IBM 1401 Tape Channel Analyzer Specification Version 1.3

Bob Feretich

Change History:

Version 1.3:

Major changes made in this version.

- Added specifics on the Windows device drivers.
- Updates to reflect the use of a web application to implement the user interface.
- Updated the description of tape operations with information we discovered during debug.

Version 1.2:

Major changes made in this version.

- Updated the description of tape operations with information we discovered during debug.

Version 1.1:

Major changes made in this version.

- Corrected names of Analyzer_configuration and Analyzer_state structures.
- Changed the low priority interrupt handler state machine to wait at the appropriate spots for PC data.
- Moved the TWRITEPULSE signal to Port B bit 3.
- Changed CE Panel description to change Read Bus Gain Levels only between records and to make LRC generation the default.
- Changed use of Timer4 to also perform the Write write-read gap delay.

Version 1.0:

Baseline version.

IBM 1401 Tape Channel Analyzer

1 Overview of Major Components

The components of the IBM 1401 to PC Interface Adapter are:

- The Adapter (the USB peripheral controller module, supporting logic, and firmware). The lowest level of the design.
- The Windows drivers that enables the above and below items to communicate. The kernel drivers consist of tapeusb.sys and generic.sys. The user mode driver is TapeUsbDll.dll.
- The Adapter Web Application. The highest level of the design.

The design strategy for this subsystem is to focus on simplicity and to push required complexity to a high of a level as possible. Since this is a one of a kind design, unit materials cost is secondary to design complexity cost.

1.1 The Adapter

The Adapter consists of:

- a USB peripheral chip
- a microcontroller (maybe included in the USB module) and its firmware
- a data buffer large enough to hold an entire tape record (max record size is 2000 characters)
- microcontroller parallel ports that map to the tape channel signals to be controlled and monitored
- voltage level translation circuits
- support circuitry for to make the above work

This is the lowest level of design. All interface functionality that requires "microsecond" response times needs to be handled at this level. This means that "Tape Read" and "Tape Write" character data transfers must occur within this level. Also, the Adapter can hold only one record of tape data (<=2000 characters). Due to relative difficulties in observing and controlling the behavior of the Adapter, the simplicity of this level is the highest priority.

The Adapter has two major functions, that of a TAU bring-up tool and a multiple tape drive emulator. The bring-up tool function is described under the heading "Tape Channel Analyzer". When performing this function, Adapter monitors the tape channel, recognizes specific activity on the tape channel for which it has been configured to respond, and respond to the activity as instructed. The response could be stimulation of the tape drive and/or recording data from the channel to be uploaded to the PC for display. For its emulation function, the Adapter shall emulate the relevant functionality of a bank of IBM 729 Model II of V tape drives. Data written to the emulated tape drives is uploaded to the PC. Data to be read from emulated tape drives is downloaded from the PC.

The 1401 assumes it is dealing with a real tape drive. Every time it reads or writes to a real tape drive it must wait several milliseconds for the tape to accelerate to speed, perform the operation, and wait several milliseconds for the tape to decelerate to a stop. These acceleration/deceleration time intervals are used to transfer records between the

Adapter and the PC. On tape read operations, the Adapter will request and receive a record from the PC via the USB bus during the tape acceleration time interval and then feed it to the 1401 a character at a time. On tape write operations, the Adapter will assemble a record from characters received from the 1401 and then send the record to the PC via the USB bus during the tape deceleration time interval.

The above architecture requires the Adapter to retain only a few bytes of tape drive state information per drive. Because of this minimal amount of state information and because the 1401 can communicate with only one tape drive at a time, the Adapter can emulate a full bank of six 729 tape drives.

1.2 Windows Drivers

The drivers handle the interface between the Adapter and the Adapter Web Application Program. Ideally, the Adapter registers with Windows as a custom USB device. The device's vendor ID and product ID are used to have Windows load our kernel driver. The kernel driver performs most of the millisecond level tape drive functions.

The kernel driver creates a memory buffer for each tape being emulated. These memory buffers are locked in memory (non-pageable) for the entire time the virtual tape is mounted on an emulated tape drive. The typical size of a tape buffer is about 20 MB. The emulation of 6 drives would then dedicate and lock 120 MB of RAM memory. Although this is a substantial amount of system resources, modern computers with gigabytes of RAM can support this requirement.

The tapeusb.sys kernel driver responds to Adapter requests and transfers records to and from these locked buffers. Note that the tape acceleration/deceleration time intervals are not long enough to move data to or from disk. They are even to short to depend upon the Windows task scheduler. To reliably satisfy the Adapter's response time requirements, the kernel driver performs its communications and tape record handling actions in interrupt service routines (ISRs) running at the IRQ2 hardware interrupt level.

The tapeusb.sys kernel driver uses the Windows Driver Model (WDM) architecture. The generic.sys driver module provides the boilerplate WDM functions. The generic.sys driver module is copied from the CD provided with the Walter Oney book, "Programming the Microsoft Windows Diver Model - Second Edition". This permits the tapeusb.sys driver module to focus on tape drive emulation and ignore most of the plug-n-play, registry, and power management driver functions.

The TapeUsbDll.dll driver is a user mode driver. It moves data between the tape buffers and the disk file system. Data is moved from disk to tape buffer when a virtual tape is mounted on an emulated tape drive. Data is moved from a tape buffer to a disk file when the virtual tape is removed from the emulated tape drive. The dll also provides an interface for calls from the Java web application and a means whereby the Java web application can peek into the kernel driver and monitor tape drive activity.

1.3 Adapter Web Application

In order to permit users to use their own notebook computers to control tape drive emulation, it was decided to have users control the Emulator by means of standard web pages and their browser. No special software need be loaded onto the user's computer. The web application is tested using the latest versions of both the Firefox and Internet Explorer browsers.

The components of the Web Application consists of:

- The Apache Tomcat 6 webserver - to provide the servlet containers
- A set of Java servlets , JSP pages, and Java Beans that implement the Model-View-Controller (MVC) architecture web application
- JavaScript extended with Ajax to provide dynamic communication between the webserver and web pages.

Apache Tomcat is a free open source webserver that is written in Java and is designed to run Java web applications efficiently. It can also serve standard html web pages. The servlets, JSP pages, Java Beans, and JavaScript combine to form the Graphical User Interface (GUI) to the Tape Channel Analyzer.

The Tape Channel Analyzer GUI should provide two types of functionality; a Tape Channel Diagnostic Panel and the Virtual Tape Drive Operator Console set. The a Tape Channel Diagnostic Panel consists of indicators and switches used to configure the bring-up and debug functions of the Adapter. Through this interface a user can monitor tape channel activity, control the Adapter's data generator, and force specific types of errors.

The Virtual Tape Drive Console GUI represents a bank of operator control panels for IBM 729V drive as closely as reasonable. This GUI provides traditional Tape Drive Operator Panel indicators and buttons, as well as a mechanism to associate a PC file with the Virtual Tape Drive. Data written from the 1401 to the tape drive will appear in this file. When the 1401 reads from the tape drive, the data from this file will be supplied to the 1401. If the associated file contains a bootable object program or a "boot loader", programs from the tape can be loaded into the 1401 and executed by pressing the "Load" button on the 1401 Operators Console. This eliminates the need to punch programs to card decks before they can be run.

2 The Adapter

The functional elements of the Adapter are:

- the IBM 1401 tape channel interface
- the data buffer
- the microcontroller
- the USB buffer
- the USB peripheral interface engine
- the USB interface

The Adapter monitors activity on the USB interface and on the tape channel interface. Activity on either side may invoke action by the Adapter.

2.1 IBM 1401 Tape Channel Interface

The 1401's Tape Adapter Unit (TAU) controls the tape channel. The 1401 CPU makes "calls" to the TAU based in the instruction being executed and the TAU translates the "call" into "tape channel operations". The overall system was designed to make the tape drive units as simple (low logic card count) as possible. As a result, much of the complex logic that is found in modern tape drives, is performed by the TAU. This was economical because the logic in the TAU benefited all of the tape drives. This architecture also makes our job of emulating a tape drive easier.

2.1.1 1401 Tape Channel Operations

Defined Tape Channel Operations are:

- Read Tape
- Write Tape
- Backspace Record
- Skip and Blank Tape
- Rewind Tape
- Rewind and Unload Tape
- Sense Status (No-Op)

All of the 1401 instructions that effect the Tape I/O Channel map into one of the above channel operations. Tape oriented conditional branch instruction do not effect the Tape I/O Channel, but rather they sense status that has been set in the 1401 Tape Adapter Unit (TAU) as a result of previous channel operations. Interlocks within the TAU enforce that only one channel operation may be in progress at a time. This makes the difference between emulating one drive and multiple drives small.

Magnetic tape is a sequential file medium. When writing a record to a tape it is/was considered "poor programming practice" to assume that there is valid data beyond the point on the tape where this record is written. Due to stretching and contraction of the tape medium, tape speed tolerance and variations in tape start and stop timing, attempting to update a record on a tape could corrupt records that follow it. Given the above, we will make the simplifying assertion that writing a record to a tape erases all data beyond that record. Note that even with this assumption, it is allowed to mix read, write, and backspace operations to the same tape. The requirement is that the tape is never read beyond the location of the last write. If an attempt is made to read beyond the last write the Adapter will respond by indicating End-Of-Reel.

Before the TAU switches a tape drive from Write mode to Read mode, it advances the tape forward a short distance so that noise caused by turning off the write head would not corrupt previously written data. The tape advance causes erasure of any data that abutted the last block written. As a result, 1401 non-write tape instructions like Tape Rewind actually begin with a short Write operation (no data is transferred).

The timing information in the below "tape operation" descriptions are for an IBM 729V. The timings may vary for other drives. See the "Critical Timings Table" has complete timing information on all supported drives.

2.1.2 Tape channel operation phases

A tape channel operation session will pass through one or more of the below phases. These phases map to one or more states in the Adapter's State Machine.

- Selection Phase (e.g. Selected Not Ready, Selected Ready)
- Start Phase (e.g. Pre-Read, Pre-Backspace, Pre-Write)
- Execution Phase (e.g. Read, Backspace, Write, Rewinding...)
- Stop Phase (e.g. Post-Read, Post-Backspace, Post-Write)

Note that the below describes the operation of the full implementation of Adapter functionality. Initially a subset of this functionality will be implemented to support Tape Channel Analyzer needs for TAU bring-up. The full implementation is described to assure that the Adapter's architecture can support anticipated future expansion.

Each operation begins with the tape reels stopped; and ends with either the tape reels stopped or the tape drive entering "Manual Control" mode. Tape channel operations can have four phases. The "selection phase" begins a channel operation session with a Virtual Tape Drive. The "start phase" provides several milliseconds for tape movement to start and achieve a stable speed. The "execution phase" performs the specified operation. And, the "stop phase" provides a few milliseconds for tape movement to stop.

A tape channel operation session begins with the TAU activating one of its six Select lines. The Adapter will monitor the levels of the Select lines for which it has been configured to respond. If the Adapter is configured to respond to a specific Select line, it is said to have a "Virtual Tape Drive" defined for that line. If a Virtual Tape Drive is selected, the Adapter will emit status for that Virtual Tape Drive on its output signal lines. The values being emitted will be updated as changes in the Virtual Tape Drive state occur. The TAU ends a channel operation session by deactivating the Select line.

Once the channel operation session is established, if tape movement is needed, the TAU will initiate tape movement by activating the "Go" signal. The activation of the "Go" signal begins the "Start" phase of the channel operation. The "Delay Counter" in the TAU controls the length of the "Start" phase. The TAU determines the type of tape drive it is communicating with by examining the tape drive's status that it received during the Selection phase. The Adapter can provide status to the TAU that describes it as an IBM 729 Model II, IV, V, or VI drive. This delay time permits tape to start moving and obtain a stable speed. The Adapter may use this delay time to exchange information with the PC.

Read Tape, Write Tape, or Backspace Tape operations have "Start" phases, that lead into their "Execution" phase. During their "Execution" phase, data is transferred between the TAU and the Adapter. Because of the real time workload demand on the Adapter during "execution phases", it does not communicate with the PC during this phase. This restriction requires the Adapter to have sufficient memory to buffer any record transfer that is likely to occur. The Adapter will provide buffering for at least 2K byte tape records. This seems to be a reasonable restriction, given the maximum memory capacity of the 1401 is 16K bytes.

After the record data has been transferred, the TAU deactivates the Go signal to begin the "Stop" phase. The deactivation of the Go signal tells the tape drive to stop tape movement. Data transfer from the tape drive to the TAU may continue into this phase for up to 2 milliseconds. Tape drive specifications require a drive to maintain full outbound data amplitude for 2 milliseconds after the Go signal has been deactivated. The TAU again uses the "Delay Counter" to provide a time interval appropriate for the type of tape unit with which it is communicating. The Adapter may also use this interval to exchange data with the PC. At the completion of this time interval, the TAU may deactivate the Select signal, thus ending the tape operation session.

2.1.3 Read Tape Operation

The 1401 instructions that map to this operation are "Read Tape", "Read Tape with Word Marks", and "Diagnostic Read". The uniqueness of the specific instructions is known only by the TAU and is not communicated to the tape drive.

This operation transfers data from the selected tape drive to the TAU. The TAU will begin this tape operation session with the Adapter by activating a Select signal and proceed through all four phases. During the "Selection Phase", the TAU sets the Adapter to read mode and then moves to the "Start Phase" by activating the Go signal. The Backward signal is inactive while GO is active for the Read operation. If it is active, the Backspace Record tape operation is being performed, rather than the Read.

A drive is at its "load point" if it is fully rewound so that the load point reflective spot is under the "load point photo detector". If the selected tape drive is not indicating that it is at its "load point", the TAU will start monitoring for the appearance of data on the Read Bus 6.6 milliseconds after Go is activated (timings in the test of this document are for a 729 Model V drive operating at 800 CPI.). If the drive is indicating that it is at its "load point", the TAU will start monitoring for data 24 milliseconds after Go. The TAU will wait indefinitely for the first character of a record to be delivered. When the tape is not at its "load point", the transfer of the first character read typically occurs 10.5 milliseconds after Go. The Adapter transitions into the "Pre-Read" state when the TAU activates the Go signal. During this state, the Adapter must request and receive the record to be read from the PC. When the data is received, the Adapter moves to the "Execution" Phase. If the PC does not deliver the data to the Adapter in 1.7 seconds, the Adapter will report an "End-of-Reel" condition to the TAU, a data "Underflow" event to the PC, and enter the "Selected Ready" state. Warning: Most programs will not expect this "End-of-Reel" condition to occur and will crash if it does.

For a Read operation, the "Execution Phase" maps to the "Read" state. In this state the Adapter will send read data to the TAU. Character transfers will occur at fixed intervals based upon the density under which the data on the tape was written. The Adapter will transfer read data at the density rate for which it is configured (200, 556, or 800 characters per inch density). This results in a new character being emitted every 16.7 microseconds.

Data being transferred on the Read Data Bus is encoded in Non-Return-to-Zero-Invert-ones (NRZI) format. The Adapter normally computes a Longitudinal Redundancy Check (LRC) character for the record being transferred. The LRC character is transmitted to the TAU after the last character in the buffer is sent.

Every time the TAU receives a data character it resets its "Delay Counter". The TAU will recognize the end of a record transfer by 37 microseconds elapsing without a new character being received. Once it detects the end of the record transfer, the TAU will deactivate the Go signal, and wait for the LRC character to be sent. It will monitor the Read Bus for the LRC for 96 microseconds.

The "Stop Phase" is initiated by the deactivation of the Go signal. The Adapter will transmit the LRC character 62 microseconds after the last data character, so the transfer will occur approximately 27 microseconds into the "Stop" phase. The TAU will deactivate the Select signal 2.1 milliseconds after the deactivation of Go, thus ending the "Stop Phase" and the tape operation.

2.1.4 Write Tape Operation

The 1401 instructions that map to this operation are "Write Tape", "Write Tape with Word Marks", "Write Tape Mark", and "Diagnostic Write". The uniqueness of the specific instructions is known only by the TAU and is not communicated to the tape drive.

When the 1401 executes a "Skip and Blank Tape" instruction no tape channel operation takes place, but it impacts the next "Write Tape" operation by lengthening the period between the "Go" signal and the presentation of the first character to be written. Since we are not moving physical tape, it does not effect operation of the Adapter.

The Write operation transfers data from the 1401 TAU to the selected tape drive. The TAU will begin this tape operation session by activating a Select signal and proceed through all four phases. During the "Selection Phase", the TAU attempts to set the

Adapter to write mode. If the Virtual drive is write enabled, the Adapter will allow the transition to occur. The TAU then activates the Go signal, moving the Adapter to the "Start Phase".

If the Virtual Tape Drive is not indicating that it is at its "load point", the TAU will start transferring data 7.5 milliseconds after Go is activated. If the drive is indicating that it is at its "load point", the TAU will start transferring data 48 milliseconds after Go is activated. The Adapter will transfer from the "Start" phase (Pre-Write state) to the "Execution" phase (Write state) 6.7 milliseconds after Go is activated, if it is at its load point; and 37 milliseconds after Go is activated if it is at its load point. Any data sent before these times will be purposely lost.

When the Adapter is in the "Execute phase" Write Pulse signal interrupts are enabled. The Adapter recognizes a character on the tape channel interface by detecting the activation of the Write Pulse signal. As characters arrive from the TAU, the Adapter will store them in a RAM buffer.

After the last data character is transferred the TAU activates the Write Check Character signal. This signal causes the tape drive to write the accumulated LRC character to the tape Each received character and the LRC character is echoed in NRZI format to the Read Data bus after a 4-millisecond delay (emulates the travel distance between the tape head's write and read gaps). Also, the bits of each Write character received by a drive are Ored together and the result of this OR is instantly transmitted back to the TAU as an Echo Pulse. Every time the TAU writes a non-zero character (and all characters are non-zero) to a drive, it expects the Echo Pulse to be received within 5.2 microseconds.

When the first Write data character is received, the Adapter starts a 4-millisecond timeout. After this timeout elapses, each time a new Write data character is received, an earlier character is echoed back to the TAU on the Read Bus. After Write Data characters stop being received, the Adapter continues to echo received characters at the configured "Frame Rate" until all data characters have been echoed back to the TAU. The Adapter accumulates the LRC of the characters that it echoes. After the last data character is echoed, the "char to LRC" time interval (62 microseconds) is permitted and the LRC is transmitted to the TAU. The Adapter ignores the Write Check Character signal transition. As soon as the Adapter receives a data character, if it is not zero, a 4-microsecond Echo Pulse is sent to the TAU.

The TAU signals the end of the "Execution Phase" by deactivating the "Go" signal 3-milliseconds after the last character is written to the drive. Thus the operation advances to the "Stop" phase (Post-Write state). However several (perhaps all) of the written data may still need to be echoed back to the TAU. These characters and the LRC continue to be echoed while in the "Stop" phase. A real tape drive is able to deal with this situation by the requirement that it maintain speed and read data amplitude for 2-miliseconds after Go deactivates.

Once in the "Stop" phase, the Adapter sends a "Write Upload Request" event to the PC. After the LRC is transferred to the TAU, THE "Stop" phase is terminated and the Adapter returns to the "Selection" phase, but is "Not Ready" and will not become "Ready" until the PC uploads the written record. If the PC is unable to upload the record, the Adapter will report an "End-of-Reel" condition to the TAU and a data "Overflow" event to the PC. Warning: Most programs will not expect this condition to occur and will crash if it does.

2.1.5 Backspace Record Tape Operation

The Backspace Record operation moves the tape backward by one record. During this operation, the drive performs a read operation with the tape moving in the backward direction. The characters transmitted to the TAU on the Read Bus are neither passed to the CPU nor error checked. However, the transmission of the characters is necessary so that the TAU can detect when to stop the tape.

If the drive is indicating that its tape is positioned at the load point, the TAU will not issue the Backspace operation. If during the Backspace operation, the drive detects the "Load Point" reflective spot, the operation is ended.

If the drive is in Write mode, the tape is first advanced forward before the mode is changed to Read. This is done to guarantee a full inter-record gap after the last written record. (The tape may have been previously recorded and the current head position might be in the middle of an old record.) In this case, this first part of the Backspace operation will appear to the tape drive as a Tape Write operation that transfers no data (erase). The Adapter will interpret this part of the Backspace operation as an erroneous Write operation, but when Go is deactivated and no characters have been received, the Adapter will cancel the erroneous Write operation. The remainder of this description assumes the drive is in Read mode.

The TAU will begin this tape operation session by activating a Select signal and proceed through all four phases. During the "Selection Phase", the TAU sets the Adapter to read mode and activates the Backward signal. It then activates the Go signal and the Adapter moves to the "Start Phase".

The TAU starts the Start Phase by ignoring the Read Data bus for 3 milliseconds. This permits the drives read head to move away from any tape noise caused by changing the drive from write to read mode. The Adapter will wait 3.5 milliseconds and then enter the execution phase.

During the "Execution Phase", the Adapter will send random read data to the TAU. The TAU does not interpret the characters on the Read Bus. It just searches for an inter-record gap. Since the Adapter will send 100 NRZI encoded "A" (0x71) characters to the TAU in using the appropriate Read Bus frame timing. (For a 75ips drive using 800 bpi density the character frame is 16.7 microseconds.) No LRC character is sent. The Adapter will then stop transmitting and wait for the TAU to respond. The TAU detects an inter-record gap by 0.6 milliseconds elapsing after the last character is received. It then deactivates the GO signal and enters the Stop Phase 5.7 milliseconds after the last character is received.

The "Stop Phase" is initiated by the deactivation of the Go signal. The Adapter transmits a Backspace Event to the PC. The TAU ends this phase and the operation by deactivating Select.

2.1.6 Skip and Blank Tape Operation

This operation is described under the Write Tape Operation.

2.1.7 Rewind Tape Operation

This operation consists only of a Selection phase. The TAU selects the tape drive and activates the "Rewind" signal. In response to this, the drive activates the "Select and Rewind" output signal to indicate that the tape has begun rewinding. While the tape is rewinding, it becomes "not ready" and all output signals that require the drive to be ready are deactivated. When the drive detects the Load Point reflective spot, the drive turns off the "Select and Rewind" output signal, reenters the Ready state, activates the "Select and At Load Point" output signal, deactivates the "Select and Not At Load Point" output signal. The TAU then deactivates the Select signal, and reactivates other Ready dependent signals as appropriate.

After the Adapter activates the "Select and Rewind" output signal, it sends a Rewind Event to the PC, waits 1.7 seconds and sets the Virtual Drive's "Tape Position" state element to zero to indicate that the tape is positioned at its Load Point".

2.1.8 Rewind and Unload Tape Operation

This operation consists only of a Selection phase. The TAU selects the tape drive and activates the "Rewind and Unload" signal. In response to this, the drive the Select activates the "Select and Rewind" output signal to indicate that the tape has begun rewinding. While the tape is rewinding, it becomes "not ready" and all output signals that require the drive to be ready are deactivated. When the drive detects the Load Point reflective spot, it remains in the "not ready" state as it removes the tape from the vacuum columns and opens its access windows. The drive is now in "manual control mode" and must be made ready by the operator pressing the "Load Rewind" button and then the "Start" button, before it will again respond to the TAU.

When the TAU detects the activation of the "Select and Rewind" signal, it deactivates Select thus ending the operation. Note that the TAU does not wait for the tape to rewind to its Load Point.

After the Adapter activates the "Select and Rewind" output signal, it clears the drive's "Mechanically_Ready" and "Electrically_Ready" state elements, sets the drive's "Tape Position" state element to zero, and sends an Unload Event to the PC.

2.1.9 Tape Channel I/O Signals

The interface is physically represented by the below input and output signals.

Table 1. Tape Channel Inputs

Input Name	Level	Description
Select [1:6]	+P	Selects the addressed tape drive. The rotating address switch on the drive's Operator's Panel selects the address to which the drive will respond. Although the switches on the 729 drives have positions 0 through 9, the 1401 TAU can only address 1 through 6. If a drive is not selected, it will ignore the below input signals.
Set Hi Density	+P	Sets drive to write in "high density" mode.
Set Lo Density	-N	Sets drive to write in "low density" mode.
Turn On TI	-N	Sets the Tape Indicator flip-flop.
Turn Off TI	+P	Resets the Tape Indicator flip-flop
Set Read Status	+P	Sets the Read-Write flip-flop to Read.
Set Write Status	-N	If the tape is not write protected, sets the Read-Write flip-flop to Write.
Go	+P	Activating this signal starts the tape moving. Deactivating this signal begins the braking process to slow and stop movement.
Backward	-N	ANDed with "select and ready" to produce "reverse" which controls the direction of tape movement. The backward line is also ANDed with "non at load point" to prevent backspacing when the tape is at its load point.
Rewind	-N	Initiates a rewind operation.
Rewind & Unload	+P	Initiates a rewind operation followed immediately by an unload operation.
Write Bus [0:6]	-N	Data to be written to the tape.
Write Pulse	-N	Strobe to indicate valid data is on the Write Bus.
Write Check Character	-N	Tells the drive to write the accumulated LRC character to the tape.
Process Meter	-N	Not connected. Supposed to run the use time meter on the tape drive.
Summary		
	+P	11 signals
	-N	14 signals, not counting Process Meter.
	Total	25 signals

Table 2. Tape Channel Outputs

Output Name	Level	Description
Select and Ready Mod 2	+P	Indicates that the selected tape drive is a 729II or 729V, mechanically ready, and electrically ready. (These drives move tape at 75 ips.)
Select and Ready Mod 4	+P	Indicates that the selected tape drive is a 729IV or 729VI, mechanically ready, and electrically ready. (These drives move tape at 112.5 ips.)
Mod 5 or 6	+P	Indicates that the selected tape drive is either a 729V or a 729VI. (These tape drives support all densities up to 800 bpi.) This signal when analyzed with the above two signals completely identifies the model drive being used.
Select and At Load Point	+P	Indicates that the selected drive's tape is positioned at its load point.
Select and Not At Load Point	+P	Indicates that the selected drive's tape is not positioned at its load point.
Select and Rewind	-N	Indicates that the selected drive is rewinding.
Select and TI Off	-N	Indicates that the selected drive's Tape Indicator state element is off.
Select and TI On	-N	Indicates that the selected drive's Tape Indicator state element is on.
Sel Ready and Read	+P	Indicates that the selected drive is ready and its Write_Mode state element is set to Read (off).
Sel Ready and Write	+P	Indicates that the selected drive is ready and its Write_Mode state element is set to Write (on).
Hi-Lo Density	+P	Indicates that the selected drive's Hi_Density state element is set to on.
Write Echo Pulse	-N	This seems to be the conditional wrap back of the Write Pulse input. If any of the Write Data bits are set to one, then the Write Pulse is sent back to the TAU using this signal. (All legal 1401 BCD characters will have at least two bits on.)
Read Data [0:6]	NRZI	Data transmitted from the selected drive to the TAU.
Summary		
	+P	8 signals
	-N	4 signals
	NRZI	7 signals
	Total	19 signals

2.2 Adapter State

The Adapter state consists of the state elements needed to emulate a tape drive and the state elements needed for Tape Channel Analyzer configuration. Note that six sets of Tape Drive State elements are maintained one for each drive that may be being emulated.

2.2.1 Tape Drive State

The below state elements are maintained for each emulated and monitored drive. The state elements may be modified by the PC application or by the effects of Tape Channel operations.

State Element	Description
Mechanically_Ready	Set means that a tape is mounted on the drive and the access window is closed.
Electrically_Ready	Set means the drive is not in manual control mode.
Write_Mode	Set means the drive is in Write mode. Reset means the drive is in Read mode.
Hi_Density	Set means that the drive will write in its High Density Mode. Note that density is completely controlled by the TAU. This seems to just be for a light on the Operator's Panel.
Tape Indicator	This bit can be read and written by 1401 assembler instructions. It is also set as a side effect of exceptions occurring during tape operations.
Error Flags	Eight bits reserved for internal error flags.
Byte Count	For Read and Backspace Operations - The number of data characters to be sent to the TAU. This field decrements as characters are sent. For Write Operations - The number of data bytes received from the TAU. This field increments as characters are received.
Tape Position	This is the number of records from the start of the tape to the current position. A value of zero means that the tape is at its "Load Point". This value is represented as a signed 32-bit integer.

2.2.2 Tape Drive Configuration

This information is set up by the application software for each emulated drive and used but not modified by the Adapter. See the "Adapter Microcontroller Timer Use" section for specific timer values.

Configuration Variable	Description
Emulation_Enabled	Set means the Adapter will respond to interface transactions for this drive address.
Read, At LP, GO to 1st Char Interval	Time in pre-scaled ticks between detection of an active going GO transition and Adapter's writing its 1st character to the Read Bus on Read Operations, if the Virtual Tape is positioned at its Load Point.
Read, Not at LP, GO to 1st Char Interval	Time in pre-scaled ticks between detection of an active going GO transition and Adapter's writing its 1st character to the Read Bus on Read Operations, if the Virtual Tape is not positioned at its Load Point.
Backspace, GO to 1st Char Interval	Time in pre-scaled ticks between detection of an active going GO transition and Adapter's writing its 1st character to the Read Bus on Backspace Operations.
Write, Ignore after GO Interval	Time in pre-scaled ticks between detection of an active going GO transition and the Adapter's enabling the receiving of data from the Tape Channel Write Data Bus.
Read Timeout	The time limit in pre-scaled ticks between the Adapter's writing its 1st character to the Read Bus and the completion of the Read Operation. If this time interval is exceeded, than the operation may be hung.
Backspace Timeout	The time limit in pre-scaled ticks between the Adapter's writing its 1st character to the Read Bus and the completion of the Backspace Operation. If this time interval is exceeded, than the operation may be hung.
Write Timeout	The time limit in pre-scaled ticks between the expiration of the "Write, Ignore after GO Interval" and the completion of the Write Operation. If this time interval is exceeded, than the operation may be hung.
Rewind Delay	The time in pre-scaled ticks between the drive becoming "not ready" due to the detection of an active going "Rewind" transition and the completion of the Rewind Operation. This is the time it takes the emulated tape to rewind.
Unload Delay	The time in pre-scaled ticks between the drive becoming "not ready" due to the detection of an active going "Rewind & Unload" transition and the completion of the Unload Operation. This is the time it takes the emulated tape to rewind. At the end of this delay, if the drive is still selected by the TAU, then the drive will return to the selected but not ready state. Otherwise, the drive will return to the idle state.
High Density Read Char to Char	The time in system ticks between data character presentations on the Read Bus during High-Density Read Bus data transfer

	operations. Used by Read Operation, Backspace Operation, and by the echo function of the Write Operation.
High Density Read Pulse Width	The time in system ticks between turning on bits in the Tape Channel Pulse Register and turning them off. The initial implementation will use the same timing for both High and Low density operation. Separate fields are provided as a contingency.
High Density Read Char to LRC Delay	The time in system ticks between presentation of the last data character and the LRC character during High-Density Read Bus data transfer operations. Used by Read Operation and by the echo function of the Write Operation.
Read Data Source	0 = The data source for Read data will be the PC. (via "Download Record" commands) 1 = The data source for Read data will be the Analyzer's Data Generator.
File Protected	False = The drive is not file protected. (The drive's tape has its write enable ring installed.) True = The drive is file protected. (The drive's tape has its write enable ring removed.)

2.2.3 Tape Channel Analyzer State

The below states have been added to permit the Tape Channel Analyzer to alter the responses from emulated drives and monitor the tape channel activity from any drive, real or emulated.

State Element	Description
Monitor_Drives[1:6]	These bits correspond with the Select [1:6] inputs. A bit being set configures the Analyzer to monitor Tape Operations addressed to the corresponding tape drive. Monitoring is a passive function and will not effect channel operation.
Respond_Drives[1:6]	These bits correspond with the Select [1:6] inputs. A bit being set configures the Analyzer to respond to Tape Operations addressed to the corresponding drive. All drives selected in this element must also be selected in the Monitor_Drives element.
Only one instance of each of the below state elements exists in the adapter. The value of these state elements will be used for all drives selected in the Respond_Drives element.	
Force_Not_Ready	If this bit is set, the controlled drives will be forced to and held in the "Electrically Not Ready" state.
Inhibit_Echo_Pulse	Inhibits output of the "Write Echo Pulse". (Used only in Write operations.)
Drive_Model	Tells the analyzer to respond as a specific model tape drive. Valid values are 729V (default), 729II, 729IV, and 729VI.

	Note that this only effects the "Select_and_Ready_Mod 2", "Select_and_Ready_Mod_4", and "Mod_5_or_6" output signals. Critical timings are controlled separately.
Force_LRC_Error	Complements the LRC character that is echoed on the Read Data lines. (Used only in Write operations.)
Force_VRC_Error	Complements the check bit of each Write Bus character as it is echoed on the Read Bus. (Used only in Write operations.)
Force_Echo_Error	Complements the units and check bit of each Write Bus character as it is echoed on the Read Bus. (Used only in Write operations.)
Generate LRC	1 = The Adapter will generate and transmit a LRC character on the Read Bus during Read operations after each record is sent. This assumes that the LRC character is not included in the data record. If "Read Data Source" is the data generator, the LRC will be transmitted as character "Gened Record Length"+1. 0 = The Adapter does not generate a LRC character. The LRC is assumed to be that last character in the data record. These values apply regardless of the selected data source.
Nominal Gain [C,B,A,8,4,2,1]	The 4-bit track amplitude setting for each track of the Read Bus during normal operation.
Test Gain [C,B,A,8,4,2,1]	The 4-bit track amplitude setting for each track during the presentation of "Test" data on the Read Bus. Note that "Test Gain" applies even if the selected data source is the PC. The LED signal will be turned on during the presentation of test data to enable the triggering of test equipment.
Test Delay	See the description of the Tape Channel Analyzer's Data Generator for a description of how this field is used. (In the Tape Channel Analyzer / Functionality section of this document. The record's length preempts this setting, it does not persist to the next record.
Test Duration	See the description of the Tape Channel Analyzer's Data Generator for a description of how this field is used. (In the Tape Channel Analyzer / Functionality section of this document. The record's length preempts this setting, it does not persist to the next record. The LED signal will be turned on during the presentation of test data to enable the triggering of test equipment.
Test Repeats	The number of times that "Test Data Delay" and "Test Data Duration" is to be repeated in a record. The record's length preempts this setting, it does not persist to the next record. Example: If "Test Data Delay"==1, "Test Data Duration"==1, and "Test Data Repeats"==100; then for up to 200 characters, starting with the second character in a record, every other character will be transmitted using the "Test Gain" settings.
Gened Record Length	If "Read Data Source" selects the data generator, this is the size

	of the records that will be generated. (Not including any generated LRC character.) See the description of the Tape Channel Analyzer's Data Generator for a description of how this field is used. (In the Tape Channel Analyzer / Functionality section of this document.)
Test Data Char	If "Read Data Source" selects the data generator, this seed character used to generate "Test Data". See the description of the Tape Channel Analyzer's Data Generator for a description of how this field is used. (In the Tape Channel Analyzer / Functionality section of this document.)
Filler Data Char	If "Read Data Source" selects the data generator, this character is presented on the Read Bus when the presentation of "Test" data is delayed or has a duration shorter than required to meet the record length specification. Filler characters are always transmitted at "Nominal Gain". See the description of the Tape Channel Analyzer's Data Generator for a description of how this field is used. (In the Tape Channel Analyzer / Functionality section of this document.)
Data Generator Mode	Repeat (0) = The "Test Data Char" seed is repeated without alteration. BCD Inc (1) = The next higher character in the BCD tape character collating sequence is used each time the "Test" data is transmitted. This sequence wraps from the higher character in the collating sequence to the lowest. (It is reinitialized to the seed character at the beginning of each record.) Binary Rotate (2) = The "Test Data Char" is rotated right by one bit each time it is transmitted. (It is reinitialized to the seed character at the beginning of each record.) When the "Test Data Char" is altered using any of the above modes, a valid even parity C-bit is generated. Note that Binary operations may result in bit patterns that are not valid BCD tape characters.

2.3 Adapter to PC Interface Commands and Events

When the PC wants to invoke action in the Adapter, it issues a Command. The Command is a message sent on the USB interface to the Adapter. When the Adapter receives a Command it answers immediately with a Response. By "immediately", it is meant that none of the Commands require the Adapter to wait for anything to occur.

The Adapter can request attention from the PC, by issuing an Event. An Event is a message sent on the USB interface to the PC. Events are one directional communications. The Adapter does not expect nor wait for a response. Events should be thought of as I/O Interrupts from the Adapter to the PC. The Adapter has the role of a slave (as opposed to a peer) to the PC.

Structures appearing in "Command specific data", "Command responses", or "Event specific data" of multiple commands are identified as *struct* elements below and are defined at the end of this section.

2.3.1 PC to Adapter Command Interface

The format of a Command message is:
- Start Flag - 1 byte (0xf0)
- Message length - 2 bytes (length of command specific data + 1)
- Command code - 1 byte
- Command specific data - variable length
- CRC - 1 byte (chosen to make the unsigned sum of all bytes modulo 256 equal zero)
- Stop Flag - 1 byte (0xf1)

The Adapter replies to each Command with a Command Response. The format of a Command Response message is:
- Start Flag - 1 byte (0xf0)
- Message length - 2 bytes (length of command response data + 5)
- Command code - 1 byte (the code of the Command that caused this response.)
- time_stamp - 4 bytes
- Command response data - variable length
- CRC - 1 byte (chosen to make the unsigned sum of all bytes modulo 256 equal zero)
- Stop Flag - 1 byte (0xf1)

The time_stamp is provided by the "Free Running Timer" for this message. The time stamp value is in scaled system ticks, which are units of about .3 milliseconds. See the description of this timer for more information.

If Command specific data or Command response data contains data intended for or from the tape data record, that data must be at the end of the data segment.

If the format of a received Command is corrupted or invalid, the Adapter will respond with an Exception Event instead of a Command Response. (To facilitate programming, Events and Command Responses have identical formats.) A received Command is processed in the following sequence.

1. A check is made to determine if the entire Command message was received. If not, a "Truncated Message" Event is transmitted.
2. The CRC of the Command message is checked. If not correct, a "CRC Error" Event is transmitted.
3. The Command Code in the message is checked. If not valid, an "Invalid Command Code" Event is transmitted.
4. Control is directed to the command specific service routine. Errors detected in these routines are reported in the appropriate Command Response message.

The Commands codes are:

• Full Reset	(0x01)
• Get Analyzer Configuration	(0x02)
• Set Analyzer Configuration	(0x03)
• Define Virtual Drive	(0x04)
• Undefine Virtual Drive	(0x05)
• Get Virtual Drive State	(0x06)
• Set Virtual Drive State	(0x07)
• Upload Record	(0x08)
• Download Record	(0x09)
• DiagUpload Loopback	(0x10)

2.3.1.1 Full Reset

All ongoing operations are aborted. The Adapter is set to its Power-On-Reset state. If the 1401 is in session with the Adapter, this could cause the 1401 to detect an error. The Adapter's last reset reason code is set to "PC Reset" at the end of this command.

```
    Command specific data:   None
    Command response data:
        struct   Adapter_status          //containing the old reset reason code
```

2.3.1.2 Get Analyzer Configuration

Reads the configuration of the analyzer. The below data is displayed and edited by the Tape Analyzer GUI.

```
    Command specific data:   None.
    Command response data:
        struct   Adapter_status
        struct   Analyzer_configuration
```

2.3.1.3 Set Analyzer Configuration

Sets the configuration for the analyzer. The below data is displayed and edited by the Tape Analyzer GUI.

```
    Command specific data:
        struct   Analyzer_configuration

    Command response data:
        struct   Adapter_status
```

2.3.1.4 Define Virtual Drive

Tells Adapter to emulate this drive, if 1401 tries to communicate with a drive at this address. Newly defined drives default to being in "manual control mode". A "Set Virtual Drive State" command must be issued to make the drive "ready".

```
    Command specific data:
        byte     drive_address                //1 through 6
        byte     drive_model                  //model of drive being emulated
                                              //  2= IBM 729 Model II
                                              //  4= IBM 729 Model IV
                                              //  5= IBM 729 Model V
                                              //  6= IBM 729 Model VI
        struct   Virtual_tape_drive_state     //initial state

    Command response data:
        byte     return_code                  // 0==success
```

2.3.1.5 Undefine Virtual Drive

Tells Adapter to stop emulating this drive. Operations in progress are aborted. This command should only be issued when the addressed drive is in "manual control mode" (not ready).

```
    Command specific data:
        byte     drive address (1 through 6)

    Command response data:
        byte     return_code                  // 0==success
```

2.3.1.6 Get Virtual Drive State

Upload the state of the virtual tape drive to the PC.

```
    Command specific data:
        byte     drive_address                //1 through 6

    Command response data:
        byte     return_code                  // 0==success
        struct   Adapter_status
        struct   Virtual_tape_drive_state     // not present if return_code>=64
```

2.3.1.7 Set Virtual Drive State

Set the addressed virtual drive's state. This may reset the drive and/or abort ongoing operations.

```
    Command specific data:
        byte     drive_address                //1 through 6
        struct   Virtual_tape_drive_state
    Command response data:
        byte     return_code                    // 0==success
```

2.3.1.8 Upload Record

Move tape record data from the Adapter to the PC.

```
    Command specific data:
        byte     drive address (1 through 6)

    Command response data:
        byte     return_code            // 0==success
        struct   Virtual_tape_drive_state  // not present if return_code>=64
        byte[]   record_data            // not present if return_code>=32
                                        // last byte if the LRC written by the 1401
```

2.3.1.9 Download Record

Move tape record data from the PC to the Adapter.

```
    Command specific data:
        byte     drive_address     //1 through 6
        byte[]   record_data       // last byte should be LRC unless Gen_LRC Flag is on

    Command response data:
        byte     return_code                // 0==success
        struct   Virtual_tape_drive_state  // not present if return_code>=64
```

2.3.1.10 DiagUpload Loopback

Store prepared tape record data into the Adapter. Wait for initial_delay seconds and then pretend that the 1401 just completed writing the stored data to drive drive_address. If repeat_count is non-zero, then repeat pretending to receive this record repeat_count times delaying repeat_ delay *100 milliseconds between repetitions.

```
    Command specific data:
        byte     drive_address  //1 through 6
        int      initial_delay   // in Timestamp format
        int      repeat_delay    // in Timestamp format
        short    repeat_count    // 0 means do this only once
        byte[]   record_data     // last byte should be LRC unless Gen_LRC Flag is on

    Command response data:
        byte     return_code                // 0==success
```

2.3.2 Adapter to PC Event Interface

The format of an Event message is:

- Start Flag - 1 byte (0xf0)
- Message length - 2 bytes (length of the Event specific data + 5)
- Event code - 1 byte
- time_stamp - 4 bytes
- Event specific data - variable length

- CRC - 1 byte (chosen to make the unsigned sum of all bytes modulo 256 equal zero)
- Stop Flag - 1 byte (0xf1)

Note that the format of the Event message is identical to the "Command Response" message.

If Event specific data contains data intended for or from the tape data record, that data must be at the end of the data segment.

The Event codes are:

• Download Request	(0x81)
• Download Underflow	(0x82)
• Upload Request	(0x83)
• Upload Overflow	(0x84)
• Backspace	(0x85)
• Rewind	(0x86)
• Unload	(0x87)
• Status Change	(0x88)
• Read Timeout	(0x89)
• Write Timeout	(0x8A)
• Backspace Timeout	(0x8B)
• Channel Transaction	(0x8C)

Exception Events

• Invalid Command Code	(0xFC)
• CRC Error	(0xFD)
• Truncated Message	(0xFE)

2.3.2.1 Download Request

The 1401 Tape Adapter Unit (TAU) has requested the next sequential record on the specified tape drive. This data is to be supplied by the PC via a "Download Record" command. The TAU will wait about 10.5 milliseconds for this data transfer to start. If the PC does not supply the requested data in time, the Adapter will send a "Download Underflow" event.

Event specific data:

```
byte    drive_address        //1 through 6
struct  Virtual_drive_state
```

2.3.2.2 Download Underflow

The 1401 TAU tried to read from an emulated tape, but the PC did not supply the data in time. An error was sent to the 1401. This event indicates that a serious error occurred. The PC should record that this error occurred.

Event specific data:

```
byte    drive_address        //1 through 6
struct  Virtual_drive_state
```

2.3.2.3 Upload Request

The 1401 TAU has written a record to the Adapter. This record is intended to be the next sequential record for the specified tape drive. This data is to be transferred to the PC via an "Upload Record" command. The TAU may begin writing another record in about 10.5 milliseconds. If the PC does not upload the record data in time, the Adapter will send a " Upload Overflow" event.

Event specific data:

```
byte     drive_address          //1 through 6
struct   Virtual_drive_state
```

2.3.2.4 Upload Overflow

The 1401 TAU wrote a record to an emulated tape, but the PC did not upload the record in time. An error was sent to the 1401. This event indicates that a serious error occurred. The PC should record that this error occurred.

Event specific data:

```
byte     drive_address          //1 through 6
struct   Virtual_drive_state
```

2.3.2.5 Backspace

The 1401 TAU has moved the tape backward by 1 record on an emulated tape. The new tape position is specified in the "virtual drive status".

Event specific data:

```
byte     drive_address          //1 through 6
struct   Virtual_drive_state
```

2.3.2.6 Rewind

The 1401 TAU has commanded an emulated drive to rewind its tape to the "Load Point". Note that the "Tape Position" in the presented "virtual drive status" is the previous position. It should be interpreted as zero.

Event specific data:

```
byte     drive_address          //1 through 6
struct   Virtual_drive_state
```

2.3.2.7 Unload

The 1401 TAU has commanded an emulated drive to rewind and unload its tape. This operation has resulted in the emulated drive becoming "Mechanically Not Ready". Note that the "Tape Position" in the presented "virtual drive status" is the previous position. It should be interpreted as zero.

Event specific data:

```
byte     drive_address          //1 through 6
struct   Virtual_drive_state
```

2.3.2.8 Status Change

A 1401 TAU operation did something to cause the virtual drive status to change. (For example it could have set or reset the Tape Indicator.) This new virtual drive status is supplied to the PC as part of this Event message.

Note that if the Status Change due to the operation has already been communicated to the PC via another Event, then this Event is not sent.

Event specific data:

```
byte    drive_address          //1 through 6
struct  Virtual_drive_state
```

2.3.2.9 Read Timeout

The 1401 TAU started a Read operation to an emulated drive, but did not complete the operation in the expected period of time. This may have been due to an error occurring in the TAU. The virtual drive status supplied by this Event indicates the progress made in he operation before the timeout occurred. If the TAU does not complete the operation, the PC should send a "Set Virtual Drive State" command to reset the drive to an "Idle" state.

Event specific data:

```
byte    drive_address          //1 through 6
struct  Adapter_status
struct  Virtual_drive_state
```

2.3.2.10 Write Timeout

The 1401 TAU started a Write operation to an emulated drive, but did not complete the operation in the expected period of time. This may have been due to an error occurring in the TAU. The virtual drive status supplied by this Event indicates the progress made in he operation before the timeout occurred. If the TAU does not complete the operation, the PC should send a "Set Virtual Drive State" command to reset the drive to an "Idle" state.

Event specific data:

```
byte    drive_address          //1 through 6
struct  Adapter_status
struct  Virtual_drive_state
```

2.3.2.11 Backspace Timeout

The 1401 TAU started a Backspace operation to an emulated drive, but did not complete the operation in the expected period of time. This may have been due to an error occurring in the TAU. The virtual drive status supplied by this Event indicates the progress made in he operation before the timeout occurred. If the TAU does not complete the operation, the PC should send a "Set Virtual Drive State" command to reset the drive to an "Idle" state.

Event specific data:

```
byte    drive_address          //1 through 6
struct  Adapter_status
struct  Virtual_drive_state
```

2.3.2.12 Channel Transaction

If the Tape Channel Analyzer is enabled, one of these events is transmitted every time a Tape Channel control input or output signal changes state for a "monitored" drive. Note that changes on the Tape Channel Read Bus, Write Bus, and data bus strobes will not cause this event.

Event specific data:

```
byte    drive_address          //1 through 6 are normal. 16 indicates ISR message.
struct  Virtual_drive_state
short   buffer_count           //count of characters in the read/write data buffer
byte    select_reg             //Select Register (Port D encode 0)
byte    port_A
byte    port_B
byte    port_C
byte    port_E
byte    port_H;
byte    machine_state          //current state of the operations state machine
byte    read_write_flags       //bits that are ORed on when a "Write Pulse" or
                               //"Write Check Character" input becomes active
                               //   bits 7-1 unused
                               //   bit 0=write character received on Write Bus
```

2.3.2.13 Invalid Command Code

The received Command is not supported.

Event specific data:

```
byte[]  received message fragment    //size is limited to the first 64 bytes
                                     // the received Command code is the 1st byte
```

2.3.2.14 CRC Error

The CRC on the received Command message is not correct. The Adapter has ignored the Command.

Event specific data:

```
byte    received CRC
byte    expected CRC
byte[]  received message fragment    //size is limited to the first 64 bytes
```

2.3.2.15 Truncated Message

One hundred milliseconds have elapsed since the first byte of a Command message was received, but the specified number of bytes has not yet been received. The adapter has reset itself to expect the beginning of a new command.

Event specific data:

```
short   length of received fragment
byte[]  received message fragment    //size is limited to the first 64 bytes
```

2.3.3 Structure definitions

Adapter_status

byte	ANLS_STATUS	exception flags
byte	ANLS_STATE	state machine state
byte	ANLS_STATE2	state machine state modifier
byte	ANLS_RESR	last reset reason code (RCON Register)
byte	ANLS_EMASK	bit mask for drives that are being "emulated"
byte	ANLS_ACTDRV	bit mask for drive that is "in session"
byte	ANLS_BSTATUS	buffer status

```
    short   ANLS_TDNXTFREE pointer to next free location
    short   ANLS_TDNXTSEND pointer to next location to be sent

Analyzer_configuration
//The below data is displayed and edited by the Tape Analyzer GUI.
    byte    ANLS_MMASK      bit mask for drives that are being "monitored"
    byte    ANLS_CNFG1      configuration flags for drive emulation
    byte    ANLS_NGAIN21    normal gain for these two tape channel tracks
    byte    ANLS_NGAIN84    normal gain for these two tape channel tracks
    byte    ANLS_NGAINBA    normal gain for these two tape channel tracks
    byte    ANLS_NGAINC     normal gain for these two tape channel tracks
    byte    ANLS_NGAIN21    test gain for these two tape channel tracks
    byte    ANLS_NGAIN84    test gain for these two tape channel tracks
    byte    ANLS_NGAINBA    test gain for these two tape channel tracks
    byte    ANLS_NGAINC     test gain for these two tape channel tracks
    byte    ANLS_DGMODE     data generator mode
    short   ANLS_DGTDLY     test delay
    short   ANLS_DGTDUR     test duration
    short   ANLS_DGTRPT     test repeat count
    short   ANLS_DGRECL     generated record length
    byte    ANLS_DGTCHR     initial test data character
    byte    ANLS_DGFCHR     filler data character

Virtual_drive_state
    byte    flags   // bits defined below
            // bit 7  1= TAU is in session with this drive
            // bit 6  1= mechanically ready
            // bit 5  1= electrically ready
            // bit 4  1= drive is in write mode, 0= drive is in read mode
            // bit 3  1= high density, 0= low density
            // bit 2  1= tapeIndicator        See console indicator description
            // bit 1  1= write enabled, 0= write disabled (ring not in tape)
            // bit 0  unused
    byte    lrc             // accumulated LRC character
    int     tape_position   // the number of records from the beginning of the tape
                            //  to the current position. 0 = tape is at the load point
```

2.4 Adapter Microcontroller Port Register Assignments

Port A	Dir	7 bits -- Not used.
0	I	Select and TI On (Monitoring)
1	I	Select and TI Off (Monitoring)
2	I	HiLo Density (Monitoring)
3	I	Select and Rewind (Monitoring)
4	I	Select & Ready Mod 2 (Monitoring) (open drain)
5	I	Select & Ready Mod 4 (Monitoring)
6	IM	Not brought out.

Port B	Dir	8 bits, interrupting -- Interrupting Signals
0		
1	I	SELECT_CHANGE - Tape channel select signals have changed state since they were last read.
2	IM	FT245-RXF# (internal to module)
3	I	TWRITEPULSE - Write Pulse from tape channel.
4	IM	FT245-TXE# (internal to module)
5	IM	SWVCC (internal to module)
6	I	Reserved (Programming Clock PGC)
7	I	Reserved (Programming Data PGD)

Port C	Dir	8 bits -- Channel State Change Requests
0	I	TSETHDEN - Set Hi Density from tape channel.
1	I	TSETLDEN - Set Lo Density from tape channel.
2	I	TSETTI - Turn On TI from tape channel.
3	I	TRESETTI - Turn Off TI from tape channel.
4	I	TSETREAD - Set Read Status from tape channel.
5	I	TSETWRITE - Set Write Status from tape channel.
6	O	Reserved (USART1 TX1)
7	I	Reserved (USART1 RX1)

Port D	Dir	8 bits, multiplexed -- Data Bus to 2nd Level Components
7:0	I/O	DB - Bi-directional data bus. Connected to: FT245 FIFO (internal to module), Latched Select[1:6] - latched Channel Select Signals, Read Channel Polarity Reg. (on analog card), Read Channel Pulse Reg. (on analog card), Read Channel Amplitude Regs A-D (on analog card)

Port E	Dir	8 bits -- Channel Inputs and Data Bus Address Encode
2 : 0	O	DBSEL[2:0] - Selects register that is to receive data from Port D data bus. Selection encodes are: 0 = FT245 FIFO 1 = SELREGOE - Select Register output enable 2 = SELRCDR - Read Channel Polarity (Direction) Register 3 = SELRCPR - Read Channel Pulse Register 4 = SELRCAA - Read Channel Amplitude A Register 5 = SELRCAB - Read Channel Amplitude B Register 6 = SELRCAC - Read Channel Amplitude C Register 7 = SELRCAD - Read Channel Amplitude D Register
3	I	TBACKWARD - Tape Channel's Backward signal
4	I	TREWIND - Tape Channel's Rewind signal
5	I	TREWUNLD - Tape Channel's Rewind and Unload signal.
6	I	TWRITELRC - Tape Channel's Write Check Character signal.
7	I	Unused

Port F	Dir	8 bits -- Tape Channel Write Data and Echo Pulse
6:0	I	TWRITEDATA - Tape Channel Write Data.
7	O	WRITEECHO -- Write Echo Pulse

Port G	Dir	5 bits -- Strobe Signals
0	O	SELSTROBE - Select Register Write Strobe
1	OM	FT245-RD# (internal to module) - Read next byte from FIFO.
2	OM	FT245-WR (internal to module) - Write byte into FIFO.
3	O	ANALOGSTRB - Write strobe to the Analog Module
4		FT245-SNDWUP
5	I	Unused

Port H	Dir	8 bits -- Virtual Drive Configuration
0	O	MOD2 - Ready Mod 2
1	O	MOD4 - Ready Mod 4
2	O	MOD5OR6 - Mod 5 or 6
3	I	Mod 5 or 6 (Monitoring)
4	I	Sel Ready & Write (Monitoring)
5	I	Sel Ready & Read (Monitoring)
6	I	Sel Ready & At Load Point (Monitoring)
7	I	Sel Ready & Not At Load Point (Monitoring)

Port J	Dir	8 bits -- Virtual Drive State
0	O	SELECTED - The Adapter is selected
1	O	READY - The Adapter is mechanically and electrically ready.
2	O	WRITE - The Adapter is in Write mode.
3	O	TAPEIND - The drives Tape Indicator (TI).
4	O	HIDENSITY - The drive is in Hi density write mode.
5	O	MODULE_LED# , TRIGGER_ANLZR
6	O	LOADPT - The drive's tape is at the load point.
7	O	REWINDING - The Tape is rewinding.

Mplxed D (encode 0)	Dir	FT245 USB Peripheral Chip (requires additional Port B controls)
7 : 0	I/O	USB data FIFO

Mplxed D (encode 1)	Dir	Select Register
0	I	TGO - Tape channel signal
1	I	SELECT REG[1]
2	I	SELECT REG[2]
3	I	SELECT REG[3]
4	I	SELECT REG[4]
5	I	SELECT REG[5]
6	I	SELECT REG[6]
7	I	0 (Grounded)

Mplxed D (encode 2)	Dir	Read Channel Polarity (Direction) Reg. (on Analog board)
6 : 0	O	Mapped to Read Channel bits CBA8421 respectively.
7	O	Unused

Mplxed D (encode 3)	Dir	Read Channel Pulse Reg. (on Analog board)
6 : 0	O	Mapped to Read Channel bits CBA8421 respectively
7	O	Unused

Mplxed D (encode 4)	Dir	Read Channel Amplitude A Reg. (on Analog board)
3 : 0	O	Gain control for Read Channel Bit 1
7 : 4	O	Gain control for Read Channel Bit 2

Mplxed D (encode 5)	Dir	Read Channel Amplitude B Reg. (on Analog board)
3 : 0	O	Gain control for Read Channel Bit 4
7 : 4	O	Gain control for Read Channel Bit 8

Mplxed D (encode 6)	Dir	Read Channel Amplitude C Reg. (on Analog board)
3 : 0	O	Gain control for Read Channel Bit A
7 : 4	O	Gain control for Read Channel Bit B

Mplxed D (encode 7)	Dir	Read Channel Amplitude D Reg. (on Analog board)
3 : 0	O	Gain control for Read Channel Bit C
7 : 4	O	Unused

2.5 Adapter Microcontroller Timer Use

Four hardware timers are used on the microcontroller. All of the timers are run off the microcontroller clock. A 10 MHz. crystal oscillator driving a 4x multiplying Phase Locked Loop creates the microcontroller clock. The resulting clock rate is 40 MHz. The internal microcontroller clock received by the timers is $F_{OSC}/4$ (10 MHz.).

2.5.1 Frame Timer

The "Frame Timer" is used during Read, Backspace, and Write operations to determine when to place a character on the 1401 Tape Channel Read Data Bus. Time intervals between 16 microseconds to 250 microseconds are programmed.

Table 2: Frame Timer Write Specifications.

Model =	729 II & V	729 II & V	729 V	729 IV & VI	729 IV & VI
Speed(ips) =	75	75	75	112.5	112.5
Density(bpi)=	200	556	800	200	556
Char to Char (uSec)	66.6	24.0	16.7	44.5	16.0
Char to LRC Ignore Period (uSec)	150	54	37	100	36
Char to LRC (uSec)	250	90	62	166	60

PIC18LF8720's Timer2 implements this timer. The input to the timer is the microcontroller's internal clock. The timer's prescaler is set to 1:16. The timer's postscaler is set to 1:1. Interrupts from this timer are configured to occur at the "Low" priority level.

The above configuration yields a timer tick of 1.6 microseconds and a full range value of 409.6 microseconds. The below tick values should be used to program this timer. Adjustments to the below table may be necessary to account for average interrupt latency time.

The 1401 will interpret any character received within the "Ignore Period" as the next data character being received. To be interpreted as an LRC, the character most arrive after the "Ignore Period" has elapsed.

The 1401 always writes its LRC "Char to LRC" microseconds after the last data character in the record. The Adapter will attempt to match this timing when it delivers an LRC character to the 1401.

Table 3: Frame Timer - Actual Values.

Model =	729 II & V	729 II & V	729 V	729 IV & VI	729 IV & VI
Speed(ips) =	75	75	75	112.5	112.5
Density(bpi)=	200	556	800	200	556
Char to Char (ticks)	42	15	10	28	10
Char to Char (uSec)	67.2	24	16	44.8	16
Char to LRC Ignore Period (ticks)	94	34	23	63	23
Char to LRC Ignore Period (uSec)	150.4	54.4	36.8	100.8	36.8
Char to LRC (ticks)	156	56	39	104	37
Char to LRC (uSec)	249.6	89.6	62.4	166.4	59.2

2.5.2 Write-Read Gap Timer

The "Write-Read Gap Timer" is used during the Write operation to delay the echo of written data on the Read Bus. 729 model II and V drives have a 4 mSec delay. 729 model IV and VI drives have a 2.7 mSec delay. The timer is started when the 1st written character is received. When the timer interrupts it is shut off, the first character is echoed, and the Frame Timer is started.

PIC18LF8720's Timer4 implements this timer. The input to the timer is the microcontroller's internal clock. The timer's prescaler is set to 1:16. The timer's postscaler is set to 1:16. Interrupts from this timer are configured to occur at the "Low" priority level.

The above configuration yields a timer tick of 25.6 microseconds and a full range value of 6528 microseconds. To achieve a 4-millisecond delay, the tick count of 156 should be used. . To achieve a 2.7-millisecond delay, the tick count of 105 should be used.

2.5.3 I/O Polling Timer

The "I/O Polling Timer" is used to check for changes in 1401 Tape Channel control input signals. The most critical of these signals are wired to interrupt producing microcontroller pins. State changes occurring to the less critical signals must be discovered by polling. Currently, a polling interval of about one millisecond is deemed sufficient. Whenever the "I/O Signal Polling" interrupt service routine is entered, this timer is reset to its "start" value.

PIC18LF8720's Timer3 implements this timer. The input to the timer is the microcontroller's internal clock. The timer's prescaler is set to 1:8. Interrupts from this timer are configured to occur at the "Low" priority level.

The above configuration yields a timer tick of 0.8 microseconds and a 16-bit full range value of 52 milliseconds. To achieve a one-millisecond timeout, a start value of -1250 ticks is used.

2.5.4 Alarm Timer

The "Alarm Timer" is used to signal the end of specific timed intervals that are important to the operation of the 1401 Tape Channel Interface.

PIC18LF8720's Timer0 implements this timer. The input to the timer is the microcontroller's internal clock. The timer's prescaler is set to 1:256. Interrupts from this timer are configured to occur at the "Low" priority level.

The above configuration yields a timer tick of 25.6 microseconds and a 16-bit full range value of 1.7 seconds. Note that this timer increments and interrupts on the transition from 0xffff to 0x0000. Therefore the complement of the below tick values plus 1 needs to be loaded into the counter.

Table 4: Alarm Timings (milliseconds / tick value)

Model =	729 II & V	729 IV & VI
Speed(ips) =	75	112.5
Density(bpi) =	All	All
Read At LP: GO to 1st Char	48 / 1875	32 / 1250
Read Not At LP: GO to 1st Char	10.5 / 410	6.7 / 262
Backspace: Go to 1st Char	3.1 / 121	2.1 / 82
Write At LP: Ignore after GO	44/ 1718	29/ 1133
Write Not At LP: Ignore after GO	6.6 / 258	4.4 / 172
Read Timeout	1.7 sec / 64k	1.7 sec / 64k
Backspace Timeout	1.7 sec / 64k	1.7 sec / 64k
Write Timeout	1.7 sec / 64k	1.7 sec / 64k
Rewind Delay	1.7 sec / 64k	1.7 sec / 64k
Unload Delay	1.7 sec / 64k	1.7 sec / 64k

2.5.5 Free Running Timer

The "Free Running Timer" is used as a time base by the Adapter's "Tape Channel Analyzer" function. When the "Tape Channel Analyzer" function is enabled, this timer is started and runs continuously. "Channel State Change" events are stamped with this timer's value so that PC based software can track relative timings of interface signal transitions. The desire is that the resolution of this timer be less than 1 millisecond and will not wrap during a debugging session.

The "Free Running Timer" is implemented using the PIC18LF8722's Timer1 (16-bit hardware timer) with a 3-byte extension of RAM memory. Every time the timer interrupts the RAM extension is incremented by one. The Timer is sampled as a four-byte integer consisting of the three bytes of RAM extension followed by the high order byte of the hardware timer. The input to the hardware timer is the microcontroller's internal clock. The timer's prescaler is set to 1:8. Interrupts from this timer are configured to occur at the "Low" priority level.

The above configuration yields a hardware timer tick of 0.8 microseconds. The RAM extension is incremented every 52.43 milliseconds and it will wrap after approximately 10 days. When the timer is initialized, the RAM extension is set to 0x8000 so that arithmetic overflow is delayed as long as possible. The resolution of the 32-timer (3 bytes of RAM plus the upper byte of the hardware timer) is 0.2 milliseconds.

The sampling of this timer's value must be performed in a way that avoids roll over errors. Sampling this timer should be performed in the following manner:

1. Copy the 3 byte RAM extension to the work area. (bytes 1 through 3, low order first)
2. Read the low order byte of the hardware timer to cause the high order byte to be latched.
3. Copy the high order byte of the hardware timer to byte 0 of a work area.
4. Read and save the timer's interrupt flag.
5. Read the low order byte of the hardware timer to cause the high order byte to be latched.
6. Compare the high order byte of the hardware timer to byte 0 of a work area. If they differ then go to step 3.

Check the saved copy of the timer's interrupt flag. If it is on, then increment the work area copy of the RAM extension.

2.6 Adapter Microcontroller Interrupt Level Use

The Microcontroller supports two priority levels thus permitting code execution on three levels, the background level, the low priority level, and the high priority level.

2.6.1 Background Level Firmware

Background level software executes at the lowest priority. It runs only when no interrupt software is active. Typically, the "Main Program" is executed at this level. The "Main Program" initializes the execution environment, initializes the interrupt service routines, enables interrupts, and starts performing work.

The work performed on this level is the communication to and from the USB controller. Work is initiated from two sources, inbound USB messages, and posts to the outbound message queue. The firmware on this level alternates checking for inbound USB messages and outbound postings.

Inbound USB messages are detected by polling the USB peripheral chip. When the chip indicates that inbound data is present, the command message is received, executed, and a response is sent. See the "Adapter to PC Interface Commands and Events" section for message definitions. All commands are defined to be executed and responded to without waiting for peripheral actions to be performed. When the last byte of a command message is received, it is responded to before the "Outbound Message Queue" is checked for new messages.

Outbound messages (Events) are posted to the "Outbound Message Queue" by interrupt service routines running at the low priority level. When a posted message is detected, it is sent to the USB chip. Due to the nature of priority level execution, if the firmware on this level detects the presence of a message in the queue, it is guaranteed that entire message has been placed into the queue. Also, when a message is detected in the "Outbound Message Queue", all messages in the queue are sent before the USB chip is examined for inbound messages.

2.6.2 Low Priority Level Firmware

Work on this level is initiated by interrupts caused by 1401 Tape Channel Interface signal changes and microcontroller timers. Six low priority interrupts are defined; the "Frame Timer", the "Free Running Timer", the "Select Change Interrupt", the "Go Changed Interrupt", the "Alarm Timer", and the "I/O Polling Timer".

Frame Timer Interrupts
Interrupts from the "Frame Timer" are routed to its service routine. The occurrence of this interrupt indicates that it is time to transmit a character on the Tape Channel Read Bus. It is used during:

- tape channel Read operations to transmit the read data on the Read Bus
- tape channel Backspace operations to transmit generated data on the Read Bus
- tape channel Write operations to echo Write data on the Read Bus

The occurrence of this interrupt means that it is time to transmit the next character on the Read Bus. This service handler shares the low priority level with other interrupts, but the other interrupts should not occur during the Read, Backspace, or Write block data transfers unless the actions of this handler need to be terminated.
The first part of this service routine is extremely timing sensitive. The other part is less sensitive time sensitive.

In the first part of the service routine, the work performed is:

1) The interrupt flag is cleared.
2) The timer is restarted for the next Read Frame.
3) Four bytes of pulse amplitude data are moved to the Read Channel Amplitude Registers.
4) The next_data_character to be sent is XORed with the LRC.
5) The LRC is moved to the Read Channel Direction Register.
6) The next_data_character is moved to the Read Channel Pulse Register.
7) The "Read Pulse Width Timer" is started.

The time interval from step 1 to step 7 should be a fixed number of microcontroller instruction cycles. All branching has been eliminated in order to accomplish this goal. It also must be executed in less than one half of a Read Data Bus Frame (target is 5 microseconds for the highest density read operations).

In the other part of the service routine, the work performed is:

1) Determine and make ready the Amplitude settings to be used for the next character to be transmitted.
2) Determine and make ready the next character to be transmitted. (Tape Channel Analyzer read data character generation occurs here.)
3) If Operation State == Read, then:
   a) If the current character is the LRC, then:
      i) Post "Read Complete".
      ii) Stop the Read Frame Timer.
   b) Else if the LRC is the next character to be sent, then:
      i) Add the difference in frame times to the Read Frame Timer.
4) Else if Operation State == Backspace, then:
   a) If the current character was the last to be sent, then:
      i) Post "Backspace Complete".
      ii) Stop the Read Frame Timer.
5) Else if Operation State == Write, then:
   a) If the current character is the LRC, then:
      i) Post "Write Complete".
      ii) Stop the Read Frame Timer.
   b) Else if the LRC is the next character to be sent, then:
      i) Add the difference in frame times to the Read Frame Timer.
6) Return from Interrupt.

The other part is not time sensitive except that both parts of the service routine must be executed in less than one frame time (16 microseconds for the highest supported densities), including time spent responding to the "Read Pulse Width Timer" and the "Write Pulse" or "Write Check Character" high priority interrupts. The Read Pulse Width Timer interrupt will always occur sometime during the execution of the second part of the service routine.

Free Running Timer Interrupts
Interrupts from the "Free Running Timer" are routed to its service routine. The occurrence of this interrupt indicates that the hardware timer has overflowed and the timer's RAM extension needs to be incremented.

This service handler shares the low priority level with other interrupts and it is essential that it cause as little timing interference as possible.

The work performed is:
1) The interrupt flag is cleared.
2) Increment the low order byte of the RAM extension.
3) Branch no carry to step 7.
4) Increment the middle byte of the RAM extension.
5) Branch no carry to step 7.
6) Increment the high order byte of the RAM extension.
7) Return from interrupt.

Other Low Priority Interrupts
A common service routine handles interrupts from the "Select Change Interrupt", the "Go Changed Interrupt", the "Alarm Timer", and the "I/O Polling Timer". The specific work to be done by this level is selected via a "current operation's state / input branch table". The operation states are:
- Idle - No virtual drive is selected. In this state the adapter will ignore all input transitions except for the activation of a select signal that corresponds to a configured virtual tape drive.
- Selected_NR - The TAU is in session with a configured virtual tape drive, but the drive is not ready to process TAU commands. If the previous tape operation was a write, the Adapter will inhibit the transition from this stare to Selected_Rdy until the data is uploaded.
- Selected_Rdy - The TAU is in session with a configured virtual tape drive, and the drive is ready to process TAU commands.
- Pre_Read - The virtual tape is moving forward while the drive is in Read state, no read data has been transferred. The Adapter will inhibit the transition from this stare to Read until the data is downloaded.
- Pre_Backspace - The tape is moving backwards while the drive is in Read state, no read data has been transferred.
- Pre_Write - The tape is moving forward while the drive is in Write state, no write data has been transferred.

- Read - The selected virtual drive is performing a Read block transfer.
- BackwardsRead - The selected virtual drive is performing a Backspace block transfer.
- Write - The selected virtual drive is performing a Write block transfer.
- Post_Read - The last byte (LRC) of a record has been sent to the TAU, waiting for GO to deactivate.
- Post_Backspace - The last byte of a backspace data has been sent to the TAU, waiting for GO to deactivate.
- Post_Write - The last byte (LRC) of a record has been received from the TAU, waiting for GO to deactivate.
- Rewinding - The Adapter has signaled the TAU that the drive is rewinding as part of a Rewind operation.
- Unloading - The Adapter has signaled the TAU that the drive is rewinding as part of a Rewind & Unload operation.

When a low priority interrupt occurs the work performed is:

1) Stop the "I/O Polling Timer".
2) If the "Alarm Timer" has interrupted, then stop it and clear its interrupt flag.
3) Clear interrupt flags for the "Select Change Interrupt", the "Go Changed Interrupt", and the "I/O Polling Timer".
4) Sample and save the Free Running Timer value (see the sampling instructions under this timer's description), the Select Register (Port D encode 0) and the A, B, C, E, and H Port registers. Input changes are detected by comparing these saved ports with an old copy of the saved ports that was made on the previous entry of this interrupt handler.
5) Handle virtual drive functions. Switch on current state:
   a) Idle:
      i) If the Select signal to a Virtual Drive has become active then:
         (1) Load the context for the selected tape drive.
         (2) initialize operations status.
         (3) Set current state to "Selected_NR".
         (4) Loop back to Step 5.
      ii) Exit the virtual drive function switch block.
   b) Selected_NR:
      i) If the virtual drive's Select signal has become inactive, then:
         (1) Set current state to "Idle".
         (2) Exit the virtual drive function switch block.
      ii) If the drive is "Electrically Ready" and "Mechanically Ready" and Force_Not_Ready is not asserted, then:
         (1) Set current state to "Selected_Rdy".
         (2) Loop back to Step 5.
      iii) Exit the virtual drive function switch block.
   c) Selected_Rdy:
      i) If the virtual drive's Select signal has become inactive, then:
         (1) Set current state to "Idle".
         (2) Exit the virtual drive function switch block.
      ii) If "Turn Off TI" has become active, then:
         (1) Set the "Tape Indicator" state element to false.

iii) If "Turn On TI" has become active, then:
   (1) Set the "Tape Indicator" state element to true.
iv) If "Set Lo Density" has become active, then:
   (1) Set the "Hi_Density" state element to false.
v) If "Set Hi Density" has become active, then:
   (1) Set the "Hi_Density" state element to true.
vi) If "Set Read Status" has become active, then:
   (1) Set the Write_Mode state element to false.
vii) If "Set Write Status" has become active and the Read_Protect state element is inactive, then:
   (1) Set the Write_Mode state element to true.
viii) If the virtual drive's Rewind signal has become active, then:
   (1) Set the current state to Rewinding.
   (2) Set the "Tape Position" state element to zero.
   (3) Set the "Mechanically_Ready" state element to false.
   (4) Set the "Alarm Timer" to the "Rewind Delay".
   (5) Post a Rewind event to the PC.
   (6) Exit the virtual drive function switch block.
ix) If the virtual drive's "Rewind & Unload" signal has become active, then:
   (1) Set the current state to Unloading.
   (2) Set the "Tape Position" state element to zero.
   (3) Set the "Mechanically_Ready" state element to false.
   (4) Set the Alarm_Time to the "Unload Delay".
   (5) Post an Unload event to the PC.
   (6) Exit the virtual drive function switch block.
x) If the GO signal has become active and the Backward input is inactive and the Write_Mode state element is false, then:
   (1) If the read data record has not been downloaded, then exit the virtual drive function switch block. (Does detecting the end-of-reel reflective spot make the drive "Not Ready"?)
   (2) Set current state to Pre_Read.
   (3) If the datasource is the PC, then Post a "Read Request" to the PC.
   (4) Set the Alarm_Time to the "Read: GO to 1st Char" delay for the virtual drive's model, density setting, and "At Load Point" state.
   (5) Initialize read data transfer fields to deliver the first read character.
   (6) Exit the virtual drive function switch block.
xi) If the GO signal has become active and the Backward input is active and the Write_Mode state element is false, then:
   (1) Set current state to Pre_Backspace.
   (2) Set the Alarm_Time to the "Backspace: GO to 1st Char" delay for the virtual drives model and density setting.
   (3) Initialize read data transfer fields to deliver the first read character.
   (4) Exit the virtual drive function switch block.
xii) If the GO signal has become active and the Backward input is inactive and the Write_Mode state element is true, then:

        (1) If the last record has not been uploaded, then Set the "Tape Indicator" to true, post a "Write Overflow" event, and exit the virtual drive function switch block. (Does detecting the end-of-reel reflective spot make the drive "Not Ready"?)
        (2) Set current state to Pre_Write.
        (3) Set the Alarm_Time to the "Write: Ignore after GO" delay for the virtual drives model and density setting.
        (4) Initialize read data transfer fields to deliver the first read character.
        (5) Exit the virtual drive function switch block.

d) Pre_Read:
    i) If the virtual drive's GO signal has become inactive, then:
        (1) Set current state to Post_Read.
        (2) Loop back to Step 5.
    ii) If the virtual drive's Select signal has become inactive, then:
        (1) Set current state to Idle.
        (2) Exit the virtual drive function switch block.
    iii) If the Alarm_Time has expired, then:
        (1) If the datasource is the PC and the record has not been downloaded, then:
            (a) Post a "Read Underflow" to the PC
            (b) What dies a 729 do when the reflective spot is detected while reading?
        (2) Set current state to Read.
        (3) Set the Alarm_Time to the "Read Timeout" delay.
        (4) Initialize and start the Frame_Timer with the appropriate frame time for the virtual drive's model and density setting.

e) Pre_Backspace:
    i) If the virtual drive's GO signal has become inactive, then:
        (1) Set current state to Post_Backspace.
        (2) Loop back to Step 5.
    ii) If the virtual drive's Select signal has become inactive, then:
        (1) Set current state to Idle.
        (2) Exit the virtual drive function switch block.
    iii) If the Alarm_Time has expired, then:
        (1) Set current state to Backspace.
        (2) Set the Alarm_Time to the "Backspace Timeout" delay.
        (3) Initialize and start the Frame_Timer with the appropriate frame time for the virtual drive's model and density setting.

f) Pre_Write:
    i) If the virtual drive's GO signal has become inactive, then:
        (1) Set current state to Post_Write.
        (2) Loop back to Step 5.
    ii) If the virtual drive's Select signal has become inactive, then:
        (1) Set current state to Idle.
        (2) Exit the virtual drive function switch block.
    iii) If the Alarm_Time has expired, then:
        (1) Set current state to Write.
        (2) Set the Alarm_Time to the "Write Timeout" delay.

        (3) Enable "Write Pulse" and "Write Check Character" interrupts.
        (4) Initialize but don't start the Frame_Timer with the appropriate echo delay time for the virtual drive's model and density setting.

g) Read:
    i) If the Read has been posted as complete, then:
        (1) Increment the "Tape Position" state element by 1.
        (2) Set current state to Post_Read.
        (3) Loop back to Step 5.
    ii) If the virtual drive's GO signal has become inactive, then:
        (1) Stop the "Frame Timer".
        (2) Set current state to Post_Read.
        (3) Loop back to Step 5.
    iii) If the virtual drive's Select signal has become inactive, then:
        (1) Stop the "Frame Timer".
        (2) Set current state to Idle.
        (3) Exit the virtual drive function switch block.
    iv) If the "Alarm Timer" expired, then:
        (1) Stop the "Alarm Timer".
        (2) Post a "Read Timeout" event to the PC.
        (3) Loop back to Step 5.

h) Backspace:
    i) If the Backspace has been posted as complete, then:
        (1) Decrement the "Tape Position" state element by 1.
        (2) Post a Backspace event to the PC.
        (3) Set current state to Post_Backspace.
        (4) Loop back to Step 5.
    ii) If the virtual drive's GO signal has become inactive, then:
        (1) Stop the "Frame Timer".
        (2) Set current state to Post_Backspace.
        (3) Loop back to Step 5.
    iii) If the virtual drive's Select signal has become inactive, then:
        (1) Stop the "Frame Timer".
        (2) Set current state to Idle.
        (3) Exit the virtual drive function switch block.
    iv) If the "Alarm Timer" expired, then:
        (1) Stop the "Alarm Timer".
        (2) Post a "Backspace Timeout" event to the PC.
        (3) Loop back to Step 5.

i) Write:
    i) If the Write has been posted as complete, then:
        (1) Disable "Write Pulse" and "Write Check Character" interrupts.
        (2) Stop the "Frame Timer".
        (3) Increment the "Tape Position" state element by 1.
        (4) Post a "Write Request" event to the PC.
        (5) Set current state to Post_Write.
        (6) Loop back to Step 5.

- ii) If the virtual drive's GO signal has become inactive, then:
  - (1) Disable "Write Pulse" and "Write Check Character" interrupts.
  - (2) Stop the "Frame Timer".
  - (3) Set current state to Post_Write.
  - (4) Loop back to Step 5.
- iii) If the virtual drive's Select signal has become inactive, then:
  - (1) Disable "Write Pulse" and "Write Check Character" interrupts.
  - (2) Stop the "Frame Timer".
  - (3) Set current state to Idle.
  - (4) Exit the virtual drive function switch block.
- iv) If the "Alarm Timer" expired, then:
  - (1) Stop the "Alarm Timer".
  - (2) Post a "Write Timeout" event to the PC.
  - (3) Loop back to Step 5.

j) Post_Read:
- i) If the virtual drive's GO signal has become inactive, then:
  - (1) Set current state to Selected_Rdy.
  - (2) Loop back to Step 5.
- ii) If the virtual drive's Select signal has become inactive, then:
  - (1) Set current state to Idle.
  - (2) Exit the virtual drive function switch block.

k) Post_Backspace:
- i) If the virtual drive's GO signal has become inactive, then:
  - (1) Set current state to Selected_Rdy.
  - (2) Loop back to Step 5.
- ii) If the virtual drive's Select signal has become inactive, then:
  - (1) Set current state to Idle.
  - (2) Exit the virtual drive function switch block.

l) Post_Write:
- i) If the virtual drive's GO signal has become inactive, then:
  - (1) Set current state to Selected_Rdy.
  - (2) Loop back to Step 5.
- ii) If the virtual drive's Select signal has become inactive, then:
  - (1) Set current state to Idle.
  - (2) Exit the virtual drive function switch block.

m) Rewinding:
- i) If the "Alarm Timer" has expired, then:
  - (1) Set current state to Selected_Rdy.
  - (2) Set the "Mechanically_Ready" state element to true.
  - (3) Loop back to Step 5.
- ii) If the virtual drive's Select signal has become inactive, then:
  - (1) Set current state to Idle.
  - (2) Invalidate the Alarm_Time.
  - (3) Set the "Mechanically_Ready" state element to true.
  - (4) Exit the virtual drive function switch block.

n) Unloading:

        i) If the "Alarm Timer" has expired, then:
            (1) Set current state to Selected.
            (2) Loop back to Step 5.
        ii) If the virtual drive's Select signal has become inactive, then:
            (1) Set current state to Idle.
            (2) Invalidate the Alarm_Time..
            (3) Exit the virtual drive function switch block.

6) If the state machine has changed state OR "Select" signal from a Monitored drive is active and any input or output has changed state since the last interrupt OR the Select signal from a Monitored drive has just become inactive OR the read_write_flag is non-zero, then:
    a) Post a "Channel I/O Status" event to the PC.
7) Replace the old copy of the saved ports with the current saved copy.
8) Write the virtual drives state to the appropriate output port pins.
9) Set the "I/O Polling Timer" to its normal interval.
10) Start the "I/O Polling Timer".
11) Return from interrupt.

2.6.3 High Priority Level Firmware

Work on this level is initiated by the "Read Pulse Width Timer", the "Write Pulse" input signal, the "Write Check Character" input signal, error exceptions, and potentially, debug actions. It is a design goal to perform as little work as possible on this level. It is also a requirement that non-error related interrupt service routines on this level must exit (return from interrupt) in less than 5 microseconds.

Read Pulse Width Interrupts
Interrupts from the "Read Pulse Width Timer" are routed to its service routine.
1. Clear the interrupt flag.
2. Store zeros in the Read Channel Pulse Register.
3. Return from Interrupt.

Write Pulse Interrupt
Interrupts from the "Write Pulse" input signal are routed to its service routine.
1) Clear the interrupt flag.
2) Read the character from the "Tape Channel Write Data" Register.
3) If the Frame Timer is not running (this is the first character to be echoed), then:
    a) Initialize the Frame Timer to 4 milliseconds and start it.
4) Store the received character in the read buffer and increment the received character count.
5) Set the "write character received" bit in read_write_flags to 1.
6) Return from Interrupt.

Write Check Character Interrupt
Interrupts from the "Write Check Character" input signal are routed to its service routine.
1) Clear the interrupt flag.

2) Read the character from the "Tape Channel Write Data" Register.
3) If the Frame Timer is not running, then:
   a) Initialize the Frame Timer to 4 milliseconds and start it.

3 The Web Application

The web application consists of two projects that are managed under Eclipse, TAPEUSB and Emulator. TAPEUSB is an Eclipse Java project that provides the Java interface to the Adapter via the TapeUsbDll.dll. When it is "made", the result is a "jar" file that is nested as a library under the Emulator project. The Emulator project is an Eclipse Dynamic Web Application project. When it is "made", the result is a "war" file, which is deployed by the Tomcat webserver.

The TAPEUSB project is documented via JavaDoc. Please refer to that document for detailed information.

The Emulator project consists of:

- index.html - the welcome web page for the Emulator.
- Console.jsp - the web page that presents the bank of tape drive operator consoles.
- console.js - the JavaScript w/Ajax that runs under the Console.jsp web page.
- Global.java - this servlet is executed when Tomcat initializes. It also provides access to application wide static fields.
- Configure.java - this servlet prepares the Console.jsp page for display.
- XmlUpdate.java - this servlet is invoked by the XMLHttpRequest made by console.js to update tape drive status displays.
- Drive.java - the Java Bean that maintains the state of a drive. An instance of Drive,java exists for each tape drive being emulated.
- DownloadFile.java - this servlet is invoked when a drive "unload" is performed and a file needs to be downloaded from the webserver to the user's PC.
- UploadFile.java - this servlet is invoked when the browser wants to upload a file from the PC to the webserver.
- SimhConv.java - contains format conversion methods called by DownloadFile and UploadFile to convert between SimH and ASCII formats.

All PC files are stored in ASCII format. This is possible because all of the characters the 1401 can write to tape can be mapped to printable ASCII characters. "Pierce's primary BCD encoding" is used to map BCD to ASCII.

```
Numeric  b    1    2    3    4    5    6    7    8    9    0     3    4    5    6     7
Rows     b                                                       8    8    8    8     8
        ===================================================================================
00-0F:  ' ', '1', '2', '3', '4', '5', '6', '7', '8', '9', '0',  '#', '@', ':', '>',  '{',
10-1F:  '^', '/', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '|',  ',', '%', '~', '\\', '"',
20-2F:  '-', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', '!',  '$', '*', ']', ';',  '_',
30-3F:  '&', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', '?',  '.', ')', '[', '<',  }'},
```

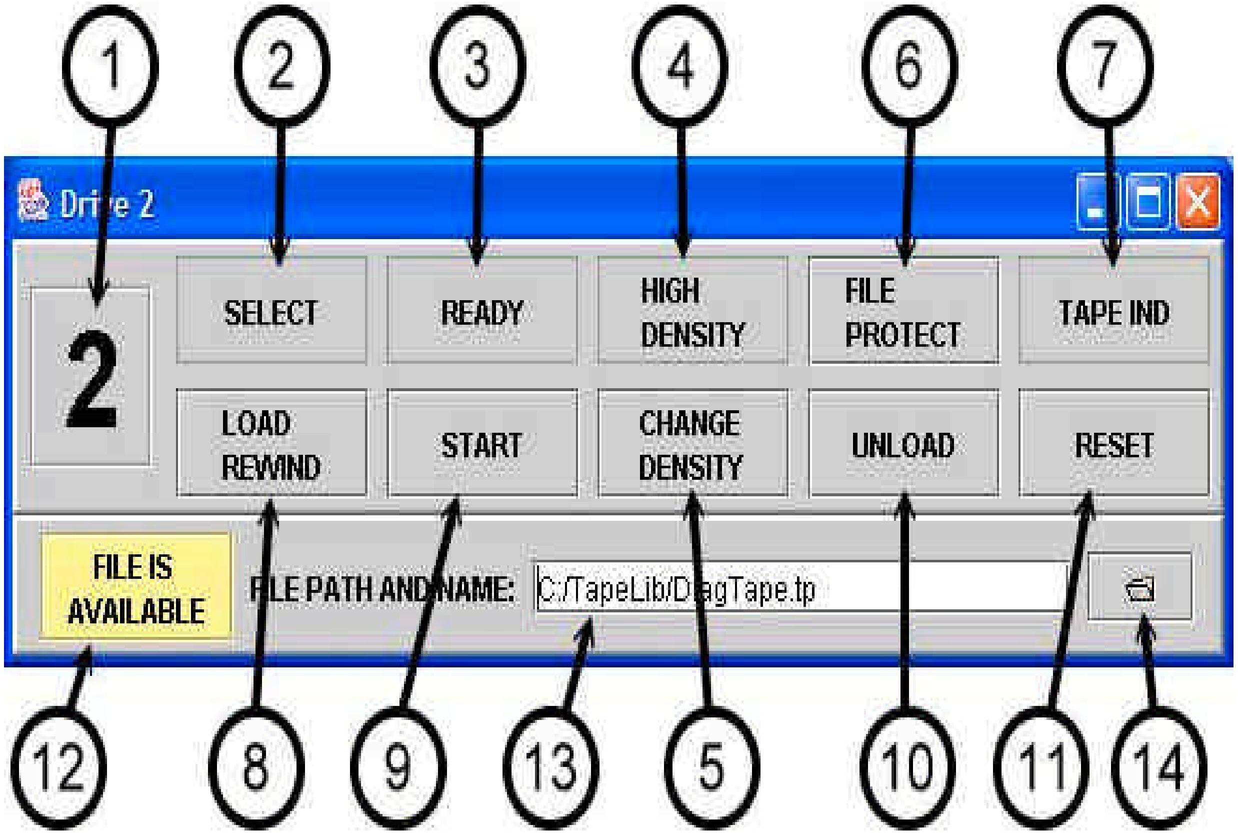
4 Virtual Tape Drive Console

The below description is for an early Swing based vision of the Virtual Tape Drive Console. The web page implementation of the GUI is similar, but not exactly consistent with the below description. Rather than correct the below description, additional web pages will be created to describe and provide instruction for using the console.

The Virtual Tape Drive Console provides basic status visibility and control of the virtual drive. Since the objective of this application is to emulate 1401 era tape drives and priority is being given to restoration of the IBM 729 Model V tape drives, the GUI should represent a IBM 729 Model V drive as closely as reasonable. One instance of this console will be created for each emulated tape drive.

4.1 Functionality

The console should emulate the operator panel of a 729V tape drive. The GUI displays the operator controls and indicators in a "JFrame" (non-Java readers should substitute the term "Window"). Each virtual tape drive must have a unique address. Legal addresses are 1 through 6. The address must be user specifiable.



1) The "Drive address" indicates the tape channel drive address that is associated with the Frame.

2) Select Indicator - "On" indicates that the 1401 has selected the virtual drive and is in session with the Adapter.
3) Ready Indicator - "On" indicates that the Adapter is Mechanically and Electrically ready. "Off" indicates that the drive is in manual control mode.
4) High Density Indicator - "On" indicates that the TAU will perform Write Operations using the "High Density" timings for the model drive that is addressed. (Write Operation timing is determined completely by the TAU's interpretation of the "Select and Ready Mod x", "Mod 5 or 6", and "Hi-Lo Density" response signals.) For Read Operations on real drives, this density indicator is meaningless. The density of Read Operations is predetermined by the data on the tape.
   For Virtual Tape Drive emulation, Write Operations will be performed in the same manner. For Read Operations, the Adapter will use the state of this indicator to select the timings that it will use to present data to the TAU. If the drive is configured as a 729 V, then read data is transferred using 800 bpi timing when in high density mode and 556 bpi timings when in low density mode.
5) Change Density Button - Toggles tape drive between high and low density. The TAU can also set the drive to the high or low density state.
6) File Protect Indicator / Button - "On" indicates that the drive will not accept Write Operations. On a real drive, this state is determined by the absence of the "write ring" being present on the tape reel. Under emulation, this indicator also functions as a button to toggle the "File Protect" state. However, if a file is associated with the Virtual Drive (see #13) and the file has the "file protect attribute" set, then the "File Protect" state is forced to "On". This button is inoperative unless the drive is in manual control mode.
7) Tape Indicator - "On" indicates that the Adapter's "Tape Indicator" state bit is on. The Adapter's "Tape Indicator" state bit can be turned on/off by 1401 interface signals. Reading a "tape mark" character will also turn it on. Pressing the "Unload" console button or a TAU initiated Unload operation will also make the Adapter turn it on.
8) Load Rewind Button - Emulates closing the drive's access window and positioning the tape at its "Load Point". Upon pressing this button, this application will also use the pressing of this button to:
   a) Verify the validity of the "file association field" (see #13). If the field is not valid, the pressing of the button is ignored and the user is informed that the field is not valid.
   b) Make "file association field" uneditable.
   c) If a file is specified, a copy of the file is sent to the Adapter Communications DLL.
   d) The file availability indicator (see #13) is set to "IN USE".

   This button is inoperative unless the drive is in manual control mode.
9) Start Button - Pressing the start button places the drive in "ready" status if tape is loaded (file is associated or CE Panel Data Generator is enabled) and the access door is closed (Load Rewind was pressed). This button is inoperative unless the drive is in manual control mode and the above conditions are met. "Ready" status takes the drive out of manual control mode.

10) Unload Button - Emulates rewinding the tape, pulling the tape out of the vacuum columns, and opening the drive's access window. When this button is pressed, the Adapter:
    a) If data was written to a file, then the file is closed and transferred from the Adapter Communications DLL to this application.
    b) The "file association field" is made editable.
    c) The file availability indicator (see #13) is set to "AVAILABLE".

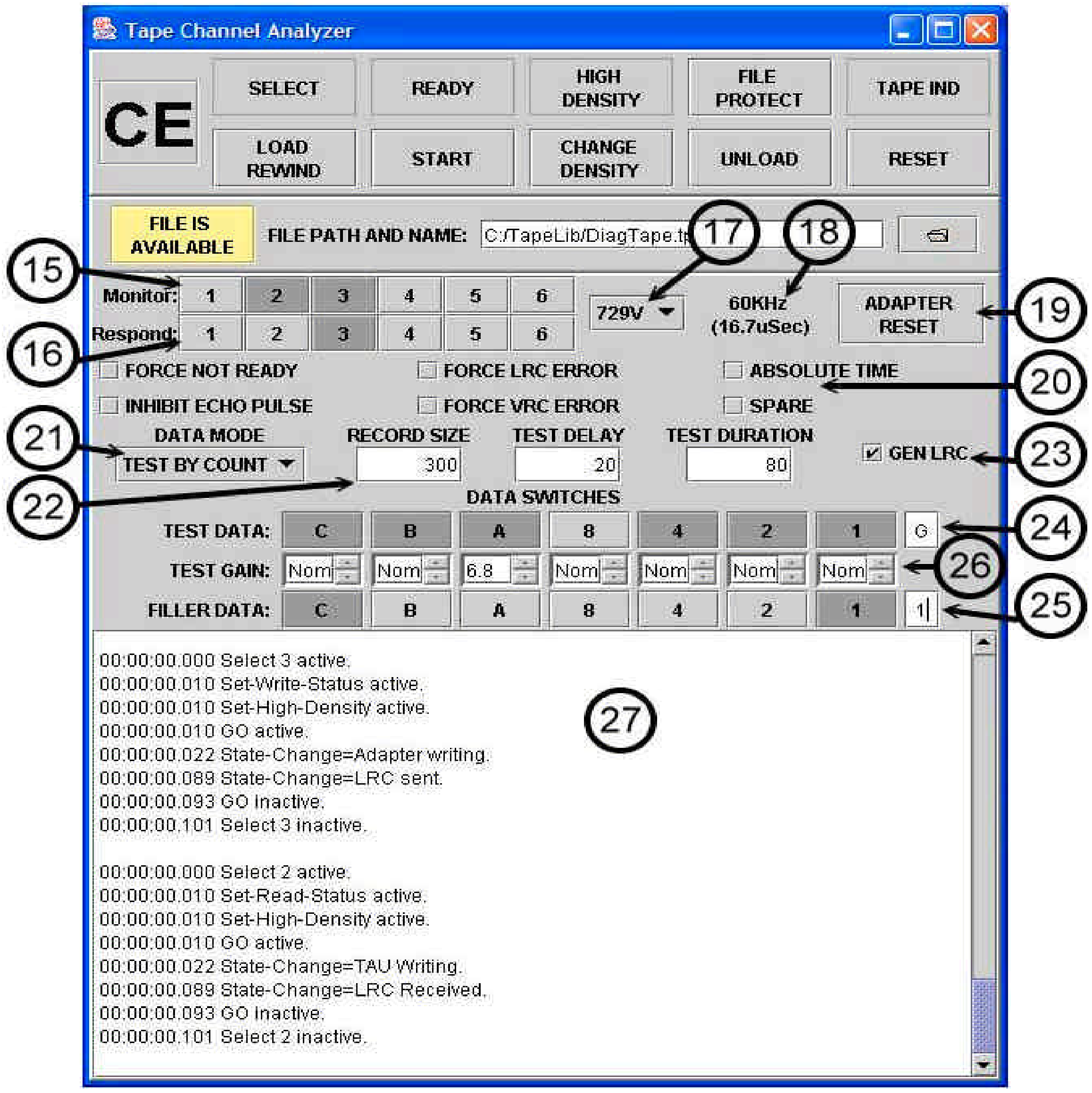
    This button is inoperative unless the drive is in manual control mode. The TAU may also initiate these actions by issuing a "Rewind & Unload" operation.
11) Reset Button - Aborts any ongoing operation and sets the tape drive in "manual control mode".
12) File availability indicator - This indicator changes state between "IN USE and "AVAILABLE" (a change in background color would help denote the state difference). The indicator state of "IN USE" means that the application's copy of the associated file may be out of sync with the copy of the file held by the Adapter Communications DLL and may not contain the data written by the TAU. The local copy of the file can be synchronized by pressing the "UNLOAD" button or the TAU issuing a "Rewind & Unload" operation.
13) File association field - To associate a PC file with the emulated tape reel, the fully qualified path/name of the file should be specified in this field. The field may be blank if the CE panel has its Data Generator enabled. If the field is not blank, it must contain a valid file path/name. The standard Java file path format is used. (The forward slash ("/") separator is used rather than the Windows backslash separator ("\").) Specifying a valid file name for a file that does not exist indicates the mounting of a blank tape. If this field specifies a file which is Windows write protected, then the "File Protect" state will be forced "On".
14) File chooser button - Pressing this button will activate a "File Chooser" pop-up window to help the user locate the file path/name to select for the "file association field".

5 Tape Channel Analyzer GUI

The below description is for a Swing based vision of the Tape Channel Analyzer GUI. It is expected that final implementation of the GUI will be similar, but not exactly consistent with the below description. The below description will be updated when the implementation is determined.

This application presents a GUI with the look and feel of a CE Panel and a logic analyzer. Switch controls override normal Adapter responses. A waveform display window is not specified, but may be a good enhancement.

5.1 Functionality



FILE PATH AND NAME: C:/TapeLib/DiagTape.t

729V

DATA MODE: TEST BY COUNT

RECORD SIZE: 300

TEST DELAY: 20

TEST DURATION: 80

[✓] GEN LRC

```
00:00:00.000 Select 3 active.
00:00:00.010 Set-Write-Status active.
00:00:00.010 Set-High-Density active.
00:00:00.010 GO active.
00:00:00.022 State-Change=Adapter writing.
00:00:00.089 State-Change=LRC sent.
00:00:00.093 GO Inactive.
00:00:00.101 Select 3 inactive.

00:00:00.000 Select 2 active.
00:00:00.010 Set-Read-Status active.
00:00:00.010 Set-High-Density active.
00:00:00.010 GO active.
00:00:00.022 State-Change=TAU Writing.
00:00:00.089 State-Change=LRC Received.
00:00:00.093 GO Inactive.
00:00:00.101 Select 2 inactive.
```

The top portion of the CE Panel operates as described in the previous section, except as described below.

15) Monitor Toggle Switches - The switches indicate which tape channel drive addresses the Adapter monitors. If a drive address is "monitored", then tape channel transaction to that address are recorded and displayed in the "Channel Trace" pane (see #27).
16) Respond Toggle Switches - The switches indicate the tape channel drive addresses to which the Adapter will emulate (respond).

The actions of the below controls apply to all Virtual Drives selected by the "Respond Toggle Switches".

17) Drive Model Selector - Selects the model drive that will be emulated. Valid Selections are 729V (default), 729II, 729IV, and 729VI.
18) Frame Rate Display - The data transfer rate resulting from the above "Drive Model Selector" and density settings.
19) Adapter Reset - Fully resets the Adapter. Any ongoing operations are aborted.
20) Check Box Controls - These controls enable or disable specific features of the Tape Channel Analyzer.
    a) Force Not Ready - The Adapter will respond to all TAU operations as "Not Ready". The Virtual Drive state is forced and held "Not_Ready".
    b) Inhibit Echo Pulse - When Write Operations occur, the Adapter will not respond to write data with echo pulses.
    c) Force LRC Error - The Adapter's LRC character will be complemented before it is transmitted. This applies to all transmitted LRC characters including those for Read Operations and the echo function of the Write Operations.
    d) Force VRC Error - The Adapter will complement the "C"-bit of all data characters that are transmitted. LRC characters will be calculated using the complemented data. This applies to all transmitted data characters including those for Read Operations and the echo function of the Write Operations. This switch does not effect LRC characters.
    e) Absolute Time - Normally timestamps displayed in the "Trace Pane" (see #27) are displayed using time relative to the previous active transition of a monitored Select signal. If "Absolute time" if checked, then absolute timestamps (relative to the last Adapter Reset) will be displayed. This facilitates the examination of Tape Channel timing across multiple operations.
21) Data Generator Mode - The Tape Channel Analyzer can generate several types of data streams. This control selects the type of data stream that is generated.
    a) Force Test - The character represented by the "Test Data" switches at the gain level established by the "Test Gain" switches will be transmitted whenever a character is sent on the Tape Channel Read Bus.
    b) Alternate - Characters in the generated data stream will alternate between the character represented by the "Test Data" switches and the character represented by the "Filler Data" switches. All characters transmitted at the gain level established by the "Test Gain" switches.
    c) Test By Count - The generated data stream will consist of records that are "Record Size" (see #22) characters in length. In each of these records, the first part of the record will consists of the "Filler Data" character. The "Test Delay"

field contains the number of "Filler Data" characters to be inserted. Then a block of "Test Data" characters will be transmitted. The "Test Duration" field contains the number of "Test Data" characters in this block. If "Record Size" is greater than "Test Delay" plus "Test Duration", the remainder of the record is composed of "Filler Data" characters. All characters transmitted at the gain level established by the "Test Gain" switches.

d) Test+1 - Same as "Test by Count" except that the "Test Data" character is the first character of the "Test Data" block. Subsequent characters in the block generated by selecting the next character in the BCD tape character collating sequence. Example: A block of 5 characters starting with "7" is "7", "8", "9", blank, and "."
e) Test Rotate - Same as "Force Test" except that the bits of "Test Data" character are rotated one position to the right after each transmission. Note that this may generate characters that are not valid BCD tape characters.

22) Count fields - The "Record Size", "Test Delay", and "Test Duration" fields are numeric text entry fields that contain data that may be used by some "Data Generator" modes. (see #21)
23) Gen LRC - When checked the Tape Channel Analyzer's Data Generator will generate a valid LRC character and transmit it after the last data character of the record. This character will be in addition to the count specified in the "Record Size" field. By default, Gen LRC is active.
24) Test Data Toggle Switches - The character represented by these switches is known as the "Test Data" character. See "Data Generator Mode" (#22) for a description of how the character is used. If the character is entered via the switches, then the text representation of the character is displayed in the right side text field. If the character is entered via the right side text field, then the bit representation is set in the switches.
25) Filler Data Toggle Switches - The character represented by these switches is known as the "Filler Data" character. See "Data Generator Mode" (#22) for a description of how the character is used. If the character is entered via the switches, then the text representation of the character is displayed in the right side text field. If the character is entered via the right side text field, then the bit representation is set in the switches.
26) Test Gain Spinner Switches - Each of these switches indicates the gain level to be used for the corresponding bit of "Test Data". These switches only apply to all data generated by the Data Generator.
    Note: We have also discovered that the Adapter is unable to change Gain in the middle of a record. This discovery is expected to effect the interpretation of some other fields and controls.
27) Trace Pane - This pane displays a textual representation of the Tape Channel Analyzer's trace file. Recorded Tape Channel control and control response signal transitions are displayed in this pane. Each transition is displayed as a line preceded by a timestamp. The timestamps may be displayed as absolute or relative (see #20).

6 Windows Kernel Driver

The Driver handles the interface between the Adapter and the Adapter Application Program. The driver follows Microsoft Windows Driver Model (WDM) rules and conventions. Time critical portions of the Driver operate in kernel mode at the Dispatch level so that Windows services can not interfere.

The driver modules are:

- tapeusb.sys - The IBM 1401 Tape Channel Analyzer (Adapter) specific part of the driver.
- Generic.sys - A foundation of driver utility routines that is provided with Microsoft's "Programming the Windows Driver Model 2nd edition" book. These routines perform most of the PnP, WMI, and PWR interrupt handling. The device queue routines are also used. This module is used straight from the CD, without modification.

The driver source modules are:

- DriverEntry.cpp - The driver load and unload routines. Wrapper routines for plug-n-play (PnP), power (PWR), and Windows management (WMI) handling. These wrappers call the Generic.sys routines to perform most of the work. Basically, this source file contains the code the boilerplate code. Very little of it has been modified.
- ReadWrite.cpp - This file contains the Adapter specific application interfaces and interrupt handling. The logic for the Read, Write, and DeviceIoControl functions are handled here. The contents focuses on the driver/kernel infrastructure.
- AdapterLogic.cpp - This file contains the program logic that performs the formatting for messages being sent to the Adapter and the interpretation of messages received from the Adapter.

6.1 Adapter Modes and Virtual Tape Drive States

From the perspective of the Driver, the Adapter has two operational modes, normal mode and diagnostic mode. When the Adapter is initialized or reset, it is placed into normal mode. The application program may place the Adapter into diagnostic mode. The operational mode of the Adapter effects the Adapter's interface to the 1401 and the operation of all Virtual Tape Drives.

In diagnostic mode, the Adapter's interface to the 1401 is disabled. The Adapter will not respond to 1401 Tape Channel operations. Currently, the only defined function that runs in diagnostic mode is "Adapter Loopback". This function is used to artificially generate simulated 1401 Tape Write traffic to test and stress the ability of the PC (Driver and application program software) to meet the 1401's response time requirements. Once in diagnostic mode, the Adapter must be reset to return to normal mode.

The Adapter can simultaneously emulate multiple tape drives. Each of these drives is described by its Virtual Tape Drive State. An important element of the Virtual Tape Drive State is whether the Virtual Tape Drive is Ready or is in Manual Control Mode (sometimes referred to as Not Ready). This distinction pertains to the entity that is managing the Virtual Tape Drive's State. The Driver and Adapter together manage the state of Ready drives. The application program is responsible for managing the state of drives that are in Manual Control Mode.

A drive becomes Ready when the application program turns control of a Virtual Tape Drive's State over to the Driver. In the process of turning over control, the Driver will turn on the Ready bit in the Virtual Tape Drive's State data structure. A drive enters Manual Control Mode when the Driver turns control of the Virtual Tape Drive's State over to the application program. This transfer of control is indicated by the Driver turning off the Ready bit. The application program must not attempt to change the state of drives that are not in Manual Control Mode.

6.2 Windows Interrupt Request Levels (IRQLs)

Windows defines several interrupt request priority levels. They are numbered 0 (lowest priority) through 31 (highest priority). These IRQLs are mapped to the underlying CPU's hardware interrupt levels. The levels that are significant to Driver operation are described below.

6.2.1 PASSIVE_LEVEL (IRQL 0)

Level 0 is known as the *PASSIVE_LEVEL*. All application programs and most Windows services operate at this level. The Windows task scheduler allocates time slices to the application programs and services. When the time slice expires or a thread "blocks" (needs to wait for an I/O operation to be completed), the scheduler preempts execution of the current thread and schedules a different program/service to be run. There is a thread priority system that the scheduler uses to select the next program/service to be executed. The priority system also determines the length of he time slice the application/service is given. This thread priority system only effects the activity of the scheduler and does not relate to execution on higher IRQLs.

When there is work to be performed on a higher IRQL, execution at any lower IRQL is immediately suspended (regardless of thread priority), and control of the CPU is given to the higher IRQL. Note that there is a difference between "suspending" execution and "preempting" execution. Preempting execution involves an order of magnitude more overhead and delay.

All application invocations of Driver methods begin on this IRQL. Invocations that perform USB I/O will eventually perform software interrupts to perform the actual I/O at the *DISPATCH_LEVEL*. As much work as possible is performed at the *PASSIVE_LEVEL*.

6.2.2 DISPATCH_LEVEL (IRQL 2)

Communication with the USB Bus occurs at this level. IRQLs 2 and above are reserved for time critical processing. The programming environment is extremely limited at these levels. Many of the operating system's services are not available to software operating at

IQL 2 and above. For example, the virtual memory paging service and file system are not available. All memory resources accessed must have been previously paged into real memory and locked. To address application memory, application buffers are mapped into the kernel's system address space.

6.2.3 IRQLs above the Dispatch Level

Microsoft (and programming best practices) directs that processing at these levels should be minimized. Typically, execution of a software routine at these levels is limited to less than a hundred microseconds. However, it is statistically possible that a confluence of I/O activity could delay Driver execution for several milliseconds. Limiting the I/O workload of the PC should prevent delays of this duration from occurring. The PC should not be tasked with handling a lot of network or disk drives.

6.3 User mode driver interface

The user mode driver (DLL) uses the below Microsoft C++ library routines to communicate with the Driver.

- CreateFile - Returns a HANDLE that provides access to the Tape Channel Analyzer.
- CloseHandle - Frees system resources associated with the Tape Channel Analyzer.
- DeviceIoControl - Communicates function requests to the driver.
- GetLastError - Returns additional information on failure of above routines.

Refer to the Microsoft documentation for more information.

The Adapter appears to the PC as a single device with a single input datastream and a single output datastream. Because of this, service requests from multiple application threads must be ordered and synchronized. This responsibility has been assigned to the application program. The application program should create a single thread, the Daemon thread, that manages I/O to the Adapter. All application program communication with the Adapter or the Driver should be performed via this thread. Data communication with the Driver is performed in "overlapped" mode so that the DLL is never blocked from execution.

The DLL performs CreateFile to establish a communication session with the Driver. The DLL performs CloseHandle to end the session. DeviceIoControl calls are used to communicate work instructions to the driver. If an error status is returned from the DeviceIoControl, GetLastError should be called to receive detailed information about the error.

The DeviceIoControl method passes an *IoControlCode* and some parameters to the driver. This *IoControlCode* specifies the function that the driver is requested to perform. The implemented *IoControlCode*s are:

- Adapter Reset (IOCTL_TXA_ADAPTER_RESET)
- Get Adapter Status (IOCTL_TXA_ADAPTER_GET_STATUS)
- Set Adapter Configuration (IOCTL_TXA_ADAPTER_SET_CONIG)
- Set Adapter Loopback Mode (IOCTL_TXA_ADAPTER_LOOPBACK)
- Drive Start (IOCTL_TXA_DRIVEx_START)

- Drive Reset (IOCTL_TXA_DRIVEx_RESET)
- Register Monitor Buffer (IOCTL_TXA_MONBUF_REG)
- Free Register Buffer (IOCTL_TXA_MONBUF_FREE)
- Watchdog Timer Configuration (IOCTL_TXA_WATCHDOG_CONFIG)

The DeviceIoControl method call provides for an input buffer and an output buffer. This is idea for our use, because the input buffer may be used to contain information that is to be sent to the Adapter, and the output buffer may be used to present the Adapter's response to the application. Both buffers are allocated and owned by the application program. The input buffer is a block of application memory whose contents is copied into a system buffer and the copy is presented to the Driver. The kernel permits the Driver to directly access the output buffer. For the period in which a DeviceIoControl is pending, this buffer can effectively be used as a block of shared memory between the application and the driver. The design of this driver makes use of this feature. For example, after a DeviceIoControl is executed to define a specific virtual tape drive. Any application thread may peek into the output buffer and monitor the progress/status of tape operations being performed between that 1401 and the Adapter. In order to use the shared memory feature, the DeviceIoControl method must be called in overlap mode so the DLL thread is not blocked and the Driver retains the ability to access the output buffer memory for extended periods.

When the DLL thread issues a DeviceIoControl, it should check the value DeviceIoControl returns. If the value is TRUE, then the DeviceIoControl has "immediately completed" successfully and the lpBytesReturned variable contains the number of bytes that were stored into the output buffer. If the value is FALSE, then the DLL must call GetLastError() to obtain more information.

Note that the hex values of the below codes are the ones that the kernel driver provides. They may be translated to other vales by the time they are delivered to the DLL.

Possible return values from GetLastError() after DeviceIoControl are: (from ntstatus.h)

- `STATUS_PENDING (0x00000103L)`
  This is not an error. It means that the DeviceIoControl is not yet complete and that the DLL should call GetOverlappedResult() if it wants to wait for the completion of the DeviceIoControl.
- `STATUS_NOT_IMPLEMENTED (0xC0000002L)`
  The Driver does not support the presented Ioctl code.
- `STATUS_BUFFER_TOO_SMALL (0xC0000023L)`
  Either the input or output buffers were smaller than the required size. If the output buffer caused the violation, then the data in the output buffer may have been truncated.
- `STATUS_INVALID_PARAMETER (0xC000000DL)`
  A data field provided in the input buffer is not valid.
- `STATUS_INVALID_USER_BUFFER (0xC00000E8L)`
  A data field provided by the DLL in the output buffer is not valid.

- `STATUS_UNRECOGNIZED_MEDIA (0xC0000014L)`
  An error was detected scanning the tape image part of the tape data buffer.
- `STATUS_DEVICE_NOT_CONNECTED (0xC000009DL)`
  The virtual tape drive specified in the request is not currently defined.
- `STATUS_INSUFFICIENT_RESOURCES (0xC000009AL)`
  The Driver was unable to obtain sufficient memory to perform the request. You probably need to reboot. This may be the result of a memory leak or extreme system load conditions.
- `STATUS_ILLEGAL_FUNCTION (0xC00000AFL)`
  The operation was illegal within the context of its use. This usually means that you have used an invalid combination of DeviceIoControl requests. Such as starting a drive that is already started or defining a buffer that is already defined.
- `STATUS_INTERNAL_ERROR (0xC00000E5L)`
  The Driver detected an internal state that should not have been possible.

When a DeviceIoControl request responds with `STATUS_PENDING`, upon completion GetOverlappedResult() may be used to obtain the completion status. If GetOverlappedResult() returns TRUE, then the associated DeviceIoControl has completed successfully and the lpNumberOfBytesTransferred variable contains the number of bytes that were stored into the output buffer. If the return value is FALSE, then the DLL must call GetLastError() to obtain more information.

Possible return values from GetLastError() after GetOverlappedResult() are: (ntstatus.h)

- `STATUS_PENDING (0x00000103L)`
  This is not an error. It means that the DeviceIoControl is still not yet complete.
- `STATUS_IO_TIMEOUT (0xC00000B5L)`
  The Adapter did not respond within the timeout limit.
- `STATUS_BUFFER_TOO_SMALL (0xC0000023L)`
  The output buffers was smaller than the required size. The data in the output buffer has been truncated.
- `STATUS_DEVICE_DATA_ERROR (0xC000009CL)`
  The response from the Adapter was garbled. Something was wrong with the message's framing (start flags, stop flags, or length field.)
- `STATUS_CRC_ERROR (0xC000003FL)`
  The response from the Adapter is framed correctly, but its CRC is incorrect.
- `STATUS_INSUFFICIENT_RESOURCES (0xC000009AL)`
  The Driver was unable to obtain sufficient memory to perform the request. You probably need to reboot. This may be the result of a memory leak or extreme system load conditions.
- `STATUS_INTERNAL_ERROR (0xC00000E5L)`
  The Driver detected an internal state that should not have been possible.
- `STATUS_ILLEGAL_FUNCTION (0xC00000AFL)`
  The operation was illegal within the context of its use. This usually means that you have used an invalid combination of DeviceIoControl requests. Such as starting a drive that is already started or defining a buffer that is already defined.

- If transmission of the outbound message failed within the USBD.sys driver, that driver's failure status will be returned. I don't know what failure statuses are likely to be returned by USBD.sys.

The below sections refer to "command specific data" of PC command messages and Adapter command responses. This refers to specific messages that will be sent/received on the USB bus. Please see the PC to Adapter Command Interface section of the Tape Channel Analyzer Specification for the message descriptions. When *structures* are referenced, the non-padded non-aligned little endian memory image of the *structure* is expected in the buffer.

The below codes begin execution in user mode at the Passive IRQ level within the application's context. As mush processing as possible is performed at this mode and level. If a specific code needs to perform USB I/O, then the I/O is enqueued and performed at the Dispatch IRQ level in kernel mode.

In a typical session, it is envisioned that an application GUI (graphic user interface) thread will allocate and initialize a "tape data buffer", then request the DLL to perform a "Drive Start" to place the virtual drive in "Ready" state. This is equivalent to pressing the "Start" button on a real IBM 729 tape drive. This tape data buffer would be presented to the Driver as a DeviceIoControl shared output buffer. The GUI thread would then monitor the Virtual Tape Drive State region of the tape data buffer and update its display to reflect this state. If the GUI thread wants to place its drive in Manual Control Mode (equivalent to pressing the Reset button on the IBM 729 tape drive) it requests the DLL to perform a "Drive Reset" DeviceIoControl procedure.

The Driver may change the state of a drive from Ready to Manual Control Mode based upon action occurring inside the Adapter, but the only way to place a drive in Ready state is via the "Drive Start" DeviceIoControl procedure.

6.3.1 Adapter Reset

This code resets the state of the Driver and the Adapter. All Adapter related resources in the Driver are reinitialized. All virtual tape drives become "undefined" and their resources are freed. Tape drive data buffers are unlocked and dereferenced by the driver, but not freed. (The application is responsible for the freeing or reusing the buffers.) All pending DeviceIoControl are *completed* and will return "Canceled" status.

A "Full Reset" command is sent to the Adapter. The command response is placed into the output buffer.

lpInBuffer:	NULL
nInBufferSize:	0
lpOutBuffer:	Pointer to response buffer. Size should be at least 15 bytes. Contents are set by the Driver.
nOutBufferSize:	Size of the above buffer.
lpBytesReturned:	Set by Driver. Meaningful only if operation is immediately completed.

lpOverlapped:	Should be set to the address of an Overlap block. This is required because the device is opened with overlap permitted. However, the DLL thread should immediately wait for overlap completion. Once this DeviceIoControl is issued, no other DeviceIoControls will be accepted until this one completes.

If GetOverlappedResult() returns TRUE, then the DeviceIoControl was successful and the output buffer contains the below information.

Output buffer contents: (valid only after overlap completion)

bytes 0-3	the response timestamp.
bytes 4-14	Adapter_status structure

The effect of this code is severe. It effects all Virtual Drives. It should only be used to reset the Adapter after diagnostic mode sessions or if the Adapter appears to be hung. The casual user should not be permitted to issue it.

6.3.2 Get Adapter Status

This code sends a "Get Analyzer Configuration" command to the Adapter. The command response is placed into the output buffer.

lpInBuffer:	NULL
nInBufferSize:	0
lpOutBuffer:	Pointer to response buffer. Size should be at least 36 bytes. Contents are set by the Driver.
nOutBufferSize:	Size of the above buffer.
lpBytesReturned:	Set by Driver.
lpOverlapped:	Should be set to the address of an Overlap block. This is required because the device is opened with overlap permitted. However, the DLL thread should immediately wait for overlap completion. Once this DeviceIoControl is issued, no other DeviceIoControls will be accepted until this one completes.

Output buffer contents:

bytes 0-3	the response timestamp.
bytes 4-13	Adapter_status structure
bytes 15-35	Adapter_configuration structure

6.3.3 Set Adapter Configuration

This code sends a "Set Analyzer Configuration" command to the Adapter. The input buffer should be set to the command specific data for this command. The command response is placed into the output buffer.

lpInBuffer:	Pointer to input buffer. Size should be at least 20 bytes.
nInBufferSize:	21
lpOutBuffer:	Pointer to response buffer. Size should be at least 16 bytes. Contents are set by the Driver.
nOutBufferSize:	Size of the above buffer.
lpBytesReturned:	Set by Driver.
lpOverlapped:	Should be set to the address of an Overlap block. This is required because the device is opened with overlap permitted. However, the DLL thread should

immediately wait for overlap completion. Once this DeviceIoControl is issued, no other DeviceIoControls will be accepted until this one completes.

Input buffer contents:

bytes 0-20	Adapter_configuration structure

Output buffer contents:

bytes 0-3	the response timestamp.
bytes 4-14	Adapter_status structure

6.3.4 Set Adapter Loopback Mode

This code sends a "DiagUpload Loopback" command to the Adapter. The input buffer should be set to the command specific data for this command. The return code from command response is placed into the output buffer.

Before this code is executed, a "Drive Start" code should be executed to define the Virtual Tape Drive to be used for this diagnostic and provide the tape data buffer for the records that will be uploaded.

Once a "Set Adapter Loopback Mode" code is executed, the Adapter will be held in diagnostic loopback mode until an "Adapter Reset" code is executed. IBM 1401 I/O to the Adapter is ignored while it is in loopback mode.

lpInBuffer:	Pointer to input buffer. Size should be sufficient to hold the contents. (described below)
nInBufferSize:	Size of the above buffer.
lpOutBuffer:	Pointer to response buffer. Size should be at least 7 bytes. Contents are set by the Driver.
nOutBufferSize:	Size of the above buffer.
lpBytesReturned:	Set by Driver.
lpOverlapped:	Should be set to the address of an Overlap block. This is required because the device is opened with overlap permitted. However, the DLL thread should immediately wait for overlap completion. Once this DeviceIoControl is issued, no other DeviceIoControls will be accepted until this one completes.

Input buffer contents:

bytes 0-n	See the definition of the command specific data for the Diag Upload Loopback command.

Output buffer contents:

bytes 0-3	the response timestamp.

6.3.5 Drive Start

This code defines a Virtual Tape Drive, configures it, and places it in Ready state. There are six Drive Start Ioctls, one for each drive address. The input buffer describes the type of drive being defined and the drive's initial state. The output buffer consists of a block of memory bytes containing the Virtual Tape Drive's state followed by the contents of the tape.

lpInBuffer:	Pointer to input buffer. Size should be sufficient to hold the contents. (described below)
nInBufferSize:	Size of the above buffer.
lpOutBuffer:	Pointer to response buffer. Size should be sufficient to hold the contents. (described below)
nOutBufferSize:	Size of the above buffer. Should be at least 4k.
lpBytesReturned:	Set by Driver. to indicate the used portion of the output buffer.
lpOverlapped:	Should be set to the address of an Overlap block. This operation is intended to be overlapped.

Input buffer contents:

byte 0	Drive model (See the "Define Virtual Drive" command specific data)
bytes 1-6	Initial Virtual tape drive state structure (See the "Define Virtual Drive" command specific data). Only the tape_position field and certain bits in the flag field are used. The used flag field bits are high/low density, tape indicator, and write enabled.

Output buffer contents: - Note that while the "Drive Start" DeviceIoControl completion is "pending". The contents of the output buffer may change at any time.

bytes 0-3	the response timestamp of the latest update of the virtual tape drive state structure.
bytes 4-5	response sequence number. This field is used to uniquely identify state updates when multiple updates occur within one tick of the timestamp. The first update received with a timestamp receives a sequence number of zero. The sequence number is increment for subsequent updates.
bytes 6-11	Virtual tape drive state structure (See the "Define Virtual Drive" command specific data). This is the current state of the virtual tape drive and may be asynchronously sampled by the GUI thread to update its status displays.
bytes 12-13	Error count - a count of the number of errors that have occurred during the tape session. If this field is non-zero, the tape may be corrupted.
bytes 14-15	Error info - the diagnostic code of the last error that occurred. This field is set when the error count is incremented.
bytes 16-31	Reserved for Driver use
bytes 32-n	The contents of the emulated magnetic tape. This region should only be accessed by the application while the tape drive is in Manual Control Mode. This region may be several megabytes in size. (The typical 2400 foot tape stores about 20 Megabytes of data.) BCD 7-bit characters are right justified within the 8-bit bytes of the buffer. The most significant bit should be set to zero (0-C-B-A-8-4-2-1). Each tape record should be prefixed and suffixed with a four byte length field (Intel x86 unaligned 32-bit integer format). These fields should be set to the number of data characters in the record plus 8. Four bytes of zeros should follow the last record on the tape. When the buffer is nearly consumed, an End-Of-Reel condition will be presented to the 1401.

If an empty (scratch) tape is mounted, bytes 12-15 should be set to zeros (0x00000000). If these bytes are detected to be zeros, the tape_position field will be set to zero.
If a tape with data is mounted, bytes 12-n should contain the tape's data records in the format described above. A four byte field of zeros should follow the last record.

When the DLL calls the Driver through the "Drive Start" DeviceIoControl method entry point:

1. The DLL's thread is blocked until the Driver returns.
2. The Driver initializes the first 12 bytes of the output buffer and validates the contents of the input and output buffers. Validation includes checking the linkage of prefix and suffix record lengths; and checking that the tape position field contains a number within the bounds of the tape. (The tape position field value of zero means that the tape is "at load point".) If validation fails, the DeviceIoControl will return with error status. Otherwise it continues to the next step.
3. The Driver transmits a Define Virtual Drive command is to the Adapter and waits for the response. If the command fails, the DeviceIoControl will return with error status. Otherwise it continues to the next step.
4. The Driver returns with the special error code *ERROR_IO_PENDING*. Microsoft defines this code to mean "no error occurred (duh), but the DeviceIoControl has not completed its work". This permits the application program to continue execution in parallel with the Driver. The Driver will stay active until the virtual tape drive leaves Ready state.
5. The DLL notifies the GUI thread that its connection to the 1401 is live and that it should start monitoring the state of its virtual drive.

The GUI thread monitors the status of its Virtual Drive by periodically examining the bytes 0-11 of the output buffer. The recommended process for determining a change in virtual drive status is:

1. Compare the timestamp and response sequence number (bytes 0-5 of the output buffer) to the previously recorded values. If they are different, there has been virtual tape drive activity and the virtual drive's state may have changed. If they are different continue to the next step. (Otherwise sleep and check again later.)
2. Sample the data by copying output buffer bytes 0-11 into a local variable(s). (Bytes 0-5 of this local variable are the "previously recorded values" referred to in step 1.)
3. Compare the contents of this local variable to bytes 0-11 of the output buffer. This step detects if the Driver changed the contents of the output buffer while the application was sampling it. If the contents have changed, then go to step 2. Otherwise continue to the next step.
4. Use the data in the local variable to update the GUI display.
5. If the sampled virtual tape drive state indicates that the drive is in Manual Control Mode, then:
   a) An "Undefine Virtual Drive" command has been sent to the Adapter.
   b) The pending "Drive Start" DeviceIoControl has completed (completion status is marked in the Overlap block).
   c) The Driver has dereferenced the output buffer (the Driver can no longer change the buffer).
   d) The application may now process the tape data in the buffer.

By definition, the "Drive Start" DeviceIoControl will complete when the Virtual Drive enters Manual Control Mode (becomes not Ready). Either the application program or the 1401 may cause the Virtual Drive to enter Manual Control Mode. The application program may force the Virtual Tape Drive to enter Manual Control Mode by executing "Drive Reset" or "Adapter Reset" DeviceIoControl codes.

The 1401 may cause the virtual tape drive to enter Manual Control Mode via a "Rewind and Unload" instruction or via reading/writing beyond the end of the reel. During read operations, End-Of-Reel (EOR) is signaled when the Adapter attempts to read a record that has a zero length prefix. During write operation, EOR is signaled if there is no room for the data in the tape data region of the output buffer. EOR will also be signaled if the Adapter issues a "Download Underflow" or "Upload Overflow" events. In this case the Virtual Drive's state *tape position* field specifies the record after the last one successfully read or written by the 1401. If EOR is being signaled, any tape operation that would result in forward tape movement will cause the drive to enter manual Control Mode. Any tape operation that results in backwards tape movement would clear the EOR condition.

6.3.6 Drive Reset

This code emulates the pressing of the Reset button on an IBM 729 tape drive. It sets the drive to Manual Control Mode (also known as not Ready state). There are six Drive Reset Ioctl codes, one for each drive address. If a "Drive Start" code is pending for the addressed drive, then that code will be forced to complete.

lpInBuffer:	Null.
nInBufferSize:	0.
lpOutBuffer:	Null.
nOutBufferSize:	0.
lpBytesReturned:	Set by Driver.
lpOverlapped:	Should be set to the address of an Overlap block. This is required because the device is opened with overlap permitted. However, the DLL thread should immediately wait for overlap completion. Once this DeviceIoControl is issued, no other DeviceIoControls will be accepted until this one completes.

Output buffer contents:

bytes 0-3	the response timestamp.
byte 4	Adapter return code

6.3.7 Register Monitor Buffer

This code provides the Driver with a buffer to be used for monitoring tape channel transactions. Once the buffer is provided to the Driver, the application should use "Set Adapter Configuration" to control the monitoring function. The output buffer defined for this code is used as a shared circular buffer. This buffer becomes shared memory between the application program and the Driver.

lpInBuffer:	Null.
nInBufferSize:	0
lpOutBuffer:	Pointer to output buffer. The size of the buffer must be a multiple of 32 bytes and aligned on a 32 byte boundary. Ideally, the buffer size should be an integer number of full pages.
nOutBufferSize:	Size of the above buffer.
lpBytesReturned:	Set to 0 by Driver.
lpOverlapped:	Should be set to the address of an Overlap block. This operation is intended to be overlapped.

Output buffer contents: (The buffer is formatted into 32 byte entries.)

Entry 0	
bytes 0-3	HEAD, the byte offset of the next available entry in the buffer. This field must not be changed or even stored into by the application program. The HEAD field is advanced by the ISR. Rather than overwrite unprocessed data in the buffer, the ISR will discard new entries if there is not room for them. (Intel x86 unaligned 32-bit integer format)
bytes 4-7	TAIL, the byte offset of the next entry to be processed. The application program advances the TAIL field. (Intel x86 unaligned 32-bit integer format) When HEAD equals TAIL, then the buffer is empty.
bytes 8-31	unused.

Entries 1-n	
bytes 0-3	the response timestamp.
bytes 4-18	see the definition of the command specific data for the "Channel Transaction" event.
byte 19	if 0 then entries are consecutive. Else one or more transactions immediately preceding this one were discarded.
bytes 23-31	unused.

To receive a new transaction to be processed:

1. The application program should copy the HEAD and TAIL fields into local variables (lclHEAD and lclTAIL). The HEAD value must be read in its entirety in one atomic operation. Since HEAD is an aligned 32-bit integer, the compiler should automatically do this.
2. If lclHEAD equals lclTAIL, then there are no new transactions to be processed at this time. Otherwise continue to the next step.
3. The application program should process the buffer entry identified by lclTAIL.
4. After processing the entry, the application should add 32 to lclTAIL. If the resulting value is greater than the buffer's size, then lclTAIL is set to 32.
5. lclTAIL is then stored back into TAIL. The store must be performed in its entirety in one atomic operation. Since TAIL is an aligned 32-bit integer, the compiler should automatically do this.

6.3.8 Free Monitor Buffer

This code commands the Driver to dereference the previously provided monitor buffer and causes the "Register Monitor Buffer" DeviceIoControl to complete (completion status is marked in the Overlap block).

If the Adapter sends transaction data, and there is no buffer provided, the data is discarded.

No parameters are passed with this code.

6.3.9 Set Watchdog Timer

This code controls the watchdog timer feature of the Driver. The timer is used to detect loss of communications with the adapter. Normally, the watchdog timer permits the

Driver to allow a specified period of time for the Adapter to respond to a command message. After that period of time, the watchdog timer causes the Driver to cancel the action. However, during debug sessions, it may be useful to permit the Driver to wait indefinitely.

Having been prompted into action by the watchdog timer, the Driver will retry an Adapter command twice (a total of three attempts), after which it will assume that the USB connection with the Adapter has been severed. The Windows Plug-n-Play feature may also inform the Driver of a planned or surprise removal of the USB connection to the Adapter. In either case, the Driver will:

1. Set all defined drives to Manual Control Mode
2. Force error completion of all pending DeviceIoControl codes.
3. Free all resources.
4. Reject all attempts to call new DeviceIoControl codes except "Adapter Reset".
5. Other DeviceIoControl codes will remain disabled until a successful response to "Adapter Reset" is received.

Note that the kernel watchdog timer operates in one-second ticks. Since the start of the tick is not synchronized with the transmission of a command message, the minimum usable timeout interval is 1.5 seconds plus or minus 0.5 seconds.

lpInBuffer:	Pointer to input buffer. Size should be at least 4 bytes.
nInBufferSize:	Size of the above buffer.
lpOutBuffer:	Null.
nOutBufferSize:	0
lpBytesReturned:	0
lpOverlapped:	Should be set to the address of an Overlap block. This is required because the device is opened with overlap permitted. However, the DLL thread should immediately wait for overlap completion. Once this DeviceIoControl is issued, no other DeviceIoControls will be accepted until this one completes.

Input buffer contents:

byte 0-3	number of ticks Driver can wait for a response to a command before taking recovery action. 0 means wait indefinitely. The number of ticks should be stored as a 32-bit unsigned integer. (Intel x86 unaligned 32-bit integer format)

6.3.10 Diag Raw Message

This code is for diagnostic purposes only. It sends the contents of the input buffer to the Adapter. The input buffer data is expected to contain all required framing and control characters. The Adapter's response is placed into the output buffer.

lpInBuffer:	Pointer to input buffer. Size should be greater than zero.
nInBufferSize:	Number of characters in the above bufer.
lpOutBuffer:	Pointer to response buffer. Size should be at least 16 bytes. Contents are set by the Driver.
nOutBufferSize:	Size of the above buffer.
lpBytesReturned:	Set by Driver.
lpOverlapped:	Should be set to the address of an Overlap block. This is required because the device is opened with overlap permitted. However, the DLL thread should

immediately wait for overlap completion. Once this DeviceIoControl is issued, no other DeviceIoControls will be accepted until this one completes.

6.4 Interface to the USB Bus driver

Windows provides a number of lower level drivers that are collectively referred to here as the USB Bus driver (USBD.SYS). All access to the physical USB Bus is performed through these drivers.

The Driver makes read or write requests to the USBD.sys driver by calling it. An important parameter to this call is the address of a subroutine, in the Driver, that USBD.sys is supposed to call when the requested operation is complete. This subroutine is called the "completion callback routine". It is important to note that the invocation is performed in kernel mode at the Dispatch IRQ level.

This has good and bad consequences. The good news is that service requests from the USB Bus driver get priority over all CPU activity except higher priority interrupt service routines belonging to other devices. There is no "scheduling delay". The bad news is that many of the operating systems services are unavailable to the ISR. For example, the virtual memory paging service and file system are not available. All memory resources used by the Driver's completion callback routine must have been previously paged into real memory and locked. Memory resources must be addressed using real addresses. Note also that the "completion callback routine" must presume that it is being executed in a random context, rather than in its own application's context.

6.5 Interrupt request packet handling

Every call to DeviceIoControl creates an interrupt request packet (Irp) that contains the parameters of the request and within which must be stored the final completion status. Once created, the Irp moves through the Driver as progress is made on satisfying the DeviceIoControl request. When work on the DeviceIoControl request is complete, the Driver stores the completion status into the Irp and calls the completion callback routine that the I/O Manager provided in the Irp. This completion callback routine takes the actions necessary to signal the application program, waking it up if necessary, that the DeviceIoControl is complete.

If the Irp was to get lost, then the application program will never see completion of its DeviceIoControl request. Management of Irps was a major task in the architecture of the Driver. This section describes how Irps move through the Driver.

Many Irps may be under management of the Driver at any instant. When the Adapter is attached to the PC, the Driver creates two "Hot Read" Irps. These Irps are unrelated to any application requests. These Irps are constantly circulated down to the USBD.sys driver to read USB packets coming from the Adapter. The reason for having two is so that one is always pending in USBD.sys while the Driver is processing the other. Thus the Adapter is always being listened to. The Driver matches up data received by these Hot Read Irps with application generated DeviceIoControl Irps.

When DeviceIoControl is called, the Windows I/O Manager creates an Irp, switches to kernel mode, and calls the Driver's DispathDeviceControl routine. The first action of this routine is to validate the parameters that were specified in the DeviceIoControl. This initial Driver thread is executed in the application's context. If the Driver can immediately complete the DeviceIoControl request, it does so, stores status into the Irp, and calls the provided completion callback routine.

Unfortunately few DeviceIoControl requests can be completed immediately. Most DeviceIoControl requests require one or more USB data packets to be sent to the Adapter. Then USB data packets must be sent from the Adapter to the PC. The time it takes to perform this sequence is much much much longer than the Driver is permitted to keep any thread active, especially since we want to perform all this processing at the DISPATCH_LEVEL IRQL.

To solve this problem, the DeviceIoControl request is performed in a series of steps. Whenever the Driver needs to wait for something to occur, the Driver parks the Irp in a known location and relinquishes the thread. Later, when the "something" occurs, the Driver is given a new (random context) DISPATCH_LEVEL thread, the Driver perform its next step of work, reparks the Irp and relinquishes the thread.

The stage of completion for each Irp is known to the Driver by the place where the Irp is parked. The DeviceIoControl Irp parking spots are:

- the command Irp location - Adapter Command in progress
- the outbound Irp queue - future Adapter Commands
- the monitor Irp location - active transaction monitoring Irp
- six virtual drive Irp locations - active Virtual Tape Drive Irps

6.5.1 A simple command and response request

Let's consider a DeviceIoControl request that will send 4 packets of data to the Adapter and then receive 3 packets of data from the Adapter. Let's also assume that the Driver is idle (all parking spots empty) at the time of this request.

DispathDeviceControl determines that the "command Irp location" is empty, meaning that the current DeviceIoControl request can begin packet transfers to the Adapter. The first packet of data is placed in a buffer to give USBD.sys. (The Irp contains a field that contains the number of bytes to be transferred to the Adapter. If this amount is greater than the device's maximum transfer size, the buffer must be split into multiple separate transfers, known packets.) USBD.sys is called to begin the transfer. USBD.sys sets up and starts a DMA (hardware controlled) data transfer, and returns presenting STATUS_PENDING (not yet complete). The Driver sees that status, and parks the Irp in the command Irp location, and returns. Since we performing the DeviceIoControl requests in overlapped mode, the application is now free to continue execution.

After the DMA hardware finishes its transfer, the hardware interrupts the processor. USBD.sys services this interrupt and directly calls the Driver. The Driver retrieves the Irp parked in the "command Irp location", sees that more data needs to be sent, places the next packet of data in the buffer, and calls USBD.sys to begin the transfer. USBD.sys sets up and starts the DMA (hardware controlled) data transfer, and returns presenting STATUS_PENDING (not yet complete). The Driver sees that status, and returns.

The above paragraph repeats while there is still more data to be transmitted. If the application makes another DeviceIoControl request while the while these outbound transfers are occurring, the DispathDeviceControl will see that the "command Irp location" is occupied and place the new Irp in the " outbound Irp queue" and return with STATUS_PENDING to the new request. Again, the application is free to continue execution.

Eventually, DMA hardware finishes its transfer, the hardware interrupts the processor. USBD.sys services this interrupt and directly calls the Driver. The Driver retrieves the Irp parked in the "command Irp location", sees that no more data needs to be sent, but the request is not yet complete, because the response must be received from the Adapter. So, the Irp is left pending and parked; and the Driver returns.

Similar activity is occurring on the USB's inbound pipe. An inbound DMA hardware transfer finishes (the result of a Hot Read Irp call to USBD.sys), the hardware interrupts the processor. USBD.sys services this interrupt and directly calls the Driver. The Driver determines that the inbound message has not yet been completely received, and calls USBD.sys to read the next packet of data. USBD.sys sets up and starts the DMA (hardware controlled) data transfer, and returns presenting STATUS_PENDING (not yet complete). The Driver sees that status, and returns.

Eventually, DMA hardware finishes its transfer, the hardware interrupts the processor. USBD.sys services this interrupt and directly calls the Driver. The Driver determines that the inbound message has now been completely received. The Driver matches the received message with the command sent by the Irp parked in the "command Irp location", the Irp is removed from its parking spot, status and the response data is stored into the Irp, and its completion routine is called (thus retiring the original DeviceIoControl request). The Driver checks to see if any Irps are in the "outbound Irp queue". If there are, the one at the head of the queue is removed, parked in the "command Irp location", places the first packet of data in the outbound buffer, and calls USBD.sys to begin the transfer. USBD.sys sets up and starts the outbound DMA data transfer, and returns presenting STATUS_PENDING. The Driver then calls USBD.sys again to reestablish this Hot Read. USBD.sys sets up and starts the inbound DMA data transfer, and returns presenting STATUS_PENDING. The Driver sees that status, and returns.

Notice that all of the processing that is initiated by a DMA completion interrupt takes place at DISPATCH_LEVEL IRQL. This architecture ensures that Window's services can not slow the Driver's performance, regardless of the scheduling priorities given to those services. The Driver's is structured so that all time critical work is performed in this manner.

6.5.2 Register/Free Monitor Buffer Irp handling

Register Monitor Buffer (IOCTL_TXA_MONBUF_REG) and Free Monitor Buffer (IOCTL_TXA_MONBUF_FREE) do not follow the above flow because they do not need to communicate with the Adapter.

When Register Monitor Buffer is requested, the DispathDeviceControl routine parks this Irp in the "monitor Irp location" and returns the STATUS_PENDING status code, freeing the application to continue execution. By leaving this Irp parked, instead of retiring it, permits the Driver continued access to the output buffer that was specified in the IOCTL_TXA_MONBUF_REG DeviceIoControl request. The driver and the application program can share the data in this output buffer.

A circular transaction buffer is defined in the output buffer. Whenever the Adapter sends a "Channel Transaction" event message, the Driver inserts the transaction data into this circular buffer.

When the DLL executes a Free Register Buffer (IOCTL_TXA_MONBUF_FREE) DeviceIoControl request, the DispathDeviceControl routine unparks the Irp saved in the "monitor Irp location" (clearing that location), stores success status into that Irp, and calls that Irp's completion routine. By doing this the original Register Monitor Buffer (IOCTL_TXA_MONBUF_REG) DeviceIoControl request is retired. The DispathDeviceControl routine then returns with status success to retire the Free Register Buffer request.

6.5.3 Device Start/Reset Irp handling

The Drive Start (IOCTL_TXA_DRIVEx_START) request starts to follow the "simple command and response request" flow. It sends a "Define Virtual Drive" command message to the Adapter and is parked until the Adapter's response is received. But, after the Irp is removed from the "command Irp location", instead of being retired, it is parked in the "virtual drive Irp location" that corresponds to the virtual drive that was defined.

Whenever the 1401 performs tape operation, these tape operations are communicated to the Driver via event messages, and the Driver moves data to or from the Tape Buffers that are defined in the output buffers of Irps parked in the "virtual drive Irp locations". Irps parked in "virtual drive Irp locations" remain pending until the Adapter sends an event that places the drive in manual control mode (such as Unload) or until the DLL performs a "Device Reset" DeviceIoControl request.

When the DLL executes a Device Reset (IOCTL_TXA_DRIVEx_RESET) DeviceIoControl request, the DispathDeviceControl routine unparks the Irp saved in the specified "virtual drive Irp location" (clearing that location), stores success status into that Irp, and calls that Irp's completion routine. By doing this the original "Drive Start" DeviceIoControl request is retired. The DispathDeviceControl routine then returns with status success to retire the Device Reset request.

6.5.4 Handling Adapter events

When the 1401 performs tape operations to the Adapter, the Adapter informs the Driver about the activity by sending event messages to the Driver. These events messages are received by one of the "Hot Read" Irps, processed as described below. To demonstrate this the flow of an Upload Request event is described.

An inbound DMA hardware transfer finishes (the result of a Hot Read Irp call to USBD.sys), the hardware interrupts the processor. USBD.sys services this interrupt and directly calls the Driver. The Driver determines that the inbound message has now been completely received. The Driver sees that the received message is an event and matches it with an Irp parked in the "virtual drive Irp location". The Driver updates the "virtual drive Irp's" virtual drive state (a block of data located in the output buffer), and makes an internal DeviceIoControl request for an Upload Record command to be sent to the Adapter. The internal DeviceIoControl request is the same as the DeviceIoControl request that the application performs, except...

- it is intended for driver to driver calls
- it executes at DISPATH_LEVEL
- if the request needs to be placed in the "outbound Irp queue" these requests are inserted at the head of the queue (giving them higher priority than the application requests)

After making the internal DeviceIoControl request, the Driver calls USBD.sys to reestablish the Hot Read and returns.

The generated "Upload Record" Irp will flow through the Driver as described in the "simple command and response request" flow. When its Adapter response is received, the record data is moved into the specified tape buffer, and the request is retired.