1.1 Windows 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 - The main module containing driver entry, add device, and major function dispatch routines.
- tapepnp - The plug and play interface routines.
- tapepwr - The power management interface routines.
- tapewmi - The Windows management and instrumentation routines.
- tapedev - The device control routines (create, close, I/O control, and selective suspend). This module implements the application program's interface to the Tape Channel Analyzer.
- tapeisr - The interrupt service routines. These routines perform the time critical Tape Channel Analyzer functions. They send commands to the Adapter and respond to Adapter events.

The tapedev and tapeisr modules contain the project specific logic for the Tape Channel Analyzer. The other modules are 98% unchanged from the samples in the Windows Driver Development Kit (DDK).

1.1.1 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.

1.1.2 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.

1.1.2.1 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*.

1.1.2.2 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. Memory resources must be addressed using real addresses and the

software must be aware that buffers that appear to be contiguous memory locations to the application program will appear to the ISR as spanning discontinuous pages.

1.1.2.3 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.

1.1.3 Application program interface

The application program 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. Certain data communication with the Driver are performed in "overlapped" mode so that the Daemon is not blocked.

The Daemon performs CreateFile to establish a communication session with the Driver. The Daemon 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_DRIVE_START)
- Drive Reset (IOCTL_TXA_DRIVE_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 two user-defined parameters. In Microsoft documentation the parameters are referred to as the input and output buffers. 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. At this time, we are making an assumption that the kernel does not prevent the application from continuing to access the output buffer while the DeviceIoControl method is in progress. The design of the above set of *IoControlCode*s (codes) depends on the ability of the Application and Driver to treat regions of the output buffer as shared memory. When the shared memory feature is to be utilized, the DeviceIoControl method should be called in overlap mode so the Daemon thread is not blocked and the Driver retains the ability to access the output buffer memory.

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 request the Daemon 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. The GUI thread would then monitor the Virtual Tape Drive State region of the output 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 Daemon 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.

1.1.3.1 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 16 bytes. Contents are set by the Driver.
nOutBufferSize:	Size of the above buffer.
lpBytesReturned:	Set by Driver.
lpOverlapped:	Null. This operation should not be overlapped.

Output buffer contents:

bytes 0-3	the response timestamp.
bytes 4-5	response sequence number (will be 0 for this code).
bytes 6-15	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.

1.1.3.2 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:	Null. This operation should not be overlapped.

Output buffer contents:

bytes 0-3	the response timestamp.
bytes 4-5	response sequence number (will be 0 for this code).
bytes 6-15	Adapter_status structure
bytes 16-35	Adapter_configuration structure

1.1.3.3 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:	Size of the above buffer.

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:	Null. This operation should not be overlapped.

Input buffer contents:

bytes 0-19	Adapter_configuration structure

Output buffer contents:

bytes 0-3	the response timestamp.
bytes 4-5	response sequence number (will be 0 for this code).
bytes 6-15	Adapter_status structure

1.1.3.4 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:	Null. This operation should not be overlapped.

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.
bytes 4-5	response sequence number (will be 0 for this code).
byte 6	response return code.

1.1.3.5 Drive Start

This code defines a Virtual Tape Drive, configures it, and places it in Ready state. 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.
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 address (1-6)
byte 1	Drive model (See the "Define Virtual Drive" command specific data)
bytes 2-7	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-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 Daemon calls the Driver through the "Drive Start" DeviceIoControl method entry point:

1. The Daemon'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 Daemon 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.

1.1.3.6 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). If a "Drive Start" code is pending for the addressed drive, then that code will be forced to complete.

lpInBuffer:	Pointer to input buffer. Size should be at least 1 byte.
nInBufferSize:	Size of the above buffer.
lpOutBuffer:	Null.
nOutBufferSize:	0.
lpBytesReturned:	Set by Driver.
lpOverlapped:	Null. This operation should not be overlapped.

Input buffer contents:

byte 0	Virtual tape drive address (1-6)

1.1.3.7 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.

1.1.3.8 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.

1.1.3.9 Watchdog Timer Configuration

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 wait 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 take 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:	Null. This operation should not be overlapped.

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)

1.1.4 Interface to the USB Bus drivers stack

This Driver's Interrupt Service Routine (ISR) is used to send and receive messages on the USB Bus. 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 ISR is invoked by the USB Bus driver upon the receipt of a message from the adapter. 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 ISR must have been previously paged into real memory and locked. Memory resources must be addressed using real addresses and the ISR must be aware that buffers that appear to be contiguous memory locations to the application program will appear to the ISR as spanning discontinuous pages. Note also that the ISR must presume that it is being executed in a random context, rather than in its own application's context.

The ISR is also invoked by this driver's tapedev module's DeviceIoControl routines to send commands to the adapter. These requests may be rejected if the ISR is unable to immediately service the request. This will occur if the ISR has sent a command to the Adapter, but the ISR has neither received the command's response nor has the command timed out. When this occurs the DeviceIoControl routine should wait a couple milliseconds, check the Adapter's device status, then if it still desires to send the message, it should retry its invocation.

Every command message that is sent to the Adapter is maintained until command's response is received or until the Driver chooses to abort waiting for the response.