(!terminate && mode != EXIT_MODE) {
        octets = tkxf_octets_available(param->ttyinp_fd);
        if(!octets) {
            usleep(50);
            continue;
        }
        if(read(param->ttyinp_fd, &user_key, 1) != 1) {
            fprintf(stderr, "TCXF Error: tkxf_rcv/read failed: %s\n",
                strerror(errno));
            mode = EXIT_MODE;
        }
        processed_octets++;
        switch(user_key) {
            // HEX data
            case '0':
            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':
            // Dedupe data
            case 'm':
            case 'n':
            case 'o':
            case 'p':
            case 'q':
                key_buffer[offset] = user_key;
                offset++;
                if(offset == 6) {
                    tkxf_process_key_buffer(param->socket_fd, key_buffer, offset, mode);
                    offset = 0;
                }
                break;
            case '.':
                if(offset) {
                    fprintf(stderr, "DEBUG: Last %d bytes\n", offset);
                    tkxf_process_key_buffer(param->socket_fd, key_buffer, offset, mode);
                }
                mode = EXIT_MODE;
                offset = 0;
                break;
            case ',':
                if(offset)
                    tkxf_process_key_buffer(param->socket_fd, key_buffer, offset, mode);
        }
    }
}
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
        mode = CONTROL_MODE;
        offset = 0;
        break;
    case ' ':
        if(offset)
            tkxf_process_key_buffer(param->socket_fd, key_buffer, offset, mode);
        mode = DATA_MODE;
        offset = 0;
        break;
    default:
        offset = 0;
        break;
    }
}

terminate = 1;

// return processed_octets;
return;
}

// ----- TGXF XMT --

#define TGXF_DEBUG 0

#define TGXF_DISPLAY_ASCII_SIMPLE 1
#define TGXF_DISPLAY_ASCII_SIMPLE_SQUARE 2
#define TGXF_DISPLAY_ASCII_COMPRESSED 3
#define TGXF_DISPLAY_ANSI_SIMPLE 4
#define TGXF_DISPLAY_ANSI_SIMPLE_SQUARE 5

// TGXf Session Parameters (TGXf State Machine)
unsigned int tgxf_session_frame_count = 0;
unsigned int tgxf_session_debug_image_counter = 0;

// TGXf Session Parameters (TGXf Global Options)
unsigned int tgxf_session_qrcode_version = 0;
unsigned int tgxf_session_qrcode_fps = 0;
unsigned int tgxf_session_qrcode_bytes = 0;

// Global Window and Image handles
#ifdef ANSIONLY
    unsigned char * image_buffer;
    unsigned int image_buffer_size;
    unsigned int display_header = 70;
    char * xpm_data[255];
    GtkWidget *window;
    GtkWidget *aspect_frame;
    GtkWidget *image;
#endif

unsigned int
tgxf_pack_data_payload_bin(unsigned char *payload, void *data, unsigned int data_len) {

    if(!payload || !data || !data_len)
        return 0;

    memcpy(payload, data, data_len);

    return data_len;
}

unsigned char *
tgxf_build_data_frame(unsigned char * packet, unsigned int counter, void * data, unsigned
int payload_size) {
    unsigned char control_byte;
    unsigned int control_bit;
    unsigned char * payload;
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
// Zero Control Byte
control_byte = 0;

// Data Frame, Control Bit = 0 (bit 0);
control_bit = 0;
control_byte = control_byte | control_bit;

// Control Byte has incremented counter (bits 4,3,2,1)
if(counter < 0 || counter > 15)
    counter = 0;
counter = counter << 1;
control_byte = control_byte | counter;

// Frame consists of Control Byte and Data Payload
payload = packet + 1;
*packet = (control_byte) & 0xFF;
if(memcpy(payload, data, payload_size) != payload) {
    // fprintf(stderr,
    // "TCXF Error: tgxf_build_data_frame/failed to build frame: %s\n",
    // strerror(errno));
    // What else can we do here?
    return packet;
}

return packet;
}

#ifdef ANSIONLY
gboolean
tgxf_window_resize(GtkWidget *widget, GdkEvent *event, GtkWidget *window) {
    GdkPixbuf *o_pixbuf;
    GdkPixbuf *n_pixbuf;

    o_pixbuf = gtk_image_get_pixbuf(GTK_IMAGE(widget));
    if(o_pixbuf == NULL) {
        return 1;
    }
    n_pixbuf = gdk_pixbuf_scale_simple(o_pixbuf,
        widget->allocation.width, widget->allocation.height,
        GDK_INTERP_TILES);
    if(n_pixbuf == NULL) {
        return 1;
    }
    gtk_image_set_from_pixbuf(GTK_IMAGE(widget), n_pixbuf);
    gdk_pixbuf_unref(o_pixbuf);

    return 0;
}

void
tgxf_window_close(GtkWidget *widget, GdkEvent *event, gpointer data) {

    terminate = 1;
    gtk_main_quit();
}
#endif

unsigned char *
tgxf_render_frame(unsigned char * data, int length, int qr_ver, int qr_ecc, int qr_scale,
int qr_margin, int in_ascii) {
    unsigned int w;
    unsigned int h;
    unsigned int imgW;
    unsigned int imgH;
    unsigned int x;
    unsigned int y;
    unsigned int double_x;
    const char padding[5] = "    ";

```


PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
    }
  }
  // Two pixel border (right)
  for(x=w+2; x<w+4; x++) {
    blit_p = image_buffer+display_header+((y+2)*(w+5))+x;
    *blit_p = '0';
  }
  // Terminal string
  blit_p++;
  *blit_p = 0;
}
// Two pixel border (bottom)
for(y=h+2; y<h+4; y++) {
  for(x=0; x<w+4; x++) {
    blit_p = image_buffer+display_header+(y*(w+5))+x;
    *blit_p = '0';
  }
  // Terminal string
  blit_p++;
  *blit_p = 0;
}

gdk_threads_enter();
o_pixbuf = gtk_image_get_pixbuf(GTK_IMAGE(image));
n_pixbuf = gdk_pixbuf_new_from_xpm_data((const char **)xpm_data);
gtk_image_set_from_pixbuf(GTK_IMAGE(image), n_pixbuf);
gtk_widget_queue_draw(image);
if(o_pixbuf != NULL)
  gdk_pixbuf_unref(o_pixbuf);
gdk_threads_leave();

#endif
}

if(code != NULL)
  QRcode_free(code);

return data;
}

unsigned int
tgxf_octets_available(int fd) {
  int available_octets;

  // Win ioctlsocket(fd, FIONREAD, &available_octets)
  ioctl(fd, FIONREAD, &available_octets);

  if(available_octets < 0)
    return 0;

  return available_octets;
}

void *
tgxf_xmt(void * thread_param) {
  tgxf_xmt_parms * param;
  unsigned int frame_rounding_correction;
  unsigned int frame_bytes_version[64];
  unsigned int read_bytes;
  unsigned int recv_bytes;
  int mode_ecc_level;
  int mode_pixel_size;
  int mode_margin_size;
  unsigned int duration;
  unsigned int frame_number;
  unsigned char * tgxf_packet;
  unsigned char * raw_data;
  unsigned int octets;
#ifdef ANSIONLY
  GdkColor background_color;
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
unsigned int display_pixels_version[64];
unsigned char * image_ptr;
unsigned int y;
#endif
int flags;

param = (tgxf_xmt_parms *)thread_param;

// Static tables
frame_rounding_correction = 4;           // Allow for variable ECC encoding
frame_bytes_version[1] = 14 - frame_rounding_correction;
frame_bytes_version[2] = 26 - frame_rounding_correction;
frame_bytes_version[8] = 152 - frame_rounding_correction;
frame_bytes_version[15] = 412 - frame_rounding_correction;
#ifndef ANSIONLY
display_pixels_version[1] = 21;          // 1 (21x21)
display_pixels_version[2] = 25;          // 2 (25x25)
display_pixels_version[8] = 49;          // 8 (49x49)
display_pixels_version[15] = 77;         // 15 (77x77)
#endif

// Customisable parameters
mode_ecc_level = QR_ECLEVEL_M;           // _L, _M, _Q, _H
mode_pixel_size = 3;                     // Size according to your display
mode_margin_size = 1;

// Calculations based on custom parameters
tgxf_session_qrcode_bytes = frame_bytes_version[tgxf_session_qrcode_version];
read_bytes = tgxf_session_qrcode_bytes - 1; // Subtract the Control Byte
duration = 100 / tgxf_session_qrcode_fps;

// Initialisation
tgxf_session_frame_count = 0;
if((tgxf_packet = (unsigned char *)malloc(tgxf_session_qrcode_bytes))
 == NULL) {
    fprintf(stderr,
        "TCXF Error: tgxf_xmit/unable to allocate frame buffer: %s\n",
        strerror(errno));
    terminate = 1;
    return;
}
if((raw_data = (unsigned char *)malloc(read_bytes)) == NULL) {
    fprintf(stderr,
        "TCXF Error: tgxf_xmit/unable to allocate raw data buffer: %s\n",
        strerror(errno));
    free(tgxf_packet);
    terminate = 1;
    return;
}
#ifndef ANSIONLY
if(!param->in_ascii) {
    image_buffer_size =
        display_header +
        ((2 + display_pixels_version[tgxf_session_qrcode_version] + 2) *
         (2 + display_pixels_version[tgxf_session_qrcode_version] + 2 + 1));
    if((image_buffer = (unsigned char *)malloc(image_buffer_size)) == NULL) {
        fprintf(stderr,
            "TCXF Error: tgxf_xmit/unable to allocate image buffer: %s\n",
            strerror(errno));
        free(raw_data);
        free(tgxf_packet);
        terminate = 1;
        return;
    }
    memset(image_buffer, 0, image_buffer_size);
    xpm_data[0] = image_buffer;
    snprintf(xpm_data[0], 10, "%02d %02d 3 1", // 9 + 1
        display_pixels_version[tgxf_session_qrcode_version] + 4,
        display_pixels_version[tgxf_session_qrcode_version] + 4);
    xpm_data[1] = xpm_data[0] + 10;
    snprintf(xpm_data[1], 14, "          c None"); // 13 + 1
    xpm_data[2] = xpm_data[1] + 14;
}
#endif
}
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
    snprintf(xpm_data[2], 23, "1      c #000000000000"); // 22 + 1
    xpm_data[3] = xpm_data[2] + 23;
    snprintf(xpm_data[3], 23, "0      c #FFFFFFFFFFFF"); // 22 + 1
    xpm_data[4] = xpm_data[3] + 23;
    for(y=1; y<(display_pixels_version[tgxf_session_qrcode_version]+4); y++) {
        xpm_data[y+4] = xpm_data[y+3] +
            2 + display_pixels_version[tgxf_session_qrcode_version] + 2 + 1;
    }
}
#endif

// Text Setup
if(param->in_ascii) {
    printf("\033[0;30;47m"); // White on Black
    printf("\033[2J"); // Clear screen
    printf("\033[0;0H");
    printf("\n");
}
#ifdef ANSIONLY
    if(!param->in_ascii) { // Set window background white
        background_color.red = 0xffff;
        background_color.green = 0xffff;
        background_color.blue = 0xffff;
        gdk_threads_enter();
        gtk_widget_modify_bg(window, GTK_STATE_NORMAL, &background_color);
        gdk_threads_leave();
    }
#endif
sleep(2); // Let camera contrast settle

// Read from socket, render TGXF frames
while(!terminate) {
    // Data to be read on network socket?
    octets = tgxf_octets_available(param->socket_fd);
    if(!octets) {
        usleep(50);
        continue;
    }
    // If so, acquire it
    recv_bytes = read_bytes;
    if(octets < recv_bytes)
        recv_bytes = octets;
    flags = 0;
    if(recv(param->socket_fd, raw_data, recv_bytes, flags) > 0) {
        memset(tgxf_packet, 0, tgxf_session_qrcode_bytes);
        // Generate Data Frame
        tgxf_build_data_frame(tgxf_packet, tgxf_session_frame_count,
            raw_data, recv_bytes);
        // Display Data Frame
        tgxf_render_frame(tgxf_packet, tgxf_session_qrcode_bytes,
            tgxf_session_qrcode_version,
            mode_ecc_level, mode_pixel_size, mode_margin_size,
            param->in_ascii);
        // Update Data Counter
        tgxf_session_frame_count++;
        if(tgxf_session_frame_count > 15)
            tgxf_session_frame_count = 0;
        // Wait for display refresh period
        usleep(duration * 10000);
    }
}

// Text Cleanup
if(param->in_ascii) {
    sleep(1);
    printf("\033[0;37;0m"); // Black on White
    printf("\033[2J");
    printf("\033[0;0H");
}

#ifdef ANSIONLY
    if(!param->in_ascii) {
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
    free(image_buffer);
}
#endif
free(raw_data);
free(tgxf_packet);

return;
}

// ----- MAIN -----

void
logo(int tagline) {
    char midnightcode_logo_ascii[26][22] = {
        "      _JNJ_ ",
        "    .JNMH ",
        "   JMMF ",
        "  .NMM )",
        "  MMM )",
        " (MMM ",
        " M I D N I G H T C O D E ",
        " (NMMF ",
        "  NMML ",
        "   NMML ",
        "   4MMNL ",
        "   `4HNNL_ ",
        "   `\"";

};
int idx;

fprintf(stderr, "\n");
for(idx=0; idx<26; idx++) {
    if((float)((float)idx/(float)2) == (int)(idx/2)) {
        fprintf(stderr, "          \033[01;31m");
    } else {
        fprintf(stderr, "\033[31m");
    }
    if(idx == 12 || idx == 13)
        fprintf(stderr, "\033[01;37m");
    fprintf(stderr, "%s\033[0m", midnightcode_logo_ascii[idx]);
    if((float)((float)idx/(float)2) != (int)(idx/2))
        fprintf(stderr, "\n");
}
fprintf(stderr, "\n");
if(tagline)
    fprintf(stderr,
        "          ThruConsoleXfer - Data Centre End - Reference Code"
        "\n\n");

return;
}

void
usage(int help) {

    logo(1);
    if(help) {
        fprintf(stderr,
            " Usage: tcxf-pce [OPTIONS]...\n"
            "   -t PORT, --tcp=PORT      TCP port to listen on (8442)\n"
#ifdef ANSIONLY
            "   -g, --graphic           Encode to Graphic/GTK output (ANSI)\n"
#endif
            "   -v #, --version={1,2,8,15} QRcode symbol version to use (8)\n"
            "   -f #, --fps={1,2,5,8,10} Symbols displayed per second (5)\n"
            "\n"
        );
    }
}
}
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
void
signal_handler(int sig) {

    terminate = 1;
    if(listener_socket_fd)
        shutdown(listener_socket_fd, SHUT_RDWR);

    return;
}

int
main(int argc, char **argv) {
    static const struct option options[] = {
        {"help",      no_argument, NULL, 'h'},
        {"tcp",       required_argument, NULL, 't'},
#ifdef ANSIONLY
        {"graphic",   no_argument, NULL, 'g'},
#endif
        {"version",   required_argument, NULL, 'v'},
        {"fps",       required_argument, NULL, 'f'},
        {NULL, 0, NULL, 0}
    };
#ifdef ANSIONLY
    static char *optstring = "ht:gv:f:";
#else
    static char *optstring = "ht:v:f:";
#endif
    struct sigaction sa;
    pthread_t pid_r = { 0 };
    pthread_t pid_w = { 0 };
    tkxf_rcv_parms tkxf_rcv_param;
    tgxf_xmt_parms tgxf_xmt_param;
    int opt;
    int val;
    struct termios save_term;
    unsigned int tcp_port;
    struct sockaddr_in listener_addr;
    struct sockaddr_in client_addr;
    socklen_t client_len;
    int socket_fd;
    int ttyinp_fd;
    int in_ascii;
    char nulbuf;

    // Establish defaults
    tcp_port = 8442; // T*
    in_ascii = 1;
    tgxf_session_qrcode_version = 8;
    tgxf_session_qrcode_fps = 5;

    // Parse command line
    while((opt = getopt_long(argc, argv, optstring, options, NULL)) != -1) {
        switch(opt) {
            case 'h':
                usage(1);
                exit(0);
                break;
            case 't':
                tcp_port = atoi(optarg);
                break;
#ifdef ANSIONLY
            case 'g':
                in_ascii = 0;
                break;
#endif
            case 'v':
                // 1 (21x21), 2 (25x25),
                val = atoi(optarg); // 8 (49x49), 15 (77x77)
                if(val != 1 && val != 2 && val != 8 && val != 15)

```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
        val = 8;
        tgfx_session_qrcode_version = val;
        break;
    case 'f':
        // 1, 2, 5, 8, 10
        val = atoi(optarg);
        if(val != 1 && val != 2 && val != 5 && val != 8 && val != 10)
            val = 5;
        tgfx_session_qrcode_fps = val;
        break;
    default:
        fprintf(stderr, "Try `%s --help' for more information.\n", argv[0]);
        exit(-1);
        break;
    }
}

// Purdee
logo(1);

// Validate user input or Default it
fprintf(stderr, "TCXF: Using terminal device [stdin]\n");
if(in_ascii) {
    fprintf(stderr, "TCXF: Using terminal device [stdout]\n");
} else {
    fprintf(stderr, "TCXF: Using GTK window canvas [~]\n");
}
fprintf(stderr,
    "TCXF: Rendering parameters (TGXF) FPS=%d, Version=%d\n",
    tgfx_session_qrcode_fps, tgfx_session_qrcode_version);
fprintf(stderr, "\n");

// Install signal handler
memset(&sa, 0, sizeof(sa));
sa.sa_handler = &signal_handler;
if(sigaction(SIGPIPE, &sa, NULL) == -1) {
    fprintf(stderr,
        "TCXF Error: sighandler setup failed: %s\n", strerror(errno));
    return -1;
}
if(sigaction(SIGINT, &sa, NULL) == -1) {
    fprintf(stderr,
        "TCXF Error: sighandler setup failed: %s\n", strerror(errno));
    return -1;
}

// Initialize hardware
ttyinp_fd = tkxf_setup_terminal(&save_term);
if(ttyinp_fd < 0) {
    fprintf(stderr, "TCXF Error: Serial setup failed\n");
    return -1;
}
#endif
#ifdef ANSIONLY
if(!in_ascii) {

    // Threads setup
    g_thread_init(NULL);
    gdk_threads_init();
    gdk_threads_enter();

    // Setup GTK and define Window
    gtk_init(&argc, &argv);
    window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(window),
        "ThruConsoleXfer - Data Centre End - Reference Code");
    gtk_container_set_border_width(GTK_CONTAINER(window), 10);
    gtk_window_set_resizable(GTK_WINDOW(window), TRUE);
    gtk_signal_connect(GTK_OBJECT(window), "delete-event",
        GTK_SIGNAL_FUNC(tgfx_window_close), NULL);

    aspect_frame = gtk_aspect_frame_new("TGXf", 0.5, 0.5, 1, FALSE);
    gtk_container_add(GTK_CONTAINER(window), aspect_frame);
    gtk_widget_show(aspect_frame);
}
}

```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
// Add the canvas
image = gtk_image_new();
gtk_widget_set_usize(image, 100, 100);
gtk_container_add(GTK_CONTAINER(aspect_frame), image);
gtk_widget_show(image);
g_signal_connect(image, "expose-event",
    G_CALLBACK(tgxf_window_resize), (gpointer>window);

    gtk_widget_show(window);
}
#endif

// Setup networking
listener_socket_fd = socket(AF_INET, SOCK_STREAM, 0);
if(listener_socket_fd < 0) {
    fprintf(stderr, "TCXF Error: socket failed: %s\n", strerror(errno));
    tkxf_restore_terminal(&save_term);
    return -1;
}

// Configure listener socket
bzero((char *)&listener_addr, sizeof(listener_addr));
listener_addr.sin_family = AF_INET;
listener_addr.sin_addr.s_addr = INADDR_ANY;
listener_addr.sin_port = htons(tcp_port);
if(bind(listener_socket_fd, (struct sockaddr *)&listener_addr,
    sizeof(listener_addr)) < 0) {
    fprintf(stderr, "TCXF Error: bind failed: %s\n", strerror(errno));
    tkxf_restore_terminal(&save_term);
    return -1;
}

// Await one connection
listen(listener_socket_fd, 1);

// Accpet the client
client_len = sizeof(client_addr);
socket_fd = accept(listener_socket_fd,
    (struct sockaddr *)&client_addr, &client_len);
if(socket_fd < 0) {
    fprintf(stderr, "TCXF Error: connection failed: %s\n", strerror(errno));
    tkxf_restore_terminal(&save_term);
    return -1;
}
close(listener_socket_fd);
listener_socket_fd = 0;

// Configure threads
tkxf_rcv_param.socket_fd = socket_fd;
tkxf_rcv_param.ttyinp_fd = ttyinp_fd;
tgxf_xmt_param.socket_fd = socket_fd;
tgxf_xmt_param.in_ascii = in_ascii;

// Launch threads
pthread_create(&pid_r, NULL, &tgxf_xmt, (void *)&tgxf_xmt_param);
pthread_create(&pid_w, NULL, &tkxf_rcv, (void *)&tkxf_rcv_param);

#ifdef ANSIOONLY
// Start the Window
if(!in_ascii) {
    gtk_main(); // blocking
    gdk_threads_leave();
    terminate = 1;
}
#endif

// Wait around for exit
while(!terminate) {
    if(recv(socket_fd, &nulbuf, sizeof(nulbuf), MSG_PEEK|MSG_DONTWAIT) == 0) {
        fprintf(stderr, "TCXF: client connection closed\n");
        terminate = 1;
    }
}
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
    }
    usleep(250);
}

// Shutdown and bail
pthread_join(pid_r, NULL);
pthread_join(pid_w, NULL);
tkxf_restore_terminal(&save_term);
close(socket_fd);

return 0;
}
```


3.9.4 C Reference Implementation – Personal Computer End (PCE)

The following C source code is for the Through Console Xfer Personal Computer End (PCE) Reference Code.

```
/*
                                     .JNJ`
                               .JNMH`
                             JMMF`
                           .NMM)
                          (MM)
                         (MMM`
M I D N I G H T       C o D E
                         (NMMF
                          NMML
                         NMML
                        4MMNL
                       `4HNNL
                      `u u u`

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

*/
/* tcxf-pce.c :: ThruConsoleXfer - Personal Computer End */
/*
Software Architecture

TCXF PCE                                TCXF DCE
socket -> r => keyb xmt [~TKXF~] keyb rcv => w -> socket
<- w <= scrn rcv [~TGXF~] scrn xmt <= r <-

*/
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <string.h>
#include <getopt.h>
#include <termios.h>
#include <pthread.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <errno.h>
#include <sys/ioctl.h>
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
#include <linux/tcp.h>
#include <signal.h>
#include <sys/time.h>
#include <zbar.h>

int terminate = 0;
int listener_socket_fd;

typedef struct {
    int socket_fd;
    int serial_fd;
} tkxf_xmt_parms;

typedef struct {
    int socket_fd;
    const char *camera_device;
    int display;
    int verbose;
} tgxf_rcv_parms;

// ----- TKXF XMT --

#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
tkxf_set_interface(int fd, int speed, int parity) {
    struct termios tty;

    memset (&tty, 0, sizeof tty);
    if(tcgetattr (fd, &tty) != 0) {
        fprintf(stderr, "TCXF Error: tkxf_set_interface/tcgetattr failed: %s\n",
            strerror(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) {
        fprintf(stderr, "TCXF Error: tkxf_set_interface/tcsetattr failed: %s\n",
            strerror(errno));
    }
}
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
    return -1;
}

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

return 0;
}

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

    memset(&tty, 0, sizeof tty);
    if(tcgetattr(fd, &tty) != 0) {
        fprintf(stderr, "TCXF Error: tkxf_set_blocking/tcgetattr failed: %s\n",
            strerror(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)
        fprintf(stderr, "TCXF Error: tkxf_set_blocking/tcsetattr failed: %s\n",
            strerror(errno));

    return;
}

int
tkxf_setup_port(const char * serial_dev) {
    int fd;

    fd = open(serial_dev, O_RDWR|O_NOCTTY|O_SYNC);
    if(fd < 0) {
        fprintf(stderr, "TCXF Error: tkxf_setup_port/open failed on %s: %s\n",
            serial_dev, strerror(errno));
        return -1;
    }
    tkxf_set_interface(fd, B38400, 0); // set speed to 38,400 bps, 8n1
    tkxf_set_blocking(fd, 1);         // set blocking

    return fd;
}

int
tkxf_data_mode(int fd) {
    unsigned char octet;

    octet = 0x2c;
    if(write(fd, &octet, 1) < 0) {
        fprintf(stderr, "TCXF Error: tkxf_data_byte/write failed: %s\n",
            strerror(errno));
        return -1;
    }

    return 0;
}

int
tkxf_stop_mode(int fd) {
    unsigned char octet;

    octet = 0x37;
    if(write(fd, &octet, 1) < 0) {
        fprintf(stderr, "TCXF Error: tkxf_stop_byte/write failed: %s\n",
            strerror(errno));
        return -1;
    }
}
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
    }
    return 0;
}

// USB keyboard 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]"
//   0x15 = letter r, meaning "as per key[5]"
//
// 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
tkxf_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
tkxf_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
tkxf_keyboard_ready(int fd) {
    unsigned char data_buffer;

    if(wire_packets == 4) {
        data_buffer = 0;
        if(read(fd, &data_buffer, 1) != 1) {
            fprintf(stderr,
                "TCXF Error: tkxf_send_keyboard_packet/read ack failed: %s\n",
                strerror(errno));
            return -1;
        }
    }
}
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
    if(data_buffer != 1) {
        fprintf(stderr,
            "TCXF Error: tkxf_send_keyboard_packet/read was non-ack: [%d]\n",
            data_buffer);
        return -2;
    }
    wire_packets = 0;
    return 1;
}

return 0;
}

int
tkxf_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; // ,
    tkxf_dedupe_key_buffer(keys, max_offset);
    for(idx = 0; idx <= max_offset; idx++) {
        keyboard_packet[idx + 1] = keys[idx];
        // printf("DEBUG [%x]\n", keys[idx]);
    }

    if(write(fd, keyboard_packet, max_offset + 2) < 0) {
        fprintf(stderr, "TCXF Error: tkxf_send_keyboard_packet/write failed: %s\n",
            strerror(errno));
        return -1;
    }
    wire_packets++;

    return wire_packets;
}

void
tkxf_close_port(int fd) {

    close(fd);

    return;
}

unsigned int
tkxf_octets_available(int fd) {
    int available_octets;

    // Win ioctlsocket(fd, FIONREAD, &available_octets)
    ioctl(fd, FIONREAD, &available_octets);

    if(available_octets < 0)
        return 0;

    return available_octets;
}

void *
tkxf_xmt(void * thread_param) {
    tkxf_xmt_parms * param;
    unsigned char data_buffer[1];
    unsigned char *data_buffer_p;
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
int data_buffer_size = 1;
unsigned int offset;
unsigned char keys[6];
unsigned int octets;
int flags;

param = (tkxf_xmt_parms *)thread_param;

data_buffer_p = data_buffer;
offset = 0;
while(!terminate) {
    // Data to be read on network socket?
    octets = tkxf_octets_available(param->socket_fd);
    if(octets) {
        // If so, pack and send
        flags = 0;
        if(recv(param->socket_fd, data_buffer_p, 1, flags) > 0) {
            offset = tkxf_pack_byte(keys, offset, data_buffer[0]);
            if(offset == 6) {
                // If full keyboard packet, send it.
                // printf("DEBUG data: sending [%d]\n", offset);
                tkxf_send_keyboard_packet(param->serial_fd, DATA_MODE, keys, offset - 1);
                // Blocking check for keyboard ready
                // printf("DEBUG data: wire_packets [%d]\n", wire_packets);
                tkxf_keyboard_ready(param->serial_fd);
                offset = 0;
            }
        }
    } else {
        // If not, check to see the keyboard is ready ..
        octets = tkxf_octets_available(param->serial_fd);
        if(octets) {
            tkxf_keyboard_ready(param->serial_fd);
            // Now if there is a partial packet available
            if(offset) {
                // Send incomplete keyboard packet
                // printf("DEBUG no data: partial [%d]\n", offset);
                tkxf_send_keyboard_packet(param->serial_fd, DATA_MODE, keys, offset - 1);
                // printf("DEBUG no data: wire_packets [%d]\n", wire_packets);
                tkxf_data_mode(param->serial_fd);
                offset = 0;
            }
        }
        usleep(100);
    }
}
if(offset) {
    // Send incomplet keyboard packet before exiting the program
    // printf("DEBUG left over bits [%d]\n", offset);
    tkxf_send_keyboard_packet(param->serial_fd, DATA_MODE, keys, offset - 1);
    tkxf_data_mode(param->serial_fd);
}

return;
}

// ----- TGXF RCV --

#define TGXF_DEBUG 0

// TGXf Session Parameters (TGXf State Machine)
unsigned int tgxf_session_last_data_counter = 15;
unsigned int tgxf_session_frame_count = 0;
unsigned int tgxf_session_errors = 0;
unsigned int tgxf_session_max_errors = 0;
unsigned int tgxf_session_debug_image_counter = 0;

// TGXf Session Parameters (TGXf Global Options)
unsigned int tgxf_session_qrcode_version = 0;
unsigned int tgxf_session_qrcode_fps = 0;
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
unsigned int tgxf_session_qrcode_bytes = 0;

static int
tgxf_decode_frame(const zbar_symbol_t *sym, const void *userdata) {
    tgxf_rcv_parms * param;
    struct itimerval tbuf;
    struct sigaction action;
    unsigned char *frame_buffer;
    unsigned char *frame_payload;
    char error_buffer[1024];
    int frame_size;
    int payload_size;
    int error_buffer_size = 1024;

    unsigned char control_byte;
    unsigned int control_bit;
    unsigned int control_type;
    unsigned int control_subtype;
    unsigned int data_counter;
    unsigned int expected_counter;
    unsigned int lost_frames;
    unsigned int transfer_bytes_remaining;
    unsigned int frames_remaining;
    unsigned int minutes_remaining;
    unsigned int seconds_remaining;
    unsigned int lost_frame_idx;
    int i;
    float progress_state;

    param = (tgxf_rcv_parms *)userdata;

    frame_size = zbar_symbol_get_data_length(sym);
    if(frame_size <= 0) {
        fprintf(stderr, "ERROR(TGXf): invalid frame size: %d\n", frame_size);
        return -1;
    }
    if((frame_buffer = (unsigned char *)malloc(frame_size)) == NULL) {
        fprintf(stderr, "ERROR(TGXf): unable to allocate frame buffer\n");
        return -1;
    }
    if(memcpy(frame_buffer, zbar_symbol_get_data(sym), frame_size)
        != frame_buffer) {
        fprintf(stderr, "ERROR(TGXf): failed to establish frame buffer\n");
        return -1;
    }
    }

    // Point to Payload
    frame_payload = frame_buffer;
    frame_payload++;
    payload_size = frame_size - 1;

    // Get Control Byte and Bit
    control_byte = *frame_buffer;
    control_bit = control_byte & 1;

    // DATA Frames with Payload Only
    if(control_bit)
        return 0;
    if(payload_size <= 0)
        return 0;

    // Duplicate Data Frame Detection - Ignore
    data_counter = (control_byte & 30) >> 1;
    if(data_counter == tgxf_session_last_data_counter) {
        if(TGXF_DEBUG)
            fprintf(stderr,
                "TCXF Warning: tgxf_decode_frame/duplicate frame, ignored\n");
        free(frame_buffer);
        return 0;
    }
}
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
// Missing Data Frame Detection - Accumulate Errors
expected_counter = tgif_session_last_data_counter + 1;
if(expected_counter == 16)
    expected_counter = 0;
if(data_counter != expected_counter) {
    lost_frames = data_counter - expected_counter;
    if(expected_counter > data_counter) {
        lost_frames = data_counter + 16 - expected_counter;
    }
    // Error Accumulation
    tgif_session_errors += lost_frames;
    if(tgif_session_max_errors) {
        // If there is a session limit, and it's exceeded ..
        if(tgif_session_errors >= tgif_session_max_errors) {
            terminate = 1;
            fprintf(stderr,
                "TCXF Error: tgif_decode_frame/max errors for session reached\n");
        }
    }
    // Error Reporting
    if(lost_frames == 1) {
        fprintf(stderr,
            "TCXF Error: tgif_decode_frame/lost 1 data frame\n");
    } else {
        fprintf(stderr,
            "TCXF Error: tgif_decode_frame/lost %d data frames\n",
            lost_frames);
    }
    // Correct the session frame count
    tgif_session_frame_count += lost_frames;
}

// Write Data to Socket
if(write(param->socket_fd, frame_payload, payload_size) < 0) {
    fprintf(stderr,
        "TCXF Error: tgif_decode_frame/socket write failed: %s\n",
        strerror(errno));
    return -1;
}

// Increment tally
tgif_session_last_data_counter = data_counter;
tgif_session_frame_count++;

free(frame_buffer);
return(1);
}

static void
tgif_barcode_handler(zbar_image_t *img, const void *userdata) {
    const zbar_symbol_t *barsym;
    zbar_symbol_type_t type;

    barsym = zbar_image_first_symbol(img);
    for(; barsym; barsym = zbar_symbol_next(barsym)) {
        if(zbar_symbol_get_count(barsym))
            continue;
        if((type = zbar_symbol_get_type(barsym)) == ZBAR_PARTIAL)
            continue;
        tgif_decode_frame(barsym, userdata);
    }
}

void *
tgif_rcv(void * thread_param) {
    tgif_rcv_parms * param;
    static zbar_processor_t *zbar_proc;
    unsigned int active;
    int keypress;
```


PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
param = (tgxf_rcv_parms *)thread_param;

// Initialise the Zbar library
if(!(zbar_proc = zbar_processor_create(1))) {
    fprintf(stderr, "TCXF Error: tgxf_rcv failed to setup zbar: %s\n",
            strerror(errno));
    terminate = 1;
    return;
}
zbar_processor_set_data_handler(zbar_proc,
    tgxf_barcode_handler, thread_param);

// Optionally increase verbosity
if(param->verbose) {
    for(; param->verbose != 0; param->verbose--)
        zbar_increase_verbosity();
}

// Process QR codes exclusively
zbar_processor_set_config(zbar_proc, 0, ZBAR_CFG_ENABLE, 0);
zbar_processor_set_config(zbar_proc, ZBAR_QRCODE, ZBAR_CFG_ENABLE, 1);

// Open video device
if(zbar_processor_init(zbar_proc, param->camera_device, param->display)) {
    // return zbar_processor_error_spew(zbar_proc, 0);
    zbar_processor_error_spew(zbar_proc, 0);
    terminate = 1;
    return;
}

// Optionally show window
if(param->display) {
    if(zbar_processor_set_visible(zbar_proc, 1)) {
        // return zbar_processor_error_spew(zbar_proc, 0);
        zbar_processor_error_spew(zbar_proc, 0);
        terminate = 1;
        return;
    }
}

// Start processing video
active = 1;
if(zbar_processor_set_active(zbar_proc, active)) {
    // return zbar_processor_error_spew(zbar_proc, 0);
    zbar_processor_error_spew(zbar_proc, 0);
    terminate = 1;
    return;
}

// Wait for exit
while(!terminate)
    usleep(250);

// Shutdown the Zbar library
zbar_processor_destroy(zbar_proc);
terminate = 1;

return;
}

// ----- MAIN --

void
logo(int tagline) {
    char midnightcode_logo_ascii[26][22] = {
        "      _JNJ`", "      ", "      ",
        "      .JNMH`", "      ", "      ",
        "      JMMF`", "      ", "      `;. ",
        "      .NMM)", "      ", "      `MN. ",
    };
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
    "      MMM)      " , "      (MML      " ,
    "      (MMM^    " , "      MMLL      " ,
    " M I D N I G H" , " T      C o D E " ,
    "      (NMMF    " , "      MHNH      " ,
    "      NMML     " , "      .MMM      " ,
    "      NMML     " , "      .NMH      " ,
    "      4MMNL    " , "      .#F      " ,
    "      `4HNNL_  " , "      `.#F      " ,
    "      `4HNNL_  " , "      `.#F      " ,
    "      `4HNNL_  " , "      `.#F      " ,
};
int idx;

fprintf(stderr, "\n");
for(idx=0; idx<26; idx++) {
    if((float)((float)idx/(float)2) == (int)(idx/2)) {
        fprintf(stderr, "          \033[01;31m");
    } else {
        fprintf(stderr, "\033[31m");
    }
    if(idx == 12 || idx == 13)
        fprintf(stderr, "\033[01;37m");
    fprintf(stderr, "%s\033[0m", midnightcode_logo_ascii[idx]);
    if((float)((float)idx/(float)2) != (int)(idx/2))
        fprintf(stderr, "\n");
}
fprintf(stderr, "\n");
if(tagline)
    fprintf(stderr,
        "          ThruConsoleXfer - Personal Computer End - Reference Code"
        "\n\n");

return;
}

void
usage(int help) {

    logo(1);
    if(help) {
        fprintf(stderr,
            " Usage: tcxf-pce [OPTIONS]...\n"
            " -t PORT, --tcp=PORT      TCP port to listen on (8442)\n"
            " -s DEVICE, --serial=DEVICE Serial device for TKXF (/dev/ttyACM0)\n"
            " -c DEVICE, --camera=DEVICE Video device for TGXF (/dev/video0)\n"
            " -n, --nodisplay         Do not display video window (0)\n"
            " -v, --verbose           Display more decoder information (0)\n"
            "\n"
        );
    }
}

void
signal_handler(int sig) {

    terminate = 1;
    if(listener_socket_fd)
        shutdown(listener_socket_fd, SHUT_RDWR);

    return;
}

int
main(int argc, char **argv) {
    static const struct option options[] = {
        {"help", no_argument, NULL, 'h'},
        {"tcp", required_argument, NULL, 't'},
        {"serial", required_argument, NULL, 's'},
        {"camera", required_argument, NULL, 'c'},
        {"nodisplay", no_argument, NULL, 'n'},
    };
}
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
    {"verbose", no_argument, NULL, 'v'},
    {NULL, 0, NULL, 0}
};
static char *optstring = "ht:s:c:nv";
struct sigaction sa;
pthread_t pid_r = { 0 };
pthread_t pid_w = { 0 };
tkxf_xmt_parms tkxf_xmt_param;
tgxf_rcv_parms tgxf_rcv_param;
char default_serial_device[32] = "/dev/ttyACM0";
char default_camera_device[32] = "/dev/video0";
int opt;
int val;
unsigned int tcp_port;
const char *serial_device;
const char *camera_device;
unsigned int display;
unsigned int verbose;
struct sockaddr_in listener_addr;
struct sockaddr_in client_addr;
socklen_t client_len;
int socket_fd;
int serial_fd;
char nulbuf;

// Establish defaults
verbose = 0;
display = 1;
tcp_port = 8442; // T*
serial_device = "";
camera_device = "";

// Parse command line
while((opt = getopt_long(argc, argv, optstring, options, NULL)) != -1) {
    switch(opt) {
        case 'h':
            usage(1);
            exit(0);
            break;
        case 'n':
            display = 0;
            break;
        case 'v':
            verbose++;
            break;
        case 't':
            tcp_port = atoi(optarg);
            break;
        case 's':
            serial_device = optarg;
            break;
        case 'c':
            camera_device = optarg;
            break;
        default:
            fprintf(stderr, "Try `%s --help' for more information.\n", argv[0]);
            exit(-1);
            break;
    }
}

// Purdee
logo(1);

// Validate user input or Default it
if(strlen(serial_device) == 0)
    serial_device = default_serial_device;
if(strlen(camera_device) == 0)
    camera_device = default_camera_device;
fprintf(stderr, "TCXF: Using serial device [%s]\n", serial_device);
fprintf(stderr, "TCXF: Using camera device [%s]\n", camera_device);
fprintf(stderr, "\n");
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
// Install signal handler
memset(&sa, 0, sizeof(sa));
sa.sa_handler = &signal_handler;
if(sigaction(SIGPIPE, &sa, NULL) == -1) {
    fprintf(stderr,
        "TCXF Error: sighandler setup failed: %s\n", strerror(errno));
    return -1;
}
if(sigaction(SIGINT, &sa, NULL) == -1) {
    fprintf(stderr,
        "TCXF Error: sighandler setup failed: %s\n", strerror(errno));
    return -1;
}

// Initialize hardware
serial_fd = tkxf_setup_port(serial_device);
if(serial_fd < 0) {
    fprintf(stderr, "TCXF Error: Serial setup failed\n");
    return -1;
}
if(access(camera_device, F_OK) != 0) {
    fprintf(stderr,
        "TCXF Error: failed to access camera device: %s\n",
        strerror(errno));
    return -1;
}

// Setup networking
listener_socket_fd = socket(AF_INET, SOCK_STREAM, 0);
if(listener_socket_fd < 0) {
    fprintf(stderr,
        "TCXF Error: socket failed: %s\n", strerror(errno));
    return -1;
}

// Configure listener socket
bzero((char *)&listener_addr, sizeof(listener_addr));
listener_addr.sin_family = AF_INET;
listener_addr.sin_addr.s_addr = INADDR_ANY;
listener_addr.sin_port = htons(tcp_port);
if(bind(listener_socket_fd, (struct sockaddr *)&listener_addr,
    sizeof(listener_addr)) < 0) {
    fprintf(stderr,
        "TCXF Error: bind failed: %s\n", strerror(errno));
    return -1;
}

// Await one connection
listen(listener_socket_fd, 1);

// Accpet the client
client_len = sizeof(client_addr);
socket_fd = accept(listener_socket_fd,
    (struct sockaddr *)&client_addr, &client_len);
if(socket_fd < 0) {
    fprintf(stderr,
        "TCXF Error: connection failed: %s\n", strerror(errno));
    return -1;
}
close(listener_socket_fd);
listener_socket_fd = 0;

// Configure threads
tkxf_xmt_param.socket_fd = socket_fd;
tkxf_xmt_param.serial_fd = serial_fd;
tgxf_rcv_param.socket_fd = socket_fd;
tgxf_rcv_param.camera_device = camera_device;
tgxf_rcv_param.display = display;
tgxf_rcv_param.verbose = verbose;

// Launch threads
```

PUBLIC
ThruConsoleXfer (TCXf) White Paper

```
pthread_create(&pid_r, NULL, &tkxf_xmt, (void *)&tkxf_xmt_param);
pthread_create(&pid_w, NULL, &tgxf_rcv, (void *)&tgxf_rcv_param);

// Wait around for exit
while(!terminate) {
    if(recv(socket_fd, &nulbuf, sizeof(nulbuf), MSG_PEEK|MSG_DONTWAIT) == 0) {
        fprintf(stderr, "TCXF: client connection closed\n");
        terminate = 1;
    }
    usleep(250);
}

// Shutdown and bail
pthread_join(pid_r, NULL);
pthread_join(pid_w, NULL);
tkxf_close_port(serial_fd);
close(socket_fd);

return 0;
}
```

3.9.5 Example Build Script

The following build script has been supplied to ease the burden of compiling the reference implementations (particularly tcxf-dce) in their various configurations.

```
#!/bin/bash

if [ "${1}x" == "px" -o "${#}x" == "0x" ]
then
    echo
    echo "----- 8< -----"
    echo "#"
    echo "# PCE"
    echo "#"
    gcc tcxf-pce.c -lpthread -lzbar -o tcxf-pce
    echo "Done."
    echo "----- >8 -----"
    echo
fi

if [ "${1}x" == "dx" -o "${#}x" == "0x" ]
then
    echo
    echo "----- 8< -----"
    echo "#"
    echo "# DCE"
    if [ "${2}x" == "ansix" -o "${3}x" == "ansix" ]
    then
        echo "# +ANSIONLY"
        opt="-DANSIONLY"
    else
        echo "# +GTK2"
        opt="$(pkg-config --cflags --libs gtk+-2.0)"
    fi
    if [ "${2}x" == "staticx" -o "${3}x" == "staticx" ]
    then
        echo "# +STATIC"
        echo "#"
        gcc tcxf-dce.c ${opt} -lpthread \
            -include "$(pwd)/qrencode-3.4.3/qrencode.h" \
            qrencode-3.4.3/.libs/libqrencode.a \
            -o tcxf-dce
    else
        echo "#"
        gcc tcxf-dce.c ${opt} -lpthread -lqrencode \
            -o tcxf-dce
    fi
    if [ -x "tcxf-dce" ]
    then
        strip tcxf-dce
    else
        echo "Not stripped."
    fi
    echo "Done."
    echo "----- >8 -----"
    echo
fi
```

3.10 End-to-End Build and Test Process for TCXf

This section includes all of the information that should be required to build and test the TCXf software. It starts with a known base – a test bench established from Ubuntu Server builds – and then works through the software builds and then the test case.

3.10.1 Establish the Test Bench

The test bench that I established was based on what was available off-the-shelf at the time of writing.

For completeness, my test host is a 64bit Linux Mint machine with an Intel CPU (8 cores) and 32GB of RAM. You shouldn't need to mirror this configuration – for example, I don't believe that you would need more than 4GB of RAM for a Linux test host.

The host platform should be viable as any x86 based 64bit platform. The guest platforms in the test bench should be 64bit Linux. This proof of concept has been tested on 64bit Linux platforms but should work on others, such as 32bit Linux platforms, without change; and still others, such as 64bit Windows platforms, with some changes to code, build processes, etc.

3.10.1.1 Acquire the Software

Acquire VMware Player 6 and Ubuntu Server 10.04 to create the test bench:

- VMware Player 6 (64bit)²²
 - I used 6.0.1 build-1379776 but there was an issue with the contents of the shared folders not updating correctly on change, so perhaps running with a later version will be better.
- Ubuntu 10.04.4 Server (64bit)²³
 - This version was chosen for long term support.

Install VMware Player on your test host and ensure that it is working correctly.

3.10.1.2 Deploy the Guest Infrastructure

The next step is to configure and deploy two guest machines/instances – one PCE and one DCE.

The following virtual machine specification was used for each:

- Hardware;
 - 8GB HDD, 1GB RAM, 1 CPU
 - Remove "Sound Card", "Printer", "Floppy"
 - On "USB Controller", set;
 - "USB Compatibility" to "USB 2.0"
 - Disable "Automatically connect new USB devices"

²² https://my.vmware.com/web/vmware/free#desktop_end_user_computing/vmware_player/6_0

²³ <http://releases.ubuntu.com/lucid/ubuntu-10.04.4-server-amd64.iso>

- Enable "Show all USB input devices"
- Disable "Share Bluetooth devices with the virtual machine"
- Software
 - Ubuntu 10.04.4 Server (LTS) 64bit
 - Make sure the virtual CD/DVD drive is referring to the ISO image that you've downloaded.

Build PCE and DCE instances according to the above specification. Once they're built (including the OS install), reboot each and then:

- Map "testbench" shared folder from both DCE and PCE to desktop folder, and;
- Login to both machines and become the super-user;

```
sudo bash
```

Note that for some reason VMware Player only has the ability to run one VM guest from its user interface (I could have sworn it used to open multiple guests in tabs in an older version). To run two simultaneous VM guests simply run the "vmplayer" process twice (either from two terminal windows, or by clicking on the launch menu icon twice, and then run once guest in each).

3.10.2 Building tcxf-pce

The following instructions will guide you through building the "tcxf-pce" binary from the content provided in this document and the base Linux build (above)

3.10.2.1 PCE – Acquire Development Packages

On the PCE console install GCC and friends (adds approximately 35MB of packages);

```
apt-get install gcc
```

3.10.2.2 PCE – Build and Install Zbar

On the PCE console install the Zbar pre-requisites (adds approximately 530MB of packages);

```
apt-get install mercurial autoconf  
apt-get install libjpeg62-dev libmagickwand-dev python-gtk2-dev libqt4-core qt4-dev-tools
```

1. Acquire the current Zbar source (this build does not work on the zbar-0.10 tarball – the latest available at the time of writing);

```
hg clone http://zbar.hg.sourceforge.net:8000/hgroot/zbar
```


2. Build Zbar;

```
cd zbar
autoreconf --install
./configure
make
```

This is a checkpoint: If the Zbar code does not compile then do not proceed. Check your development environment for missing dependencies and mitigate any new issues that may be in the current Zbar development code-base.

Note that the above worked in April 2014, but in September 2014 the Zbar build environment was broken, you will need to remove the “-Werror” entries²⁴.

3.10.2.2.1 The Zbar TGXf Patch

The following C code is in “patch” format. A change has been made to the Zbar decoder library in order to prevent the 8 bit data stream from being interpreted as UTF-8 language data.

```
--- zbar/zbar/qrcode/qrdectxt.c.orig 2014-02-02 18:05:18.811784715 +1100
+++ zbar/zbar/qrcode/qrdectxt.c      2014-02-02 18:05:50.143783857 +1100
@@ -254,7 +254,14 @@
     Does such a thing occur?
     Is it allowed?
     It requires copying buffers around to handle correctly.*/
-   case QR_MODE_BYTE:
+   case QR_MODE_BYTE:{
+       if(sa_ctext-sa_ntext>=(size_t)entry->payload.data.len){
+           memcpy(sa_text+sa_ntext,entry->payload.data.buf,
+                 entry->payload.data.len*sizeof(*sa_text));
+           sa_ntext+=entry->payload.data.len;
+       }
+       else err=1;
+   }break;
+   case QR_MODE_KANJI:{
+       in=(char *)entry->payload.data.buf;
+       inleft=entry->payload.data.len;
```

3.10.2.2.2 Applying the *Binary Clear* patch to Zbar

Assuming the Zbar patch is called “zbar-binaryclear.patch” and resides in your home directory, and that you are in the “zbar” directory per step 3 above, then the following command should apply the patch to the Zbar library;

```
patch -p1 < ~/zbar-binaryclear.patch
```

If the patch applies successfully, then re-compile and install Zbar;

```
make
make install
```

24 <http://sourceforge.net/p/zbar/discussion/664596/thread/f4b52988/>

You may also wish to update the share library cache;

```
ldconfig
```

3.10.2.3 PCE – Building tcxf-pce

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

```
gcc tcxf-pce.c -lpthread -lzbar -o tcxf-pce
```

3.10.3 Building tcxf-dce

The following instructions will guide you through building the “tcxf-dce” binary from the content provided in this document and the base Linux build (above). Obviously if you are ultimately going to upload the tcxf-dce to a foreign platform then you should build it for that platform, rather than this test environment.

3.10.3.1 DCE – Acquire Development Packages

On the DCE console install GCC and friends (adds approximately 35MB of packages);

```
apt-get install gcc
```

3.10.3.2 DCE – Building a Static libqrencode

On the DCE console install the Zbar pre-requisites (adds approximately 530MB of packages);

```
apt-get install pkg-config libpng-dev
```

1. Acquire the current libqrencode source (this build was tested successfully against qrencode-3.4.3.tar.gz – the latest available at the time of writing);

```
wget http://fukuchi.org/works/qrencode/qrencode-3.4.3.tar.gz
```

2. Build libqrencode;

```
tar xvfz qrencode-3.4.3.tar.gz
cd qrencode-3.4.3
./configure --enable-static --disable-shared
make
cd ..
```

This is a checkpoint: If the libqrencode code does not compile then do not proceed. Check your development environment for missing dependencies and mitigate any new issues that may be in the current libqrencode code-base.

3.10.3.3 DCE – Building a txcf-dce

There are two ways to build the txcf-dce software, the first being the feature rich (graphical) version that will yield the best performance and user experience, versus the small, portable ANSI ONLY version which has been created to close out the proof of concept.

Only one of the next two processes need be followed to build the txcf-dce binary.

3.10.3.3.1 Building a Complete (Graphical) txcf-dce

On the DCE console, assuming the above source code is saved as “tcxf-dce.c” and resides in current working directory, then the following command should compile the source;

```
gcc txcf-dce.c `pkg-config --cflags --libs gtk+-2.0` -lpthread \  
-include "`pwd`/qrencode-3.4.3/qrencode.h" qrencode-3.4.3/.libs/libqrencode.a -o txcf-dce
```

The resultant binary file “tcxf-dce” should be ready for use.

3.10.3.3.2 Building a Small, Portable, ANSI ONLY txcf-dce

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

```
gcc txcf-dce.c -DANSIONLY -lpthread -include "`pwd`/qrencode-3.4.3/qrencode.h" \  
qrencode-3.4.3/.libs/libqrencode.a -o txcf-dce
```

The resultant binary should be approximately 165KB. That file can be stripped;

```
strip txcf-dce
```

The resultant binary should now be approximately 52KB. That file can be compressed;

```
gzip -9 txcf-dce
```

The resultant binary file (now called “tcxf-dce.gz”) should be small enough (mine came out to 24,304 bytes) to upload via the Arduino Leonardo Keyboard Uploader – see *3.4 Uploading the Transmit Software (Reusing the First Principle is the Key)*, assuming that you have used the Freetronics Leostick, or another Leonardo with sufficient flash memory.

3.10.4 Building the Keyboard Stuffer

Follow the instructions in section 3.7.3 *Hardware Reference Implementation – Keyboard Stuffer Device – Transmit* to assemble the hardware and install the software for the Keyboard Stuffer.

3.10.5 Preparing for Testing

The following steps will ensure that the test bench is configured correctly for testing.

3.10.5.1 PCE – Making the Test Platform Graphical

On the PCE console install the X environment (adds approximately 165MB of packages);

```
apt-get install xinit lxde
```

Reboot the PCE machine and login to the X environment as your user, and start a terminal.

3.10.5.2 DCE – Making the Test Platform a Server

On the DCE console install the SSH daemon (adds approximately 1MB of packages);

```
apt-get install openssh-server
```

Make sure the setting for “GSSAPIAuthentication” is set to “no” in /etc/ssh/sshd_config.

Reboot the DCE machine and log in again.

3.10.5.3 PCE – Making the Test Platform a Client

On the PCE machine, make sure the setting for “GSSAPIAuthentication” is set to “no” in /etc/ssh/ssh_config.

3.10.5.4 DCE – Attaching the USB Keyboard

Physically connect the Keyboard Stuffer USB device (keyboard side) to the test host. Then, in VMware Player, virtually connect “Arduino Leo” to the DCE guest.

3.10.5.5 PCE – Attaching the USB Serial

Physically connect the Keyboard Stuffer USB device (serial side) to the test host. Then, in VMware Player, virtually connect “www.freertronics.com Modem” to the PCE guest.

3.10.5.6 PCE – Attaching the USB Camera

Physically connect the USB camera device to the test host.

In the PCE guest, ensure that the USB Audio module is not loaded when the camera is attached per the advisory²⁵.

```
echo "blacklist snd-usb-audio" >> /etc/modprobe.d/blacklist.conf
```

Then, in VMware Player, virtually connect “Microsoft LifeCam HD-3000” to the PCE guest. If you have problems with the device failing to work, for example, with "ERROR: zbar processor in zbar_processor_init(): unsupported request: no compatible image format" errors, then try;

```
sudo rmmod uvcvideo  
sudo modprobe uvcvideo
```

And then unplug the camera, wait three seconds and plug it in per the advisory²⁶.

One solution that worked for me in Linux Mint LiveCD environment was pre-loading a Video4Linux2 Conversion library²⁷;

```
LD_PRELOAD=/usr/lib/x86_64-linux-gnu/libv4l/v4l2convert.so ./tcxf-pce -c /dev/video1
```

²⁵ <https://bugs.launchpad.net/ubuntu/+source/guvcview/+bug/1033146>

²⁶ <https://bugs.launchpad.net/ubuntu/+source/linux/+bug/757074>

²⁷ <http://sourceforge.net/p/zbar/support-requests/48/>

3.10.6 Running the Tests – Reset Point

Please ensure that each step is successful before continuing on to the subsequent step.

3.10.6.1 Install the Hardware

Be sure that you've followed the instructions from the previous section and have the hardware correctly attached. Confirm that the devices have been made available within the guest operating system;

On the PCE you should have;

- USB Camera (Microsoft LifeCam HD-3000) /dev/video0
- USB Serial Adapter /dev/ttyACM0

On the DCE you should have;

- USB Keyboard Adapter

Note that if you chose different hardware your device entries may differ from those depicted here. Also note that you must wait until after the Keyboard Stuffer has beeped three times (about 10 seconds) before attempting to transfer any data. If data is not transferred through the Keyboard Stuffer, then unplug and replug the Keyboard Stuffer and try again.

3.10.6.2 Run the Software

Note that you will require four terminal sessions in this test – two terminal windows on PCE and two consoles on the DCE. To switch consoles in text mode Linux, you press CTRL+ALT+F1 for console one and CTRL+ALT+F2 for console two.

1. Start the DCE TCXf

On the DCE console, console one, start the tcxf-dce program;

```
./tcxf-dce -f 5 -v 1
```

Note that the camera must be pointed at this terminal window. Note too that keyboard keystrokes will go to this terminal. Consider also that this test is for stability not performance (5 FPS @ Version 1 = 50 bytes per second, or 400bps).

2. Start the PCE TCXf

In the first PCE terminal window, start the tcxf-pce program;

```
./tcxf-pce
```

At this point everything is ready to go. All you need now is a test client.

3.10.7 Test Case #1 – PCE to DCE Connectivity

This test confirms successful PCE to DCE data connectivity.

Before you begin this section, please follow section 3.10.6 *Running the Tests – Reset Point* if you haven't done so already.

3. Start the DCE client

On the DCE console, switch to console two, and start the netcat session:

```
sleep 5 ; netcat localhost 8442
```

Switch back to console one, and in a couple of seconds you should see the screen turn white when the netcat client connects. It is important that you keep console one (the one the with tcxf-dce program running in it) active, in order for it to visible show QR codes to the screen and for it to receive the keyboard keystrokes.

4. Start the PCE client

In the second PCE terminal window, start the netcat session;

```
sleep 5 ; netcat localhost 8442
```

If you didn't include the “-n” on the command line at step 2, then a video window should appear on the PCE in a couple of seconds. You can use this window to aim the camera at the DCE display. When data is sent and the DCE is showing a QR code, you will see a bounding box in this video window when the PCE software recognises the QR code. Note that you won't see any QR codes in this test.

5. Enter the test PCE->DCE data

In the second PCE terminal window (the one with the netcat) type the following and press enter;

```
Hello World!
```

You should see lights flash on the Keyboard Stuffer.

6. Confirm the result at the DCE

On the second DCE console (the one with the netcat) you should see the text “Hello World!”.

7. Reset

On all four terminals (two on the DCE and two on the PCE), break out of the running programs (CTRL+C).

3.10.8 Test Case #2 – DCE to PCE Connectivity

This test confirms successful DCE to PCE data connectivity.

Before you begin this section, please follow section 3.10.6 *Running the Tests – Reset Point* if you haven't done so already.

3. Start the PCE client

In the second PCE terminal window, start the netcat session;

```
netcat localhost 8442
```

If you didn't include the “-n” on the command line at step 2, then a video window should appear on the PCE. You can use this window to aim the camera at the DCE display. When data is sent and the DCE is showing a QR code, you will see a bounding box in this video window when the PCE software recognises the QR code.

4. Start the DCE client

On the DCE console, switch to console two, and start the netcat session:

```
sleep 10 ; echo "Hello World!" | netcat localhost 8442
```

Switch back to console one, and in a couple of seconds you should see the screen turn white when the netcat client connects. It is important that you keep console one (the one the with txcf-dce program running in it) active, in order for it to visible show QR codes to the screen and for it to receive the keyboard keystrokes.

You should see one or two QR codes flash past on the DCE console.

5. Confirm the result at the PCE

On the second PCE console (the one with the netcat) you should see the text “Hello World!”.

6. Reset

On all four terminals (two on the DCE and two on the PCE), break out of the running programs (CTRL+C).

3.10.9 Test Case #3 – PCE to DCE File Transfer

This test confirms successful PCE to DCE volume transfers.

Before you begin this section, please follow section 3.10.6 *Running the Tests – Reset Point* if you haven't done so already.

3. Start the DCE client

On the DCE console, switch to console two, and start the netcat session:

```
netcat localhost 8442 > /tmp/download.bin
```

Switch back to console one, where the screen should be white from when the netcat client connected. It is important that you keep console one (the one with txcf-dce program running in it) active, in order for it to visible show QR codes to the screen and for it to receive the keyboard keystrokes.

4. Start the PCE client

In the second PCE terminal window, start the netcat session;

```
cat ./tcxf-pce | netcat localhost 8442
```

If you didn't include the “-n” on the command line at step 2, then a video window should appear on the PCE.

5. Observe the transfer

The netcat session in the second terminal window of the PCE should exit once the transfer has been completed, though buffering may see the Keyboard Stuffer active (lights flashing) for longer.

6. Confirm the result at the DCE

In the second PCE terminal window check the md5 value for the “tcxf-pce” file;

```
md5sum ./tcxf-pce
```

On the second DCE console, check the md5 value for the “download.bin” file;

```
md5sum /tmp/download.bin
```

The two md5 values should be the same.

7. Reset

On all four terminals (two on the DCE and two on the PCE), break out of the running programs (CTRL+C).

3.10.10 Test Case #4 – DCE to PCE File Transfer

This test confirms successful DCE to PCE volume transfers.

Before you begin this section, please follow section 3.10.6 *Running the Tests – Reset Point* if you haven't done so already.

3. Start the PCE client

In the second PCE terminal window, start the netcat session;

```
netcat localhost 8442 > /tmp/download.bin
```

If you didn't include the “-n” on the command line at step 2, then a video window should appear on the PCE.

4. Start the DCE client

On the DCE console, switch to console two, and start the netcat session:

```
sleep 30 ; cat ./tcxf-dce | netcat localhost 8442
```

Switch back to console one, and in a couple of seconds you should see the screen turn white when the netcat client connects. It is important that you keep console one (the one the with tcxf-dce program running in it) active, in order for it to visible show QR codes to the screen and for it to receive the keyboard keystrokes.

5. Observe the transfer

You will see QR codes flashing “past” on the DCE console for the duration of the transfer. Once the transfer has been completed, though you won't be able to see it, the netcat session on the second console of the DCE should exit (though buffering may see the QR codes run longer).

6. Confirm the result at the PCE

On the second DCE console check the md5 value for the “tcxf-dce” file;

```
md5sum ./tcxf-dce
```

On the second PCE terminal window, check the md5 value for the “download.bin” file;

```
md5sum /tmp/download.bin
```

The two md5 values should be the same.

6. Reset

On all four terminals (two on the DCE and two on the PCE), break out of the running programs (CTRL+C).

3.11 Milestone Achievement with TCXf

As demonstrated through the test cases above, the TCXf implementation has successfully created the asymmetric digital communications link (maximum of 32kbps down, 12kbps up, from the consumer's perspective), capable of relaying full-duplex (bi-directional) data, that we thought it would.



Using the TCXf reference code described above you now have unfettered access to bi-directional (ingress and egress) binary data flows to/from the target computer, presented as a TCP socket at each end.

In an Enterprise Security Architecture context, this has significant implications.

3.11.1 But I could send files before; Evolution to IP Networking

In production use, none of the above steps required privilege escalation at the DCE. If we were to allow the DCE to create a new network interface then we could establish an IP network over the TCXf link. This is actually very easy to do.

Before you begin this section, please follow section *3.10.6 Running the Tests – Reset Point* if you haven't done so already.

3. Start the DCE client

On the DCE console, switch to console two, and start the PPP session:

```
sudo pppd 10.1.1.1:10.1.1.2 debug noccp nodetach pty "netcat localhost 8442"
```

Switch back to console one, where the screen should be white from when the netcat client connected. It is important that you keep console one (the one the with txcf-dce program running in it) active, in order for it to visible show QR codes to the screen and for it to receive the keyboard keystrokes. QR codes should be appearing here as the PPP daemon attempts to negotiate with its peer.

4. Start the PCE client

In the second PCE terminal window, start the PPP session;

```
sleep 2 ; sudo pppd noipdefault debug nodetach noccp pty "netcat localhost 8442"
```

If you didn't include the “-n” on the command line at step 2, then a video window should appear on the PCE.

5. Observe the PPP negotiation

You should see;

- The Keyboard Stuffer will be active (lights flashing), and;
- The DCE console will be active (QR codes passing), and;
- There will be debug output from the PPP daemon in the second terminal window on the PCE.

At some point the PPP daemon will announce that it has successfully established a session.

6. Confirm the result at the PCE

In a third PCE terminal window you should now be able to (very slowly, 400 bits per second slowly) ssh to the DCE. Assuming your DCE username is “bob”;

```
ssh bob@10.1.1.2
```

Be patient. Remember there are options available to go at almost 100 times this rate built into the implementation already, this is just a proof of concept at this stage.

7. Reset

When you are done you can exit out of your ssh session normally. On the other four terminals (two on the DCE and two on the PCE), break out of the running programs (CTRL+C).

3.12 Enterprise Implications of this Technology

At first glance there's no risk to the enterprise from what has been described in the above sections. The models proposed involve two PCs where the user who owns both has obtained nothing more than he started with. But it's not as clear-cut as this in the enterprise, where this first stems from the way technology is abstracted and then the way that security controls are applied to the result.

3.12.1 The Enterprise Context

Please consult the following key when consulting the diagram below:















Icon	Represents
	The target, the thing you value the most
	The user, the reason why you put holes in otherwise perfectly smooth walls
	A whole network of a given classification
	A connection or flow of network data
	A service or application hosting platform (Host / PC / Server Infrastructure)
	An authentication validation function
	An authentication token
	A gateway platform (Dedicated Security Infrastructure)
	A layer 3-7 (OSI) packet filtering function (Firewall)
	An encrypted connection or flow of network data
	A harmful code detection/containment function (Anti-Virus)
	An intrusion detection/prevention function (IDS / IPS)
	An unauthorised change detection function
	A data leakage detection / prevention function

Table 8: Enterprise Security Architecture Components (in approximate order of genesis / maturity)

PUBLIC
ThruConsoleXfer (TCXf) White Paper

TCXf Role	PCE													DCE		
Application		← Terminal + VPN Clients											Receiver/Viewer/RDP/SSH	Receiver/Viewer/RDP/SSH		
Transport	TKXf TGXf														TKXf TGXf	
Security Controls																
			End-point Security			Network Security	End-point +Network	Network Security			Server Security	Network Security	Server Security	Server Security		
Infrastructure																
Control (Owner)																

Illustration 5: TCXf in the Enterprise Security Architecture (Application and Technology) context

3.12.2 Binary Data over Screen and Keyboard, Where's the Danger?

Let's pretend the crown jewels (or beer, please see *Illustration 5: TCXf in the Enterprise Security Architecture (Application and Technology) context*) are accessible from your application servers, buried deep within your organisation - something full of (or having direct access to) customer or personally identifiable information (PII), and to make it easy for you to protect, let's include some structured data - like social security numbers.

Opportunity: Now let's assume the role of an offshore application or platform support resource. I have access to either the jewels themselves (via access to DB credentials from the application for example) or other means (I can take a copy of /dev/mem or the swap device from the application server as the platform admin), but I don't have a safe way to get out of your secure environment with the goods.

Motive: I have nothing but contempt for your security and if I can export your data I will sell it to your competitors and give it to my government.

Means: You've given me a screen and keyboard so I'm going to export your data (in this example, using TCXf).

You've worked hard, so hard in fact that my daily journey into your bullion depository would keep Auric Goldfinger²⁸ himself at bay;

1. Start from the PC / SOE

I boot-up, and login to my corporate desktop where the AV/Malware scanner runs and finds nothing. Everything that comes through this computer is scanned as part of the Data Loss Prevention (DLP) program (which I know has an algorithm for SSNs, it says so on the vendor's site). It quarantine's email attachments that it can't open, it won't let me print or save files locally. I'm not allowed (or able) to connect external devices - Bluetooth, USB HDD, Wireless - they all fail and raise alarms. There's a kiosk version of this desktop SOE/COE that even deletes itself when I log out. Despite the odds already being against me, I start my VPN client (SSL/IPSEC) and pass through my local firewall as I head toward the company ..

2. In to the Perimeter

I VPN in to the company (passing the external IPS/IDS, the outer Firewall cluster and inner IPS/IDS) and authenticate with my 2FA token. With success comes the End-point Security agent which hurls itself at my computer like a juicy bug aiming for my highway-speed windscreen: It scans my desktop in a frenzy and finds more nothing. I open up my remote Desktop client (ICA/RDP/SSH/VNC/VMwareView) to log on to the enterprise LAN ..

3. In to that big flat LAN

I pass another firewall, and if I'm lucky more IPS/IDS infrastructure and potentially into a new authentication realm so that I can login to my hosted applications or my Virtual

28 <http://www.imdb.com/character/ch0000376/>

Desktop (VDI - be that Citrix, VMware or a shell on a Unix gateway, etc) or even my office PC (work station). This platform may or may not have varying degrees of AV/Malware tools depending on how strictly that desktop policy is enforced and whether or not it is a full Desktop platform (there's nothing to detect anyway). Now I can open a connection (SSH/RDP) to the Jump Box (not depicted) ..

4. Enter the Data Centre

If I'm really lucky I will pass through another firewall cluster and I may traverse another IPS/IDS instance, and possibly even enter a third authentication realm. Here I land on a host that has that no apparent purpose other than to frustrate my day-to-day productivity, as I'm either automatically or manually thrown over the next firewall to my final destination ..

5. On to the Target

Passing another firewall (this one might even authenticate me) I finally get to logon to the Corporate Application Server. This device I know, and I can affect ..

All of these devices would of course be hardened, centrally time sourced, centrally logged for security incident and event management, under change management, constantly checked for unauthorised change, etc ad-nausea. So what can I possibly do to you?

3.12.3 Abstraction of the Screen and Keyboard in Enterprise Architectures

The problem begins with the abstraction of the screen and keyboard in enterprise architectures. Let's look at this in terms of the TCXf software architecture:

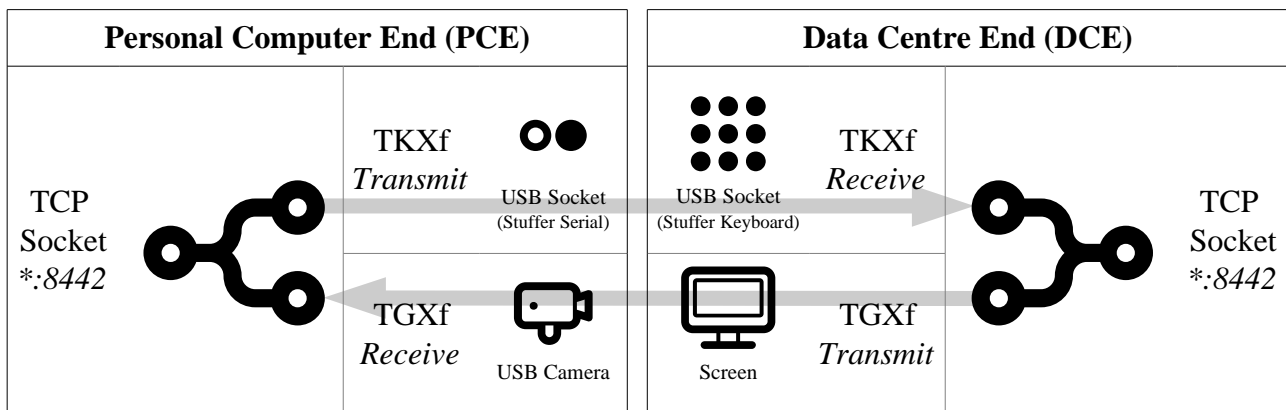


Table 9: Abstraction – Physical Screen/Keyboard versus Logical Console



The abstraction doesn't occur precisely where we instinctively think it does. The DCE screen and keyboard are physically with me, off-shore. Here, between the physical screen/keyboard and the logical operating system “console”, is where the abstraction occurs. The physical interface is technically or substantially outside of your control, while your entire service delivery capability is working to make sure that my console data his low latency, high through-put and very much uninterrupted.

3.12.4 Executing the Attack

So, I'm now a malicious user on the target system. Let's finish the job:

6. Installing the DCE software

I upload the TCXf DCE software by plugging my home-brew digital USB keyboard and “typing in” the program in the local enterprise-controlled PC, per section 3.4 *Uploading the Transmit Software (Reusing the First Principle is the Key)*. My keyboard keystrokes have been abstracted by the VPN software and subsequent terminal applications such that I'm “typing” directly into the target system.

This takes a minute or two to upload (/tmp/phpD104.tmp), and another minute to unpack (as /tmp/phpD382.tmp) and run. The following is what you'll find in my shell history:

```
echo > /tmp/phpD104.tmp
chmod 700 /tmp/phpD104.tmp
cat - > /tmp/phpD104.tmp
/tmp/phpD104.tmp > /tmp/phpD382.tmp
echo > /tmp/phpD104.tmp
chmod 700 /tmp/phpD382.tmp
/tmp/phpD382.tmp -f 5 -v 1 ; echo > /tmp/phpD382.tmp ; chmod 777 /tmp/phpD*.tmp ; exit
```

7. Running the PCE software

I unplug the home-brew digital USB keyboard and re-flash it on my personal laptop (one not supplied by your organisation). As a reconfigured Keyboard Stuffer, I connect the keyboard end to the the local enterprise-controlled PC again, and the serial end into my personal laptop. I've got a high-definition webcam connected to my personal laptop, aimed at the local enterprise-controlled PC screen.

In another terminal window (launched from the desktop I have virtually open on the enterprise LAN) I've run a query to export the “customer_details” table from the “custdata” database (using the credentials gleaned from my application management access), compressing it on-the-fly (approximately 7:1 saving) and sending it straight to the TCXf.

```
mysql -u dba -p custdata -e "select * from customer_details" | gzip -c -9 | nc localhost 8442
```

I'm receiving this data on my personal laptop, uncompressing it on the fly. In 45 minutes I have the data I wanted and I'm logged out.

Your data is now outside of your organisation and your control, and the only record you have of the transfer itself is the above logs.

3.12.5 Why wouldn't existing Enterprise Security Controls detect/prevent this?

When data was being “typed” to the enterprise at step 6, above, the relevant controls were circumvented:

- Data Loss Prevention (DLP) on the off-shore enterprise-controlled PC and the enterprise LAN based virtual desktop won't trigger as long as the keyboard encoding of the upload doesn't contain trigger words like “confidential” (not possible in a hexadecimal encoding scheme), and;
- End-Point Security (AV/Anti-Malware particularly) on the off-shore enterprise-controlled PC and the enterprise LAN based virtual desktop won't trigger as long as the keyboard encoding of the upload doesn't contain a string that can be found in a known virus, and;
- Intrusion Detection/Prevention functions (assuming they can see inside the encrypted sessions, or that the sessions aren't encrypted) won't trigger as long as the keyboard encoding of the upload doesn't contain a signature known/recognised by that system, and;
- Creating files in the server's scratch space using a common temp file naming convention (/tmp/phpD104.tmp) ensures that the unauthorised change detection software doesn't get upset.

When data was being “displayed” to the off-shore user at step 7, above, the relevant controls were circumvented:

- Data Loss Prevention (DLP) on the off-shore enterprise-controlled PC and the enterprise LAN based virtual desktop won't trigger as long as the display encoding of the upload doesn't contain trigger words like “confidential” or commonly identifiable patterns like Social Security or Credit Card numbers (not possible in a QR code encoding scheme which is constructed from white-space in this text-based example), and;
- End-Point Security (AV/Anti-Malware particularly) on the off-shore enterprise-controlled PC and the enterprise LAN based virtual desktop won't trigger as long as the display encoding of the upload doesn't contain a string that can be found in a known virus.
- Intrusion Detection/Prevention functions (assuming they can see inside the encrypted sessions, or that the sessions aren't encrypted) won't trigger as long as the display encoding of the upload doesn't contain a signature known/recognised by that system (again, not going to be a problem in a protocol founded on white-space characters).

Where did you have any chance at all:

- Detecting the hardware event on the enterprise-controlled PC, though this would be a keyboard or mouse connection.
- If the application server had AV installed and the reference “tcxf-dce” application was a known signature, then it would have been quarantined. However, if you've ever done pen-testing you know that this isn't necessarily a problem, or;
- If the scratch space was mounted as a temporary file-system that had noexec attributed to it, the code would not have run, in which case the user's home directory is probably sufficient, etc.

Fundamentally there are few controls operating in or over the communications channel/s being used and those that are have no understanding of the protocols in use and are thus ineffective.

3.12.6 So what's the Quick-Fix?

There isn't an off-the-shelf solution to this problem.

You could reduce the display rate and resolution for your remote and off-shore users, but it is likely that most of your users will balk at displays smaller than 21 x 21 pixels and/or a refresh rate of 5 frames per second or less.

Asking your End-Point Security, DLP or IDS/IDP vendor to disable QR codes might look like the answer, but it isn't. The PCE end of the TGXf/TCXf reference implementations already supports dozens of machine recognisable bar-code formats. That may tempt you to develop a taxonomy of *evil* bar-codes and block all of those accordingly.

But then how would defeat a version of the TGXf based on OpenCV - the Computer Vision library - that allows people to train²⁹ their own generic implementation on any set of pictures that they want and in any numeric volume? Imagine one user training it on two cat pictures, representing "1" and "0", or another user training it on 17 motorcycle pictures to represent his own character set, or another user choosing 312 farm animal pictures to represent his own character set. And then what if I were to release a version that was pre-trained on the logos of the Fortune 500 companies? Are you really going to filter corporate logos, farm animals, motorcycles and cat pictures from your screens?

And all of that precludes general improvements such as unplugging the HDMI interface and reading pixels as digital bits directly from the cable (potentially raising the bar to hundreds of megabits per second). Do you know which pixels I'm using to communicate on? All user-controlled bits are communications channels.

The whack-a-mole has already started and the game is open to all players. The community needs to look at the problem architecturally to respond to it maturely.

²⁹ <http://www.youtube.com/watch?v=WEzm7L5zoZE>

4 Architectural Analysis

The technology context described, broadly speaking, an implementation specific example of the methods and impacts that are at play. In this section we're going to look at the architectural view of those methods.

4.1 Human versus Machine

It is interesting to consider that there is no variation in the access awarded to the user in the solution provided.

What has been changed can be measured through the following four distinct properties:

- Volume - transfer rate in bits per second, or number of bits at rest, and;
- Accuracy - of data transferred or stored, and;
- Structure - of data transferred, and;
- Utility - of the over-arching capability.

4.1.1 Volume and Accuracy

A human reading from a screen is downloading information at a rate of about 200 words per minute, or 133 bits per second (200 words x 5 bytes x 8 bits / 60 seconds). A human typing into a keyboard is uploading information at a rate of about 120 words per minute, or 80 bits per second (120 words x 5 bytes x 8 bits / 60)³⁰. To provide a scale – Stephen Hawking was uploading at 15 words per minute (10bps) before he moved to the infrared camera solution that interprets his eye blinks, and the stenotype writing world record was set by Mark Kislingbury in 2004, at 360 words per minute (240bps) with 97.23% accuracy³¹.

There's nothing stopping you from typing in the TCXf DCE or TGXf transmit software, today, without any additional technology (i.e. without the Keyboard Stuffer). There's also nothing stopping you from typing in IP packets (though it would still require privilege escalation to put those packets on the wire on the DCE). In fact there's even a well-known standard that considers low through-put high latency carriage for IP: *A Standard for the Transmission of IP Datagrams on Avian Carriers*³².

The primary threat from machine to machine interfacing appears to stem from the super-human limits of the machine, i.e. as a human;

- How long can you read and write at your peak rate?
- What percentage of that data will be errors, or throughput impacts as you resend (re-read or re-type) corrections?
- How much of what you have read can you repeat, letter for letter?

³⁰ https://en.wikipedia.org/wiki/Words_per_minute

³¹ <http://www.guinnessworldrecords.com/records-3000/fastest-realtime-court-reporter-%28stenotype-writing%29/>

³² <http://www.faqs.org/rfcs/rfc1149.html>

4.1.2 Codify, Encipher versus Structure

The link to a US Naval Postgraduate School paper was sent to me by a friend who had some understanding of what this paper (TCXf) would be about. The Naval paper has considered the direct application of “clandestine” message passing to virtual environments such as Second Life³³ where automation and highly articulate characters (avatars) combine to create an infinite number of cyber environments capable of electronically conveying techniques used since ~500BC³⁴. While valid, I believe that this view overlooks the fundamental principle that *every* user controlled bit conveys information. In the context of gaming or other virtual environments this include obvious intermediaries, i.e. that the number of letters in the user's login name might be significant, the date of last sign-on might be significant, the number of sign-ons; their score, the number of zero's in their score, etc.

More interesting are the notions of message passing and clandestine communications.

4.1.2.1 Message Passing and Distinct Channels

When communicating – human to human – we convey messages through a data stream that goes from voice to ear, but we also send side-channel data through touch, gesture³⁵, facial expression, body posture, etc. Gestures are supplemental bits of information that, when structured, can even be exploited for covert or clandestine message passing – say one thing, then wink and it may now mean the opposite, or twice as much, depending on who's winking. Simply partaking in the primary discourse (hearing the voice) has not conveyed the message. The wink has meaning, but don't dwell on it (see the next section). The important part of the problem is the user controlled bit. What happens when the wink becomes the primary communications channel, and we've both agreed on the message structure.

4.1.2.2 Clandestine Communications versus Misunderstood Structure

How clandestine or covert a conveyed “bit” may appear depends on how you see the world and how hard you look at it. If I were to wink erratically at you, you may think me mad. If you understood that I was conveying Morse code, we would have established an effective unidirectional communications channel. In this context I will draw an awkward distinction between types of codified messages. What this paper proposes (and demonstrates) is structuring data to fit a bit manipulator (like a wink) of which Morse would be a good example (it is a protocol that codifies an alphabet into a bit stream). This is not the same as applying a cipher to a communications channel, where a vocal example might be Cockney rhyming slang³⁶ or the Navajo Code Talkers³⁷.

The problem with this distinction is that, architecturally, there is no difference between the two. If you don't understand the Morse protocol then “structured” messages are “encrypted” to you. Similarly if you understand Navajo, then their radio messages are simply “structured” to you. The

33 <http://www.dtic.mil/dtic/tr/fulltext/u2/a488794.pdf>

34 https://en.wikipedia.org/wiki/Aristagoras#Ionian_Revolt

35 https://en.wikipedia.org/wiki/Types_of_gestures

36 https://en.wikipedia.org/wiki/Cockney_rhyming_slang

37 https://en.wikipedia.org/wiki/Code_talker

distinction is clarified through intent, i.e. what is proposed here is founded in the intent to communicate, rather than the intent to conceal. Aside from Morse, another good example might be a phone number – not intended to hide identity behind some high dimensional mathematical cipher, it merely codifies an electronic routing scheme used to establish the communications channel – conveying the message “I want to speak to my mate Dave”.

Therefore the inherent difference between the human operator and the machine operator is the structure of the communication. As a human you are able to parse substantially less structured data (you can read this document including its pictures, and noting the fact that you skipped the first chapter, you weren't trained on the structure of this document specifically), but that wall is coming down. Devolution of cultural references, near universal distribution of homogeneous technology (you are reading this on one of millions of identical devices, amongst billions³⁸ of devices that are already using common electronic communications and display protocols) and abstraction of machine learning are rapidly reducing the identifiable gap.

Perhaps it is sufficient to say then, that the machine to machine communication channel currently requires “additional structure” beyond that required by the human counterpart.

4.1.3 Utility

Humans arguably have greater utility than their machine replacement (you can take bits, pass a metal detector with them, swim with them, etc), however data is not seen as valuable outside of the machine in terms of utility (in addition to the feeble accuracy of your declarative memory³⁹).

In terms of threat, the most substantial impacts stem from:

- One end of the attack being outside of the controlled environment entirely (beyond identification or containment), and;
- The communications channel leveraged being effectively unchecked – so long as the encoding schemes employed don't invoke random assignments of data segments of interest (such as a viral/malware signature⁴⁰, offensive or culturally unacceptable terminology, or common strings of value like social security numbers, security classifications or other internal handles, signatures or watermarks), and;
- The completely digital preservation of the source data, in a mobile/portable format (not constrained to a specific time, physical or logical environment, or person) – meaning that it is immediately reusable in all available communications channels without restraint.

38 <http://www.forbes.com/sites/quora/2013/01/07/how-many-things-are-currently-connected-to-the-internet-of-things-iot/>

39 <http://www.plosone.org/article/info%3Adoi%2F10.1371%2Fjournal.pone.0033079>

40 <http://www.eicar.org/86-0-Intended-use.html>

4.2 Mitigation Strategies

Mitigation strategies would be necessarily aligned to the four distinct measurable properties:

- Volume
 - Reduce the resolution of display data
 - Decrease the number of USB HID packets that can be received per second
 - Increase latency into data streams in either direction
- Accuracy
 - Add visual distortions to display data
 - Introduce errors into key strokes and add noise to mouse pointer data
- Structure
 - Analyse display data for excessive structure (Consider a “deep packet inspection” equivalent for displays – looking inside video and image data for additional structure)
 - Perform frequency analysis on key strokes or bayesian analysis on typed words
- Utility
 - Seal (glue/weld) electronic interfaces
 - Apply secondary controls to exposed user interfaces

These controls should be deployed appropriately. For example electronic and user interfaces exist at the Data Centre, the PC and most places in between, but most threat would be at the PC end. USB HID packets exist at the PC end but may be more effectively controlled/limited at the Data Centre.

4.2.1 Counter Culture

You'll note all of these mitigation proposals run almost completely counter to your current remote access strategy, namely;

- Reducing latency / improving responsiveness
 - Increased network infrastructure capacities
 - Ever increasing CPU core-count per user
- Increasing throughput (particularly in terms of display performance)
 - Increased display resolution and refresh rates
 - Video acceleration
 - Stream compression / Link compression
- Increased utility
 - Less structured graphical interfaces
 - Allowing users to bring uncontrolled devices into controlled environments or even into direct connectivity with controlled environments.
- Apathy toward physical controls
 - Physical security lax, weak or absent at end points
 - Anything/Anywhere “to be cool like company x” publishing controlled interfaces into uncontrolled environments (internal APIs to the Internet, UTP ports to waiting rooms)
 - Introduction of photographic equipment into controlled environments

4.2.2 Forever War

Be prepared for war. There is no end to this problem and human consumers will be on the losing side. Evolutionary strategies are likely to include:

- Volume
 - Using multiple interfaces to increase through-put;
 - For example, the caps-lock, num-lock, scroll-lock lights on key-boards are uploading user controlled bits.
 - Sound has not been explored in this paper⁴¹
 - Using software interfaces to increase through-put;
 - The virtualised ATAPI interface that allows a user to map send/receive bits from one environment to another
 - The Printer driver that allows a user to map send/receive bits from one environment to another
- Accuracy
 - Making noise
 - Random errors in displays would not be unheard of (we've all persisted with poor quality television/cable reception at one point or another), but then neither is error correction (and the Shannon-Hartley Theorem⁴²).
 - How long do you think you're users will put up with you injection errors into email, documents and chat conversations?
- Structure
 - Analyse input/output data for excessive structure
 - Observing highly structured data (like only 20 keys ever being used on a keyboard, excluding certain vowels) is an obvious detection method.
 - Consider a “deep packet inspection” equivalents (for displays this might include looking inside video and image data for additional structure).
 - Also be prepared for;
 - OCR and encoded blocks of text – such as English stories (see the next section)
 - Non-contextual use of “foreign” languages (you might search for English while I switch to Japanese)
 - Data in “random” pixels, similar to the faint yellow printer serial dots⁴³ (relevant when I interface directly with the HDMI port, particularly).
- Utility
 - Cutting or melting equipment;
 - Glue a USB socket and I'll cut the keyboard cord – exposing the USB interface again.
 - Bury the cord in a metal conduit, or encrypt the keyboard-to-PC communications channel and I'll remove the keys.
 - Disconnected accessories;
 - How long before you are able to replace every keyboard with a virtual one (under glass), two budgets? four?
 - And then what will you do when I release an electrostatic overlay that answers to

41 <http://pipedot.org/story/2014-09-19/quietnet-a-simple-chat-program-using-inaudible-sounds>

42 http://en.wikipedia.org/wiki/Shannon%E2%80%93Hartley_theorem

43 <https://www.eff.org/wp/investigating-machine-identification-code-technology-color-laser-printers>

- capacitive/resistive displays, or a reflective interface that answers to reflective optical displays?
- Morphing infrastructure;
 - Are you really prepared to randomly move the virtual keyboard so that the user has to visually follow it to keep typing?
 - Strobe light sources that prevent auto-focus cameras from picking up your displays?

4.3 Skipping to the Punch Line

Where does it all end? This won't end – but it seems likely to reach a new equilibrium. A mechanised attacker that interfaces like a human (given the context of the peripheral) and performs like a human – entering data in and retrieving data like a human would, both statistically and structurally – i.e. in human languages.

If this sounds far fetched then consider that computer programs have been writing academic papers for almost a decade. The first to gain notoriety was the Computer Science research paper “Router: A Methodology for the Typical Unification of Access Points and Redundancy” which was accepted as a “non-reviewed” paper to the *2005 World Multiconference on Systemics, Cybernetics and Informatics (WMSCI)*⁴⁴. Three years later the paper “Towards the Simulation of E-commerce” was accepted, after peer review, for presentation at the *2008 International Conference on Computer Science and Software Engineering (CSSE)*⁴⁵. Computers will read and write text shortly and you won't know whether you've got a human or a machine on the line.

In terms of electro-mechanical peripheral interfaces, the architecturally pragmatic solution is a single common interface with software defined features that can be controlled consistently regardless of the attached peripheral. Sounds like a data network, right? All that does is change the transport whilst leaving the problem in place.

What I believe we truly lack is the controls framework that is articulate enough to respond to the configuration required for mechanised threats;

- Volume - transfer rate in bits per second, or number of bits at rest, and;
- Accuracy - of data transferred or stored, and;
- Structure - of data transferred, and;
- Utility - of the over-arching capability.

Furthermore, I would be surprised if this controls framework wasn't broadly useful to society as “intelligent” computers with integrated robotics (or at the very least, flexible human machine interfaces) become commonplace (both offensively and defensively); i.e. the above could be considered as the basis of a machine intelligence controls framework.

44 <http://pdos.csail.mit.edu/scigen/>

45 <http://slashdot.org/story/08/12/23/2321242/software-generated-paper-accepted-at-ieee-conference>

4.4 Historical Considerations and Support

After writing up my own analysis (above), I researched the problem space, the exploit and the proposed controls frame-work to see if there was research that was supportive or contradictory.

What I found was refreshingly intelligent consideration to this problem space from forty years ago, stemming from the “Covert Channel” knowledge domain.

4.4.1 Anderson, 1972

In his “Computer Security Technology Planning Study” (October 1972)⁴⁶, James Anderson (and panel) looked deeply into the problem of defence against a malicious user attack – see these quotes from section 3.2 *The Malicious User Threat*:

As a malicious user is able to exercise more direct control over a computer through programming, he has the use of the computer as a tool to help his penetration and sub-sequent exploitation of the system. IF he has a full programming capability using assembly or most of the higher order languages, he has the maximum possible user control of the system, and has available all but a few of the tools needed to aid him in his penetration. As the users capability is reduced by such means as forcing him to use interpretive systems, transaction processing systems and the like, his opportunities for direct control of the machine through his programming actions is correspondingly reduced because these tools are not sufficient for that purpose. His threat is reduced but unfortunately not eliminated through use of such techniques. Although the scope of actions directed to achieve penetration is reduced, he can still probe the system for exploitable design or implementation flaws using non-sequitor commands, false of 'nonsense' parameters, unanticipated interruptions, and the like. If the malicious user is a supported agent, he may merely exercise a 'trapdoor' placed in the system by another agent to gain access to classified data.

[...]

The sheer size of contemporary operating systems (on the order of 100,000+ instructions) and their complexity makes it virtually impossible to validate the static design and implementation of the system. When the dynamic behavior of the system is contemplated as well, there is no practical way to validate that all of the possible control paths of the operating system in execution produce correct, error-free results.

Anderson doesn't seem to believe that a solution exists – i.e. systems will be flawed and they can't be properly assessed (for reference, the Linux kernel had 15 million lines of code in 2011⁴⁷). The study also advances the notion that the less utility available to the user, the greater the system security. A small treatise on this principle was published as Appendix IV “SECURITY VULNERABILITY AS A FUNCTION OF USER CONTROL OF SHARED RESOURCES” in the same study, some of which has been quoted here:

USER ISOLATION IN A SHARED RESOURCE ENVIRONMENT

It has been advanced as a working hypothesis that a “security perfect” computer system is vulnerable only to physical threats. In the real world of imperfect instruments, vulnerability is extended to include errors in design, implementation, operation and maintenance, and design and fabrication incompleteness to handle actual operational loads. It is then axiomatic that the more complex the operational system:

1. the greater the probability of error,

46 <http://seclab.cs.ucdavis.edu/projects/history/papers/ande72.pdf>

47 <http://arstechnica.com/business/2012/04/linux-kernel-in-2011-15-million-total-lines-of-code-and-microsoft-is-a-top-contributor/>

PUBLIC
ThruConsoleXfer (TCXf) White Paper

2. the greater the resources available to the interloper to probe the system for weakness, and
3. the increased sharing of resources increases the potential security exposure of the common user community to discovered system flaws.

Thus, if we can build better "firewalls" between users we can limit the extent of security compromise in multi-user, multi-level systems.

This appendix tries to increase our understanding of security failure modes and possible design strategies that offer promise of ameliorating the security vulnerability of failure. The thesis advanced here is that better isolation of shared resources offers the best, and possibly only, solution.

[...]

VULNERABILITY INCREASE WITH INCREASED USER CONTROL

User control over the real hardware resources ranges from the user just watching a computer display, to total control of hardware where physical wiring can be modified. Figure VI-1 discusses these levels of control in terms of the resources shared, the direct vulnerability (1st level), and the security payoff to the interloper.

06

USER CONTROL	SHARED RESOURCE	1ST LEVEL VULNERABILITY	PAYOFF
1. Just Watch	Display Surface	<ul style="list-style-type: none"> ● Malfunction/BUG/Residue Access ● Destruction/Jam ● Sophisticated Jam 	<ul style="list-style-type: none"> ● Gain (Random) ● Deny (Random) ● Falsify (Random)
2. Initiate Program (1 + Limited Push Buttons) Manual Probes	<ul style="list-style-type: none"> ● {OS {Appl. Prog.} CPU ● Data STORE 	<ul style="list-style-type: none"> ● Insufficient Legality Check ● Illegal Sequencing ● Crash System 	<ul style="list-style-type: none"> ● Gain (Directed) ● Gain (Random) ● Deny
3. Transaction Only (2 + Enter Parameters) Machine-Aided Probes	<ul style="list-style-type: none"> ● Time (Response Feedback) ● Increased I/O Bandwidth 	<ul style="list-style-type: none"> ● Logic Path Complexity ● Data Aggregation 	<ul style="list-style-type: none"> ● Gain ● Deny ● Falsify
4. Interpretive Code (3 + Code Sequences) Machine-Generated Probes	<ul style="list-style-type: none"> ● Limited Psuedo-Machine (Interpreter) 	<ul style="list-style-type: none"> ● Higher Order Complexity ● STORE Overload (Data Cancer) ● CPU Overload (Program Loop) 	<ul style="list-style-type: none"> ● Gain, Deny, Falsify ● Deny
5. Compiled Code	<ul style="list-style-type: none"> ● Limited Real-Machine (Compiler) 	<ul style="list-style-type: none"> ● Break into Machine Code (see 6) 	<ul style="list-style-type: none"> ● Gain, Deny, Falsify
6. Machine Code	<ul style="list-style-type: none"> ● Near-Total System Control ● Real Addresses ● Real Op Codes 	<ul style="list-style-type: none"> ● Violate Software (OS) Integrity ● Incomplete System Design 	<ul style="list-style-type: none"> ● Gain, Deny, Falsify
7. Machine Code (Monitor State)	<ul style="list-style-type: none"> ● Total System Control 	<ul style="list-style-type: none"> ● No System Checks & Balances ● Modify Software (OS) Integrity 	<ul style="list-style-type: none"> ● Gain, Deny, Falsify
8. Hardware	<ul style="list-style-type: none"> ● Total System Control 	<ul style="list-style-type: none"> ● Modify System Integrity 	<ul style="list-style-type: none"> ● Gain, Deny, Falsify

Figure IV-1. Increasing Security Vulnerability With Cumulative User Resource Control

Anderson's view that system security can be improved through reduced user control (or greater control of system interfaces, or lesser user utility) and increased isolation of shared resources (reduced opportunity/exposure to vulnerability/failure and reduced impact when exploited) is significant.

4.4.2 Lampson, 1973

Butler Lampson's "A Note on the Confinement Problem" (published October 1973)⁴⁸ considered seven example scenarios whereby a process owner could create covert channels around bits he could manipulate including the CPU load of the user-controlled process (6); a read/write lock on a shared/common file (5), and; manipulating output in the bill issued for the service consumed (4).

Lampson's analysis led him define a set of "Confinement Rules";

We begin by observing that a confined program must be memoryless. In other words, it must not be able to preserve information within itself from one call to another. Most existing systems make it quite easy to enforce this restriction. In ALGOL, for example, a procedure which has no own variables and which references no global variables will be memoryless.

Taking this for granted, it is easy to state a rule which is sufficient to ensure confinement.

Total isolation: A confined program shall make no calls on any other program.

By a "call" we mean any transfer of control which is caused by the confined program. Unfortunately this rule is quite impractical. Example 5 above shows that supervisor calls must be forbidden, since quite innocuous looking ones can result in leaks. Example 6 shows that even the implicit supervisor calls resulting from input/output, paging or time-slicing can cause trouble.

To improve on this situation, we must make a distinction between programs which are confined and those which are not. Recall that being confined is not an intrinsic property of a program but a constraint which can be imposed on it when it runs. Hence if every program called by a confined program is also confined, there can be no leakage. This is still not good enough, since the supervisor, for example, cannot be confined. It is at least conceivable, however, that it is trusted, i.e. that the customer believes it will not leak his data or help any confined program which calls it to do so. Granting this, we can formulate a new confinement rule.

Transitivity: If a confined program calls another program which is not trusted, the called program must also be confined.

Examples 5 and 6 show that it is hard to write a trustworthy supervisor, since some of the paths by which information can leak out from a supervisor are quite subtle and obscure. The remainder of this note argues that it is possible.

A trustworthy program must guard against any possible leakage of data. In the case of a supervisor, the number of possible channels for such leakage is surprisingly large, but certainly not infinite. It is necessary to enumerate them all and then to block each one. There is not likely to be any rigorous way of identifying every channel in any system of even moderate complexity. The six examples above, however, were chosen to illustrate the full range of possibilities which I have been able to think of, and for the present plausibility argument they will be regarded as a sufficient enumeration. The channels used there fall into three categories:

Storage of various kinds maintained by the supervisor which can be written by the service and read by an unconfined program, either shortly after it is written or at some later time.

Legitimate channels used by the confined service, such as the bill.

Covert channels, i.e. those not intended for information transfer at all, such as the service program's effect on the system load.

All the examples except 4 and 6 use storage channels. Fairly straightforward techniques can be used by the supervisor to prevent the misuse of storage as an escape route for a confined program. (For instance, example 5 can be taken care of by making a copy of a file which is being read by a confined program if someone else tries to write it; the confined program can then continue to read the copy, and the writer can proceed without receiving any indication that the file was being read.) The main difficulty, as example 5 illustrates, is to identify all the kinds of storage which the supervisor implements. This

48 <http://research.microsoft.com/en-us/um/people/blampson/11-Confinement/ Acrobat.pdf>

PUBLIC
ThruConsoleXfer (TCXf) White Paper

class of channels will not be considered further.

The following simple principle is sufficient to block all legitimate and covert channels. Masking: A program to be confined must allow its caller to determine all its inputs into legitimate and covert channels. We say that the channels are masked by the caller.

At first sight it seems absurd to allow the customer to determine the bill, but since the service has the right to reject the call, this scheme is an exact model of the purchase order system used for industrial procurement. Normally the vendor of the service will publish specifications from which the customer can compute the bill, and this computation might even be done automatically from an algorithmic specification by a trusted intermediary.

In the case of the covert channels one further point must be made.

Enforcement: The supervisor must ensure that a confined program's input to covert channels conforms to the caller's specifications.

This may require slowing the program down, generating spurious disk references, or whatever, but it is conceptually straightforward.

The cost of enforcement may be high. A cheaper alternative (if the customer is willing to accept some amount of leakage) is to bound the capacity of the covert channels.

I'm a big fan of Lampson's thinking, and given that these rules appear to have become the basis of the TCSEC evaluation criteria (see below), the US DoD took his message to heart as well.

In one discussion with a peer of mine before I began writing this paper, we tested my version of what I later saw that Lampson had coined as *Transitivity*. Transitivity can also be expressed more simply in terms of one program and one data file. i.e. The unix command “cat” is inoffensive when it sends data to /dev/null, but may increase risk when sending data to STDOUT. My theory was that a program should inherit the sensitivity (rights) of the data that it is working with. I dropped this idea because the TGXf exploit does not fundamentally alter the privileges that the user already has (the user could read and view the file before TGXf). Whereas I believe that the four properties described earlier in this chapter (volume, accuracy, structure and utility) do fundamentally encapsulate the change in posture.

Another way to get the same outcome is to regard transitivity as a poorly defined interface. If, for example, file permissions included the right to “display” a file versus the right to “download” a file then perhaps “cat” could operate within the bounds and constraints of the intention of the system (performing accordingly against various system devices as defined by their capabilities, for example). But this, too, fails the common sense tests – if I can “see” data then I have effectively “downloaded” that data (by what other means, beyond the acquisition of data, was the content on my screen rendered?). You can take this concept as deeply as you like – syscall, instruction, etc.

It is also interesting to note that Lampson's final recommendation (not pursuing Enforcement to the last bit) was also accepted by the US DoD.

4.4.3 Vleck, 1976

In May 1990 Van Vleck presented his 1976 exploration on “Timing Channels” in Multics Systems to the IEEE's Technical Committee on Security and Privacy (TCSP)⁴⁹.

49 <http://www.multicians.org/timing-chn.html>

PUBLIC
ThruConsoleXfer (TCXf) White Paper

About 1976, I was explaining Multics security to a colleague at Honeywell. (This was before it got the B2 rating, but the mandatory access control mechanism was present.) I explained the difference between storage channels and timing channels, and said that timing channels weren't important because they were so slow, noisy, and easily clogged, that you couldn't send a signal effectively.

My friend, Bob Mullen, astounded me a few days later by showing me two processes in the terminal room. You could type into one and the other would type out the same phrase a few seconds later. The processes were communicating at teletype speed by causing page faults and observing how long they took. The sender process read or didn't read certain pages of a public library file. The receiver process read the pages and determined whether they were already in core by seeing if the read took a long or short real time. The processes were running on a large Multics utility installation at MIT under average load; occasionally the primitive coding scheme used would lose sync and the receiver would type an X instead of the letter sent. If you typed in "trojan horse," the other process would type "trojan horse" a little later, with occasional X's sprinkled in the output.

Bob pointed out that noise in the channel was irrelevant to the bandwidth (Shannon's Law). Suitable coding schemes could get comparable signaling rates over many different resource types, despite countermeasures by the OS: besides page faulting, one could signal via CPU demand, segment activation, disk cache loading, or other means.

When I thought about this I realized that any dynamically shared resource is a channel. If a process sees any different result due to another process's operation, there is a channel between them. If a resource is shared between two processes, such that one process might wait or not depending on the other's action, then the wait can be observed and there is a timing channel.

Closing the channel Bob demonstrated would be difficult. The virtual memory machinery would have to do extra bookkeeping and extra page reads. To close the CPU demand channel, the scheduler would have to preallocate time slots to higher levels, and idle instead of making unneeded CPU time available to lower levels.

Bob and I did not compute the bandwidth of the page fault channel. It was about as fast as electronic mail for short messages. A spy could create a Trojan Horse program that communicated with his receiver process, and if executed by a user of another classification, the program could send messages via timing channels despite the star-property controls on data.

The rules against writing down were designed to keep Trojan Horse programs from sending data to lower levels, by ensuring that illegal actions would fault. In this case, the sender is doing something legal, looking at data he's allowed to, consuming resources or not, and so on. The receiver is doing something legal too, for example looking at pages and reading the clock. Forbidding repeated clock reading or returning fake clock values would restrict program semantics and introduce complexity.

I believe we need a mechanism complementary to the PERMISSION mechanism, called a CERTIFICATION mechanism. This controls the right to Execute software, while permission controls the Read and Write flags. A user classified Secret is not allowed to execute, at Secret level, a program certified only to Unclassified. Programs would only be certified to a given level after examination to ensure they contained no Trojan Horses. This kind of permission bit handling is easy to implement, and if the site only certifies benign programs, all works perfectly.

How do we examine programs to make sure they have no Trojan Horses? The same way we examine kernels to make sure they don't. Examining application code to ensure it doesn't signal information illegally should be easier than the work we have to do on a kernel.

Do we need to forbid write down for storage, if we allow write down via timing channels? It's inconsistent to forbid some write down paths and allow others. The main reason for preventing write down is not to stop deliberate declassification (a spy can always memorize a secret and type it into an unclassified file), it's to protect users from Trojan Horse programs leaking information. The certification approach solves this problem a different way, by providing a way to prevent execution of Trojan Horse programs.

Who should be allowed to certify a program to a given level? I think anyone who can create a program to execute at level X should be able to take a lower level program and certify it to level X. We can't stop a malicious user from creating a transmitter program, unless we forbid all program creation at level X. Other policies are possible; a site could centralize all program certification as a security officer responsibility. In any case, the OS should audit every certification and store the identity of the certifier as a file attribute.

Mullen's note regarding Shannon's Law suggests that Lampson's Enforcement principle is probably futile. Vleck's conclusion (to review/authorise code for execution and communication at a given classification or level within the executive) is equally futile (in an open computing model) when considering Anderson's experience (which did include off-the-shelf hardware and software systems in complex deployments).

4.4.4 US DoD Standard, 1983/1985

Lampson's work contributed a controls taxonomy that seems to have been adopted by the August 1983 *Trusted Computer Security Evaluation Criteria* (TCSEC)⁵⁰ which directly addresses *Covert storage channels* and *Covert timing channels* in B2 and B3 evaluations, respectively.

The following quotation comes from the December 1985 version, page 80;

8.0 A GUIDELINE ON COVERT CHANNELS

A covert channel is any communication channel that can be exploited by a process to transfer information in a manner that violates the system's security policy. There are two types of covert channels: storage channels and timing channels. Covert storage channels include all vehicles that would allow the direct or indirect writing of a storage location by one process and the direct or indirect reading of it by another. Covert timing channels include all vehicles that would allow one process to signal information to another process by modulating its own use of system resources in such a way that the change in response time observed by the second process would provide information.

From a security perspective, covert channels with low bandwidths represent a lower threat than those with high bandwidths. However, for many types of cover channels, techniques used to reduce the bandwidth below a certain rate (which depends on the specific channel mechanism and the system architecture) also have the effect of degrading the performance provided to legitimate system users. Hence, a trade-off between system performance and covert channel bandwidth must be made. Because of the threat of compromise that would be present in any multilevel computer system containing classified or sensitive information, such systems should not contain covert channels with high bandwidths. This guideline is intended to provide system developers with an idea of just how high a "high" covert channel bandwidth is.

A covert channel bandwidth that exceeds a rate of one hundred (100) bits per second is considered "high" because 100 bits per second is the approximate rate at which many computer terminals are run. It does not seem appropriate to call a computer system "secure" if information can be compromised at a rate equal to the normal output rate of some commonly used device.

In any multilevel computer system there are a number of relatively low-bandwidth covert channels whose existence is deeply ingrained in the system design. Faced with the large potential cost of reducing the bandwidths of such covert channels, it is felt that those with maximum bandwidths of less than one (1) bit per second are acceptable in most application environments. Though maintaining acceptable performance in some systems may make it impractical to eliminate all covert channels with bandwidths of 1 or more bits per second, it is possible to audit their use without adversely affecting systems performance. This audit capability provides the system administration with a means of detecting – and procedurally correcting – significant compromise. Therefore, a Trusted Computing Base should provide, wherever possible, the capability to audit the use of covert channel mechanisms with bandwidths that may exceed a rate of one (1) bit in ten (10) seconds.

The covert channel problem has been addressed by a number of authors. The interested reader is referred to references [5], [6], [19], [21], [22], [23], and [29]

50 <http://csrc.nist.gov/publications/history/dod85.pdf>

In the end (the end being 1985, though I believe this document was retired in 2002) it appears that Lampson has had the first and last say on covert channels, with the US DoD agreeing to a Trusted Computing Base retaining covert channels of between 0.1bps and 1bps so long as they are audit-able and less than 0.1bps in any other permissible case.

One note on “high” bandwidth being benchmarked against the “computer terminal” at 100 bits per second (which I agree with as a benchmark, as it aligns to the figures I collated at the beginning of this chapter). A modern computer will use HDMI to display data on a 1080p screen. That equates to 1920 pixels wide x 1080 pixels high x 24 bit (True Color) x 24 frames per second which is 1,194,363,600 bits per second or approximately 150 Megabytes per second⁵¹. As an optimist you could consider that an increase in the threshold for concern, but another view is that if the modern computer terminal is now almost 12 million times faster then imagine the comparative scale of all previously known/agreed covert channels (i.e. something that was emitting at 0.1bps in 1985 may now be capable of 1,200,000bps).

4.4.5 Further Reading

For those interested in further reading there are two leads that might be worth following.

4.4.5.1 Millen, 1999

Jonathan Millen published a paper “20 Years of Covert Channel Analysis” in 1999 that I haven't been able to acquire unencumbered⁵². Millen's background⁵³ puts him in the right places and the right times to bring the covert channel story forward by precisely 20 years. There is an abstract available at his site⁵⁴.

4.4.5.2 Jitterbugs, 2006+

Jitterbugs were introduced in the 2006 via the paper “Keyboards and Covert Channels”⁵⁵ by Gaurav Shah, Andres Molina and Matt Blaze. Jitterbugs invigorated the class of timing based covert channels and the reignited the public interest in covert channels. By way of example, see the subsequent paper “An Entropy-Based Approach to Detecting Covert Timing Channels”⁵⁶ and the counter-point paper “Liquid: A Detection-Resistant Covert Timing Channel Based on IPD Shaping”⁵⁷.

51 <http://www.zytrax.com/tech/pc/resolution.html#overview>

52 http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=766906

53 <http://www.csl.sri.com/users/millen/>

54 <http://www.csl.sri.com/users/millen/papers/20yrcc.ps>

55 <http://www.crypto.com/papers/jbug-Usenix06-final.pdf>

56 <http://www.cs.wm.edu/~hnw/paper/tdsc10.pdf>

57 <http://isec.uta.edu/mwright/papers/liquid.pdf>

4.4.6 Impact of Historical Considerations

Although unaware of it at the outset of this paper – I now believe that the solution proposed is properly classified as a storage-based covert channel. Assuming the Trusted Computing guidelines were calculated based on human information processing rates (rather than literal computer terminal data processing rates), then a benchmark for best practice exists: covert channels above 1bps must be mitigated, and there are practical/feasibility reasons for not completely preventing all covert channel opportunities.

All of the (publicly accessible) research that I've uncovered focuses on either inter-process communication on the same host (shared computing resource) or leaking information through numerous network packet manipulation techniques. There doesn't seem to have been consideration given to a malicious user injecting tools into, or extracting information from, systems that he already has access to, via covert channels (i.e. Where is the threat from someone communicating from a program they own to a program they own? Or jitterbugging their own keyboard?). I suspect that this is because of the unique set of circumstances that are created from modern enterprise architectures and the current governance frameworks/models and incentives.

It also seems likely that the change in focus of the problem and the performance improvements made in almost 50 years of computer advances have made the covert channel problem much worse today than it was when first discovered in the early 1970s.

All of that aside, there does not appear to be a generic solution to the covert channel problem that can be readily adopted and applied to the existing enterprise security architecture to mitigate the technology solution set out in this paper. At the very least, then, we must heed Lampson's advice – public since 1973 – and prominently include covert channels in the risk assessment frameworks:

The cost of enforcement may be high. A cheaper alternative (if the customer is willing to accept some amount of leakage) is to bound the capacity of the covert channels.

5 Framework Outcomes

The paradigm shift that comes from re-imagining and technically demonstrating user-controlled interfaces as data networks (for example) may have subtle implications upon multiple frameworks, some depending entirely on the demarcations created by definitions.

5.1 Australian Legal Context

The revised Australian Privacy Principles came into effect in 2014. I will refer you to the *Australian Privacy Principles guidelines (Privacy Act 1988)*⁵⁸ - version 1.0, February 2014.

5.1.1 Australian Privacy Principles

See “Chapter 6: Australian Privacy Principle 6 – Use or disclosure of personal information”, beginning page 107 of 211;

What does APP 6 say?

6.1 APP 6 outlines when an APP entity may use or disclose personal information. The intent is that an entity will generally use and disclose an individual’s personal information only in ways the individual would expect or where one of the exceptions applies.

6.2 An APP entity that holds personal information about an individual can only use or disclose the information for a particular purpose for which it was collected (known as the ‘primary purpose’ of collection), unless an exception applies. Where an exception applies the entity may use or disclose personal information for another purpose (known as the ‘secondary purpose’). Exceptions include:

- the individual consented to a secondary use or disclosure (APP 6.1(a))
- the individual would reasonably expect the secondary use or disclosure, and that is related to the primary purpose of collection or, in the case of sensitive information, directly related to the primary purpose (APP 6.2(a))
- the secondary use or disclosure of the personal information is required or authorised by or under an Australian law or a court/tribunal order (APP 6.2(b))
- a permitted general situation exists in relation to the secondary use or disclosure of the personal information by the APP entity (APP 6.2(c))
- the APP entity is an organisation and a permitted health situation exists in relation to the secondary use or disclosure of the personal information by the organisation (APP 6.2(d))
- the APP entity reasonably believes that the secondary use or disclosure is reasonably necessary for one or more enforcement related activities conducted by, or on behalf of, an enforcement body (APP 6.2(e))
- the APP entity is an agency (other than an enforcement body) and discloses personal information that is biometric information or biometric templates to an enforcement body, and the disclosure is conducted in accordance with guidelines made by the Information Commissioner for the purposes of APP 6.3 (APP 6.3).

6.3 An APP entity may disclose personal information, other than sensitive information, to a related body corporate (s 13B(1)(b)).

6.4 APP 6 does not apply to the use or disclosure by an organisation of :

- personal information for the purpose of direct marketing (this is covered by APP 7), or
- government related identifiers (this is covered by APP 9) (APP 6.7).

‘Holds’, ‘use’, ‘disclose’ and ‘purpose’

6.5 Each of the terms ‘holds’, ‘use’, ‘disclose’ and ‘purpose’ which are used in APP 6

58 <http://www.oaic.gov.au/images/documents/privacy/applying-privacy-law/app-guidelines/APP-guidelines-combined-set-v1.pdf>

PUBLIC
ThruConsoleXfer (TCXf) White Paper

and other APPs, are discussed in more detail in Chapter B (Key concepts). The following is a brief analysis of the meaning of these terms in the context of APP 6.

‘Holds’

6.6 APP 6 only applies to personal information that an APP entity ‘holds’. An APP entity ‘holds’ personal information ‘if the entity has possession or control of a record that contains the personal information’ (s 6(1)).

6.7 The term ‘holds’ extends beyond physical possession of a record to include a record that an entity has the right or power to deal with. For example, an APP entity that outsources the storage of personal information to a third party, but retains the right to deal with that information, including to access and amend it, holds that personal information. The term ‘holds’ is discussed further in Chapter B (Key concepts).

‘Use’

6.8 The term ‘use’ is not defined in the Privacy Act. An APP entity ‘uses’ information where it handles or undertakes an activity with the information, within the entity’s effective control. For further discussion of use, see Chapter B (Key concepts). Examples include:

- the entity accessing and reading the personal information
- the entity searching records for the personal information
- the entity making a decision based on the personal information
- the entity passing the personal information from one part of the entity to another
- unauthorised access by an employee of the entity.

‘Disclose’

6.9 The term ‘disclose’ is not defined in the Privacy Act. An APP entity ‘discloses’ personal information where it makes it accessible to others outside the entity and releases the subsequent handling of the information from its effective control. This focuses on the act done by the disclosing party. The state of mind or intentions of the recipient does not affect the act of disclosure. Further, there will be a disclosure in these circumstances even where the information is already known to the recipient. For further discussion of disclosure, see Chapter B (Key concepts).

6.10 The release may be a proactive release or publication, a release in response to a specific request, an accidental release or an unauthorised release by an employee. Examples include where an APP entity:

- shares the personal information with another entity or individual
- discloses personal information to themselves, but in their capacity as a different entity
- publishes the personal information on the internet, whether intentionally or not, and it is accessible by another entity or individual
- accidentally provides personal information to an unintended recipient
- reveals the personal information in the course of a conversation with a person outside the entity
- displays a computer screen so that the personal information can be read by another entity or individual, for example, at a reception counter or in an office

6.11 ‘Disclosure’ is a separate concept from:

- ‘unauthorised access’ which is addressed in APP 11. An APP entity is not taken to have disclosed personal information where a third party intentionally exploits the entity’s security measures and gains unauthorised access to the information. Examples include unauthorised access following a cyber-attack or a theft, including where the third party then makes that personal information available to others outside the entity. However, where a third party gains unauthorised access, the APP entity may breach APP 11 if it did not take reasonable steps to protect the information from unauthorised access (see Chapter 11 (APP 11))
- ‘use’, which is discussed in paragraph 6.8 above. APP 6 generally imposes the same obligations on an APP entity for uses and disclosures of personal information. Therefore, this distinction is not relevant in interpreting this principle (except in relation to APP 6.3). However, the distinction is relevant to APP 8, which applies to the disclosure of personal information to an overseas recipient (see Chapter 8 (APP 8)).

See “Chapter 8: Australian Privacy Principle 8 – Cross-border disclosure of personal information”, beginning page 136 of 211;

What does APP 8 say?

8.1 APP 8 and s 16C create a framework for the cross - border disclosure of personal information . The framework generally requires an APP entity to ensure that an overseas recipient will handle an individual's personal information in accordance with the APPs, and makes the APP entity accountable if the overseas recipient mishandles the information. This reflects a central object of the Privacy Act, of facilitating the free flow of information across national borders while ensuring that the privacy of individuals is respected (s 2A(f)) .

8.2 APP 8.1 provides that before an APP entity discloses personal information about an individual to an overseas recipient, the entity must take reasonable steps to ensure that the recipient does not breach the APPs in relation to that information. Where an entity discloses personal information to an overseas recipient, it is accountable for an act or practice of the overseas recipient that would breach the APPs (s 16C) .

8.3 There are exceptions to the requirement in APP 8.1 and to the accountability provision in s 16C (see paragraphs 8.19 – 8.55 below) .

8.4 When an APP entity discloses personal information to an overseas recipient it will also need to comply with APP 6. That is, it must only disclose the personal information for the primary purpose for which it was collected unless an exception to that principle applies (see Chapter 6 (APP 6)). A note to APP 6.1 cross - references the requirements for the cross - border disclosure of personal information in APP 8. It is implicit in this note, that APP 8 only applies to personal information covered by APP 6. That is, it only applies to personal information ‘held’ by an APP entity. The term ‘holds’ is discussed in Chapter B (Key concepts).

‘Overseas recipient’

8.5 Under APP 8.1, an ‘overseas recipient’ is a person who receives personal information from an APP entity and is:

- not in Australia or an external Territory
- not the APP entity disclosing the personal information, and
- not the individual to whom the personal information relates.

8.6 This means that where an APP entity in Australia sends information to an overseas office of the entity, AP P 8 will not apply as the recipient is the same entity. This is to be distinguished from the case where an APP entity in Australia sends personal information to a ‘related body corporate’ located outside of Australia. In that case, the related body corporate is a different entity to the APP entity in Australia. It will therefore be an ‘overseas recipient’ and APP 8 will apply.

When does an APP entity ‘disclose’ personal information about an individual to an overseas recipient?

8.7 The term ‘disclose’ is not defined in the Privacy Act.

8.8 An APP entity discloses personal information where it makes it accessible to others outside the entity and releases the subsequent handling of the information from its effective control. The release of the information may be a proactive release or publication, a release in response to a specific request, an accidental release or an unauthorised release by an employee. This focuses on the act done by the disclosing party. The state of mind or intentions of the recipient does not affect the act of disclosure. Further, there will be a disclosure in these circumstances even where the information is already known to the overseas recipient.

8.9 In the context of APP 8, an APP entity will disclose personal information to an overseas recipient where it, for example:

- shares the personal information with an overseas recipient
- reveals the personal information at an international conference or meeting overseas
- sends a hard copy document or email containing an individual’s personal information t o a n overseas client
- publishes the personal information on the internet, whether intentionally or not, and it is accessible to an overseas recipient.

8.10 'Disclosure' is a separate concept from:

- 'unauthorised access' which is addressed in APP 11. An APP entity is not taken to have disclosed personal information where a third party intentionally exploits the entity's security measures and gains unauthorised access to the personal information. Examples include unauthorised access following a cyber - attack or a theft, including where the third party then makes that personal information available to others outside the entity. However, where a third party gains unauthorised access, the APP entity may breach APP 11 if it did not take reasonable steps to protect the personal information from unauthorised access (see Chapter 11 (APP 11))
- 'use'. An APP entity uses personal information where it handles, or undertakes an activity with the personal information, within the entity 's effective control . For example, where an entity provides personal information to an overseas recipient, via a server in a different overseas location , there would not usually be a disclosure until the personal information reaches the overseas recipient . That is , routing personal information , in transit, through servers located outside Australia , would usually be considered a 'use'. In limited circumstances, the provision of personal information to a contractor may also be a 'use' of that personal information (see paragraphs 8.12 – 8.15 below).

8.11 For further information about the concepts of 'use' and 'disclosure' of personal information, see Chapter B (Key concepts).

See "Chapter 11: Australian Privacy Principle 11 – Security of personal information", beginning page 168 of 211;

What does APP 11 say?

11.1 APP 11 requires an APP entity to take active measures to ensure the security of personal information it holds, and to actively consider whether it is permitted to retain personal information.

11.2 An APP entity that holds personal information must take reasonable steps to protect the information from misuse, interference and loss, as well as unauthorised access, modification or disclosure (APP 11.1) .

11.3 An APP entity must take reasonable steps to destroy or de - identify the personal information it holds once the personal information is no longer needed for any purpose for which the personal information may be used or disclosed under the APPs . This requirement does not apply where the personal information is contained in a Commonwealth record or where the entity is required by law or a court/tribunal order to retain the personal information (APP 11.2) .

'Holds'

11.4 APP 11 only applies to personal information that an APP entity holds. An entity holds personal information ' if the entity has possession or control of a record that contains the personal information' (s 6(1)) .

11.5 The term 'holds' extends beyond physical possession of a record to include a record that an APP entity has the right or power to deal with. For example, an entity that outsources the storage of personal information to a third party, but retains the right to deal with that information, including to access and amend it , holds that personal information .

11.6 The term 'holds' is discussed in more detail in Chapter B (Key concepts).

Taking reasonable steps

11.7 The 'reasonable steps' that an APP entity should take to ensure the security of personal information will depend upon circumstances that include :

- the amount and sensitivity of the personal information . More rigorous steps may be required as the quantity of personal information increases , or if the information is 'sensitive information' (defined in s 6(1) and discussed in Chapter B (Key concepts)) or other personal information of a sensitive nature
- the nature of the entity. Relevant considerations include an entity's size, resources and its business model. For example, the reasonable steps expected of an entity that operates through franchises or dealerships, or gives database and network access to contractors, may differ from the reasonable steps required of a centralised entity

PUBLIC
ThruConsoleXfer (TCXf) White Paper

- the possible adverse consequences for an individual. More rigorous steps may be required as the risk of adversity increases
- the entity's information handling practices, such as how it collects, uses and stores personal information. This includes whether personal information handling practices are outsourced to third parties, and whether those third parties are subject to the Privacy Act. If a third party is not subject to the Privacy Act, it may be reasonable for the entity to take steps to ensure the third party meets the entity's obligations under the Privacy Act, for example through specific privacy obligations in contracts and mechanisms to ensure these are being fulfilled
- the practicability, including time and cost involved. However an entity is not excused from taking particular steps to protect information by reason only that it would be inconvenient, time - consuming or impose some cost to do so. Whether these factors make it unreasonable to take particular steps will depend on whether the burden is excessive in all the circumstances
- whether a security measure is in itself privacy invasive. For example, while an APP entity should ensure that an individual is authorised to access information, it should not require an individual to supply more information than is necessary to identify themselves when dealing with the entity (see also Chapter 12 (APP 12)).

11.8 Reasonable steps could including taking steps and implementing strategies to manage the following:

- governance
- ICT security
- data breaches
- physical security
- personnel security and training
- workplace policies
- the information life cycle
- standards
- regular monitoring and review.

11.9 For further discussion of the relevant considerations, and examples of steps that may be reasonable for an APP entity to take, see the Office of the Australian Information Commissioner's Guide to information security: 'reasonable steps' to protect personal information (OAIC Information Security Guide).

What are the security considerations ?

11.10 The six terms listed in APP 11, 'misuse', 'interference', 'loss', 'unauthorised access', 'modification' and 'disclosure', are not defined in the Privacy Act. The following analysis and examples of each term draws on the ordinary meaning of the terms. As the analysis indicates, there is overlap in the meaning of the terms.

Misuse

11.11 Personal information is misused if it is used by an APP entity for a purpose that is not permitted by the Privacy Act. APP 6 sets out when an entity is permitted to use personal information (see Chapter 6). APPs 7 and 9 also contain requirements relating to an organisation's use of personal information for the purpose of direct marketing, and use of government related identifiers, respectively (see Chapters 7 and 9).

11.12 'Use' is discussed in more detail in Chapter B (Key concepts).

Interference

11.13 'Interference' with personal information occurs where there is an attack on personal information that an APP entity holds that interferes with the personal information but does not necessarily modify its content. 'Interference' includes an attack on a computer system that, for example, leads to exposure of personal information.

Loss

11.14 'Loss' of personal information covers the accidental or inadvertent loss of personal information held by an APP entity. This includes when an entity:

- physically loses personal information, such as by leaving it in a public place, or
- electronically loses personal information, such as failing to keep adequate backups of personal information in the event of a systems failure.

11.15 Loss of personal information could also potentially occur following unauthorised access or modification of the personal information. However, it does not apply to intentional destruction or de - identification of that personal information that is done in accordance with the APPs.

Unauthorised access

11.16 'Unauthorised access' of personal information occurs when personal information that an APP entity holds is accessed by someone who is not permitted to do so. This includes unauthorised access by an employee of the entity.

Unauthorised modification

11.17 'Unauthorised modification' of personal information occurs when personal information that an APP entity holds is altered by someone who is not permitted to do so, or is altered in a way that is not permitted under the Privacy Act.

Unauthorised disclosure

11.18 'Unauthorised disclosure' occurs when an APP entity releases the subsequent handling of that personal information from its effective control in a way that is not permitted under the APPs. This includes an unauthorised disclosure by an employee of the entity. The term 'disclosure' is discussed in more detail in Chapter B (Key concepts).

See "Chapter B: Key concepts", beginning page 11 of 211;

Disclosure

B.57 Disclosure is not defined in the Privacy Act.

B.58 An APP entity discloses personal information when it makes it accessible to others outside the entity and releases the subsequent handling of the personal information from its effective control. This focuses on the act done by the disclosing party. The state of mind or intentions of the recipient does not affect the act of disclosure. Further, there will be a disclosure in these circumstances even where the personal information is already known to the recipient.

B.59 The release may be a proactive release, a release in response to a specific request, an accidental release or an unauthorised release by an employee.

B.60 Examples include where an APP entity:

- shares a copy of personal information with another entity or individual
- discloses personal information to themselves, but in their capacity as a different entity
- publishes personal information whether intentionally or not

5.1.2 Implications from this Solution

An Australian organisation (APP entity) that "holds" personal information (6.6), and makes that personal information available for "use" (6.8) but not "disclosure" (6.11(b)) to an "overseas recipient" (8.5) is accountable (8.1) for ensuring that the overseas recipient does not breach the Australian Privacy Principles (8.2). Personal information is disclosed when the Australian organisation "releases the subsequent handling of the information from its effective control" (8.8), which includes "disclosure" through "unauthorised access", "if it did not take reasonable steps to protect the personal information from unauthorised access" (8.10). The "reasonable steps" include "implementing strategies to manage", amongst other things, "ICT Security", "data breaches" and "regular monitoring and review" (11.8).

Of the "six security considerations" (11.10) set out in the Australian National Privacy Principles, the technology solution presented in this paper appears to directly instantiate:

- "interference" when an overseas recipient instigates change to a computer system that "leads to exposure of personal information" (11.14), and;
- "unauthorised disclosure" when an overseas recipient "releases the subsequent handling of that personal information from its effective control".

5.2 NIST Publications

The US National Institute of Standards and Technology (part of the US Department of Commerce) has a mission to promote U.S. innovation and industrial competitiveness by advancing measurement science, standards, and technology. NIST shares its ITL publicly⁵⁹;

The Information Technology Laboratory (ITL) at the National Institute of Standards and Technology (NIST) promotes the U.S. economy and public welfare by providing technical leadership for the nation's measurement and standards infrastructure. ITL develops tests, test methods, reference data, proof of concept implementations, and technical analysis to advance the development and productive use of information technology. ITL's responsibilities include the development of technical, physical, administrative, and management standards and guidelines for the cost-effective security and privacy of sensitive unclassified information in Federal computer systems. This Special Publication 800-series reports on ITL's research, guidance, and outreach efforts in computer security and its collaborative activities with industry, government, and academic organizations.

This section looks at the relevant aspects of the following Special Publications from the 800 series;

- Special Publication 800-46 Revision 1 (June 2009) – Guide to Enterprise Telework and Remote Access Security⁶⁰
- Special Publication 800-114 (November 2007) – User's Guide to Securing External Devices for Telework and Remote Access⁶¹
- Special Publication 800-53 Revision 4 (April 2013, updates 15 January 2014) - Security and Privacy Controls for Federal Information Systems and Organizations⁶²

5.2.1 Special Publication 800-46 Revision 1, Guide to Enterprise Telework and Remote Access Security

See “Executive Summary”, beginning page 7 of 46;

All the components of telework and remote access solutions, including client devices, remote access servers, and internal resources accessed through remote access, should be secured against expected threats, as identified through threat models. Major security concerns include the lack of physical security controls, the use of unsecured networks, the connection of infected devices to internal networks, and the availability of internal resources to external hosts. This publication provides information on security considerations for several types of remote access solutions, and it makes recommendations for securing a variety of telework and remote access technologies. It also gives advice on creating telework security policies.

To improve the security of their telework and remote access technologies, organizations should implement the following recommendations:

Plan telework security policies and controls based on the assumption that external environments contain hostile threats.

An organization should assume that external facilities, networks, and devices contain hostile threats that will attempt to gain access to the organization's data and resources. Organizations should assume that telework client devices, which are used in a variety of

59 <http://csrc.nist.gov/publications/PubsSPs.html>

60 <http://csrc.nist.gov/publications/nistpubs/800-46-rev1/sp800-46r1.pdf>

61 <http://csrc.nist.gov/publications/nistpubs/800-114/SP800-114.pdf>

62 <http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r4.pdf>

PUBLIC
ThruConsoleXfer (TCXf) White Paper

external locations and are particularly prone to loss or theft, will be acquired by malicious parties who will attempt to recover sensitive data from them. Options for mitigating this type of threat include encrypting the device's storage and not storing sensitive data on client devices. Organizations should also assume that communications on external networks, which are outside the organization's control, are susceptible to eavesdropping, interception, and modification. This type of threat can be mitigated, but not eliminated, by using encryption technologies to protect the confidentiality and integrity of communications, as well as authenticating each of the endpoints to each other to verify their identities. Another important assumption is that telework client devices will become infected with malware; possible controls for this include using anti-malware technologies, using network access control solutions that verify the client's security posture before granting access, and using a separate network at the organization's facilities for telework client devices brought in for internal use.

See "Vulnerabilities, Threats, and Security Controls", beginning page 12 of 46;

Threat modeling involves identifying resources of interest and the feasible threats, vulnerabilities, and security controls related to these resources, then quantifying the likelihood of successful attacks and their impacts, and finally analyzing this information to determine where security controls need to be improved or added. Threat modeling helps organizations to identify security requirements and to design the remote access solution to incorporate the controls needed to meet the security requirements. Major security concerns for these technologies that would be included in most telework threat models are as follows:

- **Lack of Physical Security Controls.** Telework client devices are used in a variety of locations outside the organization's control, such as employees' homes, coffee shops, hotels, and conferences. The mobile nature of these devices makes them likely to be lost or stolen, which places the data on the devices at increased risk of compromise. When planning telework security policies and controls, organizations should assume that client devices will be acquired by malicious parties who will attempt to recover sensitive data from the devices. The primary mitigation strategies are either to encrypt the client device's storage so that sensitive data cannot be recovered from it by unauthorized parties, or to not store sensitive data on client devices. Even if a client device is always in the possession of its owner, there are other physical security risks, such as an attacker looking over a teleworker's shoulder at a coffee shop and viewing sensitive data on the client device's screen.

[...]

- **External Access to Internal Resources.** Remote access provides external hosts with access to internal resources, such as servers. If these internal resources were not previously accessible from external networks, making them available via remote access will expose them to new threats, particularly from untrusted client devices and networks, and significantly increase the likelihood that they will be compromised. Each form of remote access that can be used to access an internal resource increases the risk of that resource being compromised. Organizations should carefully consider the balance between the benefits of providing remote access to additional resources and the potential impact of a compromise of those resources. Organizations should ensure that any internal resources they choose to make available through remote access are hardened appropriately against external threats and that access to the resources is limited to the minimum necessary through firewalling and other access control mechanisms.

See "Telework Client Device Security", beginning page 29 of 46;

In today's computing environment, there are many threats to telework client devices. These threats are posed by people with many different motivations, including causing mischief and disruption, and committing identity theft and other forms of fraud. The primary threat against most telework client devices is malware, including viruses, worms, malicious mobile code, Trojan horses, rootkits, and spyware. Malware threats can infect client devices through many means, including email, Web sites, file downloads and file sharing, peer-to-peer software, and instant messaging. The use of unauthorized removable media, such as flash drives, is an increasingly common transmission mechanism for malware. Another common threat against telework client devices is loss or theft of the device. Someone with physical access to a device has many options for attempting to view or copy the information stored on it. An attacker with physical access can also add malware to a device that gives them access to data accessed from or entered into the device, such as users' passwords typed into a laptop keyboard.

Permitting teleworkers to remotely access an organization's computing resources gives

attackers additional opportunities to breach the organization's security. When a client device uses remote access, it is essentially an extension of the organization's own network. If the device is not secured properly, it poses additional risk not only to the information that the teleworker accesses, but also to the organization's other systems and networks. Therefore, telework client devices should be secured properly and have their security maintained regularly.

5.2.2 Special Publication 800-114, User's Guide to Securing External Devices for Telework and Remote Access

See "Securing Information", beginning page 16 of 46;

- **Using physical security controls** for telework devices and removable media. For example, an organization might require that laptops be physically secured using cable locks when used in hotels, conferences, and other locations where third parties could easily gain physical access to the devices. Organizations may also have physical security requirements for papers and other non-computer media that contain sensitive information and are taken outside the organization's facilities.

[...]

Each situation may require a different combination of protection options: for example, an organization might require one combination for IPsec VPN access from teleworker-owned computers and another combination for access to an individual application from third party-secured devices. Teleworkers should follow their organizations' requirements and recommendations for protecting sensitive information accessed with telework devices. Some organizations use the same requirements and recommendations for all types of information because of the difficulties in differentiating sensitive and nonsensitive information.

See "Protect User Sessions from Unauthorized Physical Access", beginning page 24 of 46;

5.2.3 Protect User Sessions from Unauthorized Physical Access

It is important that user sessions be protected against unauthorized physical access. For example, if a PC is sitting unattended in an area that other people can access, anyone could walk up to the PC and masquerade as the user, such as sending email from the user's account, accessing the organization's remote access resources, making purchases from Web sites, or accessing sensitive information stored on the PC. To prevent such events, most OSs allow the user to lock the current session through menu options or a combination of keystrokes. Also, many OSs offer screensavers that activate automatically after the PC has been idle for a certain number of minutes, and can also be activated manually by the user on demand. Some of these screensavers can be configured to lock the PC and require the user to enter his or her password to unlock it. If a PC will be left unattended in an accessible area at any time, users should use a password-protected screensaver or manually lock their user sessions. However, users should be aware that these security features provide only short-term protection; someone who has access to the PC for an extended period of time can bypass these features and gain access to the user's session and data.

See "Securing Telework Devices", beginning page 35 of 46;

- **Limit access to the device.** Most consumer devices allow the owner to restrict access by setting a PIN or password; some also support more sophisticated authentication mechanisms, such as biometrics (e.g., the owner's thumbprint). Using some sort of authenticator prevents or deters access to the teleworker's information and service by a person who gains unauthorized physical access to the device. Some devices can also be configured to lock themselves automatically after an idle period; a person attempting to use the device when it is locked must authenticate again to unlock it. 24 PINs and passwords should be changed periodically and whenever teleworkers suspect that someone else may know them.

5.2.3 Special Special Publication 800-53 Revision 4, Security and Privacy Controls for Federal Information Systems and Organizations

See “External Service Providers”, beginning page 39 of 460;

2.5 EXTERNAL SERVICE PROVIDERS

Organizations are becoming increasingly reliant on information system services provided by external providers to conduct important missions and business functions. External information system services are computing and information technology services implemented outside of the traditional security authorization boundaries established by organizations for their information systems. Those traditional authorization boundaries linked to physical space and control of assets, are being extended (both physically and logically) with the growing use of external services. In this context, external services can be provided by: (i) entities within the organization but outside of the security authorization boundaries established for organizational information systems; (ii) entities outside of the organization either in the public sector (e.g., federal agencies) or private sector (e.g., commercial service providers); or (iii) some combination of the public and private sector options. External information system services include, for example, the use of service-oriented architectures (SOAs), cloud-based services (infrastructure, platform, software), or data center operations. External information system services may be used by, but are typically not part of, organizational information systems. In some situations, external information system services may completely replace or heavily augment the routine functionality of internal organizational information systems.

FISMA and OMB policies require that federal agencies using external service providers to process, store, or transmit federal information or operate information systems on behalf of the federal government, assure that such use meets the same security requirements that federal agencies are required to meet. Security requirements for external service providers including the security controls for external information systems are expressed in contracts or other formal agreements. Organizations are responsible and accountable for the information security risk incurred by the use of information system services provided by external providers. Such risk is addressed by incorporating the Risk Management Framework (RMF) as part of the terms and conditions of the contracts with external providers. Organizations can require external providers to implement all steps in the RMF except the security authorization step, which remains an inherent federal responsibility directly linked to managing the information security risk related to the use of external information system services. Organizations can also require external providers to provide appropriate evidence to demonstrate that they have complied with the RMF in protecting federal information. However, federal agencies take direct responsibility for the overall security of such services by authorizing the information systems providing the services.

Relationships with external service providers are established in a variety of ways, for example, through joint ventures, business partnerships, outsourcing arrangements (i.e., through contracts, interagency agreements, lines of business arrangements, service-level agreements), licensing agreements, and/or supply chain exchanges. The growing use of external service providers and new relationships being forged with those providers present new and difficult challenges for organizations, especially in the area of information system security. These challenges include:

- Defining the types of external information system services provided to organizations;
- Describing how those external services are protected in accordance with the information security requirements of organizations; and
- Obtaining the necessary assurances that the risk to organizational operations and assets, individuals, other organizations, and the Nation arising from the use of the external services is acceptable.

The degree of confidence that the risk from using external services is at an acceptable level depends on the trust that organizations place in external service providers. In some cases, the level of trust is based on the amount of direct control organizations are able to exert on external service providers with regard to employment of security controls necessary for the protection of the service/information and the evidence brought forth as to the effectiveness of those controls.

The level of control is usually established by the terms and conditions of the contracts or service-level agreements with the external service providers and can range from extensive control (e.g., negotiating contracts or agreements that specify detailed security requirements for the providers) to very limited control (e.g., using contracts or service-level agreements to obtain commodity services such as commercial telecommunications services). In other cases, levels of trust are based on factors that convince organizations

PUBLIC
ThruConsoleXfer (TCXf) White Paper

that required security controls have been employed and that determinations of control effectiveness exist. For example, separately authorized external information system services provided to organizations through well-established lines of business relationships may provide degrees of trust in such services within the tolerable risk range of the authorizing officials and organizations using the services.

The provision of services by external providers may result in certain services without explicit agreements between organizations and the providers. Whenever explicit agreements are feasible and practical (e.g., through contracts, service-level agreements), organizations develop such agreements and require the use of the security controls in Appendix F of this publication. When organizations are not in a position to require explicit agreements with external service providers (e.g., services are imposed on organizations, services are commodity services), organizations establish and document explicit assumptions about service capabilities with regard to security. In situations where organizations are procuring information system services through centralized acquisition vehicles (e.g., governmentwide contracts by the General Services Administration or other preferred and/or mandatory acquisition organizations), it may be more efficient and cost-effective for contract originators to establish and maintain stated levels of trust with external service providers (including the definition of required security controls and level of assurance with regard to the provision of such controls). Organizations subsequently acquiring information system services from centralized contracts can take advantage of the negotiated levels of trust established by the procurement originators and thus avoid costly repetition of activities necessary to establish such trust. Centralized acquisition vehicles (e.g., contracts) may also require the active participation of organizations. For example, organizations may be required by provisions in contracts or agreements to install public key encryption-enabled client software recommended by external service providers.

Ultimately, the responsibility for adequately mitigating unacceptable risks arising from the use of external information system services remains with authorizing officials. Organizations require that appropriate chains of trust be established with external service providers when dealing with the many issues associated with information system security. Organizations establish and retain a level of trust that participating service providers in the potentially complex consumer-provider relationship provide adequate protection for the services rendered to organizations. The chain of trust can be complicated due to the number of entities participating in the consumer-provider relationship and the types of relationships between the parties. External service providers may also outsource selected services to other external entities, making the chain of trust more difficult and complicated to manage. Depending on the nature of the services, organizations may find it impossible to place significant trust in external providers. This situation is due not to any inherent untrustworthiness on the part of providers, but to the intrinsic level of risk in the services.

Where a sufficient level of trust cannot be established in the external services and/or providers, organizations can: (i) mitigate the risk by employing compensating controls; (ii) accept the risk within the level of organizational risk tolerance; (iii) transfer risk by obtaining insurance to cover potential losses; or (iv) avoid risk by choosing not to obtain the services from certain providers (resulting in performance of missions/business operations with reduced levels of functionality or possibly no functionality at all). For example, in the case of cloud-based information systems and/or services, organizations might require as a compensating control, that all information stored in the cloud be encrypted for added security of the information. Alternatively, organizations may require encrypting some of the information stored in the cloud (depending on the criticality or sensitivity of such information)—accepting additional risk but limiting the risk of not storing all information in an unencrypted form.

2.6 ASSURANCE AND TRUSTWORTHINESS

Assurance and trustworthiness of information systems, system components, and information system services are becoming an increasingly important part of the risk management strategies developed by organizations. Whether information systems are deployed to support, for example, the operations of the national air traffic control system, a major financial institution, a nuclear power plant providing electricity for a large city, or the military services and war fighters, the systems must be reliable, trustworthy, and resilient in the face of increasingly sophisticated and pervasive threats. To understand how organizations achieve trustworthy systems and the role assurance plays in the trustworthiness factor, it is important to first define the term trust. Trust, in general, is the belief that an entity will behave in a predictable manner while performing specific functions, in specific environments, and under specified conditions or circumstances. The entity may be a person, process, information system, system component, system-of-systems, or any combination thereof.

From an information security perspective, trust is the belief that a security-relevant

PUBLIC
ThruConsoleXfer (TCXf) White Paper

entity will behave in a predictable manner when satisfying a defined set of security requirements under specified conditions/circumstances and while subjected to disruptions, human errors, component faults and failures, and purposeful attacks that may occur in the environment of operation. Trust is usually determined relative to a specific security capability and can be decided relative to an individual system component or the entire information system. However, trust at the information system level is not achieved as a result of composing a security capability from a set of trusted system components—rather, trust at the system level is an inherently subjective determination that is derived from the complex interactions among entities (i.e., technical components, physical components, and individuals), taking into account the life cycle activities that govern, develop, operate, and sustain the system. In essence, to have trust in a security capability requires that there is a sufficient basis for trust, or trustworthiness, in the set of security-relevant entities that are to be composed to provide such capability.

Trustworthiness with respect to information systems, expresses the degree to which the systems can be expected to preserve with some degree of confidence, the confidentiality, integrity, and availability of the information that is being processed, stored, or transmitted by the systems across a range of threats. Trustworthy information systems are systems that are believed to be capable of operating within a defined risk tolerance despite the environmental disruptions, human errors, structural failures, and purposeful attacks that are expected to occur in the environments in which the systems operate—systems that have the trustworthiness to successfully carry out assigned missions/business functions under conditions of stress and uncertainty.

Two fundamental components affecting the trustworthiness of information systems are security functionality and security assurance. Security functionality is typically defined in terms of the security features, functions, mechanisms, services, procedures, and architectures implemented within organizational information systems or the environments in which those systems operate. Security assurance is the measure of confidence that the security functionality is implemented correctly, operating as intended, and producing the desired outcome with respect to meeting the security requirements for the system—thus possessing the capability to accurately mediate and enforce established security policies. Security controls address both security functionality and security assurance. Some controls focus primarily on security functionality (e.g., PE-3, Physical Access Control; IA-2, Identification and Authentication; SC-13, Cryptographic Protection; AC-2, Account Management). Other controls focus primarily on security assurance (e.g., CA-2, Security Assessment; SA-17, Developer Security Architecture and Design; CM-3, Configuration Change Control). Finally, certain security controls can support security functionality and assurance (e.g., RA-5, Vulnerability Scanning; SC-3, Security Function Isolation; AC-25, Reference Monitor). Security controls related to functionality are combined to develop a security capability with the assurance-related controls implemented to provide a degree of confidence in the capability within the organizational risk tolerance.

See “Covert Storage Channel”, beginning page 83 of 460;

Covert Storage Channel [CNSSI 4009]	Covert channel involving the direct or indirect writing to a storage location by one process and the direct or indirect reading of the storage location by another process. Covert storage channels typically involve a finite resource (e.g., sectors on a disk) that is shared by two subjects at different security levels.
[...]	
Exfiltration	The unauthorized transfer of information from an information system.

See “Remote Access”, beginning page 184 of 460;

AC-17 REMOTE ACCESS

Control: The organization:

- a. Establishes and documents usage restrictions, configuration/connection requirements, and implementation guidance for each type of remote access allowed; and
- b. Authorizes remote access to the information system prior to allowing such connections.

Supplemental Guidance: Remote access is access to organizational information systems by users (or processes acting on behalf of users) communicating through external networks (e.g., the Internet). Remote access methods include, for example, dial-up, broadband, and wireless. Organizations often employ encrypted virtual private networks (VPNs) to enhance

PUBLIC
ThruConsoleXfer (TCXf) White Paper

confidentiality and integrity over remote connections. The use of encrypted VPNs does not make the access non-remote; however, the use of VPNs, when adequately provisioned with appropriate security controls (e.g., employing appropriate encryption techniques for confidentiality and integrity protection) may provide sufficient assurance to the organization that it can effectively treat such connections as internal networks. Still, VPN connections traverse external networks, and the encrypted VPN does not enhance the availability of remote connections. Also, VPNs with encrypted tunnels can affect the organizational capability to adequately monitor network communications traffic for malicious code. Remote access controls apply to information systems other than public web servers or systems designed for public access. This control addresses authorization prior to allowing remote access without specifying the formats for such authorization. While organizations may use interconnection security agreements to authorize remote access connections, such agreements are not required by this control. Enforcing access restrictions for remote connections is addressed in AC-3. Related controls: AC-2, AC-3, AC-18, AC-19, AC-20, CA-3, CA-7, CM-8, IA-2, IA-3, IA-8, MA-4, PE-17, PL-4, SC-10, SI-4.

Control Enhancements: The organization:

(1) REMOTE ACCESS | AUTOMATED MONITORING / CONTROL

The information system monitors and controls remote access methods.

Supplemental Guidance: Automated monitoring and control of remote access sessions allows organizations to detect cyber attacks and also ensure ongoing compliance with remote access policies by auditing connection activities of remote users on a variety of information system components (e.g., servers, workstations, notebook computers, smart phones, and tablets). Related controls: AU-2, AU-12.

(2) REMOTE ACCESS | PROTECTION OF CONFIDENTIALITY / INTEGRITY USING ENCRYPTION

The information system implements cryptographic mechanisms to protect the confidentiality and integrity of remote access sessions.

Supplemental Guidance: The encryption strength of mechanism is selected based on the security categorization of the information. Related controls: SC-8, SC-12, SC-13.

(3) REMOTE ACCESS | MANAGED ACCESS CONTROL POINTS

The information system routes all remote accesses through [Assignment: organization-defined number] managed network access control points.

Supplemental Guidance: Limiting the number of access control points for remote accesses reduces the attack surface for organizations. Organizations consider the Trusted Internet Connections (TIC) initiative requirements for external network connections. Related control: SC-7.

(4) REMOTE ACCESS | PRIVILEGED COMMANDS / ACCESS

The organization:

(a) Authorizes the execution of privileged commands and access to security-relevant information via remote access only for [Assignment: organization-defined needs]; and

(b) Documents the rationale for such access in the security plan for the information system.

Supplemental Guidance: Related control: AC-6.

(5) REMOTE ACCESS | MONITORING FOR UNAUTHORIZED CONNECTIONS

[Withdrawn: Incorporated into SI-4].

(6) REMOTE ACCESS | PROTECTION OF INFORMATION

The organization ensures that users protect information about remote access mechanisms from unauthorized use and disclosure.

Supplemental Guidance: Related controls: AT-2, AT-3, PS-6.

(7) REMOTE ACCESS | ADDITIONAL PROTECTION FOR SECURITY FUNCTION ACCESS

[Withdrawn: Incorporated into AC-3 (10)].

(8) REMOTE ACCESS | DISABLE NONSECURE NETWORK PROTOCOLS

[Withdrawn: Incorporated into CM-7].

(9) REMOTE ACCESS | DISCONNECT / DISABLE ACCESS

The organization provides the capability to expeditiously disconnect or disable remote access to the information system within [Assignment: organization-defined time period].

Supplemental Guidance: This control enhancement requires organizations to have the capability to rapidly disconnect current users remotely accessing the information system and/or disable further remote access. The speed of disconnect or disablement varies based on the criticality of missions/business functions and the need to eliminate immediate or future remote access to organizational information systems.

References: NIST Special Publications 800-46, 800-77, 800-113, 800-114, 800-121.

See “Covert Channel Analysis”, beginning page 362 of 460;

SC-31 COVERT CHANNEL ANALYSIS

Control: The organization:

- a. Performs a covert channel analysis to identify those aspects of communications within the information system that are potential avenues for covert [Selection (one or more): storage; timing] channels; and
- b. Estimates the maximum bandwidth of those channels.

Supplemental Guidance: Developers are in the best position to identify potential areas within systems that might lead to covert channels. Covert channel analysis is a meaningful activity when there is the potential for unauthorized information flows across security domains, for example, in the case of information systems containing export-controlled information and having connections to external networks (i.e., networks not controlled by organizations). Covert channel analysis is also meaningful for multilevel secure (MLS) information systems, multiple security level (MSL) systems, and cross-domain systems. Related controls: AC-3, AC-4, PL-2.

Control Enhancements:

- (1) COVERT CHANNEL ANALYSIS | TEST COVERT CHANNELS FOR EXPLOITABILITY

The organization tests a subset of the identified covert channels to determine which channels are exploitable.

- (2) COVERT CHANNEL ANALYSIS | MAXIMUM BANDWIDTH

The organization reduces the maximum bandwidth for identified covert [Selection (one or more); storage; timing] channels to [Assignment: organization-defined values].

Supplemental Guidance: Information system developers are in the best position to reduce the maximum bandwidth for identified covert storage and timing channels.

- (3) COVERT CHANNEL ANALYSIS | MEASURE BANDWIDTH IN OPERATIONAL ENVIRONMENTS

The organization measures the bandwidth of [Assignment: organization-defined subset of identified covert channels] in the operational environment of the information system.

Supplemental Guidance: This control enhancement addresses covert channel bandwidth in operational environments versus developmental environments. Measuring covert channel bandwidth in operational environments helps organizations to determine how much information can be covertly leaked before such leakage adversely affects organizational missions/business functions. Covert channel bandwidth may be significantly different when measured in those settings that are independent of the particular environments of operation (e.g., laboratories or development environments).

References: None.

See “Port and I/O Device Access”, beginning page 368 of 460;

SC-41 PORT AND I/O DEVICE ACCESS

Control: The organization physically disables or removes [Assignment: organization-defined connection ports or input/output devices] on [Assignment: organization-defined information systems or information system components].

Supplemental Guidance: Connection ports include, for example, Universal Serial Bus (USB) and Firewire (IEEE 1394). Input/output (I/O) devices include, for example, Compact Disk (CD) and Digital Video Disk (DVD) drives. Physically disabling or removing such connection ports and I/O devices helps prevent exfiltration of information from information systems and the introduction of malicious code into systems from those ports/devices.

Control Enhancements: None.

References: None.

See “Accounting of Disclosures”, beginning page 445 of 460;

AR-8 ACCOUNTING OF DISCLOSURES

Control: The organization:

- a. Keeps an accurate accounting of disclosures of information held in each system of records under its control, including:

PUBLIC
ThruConsoleXfer (TCXf) White Paper

- (1) Date, nature, and purpose of each disclosure of a record; and
- (2) Name and address of the person or agency to which the disclosure was made;
- b. Retains the accounting of disclosures for the life of the record or five years after the disclosure is made, whichever is longer; and
- c. Makes the accounting of disclosures available to the person named in the record upon request.

Supplemental Guidance: The Senior Agency Official for Privacy (SAOP)/Chief Privacy Officer (CPO) periodically consults with managers of organization systems of record to ensure that the required accountings of disclosures of records are being properly maintained and provided to persons named in those records consistent with the dictates of the Privacy Act. Organizations are not required to keep an accounting of disclosures when the disclosures are made to individuals with a need to know, are made pursuant to the Freedom of Information Act, or are made to a law enforcement agency pursuant to 5 U.S.C. § 552a(c)(3). Heads of agencies can promulgate rules to exempt certain systems of records from the requirement to provide the accounting of disclosures to individuals. Related control: IP-2.

Control Enhancements: None.

References: The Privacy Act of 1974, 5 U.S.C. § 552a (c)(1), (c)(3), (j), (k).

Additional references that are relevant include:

- PE-3 “Physical Access Control”
 - See control enhancement 2; PHYSICAL ACCESS CONTROL | FACILITY / INFORMATION SYSTEM BOUNDARIES
- SC-7 “Boundary Protection”
 - See control enhancement 7; BOUNDARY PROTECTION | PREVENT SPLIT TUNNELING FOR REMOTE DEVICES
 - See control enhancement 10; BOUNDARY PROTECTION | PREVENT UNAUTHORIZED EXFILTRATION – particularly;
 - (ii) monitoring for beaconing from information systems;
 - (iii) monitoring for steganography;
 - (vi) employing traffic profile analysis to detect deviations from the volume/types of traffic expected within organizations or call backs to command and control centers.

5.2.4 Implications from this Solution

There are two use cases that will affect the implications from this solution.

The first is for those who fall within the scope of the NIST Special Publications – i.e. *The security controls in NIST Special Publication 800-53 are designed to facilitate compliance with applicable (US) federal laws, Executive Orders, directives, policies, regulations, standards, and guidance.* In this first case, (US) federal agencies that are off-shoring (engaging external service providers) take direct responsibility for the overall security of the service. The questions that these agencies will need to resolve will revolve around the practical inability to assert security assurance (suggesting that the trustworthiness of external service providers would be limited) and their ability to account for disclosure (AR-8) – amongst any number of confidentiality imposts (i.e. from Controlled Unclassified Information, Medium/High Impact Systems, etc). However, the controls framework (set out in 800-53) suggests that there are at least controls (such as PE-3, SC-7, SC-31 and SC-41) that provide a mechanism to articulate and quantify the implications from this solution, in those cases, though these controls seem to be tuned to timing-based covert channels (even the definition of *covert storage channel* involves “two subjects” and there is no definition or use of *infiltration*).

The second use case is for those organisations consuming the NIST standards as a suite of “good practice” in the information security domain, but without the mandate to do so. In this case, cherry-picking the special publications for remote access (800-46 and 800-114) will not be effective in mitigating the solution described in this paper. Despite an assumption of hostile environments and physical security considerations, covert channels are neither mentioned nor mitigated in these documents.

6 Other

For completeness I have provided additional information below.

6.1 Why did you publish this?

I have sat on this problem for the last 15 years assuming that the industry would clean it up (I thought Deep Packet Inspection would get there first), but I'm so sick of explaining the vulnerability to unimaginative security personnel who ignore the potential impacts because it doesn't have a CVE number attached to it, that this year I wrote the specification and the code and even paid for an app developer to package the application for the iPhone and Android platforms, in the hope that we can mature the industry and move on to the real discussion of how not to suck at enterprise security architecture and its implementation.

6.2 But why now?

The release of the revised Australian National Privacy Principles suggested that the Australian Government was taking a genuine interest in, amongst other things, the Information Technology impact on sovereignty, despite being constantly berated^{63 64 65} by off-shore interests for taking a protective stance on citizens' personal and private data. The problem is the line of demarcation between “use” and “disclosure” suggesting that although the problem has been taken seriously, there is a misunderstanding in the meaning of, or the premise behind, “effective control” and “reasonable steps”. This paper was written to provide not only a theoretical view of the vulnerability that comes from this misunderstanding, but provides a practical implementation that can be tested by security personnel that you (or in this case the Australian Government) trusts.

I was also fortunate to have my employer sponsor my attendance at the SABSA Foundation course, presented by David Lynas himself, late 2013. In the first day of that course David discusses the problems with multi-jurisdictional and multi-cultural policy such as preventing the viewing of pornographic content, depending on how offensive, culturally insensitive, or illegal it is perceived, globally in an enterprise organisation. In Australia this is done for OH&S reasons, where a safe workplace means that you should not fear seeing offensive content – though the degree to which this is constrained is necessarily set out in corporate policy and aligned to how offensive the content is⁶⁶. By contrast, other countries consider the viewing pornographic content during an employee lunch break a right, whilst yet other countries the act could cost you your liberty entirely. David argues that it therefore doesn't make sense to filter content at the network layer – it's not offensive to have pornography traversing the wires in your network – if your policy objective is to prevent the display of content in your jurisdiction, then it should be filtered at the screen – the point of display in that jurisdiction.

63 <http://delimiter.com.au/2012/04/13/us-slams-australias-on-shore-cloud-fixation/>

64 <http://www.itnews.com.au/News/374016,microsoft-criticises-australias-offshore-cloud-policy.aspx>

65 <http://www.itnews.com.au/News/374668,servicenow-speaks-out-on-government-cloud-policy.aspx>

66 <http://nationalworkplacelawyers.com.au/8/News/view/130/Employers-hard-line-policy-on-porn-does-not-always-justify-dismissal>

Although this is a damn good example of enterprise policy architecture, I believed that I could present a much stronger argument for effective controls deployment and a better example of the threat from an unfiltered user-controlled display.

6.3 About Midnight Code

Midnight Code⁶⁷ 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.

⁶⁷ <http://midnightcode.org/>

7 Licensing

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

7.1 Midnight Code Trademark

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

7.2 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>

*/

7.3 The Reference Code

The reference code (included in this document and on the information web site) is released under the GNU GENERAL PUBLIC LICENSE Version 2, June 1991.

PUBLIC
ThruConsoleXfer (TCXf) White Paper

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

PUBLIC
ThruConsoleXfer (TCXf) White Paper

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,

PUBLIC
ThruConsoleXfer (TCXf) White Paper

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.

PUBLIC
ThruConsoleXfer (TCXf) White Paper

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

PUBLIC
ThruConsoleXfer (TCXf) White Paper

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:

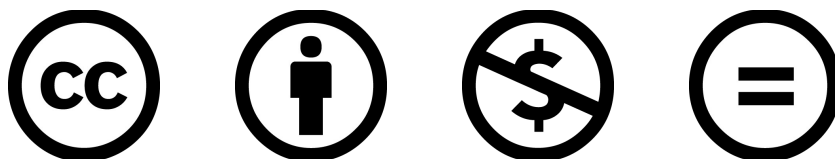
```
Yoyodyne, Inc., hereby disclaims all copyright interest in the program  
'Gnomovision' (which makes passes at compilers) written by James Hacker.
```

```
<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.

7.4 This Document

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



7.5 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;

ANSI	http://ascii-table.com/ansi-escape-sequences.php
ANSI VT100	http://ascii-table.com/ansi-escape-sequences-vt-100.php
Arduino IDE	http://arduino.cc/
CRC32	http://tools.ietf.org/html/rfc1952#section-8
GD Library	http://libgd.bitbucket.org/
Gif Creator	https://github.com/Sybio/GifCreator
libqrencode	http://fukuchi.org/works/qrencode/
PHP QRcode	http://phpqrcode.sourceforge.net/
Zbar	http://zbar.sourceforge.net/