WA8DED HOST MODE USER'S GUIDE

Original by Paul T. Williamson, KB5MU, 23 May 1986
Updated by Peter Guelzow, DB2OS, 8 Aug 1993

Abstract: This document explains the use of the WA8DED Host mode with WA8DED's AX25 Version 2 Multi-channel TNC Firmware, Version 1.1 for the TAPR TNC-1 and the TheFirmware TF Version 2.x for the TNC-2 by NORD><LINK e.V. TheFirmware TF is also available for the TNC-3 (68302) Hardware.

It is based on the information provided in the WA8DED document (Version 1.0), the W6IXU mailbox software (13 May 1986 revision), and the WA8DED Split Screen Terminal Emulator (SS, 28 April 1986 version). It is intended for use by an application programmer attempting to communicate with the TNC in Host mode, and assumes that the programmer is already familiar with user-mode operation of the WA8DED firmware and TheFirmware.

Chapter 1
What Is Host Mode?

WA8DED's TNC firmware provides two entirely independent user interfaces. The default user interface is designed for use by a human operator at a terminal. Host mode, on the other hand, is designed for use by a specially-designed computer program. It allows the computer to maintain strict control of the computer-TNC link, and to obtain control information that would otherwise be difficult to extract from the normal data stream.

One of the most common bugaboos of computer-TNC interfacing has always been flow control. The TNC is capable of sending data faster than the computer can process it, and likewise the computer is capable of sending data faster than the TNC can transmit it. It is theoretically possible to solve this problem with either hardware flow control (RTS/CTS wires in the RS-232 link) or software flow control (XON/XOFF characters inserted in the data stream). Both of these methods are tricky and error-prone, however. Hardware flow control may not always be available, and software flow control, even when it works adequately, leads to data transparency problems (i.e., what if the data being transmitted contains XON or XOFF?). If everything is not exactly right, data will be lost.

WA8DED's Host mode solves this problem by

1. Speaking only when spoken to, and

2. Limiting all data bursts to 256 bytes.

The first point allows the computer program to go off and do other things (write data out to disk, handle the console, make coffee) without worrying about data coming from the TNC and dropping on the floor. The second point allows computers that have a hard time handling fast serial data (e.g., a Commodore 64 with no UART) to concentrate on the receiving task to fill a 256 byte buffer, then go off and process the data without worrying about additional data arriving before it has time to process that batch. This scheme is very forgiving of inefficient serial routines, so that faster computers can get away with simple byte-level polling (e.g., a program for the IBM PC can use the BIOS serial port routines).

On the subject of serial routines, notice that the TNC does not handle either hardware or software flow control when in Host mode. You must make sure that your serial drivers will not send unwanted XON and XOFF characters to the TNC.

Since the TNC speaks only when spoken to, the computer must constantly poll the TNC to find out what is happening on the link. In user mode, connected data received over the AX.25 link is immediately displayed (if that channel is selected). In Host mode, the TNC is not allowed to send information unsolicited, so the program must ask the TNC for data periodically. This

implies that the computer-TNC link will be busy most of the time. Frequent polling and a relatively high baud rate (say 4800 or 9600 baud) are desirable for applications that require quick response times. A user monitoring data in real time will notice delays if the delay between polls is too great.

The format of these short computer-TNC exchanges is carefully specified to eliminate ambiguity and minimize overhead. The computer sends to the TNC in the following format:

{channel}{info/cmd}{count}{data...}

Channel is the channel number to which the transmission pertains. Info/cmd distinguishes between information to be transmitted and commands to be acted upon by the TNC. Count specifies the number of bytes to follow, and Data... are the actual data bytes or command bytes. The details of this format will be discussed later. All computer-to-TNC transmissions must follow this format.

The TNC sends to the computer one of several different formats:

{channel}{code=0} short format

or

{channel}{code=1-5}{data...}{0} null-terminated format

or

{channel}{code=6-7}{count}{data...} byte-count format]

Notice that you can always tell what is to follow from the code received. The last format is very similar to the computer-to-TNC format. The details of these formats will be described later. All computer-to-TNC transmissions will follow one of these formats.

Chapter 2
Entering and Exiting Host Mode

Host mode is entered by giving a JHOST 1 command to the TNC in the usual way. This sounds simple enough. Just send <ESC>JHOST1<CR> to the TNC. But what if something has already been sent to the TNC (say by powerup transients)? The <ESC> will be treated as data (echoed as ^[by the TNC) and the command will be lost. We can work around this by sending a ^U or ^X first, to clear the junk out. It might also be a good idea to send an XON, in case the junk included an XOFF. So we send ^Q^X^[JHOST1^M (or, if you prefer the ASCII mnemonics, <DC1><CAN><ESC>JHOST1<CR>):

```
11 18 1B 4A 48 4F 53 54 31 0D
^Q ^X ^[ J  H  O  S  T  1  CR
```

This will work every time, if the TNC is in user mode. If the TNC is in Host mode already, we should detect that fact and recover. Host mode recovery will be discussed later in this document.

Exiting Host mode is a simple matter of sending a JHOST0 command to the TNC. The TNC will acknowledge the JHOST0 command in the usual Host-mode fashion before returning to user mode, so you can use your regular Host-mode transaction routine.

Note that a QRES command issued in Host mode will dump you back into user mode. It does not acknowledge in Host-mode fashion before doing so. The Host mode parameter is not stored in NOVRAM, so it is not possible to "wake up" in Host mode. It is, however, possible to PERM other parameters from Host mode.

Chapter 3
Commanding the TNC

OK, now that we're in Host mode, let's look at a simple command exchange. We

will gloss over some of the details for this first look. Suppose we want to turn off Unattended mode. In normal user mode, we would type <ESC>U0<CR>. That is, we would send a U0 command to the TNC. To send a U0 command in Host mode, we have to assemble a transmission for the TNC according to the computer-to-TNC format.

Recall that the computer-to-TNC format is:

{channel}{info/cmd}{count}{data...}

The first byte, channel, is the channel number. In user mode, it doesn't matter what channel you are on when you give a U command. Likewise, in Host mode, it doesn't matter which channel you send this command to. Let's choose channel 0. The second byte is info/cmd; this is a command, so this byte has to be 1. The command data we are going to send is simply "U0", the same ASCII string we would have used between <ESC> and <CR> to do the same thing in user mode. This is two bytes long, so the count byte should indicate two bytes. Count is encoded as (length - 1), so the third byte of the message should be 2 - 1 = 1.

The following is an illustration of the required transmission. The top line lists the bytes to be sent in hex notation. The first byte sent is on the left. Below each byte of the top line is an explanation of that byte. This format will be used throughout this document to illustrate Host mode transmissions.

```
00 01 01 55 30
!  !  !  U  0
!  !  !
!  !  +-- Count = 1    (two bytes)
!  +----- Info/cmd = 1 (this is a command)
+-------- Channel = 0
```

Note that no carriage return is required or allowed. As soon as the byte count is fulfilled (i.e., when the last byte has been received) the TNC accepts the transmission as complete.

How do we know the TNC got the command OK? Simple: it responds with a status message. A U0 command can't fail, so we can guess that the status message will be a success message. A U0 command doesn't return any information, either, so no data field will be present. So, we can guess that the TNC will respond with a transmission in the short format:

```
00 00
!  !
!  +--- Code = 0 (Success, no data follows)
+------ Channel 0
```

Now we can see the general shape of any computer-TNC transaction: the computer sends a command to the TNC, and the TNC responds with an appropriate message. In writing a program to interface with Host mode, we will have to take care of building these outgoing frames and interpreting the incoming frames. If we take a hint from WA8DED and W6IXU, we will implement a single routine that does the dirty work of sending a transmission and receiving the response.

In receiving the response, the routine will have to decide how many characters to read from the serial port by looking at the code in the second byte of the TNC's message. This code tells it in which format the transmission will be: short format with no following data, null-terminated format with some data ending in a null, or byte-count format with a count and the specified number of bytes of information.

Notice that the routine that receives the response doesn't have to have any knowledge of the current state of the TNC or the application program. The meaning and encoding of the TNC's response is completely determined by the data in that response. This simplifies the routine that receives the response by isolating it from the complexities of the application program.

Chapter 4
Polling the TNC: the G command

The most essential, basic transaction in Host mode is the G command, which polls a specific channel of the TNC for events and data. A typical Host-mode application program will constantly send G commands to the TNC and interpret the responses. Since the G command is a query to a specific channel of the TNC, it is necessary to send G commands to all of the channels that may be in use. Data received from a station connected on channel 1 (or 2, or 3, or 4) will be sent to the computer as a response to a G command directed to channel 1 (or 2, or 3, or 4). Monitored data will be sent to the computer as a G command directed to channel 0.

The G command has three forms. A command of "G" will return any of the possible responses that is available. A command of "G0" will return only information. A command of "G1" will return only link status responses. This is useful when you are waiting for a link status change and do not wish to handle data, or vice versa.

The only other thing that changes in a G command is the channel number. Be sure to send G commands to all channels that may be active. You need not send G commands to channel 4 if Y is 3, for instance, but you must send G commands to channels 0 through 3. The unrestricted G command has the following format:

```
0x 01 00 47
!  !  !  G
!  !  !
!  !  +--- Count = 0    (one byte of data)
!  +------ Info/cmd = 1 (this is a command)
+--------- Channel = x  (where x is 0, 1, 2, 3, or 4)
```

There are only a few possible responses to a G command. If the program polls the TNC constantly with G commands, most of the replies will be "nothing available". This simply means nothing new has happened on the link since the last poll. This response applies to any of the three forms of the G command. Each of the other possible responses indicates an event on the AX.25 channel: a change in Link Status (such as "DISCONNECTED"), which is evoked by G or G1, or receipt of a frame of monitored or connected data, which is evoked by G or G0. The detailed formats of these responses will be discussed later in this document.

Monitored information is handled specially by the G command. Monitored data is handled on channel 0 of the TNC. The monitor header is transmitted separately from the monitor information. The monitor header transmission indicates whether or not there is an associated information transmission to come. If there is, it will be the next response on channel 0. This scheme has two results: it spares the programmer the task of parsing out monitor information from the monitor header (if such is desired), and it keeps the maximum transmission length down to the maximum data field size, rather than the data field size plus the monitor header size (and thus allows the count to be a single byte).

This scheme for monitored data might be considered a violation of the rule that each computer-TNC transmission is self-contained and self-explanatory. The monitored information transmission is qualified by the immediately preceding monitor header transmission. You many want to treat a monitor header with information as a special case in the application program, and grab the corresponding monitor information transmission immediately.

Chapter 5
Details of Computer-to-TNC Format

As we said before, the computer-to-TNC transmissions must always follow this format:

{channel}{info/cmd}{count}{data...}

The three header bytes (channel, info/cmd, and count) are expressed in binary. For instance, to specify 0 for any of these values, send a NULL (^^U, '^Y0' for C fans, CHR$(0) for BASIC fans). DON'T send the ASCII character '0'.

The data field will contain one of the following:

1. Data to be transmitted via the AX.25 link.

2. A command string just like you might issue from the normal user mode (the part strictly between the <ESC> and the <CR>).

3. "G", the Host-mode poll command.

If data for the AX.25 link is to be sent, the info/cmd byte must be zero (binary zero, remember, not ASCII "0"). The data field may contain any bytes whatsoever, but not more than 256 of them. Count up the data bytes and subtract one to find the value of the count byte. So if you're connected to me on channel 2, and you want to say

```
Hello
```

to me, send the TNC this transmission:

```
02 00 05 48 65 6C 6C 6F 0D
!  !  !  H  e  l  l  o  CR
!  !  !                 !
!  !  !                 +---- (this is a data byte)
!  !  +- Count = 5    (six bytes of data)
!  +---- Info/cmd = 0 (this is information to be sent)
+------- Channel = 2  (since I'm on your channel 2)
```

Note that the CR is sent as data, and will be transmitted through the AX.25 link. It is not part of the transmission format.

If the information is sent to channel 0, it will go out unproto to the address on channel 0 (default CQ). If information is sent on channel 1 - 4 while that channel is not connected, it is ignored.

If a command string is to be sent, the info/cmd byte must be one. The data field should contain the ASCII command string. It does not contain <ESC> or <CR>, just the characters that would go between them to form a normal user-mode command. For instance, to set TXDELAY to 30, send the TNC this transmission:

```
00 01 02 54 33 30
!  !  !  T  3  0
!  !  !
!  !  +- Count = 2    (three bytes of data)
!  +---- Info/cmd = 1 (this is a command)
+------- Channel = 0  (doesn't matter here)
```

The channel number is significant for the C, D, F, G, L, N, O, and V commands. C, D, F, N, O, and V commands are channel specific in exactly the same way as they are in user mode. The L command is different in Host mode, and will be discussed later in this document. The G command has already been discussed. The channel number is not significant for other commands. You may use whatever channel number is convenient. (WA8DED and W6IXU send these commands to channel 0).

The byte-count format for the computer-to-TNC transmissions is used because it provides "data transparency". This means that any bytes whatsoever may appear in the data field. The TNC can just accept the next so many bytes from the computer without thinking about it. If a special character were used to signal the end of the data, that character could not occur in the data. The null-terminated format can use this scheme, since the data in these is guaranteed not to contain a null.

Unfortunately, the byte-count scheme is not very robust. If a character is lost or a spurious character is inserted, the computer and TNC could stay out

of synch forever. We must take pains to detect loss of synch and to recover from the fault. Fortunately, the first byte of a transmission has only five valid values (0-4) and the second byte has only two valid values (0 and 1), so it is possible to detect a loss of synchronization in most cases. Timeouts can also be used to detect synch failures. Once the error is detected, recovery is possible. Host mode recovery will be discussed later in this document.

Chapter 6
Details of TNC-to-Computer Format

The TNC-to-computer transmission format is a bit more complicated than the computer-to-TNC format. The key to decoding a transmission from the TNC is the code byte that follows the channel number byte. The following table gives the code values:

TNC-TO-COMPUTER CODES

Code	Meaning	Format
0	Success, nothing follows (Nothing available)	short format
1	Success, message follows	null-terminated format
2	Failure, message follows	null-terminated format
3	Link status	null-terminated format
4	Monitor header/no info	null-terminated format
5	Monitor header/info	null-terminated format
6	Monitor information	byte-count format
7	Connected information	byte-count format

These various formats are used for different purposes. Codes 0, 1, and 2 are responses to an information or command transmission from the computer. If a batch of information was queued for transmission successfully or if a command completes successfully without returning any information, a code 0 transmission will be the reply:

```
02 00
!  !
!  +---- Code = 1    (Success, nothing follows)
+------- Channel = 2 (same as the info or command sent)
```

If the command is successful and returns information (like any command without an argument), a code 1 transmission will be the reply. The returned data will follow the code, "null terminated". This just means that a 0 byte will appear after the last information byte. For example, consider an M command without an argument. In user mode, the user would type <ESC>M<CR>, and the TNC would display something like

```
    * IUSCRT *
```

In Host mode, the command would be

```
00 01 00 4D
!  !  !  M
!  !  !
!  !  +---- Count = 0    (one data byte)
!  +------- Info/cmd = 1 (this is a command)
+---------- Channel = 0  (Doesn't matter here)
```

and the response might be

```
00 01 49 55 53 43 52 54 00
!  !  I  U  S  C  R  T  !
!  !                    +---- Null termination
!  !
!  +---- Code = 1    (Success with info)
+------- Channel = 0 (same as in the query)
```

If the command failed, or the information cannot be queued for transmission, a

code 2 transmission with one of these error messages will be the reply:

FAILURE MESSAGES

INVALID COMMAND
TNC BUSY - LINE IGNORED
CHANNEL ALREADY CONNECTED
STATION ALREADY CONNECTED

For example if you send a data line:

```
03 00 0C 48 65 6C 6C 6F 20 74 68 65 72 65 2E 0D
!  !  !  H  e  l  l  o     t  h  e  r  e  .  CR
!  !  !                                       !
!  !  !            (this is a data byte) ----+
!  !  +---- Count = 12   (thirteen data bytes)
!  +------- Info/cmd = 0 (this is information)
+---------- Channel = 3  (for instance)
```

when the TNC has no RAM available to buffer the data line, it will respond with a code 2 failure message to inform you of the condition:

```
03 02 544C432042555359202D204C494C452049474C4F524544 00
!  !  T N C   B U S Y   -   L I N E   I G N O R E D  !
!  !                                                 !
!  +---- Code = 2 (Failure)      Null terminator ----+
+------ Channel = 3 (same as the info sent)
```

If you send a bad command:

```
00 01 03 4A 55 4C 4B
!  !  !  J  U  N  K
!  !  !
!  !  +---- Count = 3    (four data bytes)
!  +------- Info/cmd = 1 (this is a command)
+---------- Channel = 0  (Doesn't matter here.)
```

It will complain about it with a code 2 failure message:

```
00 02 49 4E 56 41 4C 49 44 20 43 4F 4D 4D 41 4E 44 00
!  !  I  N  V  A  L  I  D     C  O  M  M  A  N  D  !
!  !                                               !
!  +---- Code = 2 (Failure)     Null termination --+
+------- Channel = 0 (same as the command sent)
```

The remaining codes, 3 through 7, are used only as responses to a "G" poll command. A 0 code can also be a response to a G poll. A 0 code means that nothing is available for that channel: that is, no events worthy of notice have taken place on the channel polled. This is the most common response to a G command.

```
0x 00
!  !
!  +---- Code = 0    (Nothing available)
+------- Channel = x (same as the G command)
```

A code of 3 signifies a Link Status message. These messages are:

LINK STATUS MESSAGES

({n}) BUSY fm {call} via {digipeaters}
({n}) CONNECTED to {call} via {digipeaters}
({n}) LINK RESET fm {call} via {digipeaters}
({n}) LINK RESET to {call} via {digipeaters}
({n}) DISCONNECTED fm {call} via {digipeaters}
({n}) LINK FAILURE with {call} via {digipeaters}
CONNECT REQUEST fm {call} via {digipeaters}
({n}) FRAME REJECT (x y z) fm {call} via {digipeaters}
({n}) FRAME REJECT (x y z) to {call} via {digipeaters}

These messages should look familiar, since they are identical to the messages displayed by the TNC in user mode. {n} is the channel number. Notice that the channel number in the message is redundant information. The channel number may be omitted from this message in a future release.

For instance, if I connect to you direct on channel 2, a poll of channel 2 (not necessarily the next one, depending on what else is going on) will return the event:

```
02 03 28322920434F4E4E454354454420746F204B42354D55 00
!  !  ( 2 )   C O N N E C T E D   t o   K B 5 M U  !
!  !                                                !
!  +---- Code = 3 (Link Status) Null termination --+
+------- Channel = 2
```

Codes of 4 and 5 both signify a monitor header. This is a null-terminated format message containing the

fm {call} to {call} via {digipeaters} ctl {name} pid {hex}

string that forms a monitor header. The monitor header is also identical to the monitor header displayed in user mode. If the code was 4, the monitored frame contained no information field, so the monitor header is all you get. If you monitor KB6C responding to a connect request from me and then poll channel 0, you'll get:

```
0004666D204B42364320746F204B42354D552063746C2055612070494420463000
! ! f m   K B 6 C   t o   K B 5 M U   c t l   U A   p i d   F 0 !
! !                                                             !
! +---- Code = 4 (Monitor, no info)        Null termination ----+
+------- Channel = 0 (Monitor info is always on channel 0)
```

If the code was 5, the monitored frame did contain an information field. In this case, another G command to channel 0 will return the monitored information with a code of 6. Since data transmissions must be transparent, the monitored information is passed as a byte-count format transmission. That is, it is preceded by a count byte (one less than the number of bytes in the field). No null terminator is used in this case. Since codes 4, 5, and 6 pertain to monitored information, they will be seen only on channel 0. If you hear KB6C say "Hi" to NK6K, and then poll channel 0, you'll get:

```
0005666D204B42364320746F204E4B364B2063746C2049303020704944204630 00 0
! ! f m   K B 6 C   t o   N K 6 K   c t l   I 0 0   p i d   F 0 !
! !                                     ! !                     !
! !                          or whatever ----+-+                !
! !                                                             !
! +---- Code = 5 (Monitor, info follows)   Null termination ----+
+------ Channel = 0 (Monitor info is always on channel 0)
```

and then the very next poll to channel 0 will get:

```
00 06 02 48 69 0D
!  !  !  H  i  CR
!  !  !        !
!  !  !        +----    (this is a data byte)
!  !  +---- Count = 2   (three bytes of data)
!  +------- Code = 6    (monitored information)
+---------- Channel = 0 (Monitor info is always on channel 0)
```

A code of 7 signifies connected information, so it will only be seen on a connected channel 1 through 4. It is a byte-count format transmission, for data transparency. If you're connected to me on channel 4 and I say "Hi", a poll of channel 4 will return:

```
04 07 02 48 69 0D
!  !  !  H  i  CR
!  !  !        !
!  !  !        +----    (this is a data byte)
```

```
          !  !  +---- Count = 2   (three bytes of data)
          !  +------- Code = 7    (connected information)
          +---------- Channel = 4 (info was received on channel 4)
```

Chapter 7
Querying Channel Status: the L command

One of the most powerful features of the Host mode is the ability to ask the TNC for a status report on a given channel. This report is similar to the report given by the L command in user mode, but more extensive. To obtain the Channel Status report, send an L command to the channel of interest. The TNC will (barring error) respond with a code 1 packet. The information field of this packet will contain six decimal number in ASCII delimited by spaces.

A typical Channel Status query:

```
          01 01 00 4C
          !  !  !  L
          !  !  !
          !  !  +---- Count = 0    (one data byte)
          !  +------- Info/cmd = 1 (this is a command)
          +---------- Channel = 1  (give me a report on channel 1)
```

A typical Channel Status report:

```
          01 01 30 20 30 20 30 20 30 20 30 20 30 00
          !  !  0     0     0     0     0     0  !
          !  !                                   !
          !  !              null termination ----+
          !  +------- Code = 1    (Success with info)
          +---------- Channel = 1 (this is a report on channel 1)
```

Channel 0 does not have as much state information as a regular connectable channel, so a Channel Status query to channel 0:

```
          00 01 00 4C
          !  !  !  L
          !  !  !
          !  !  +---- Count = 0    (one data byte)
          !  +------- Info/cmd = 1 (this is a command)
          +---------- Channel = 0  (give me a report on channel 0)
```

Will return a Channel Status report similar to this one.

```
          00 01 30 20 33 00
          !  !  0     3  !
          !  !           !
          !  !           +---- null termination
          !  +------- Code = 1    (Success with info)
          +---------- Channel = 0 (this is a report on channel 0)
```

The interpretation of the returned information is as follows:

CHANNEL STATUS FORMAT

```
        a b c d e f
        a = Number of link status messages not yet displayed
        b = Number of receive frames not yet displayed
        c = Number of send frames not yet transmitted
        d = Number of transmitted frames not yet acknowledged
        e = Number of tries on current operation
        f = Link state

            Possible link states are:
             0 = Disconnected
             1 = Link Setup
             2 = Frame Reject
```

```
  3 = Disconnect Request
  4 = Information Transfer
  5 = Reject Frame Sent
  6 = Waiting Acknowledgement
  7 = Device Busy
  8 = Remote Device Busy
  9 = Both Devices Busy
 10 = Waiting Acknowledgement and Device Busy
 11 = Waiting Acknowledgement and Remote Busy
 12 = Waiting Acknowledgement and Both Devices Busy
 13 = Reject Frame Sent and Device Busy
 14 = Reject Frame Sent and Remote Busy
 15 = Reject Frame Sent and Both Devices Busy

NOTE 1:  Only items a and b are displayed for channel 0.
NOTE 2:  Only states 0 - 4 are possible if version 1 is in use.
```

How you will use the Channel Status report will depend on your application. An interactive terminal program might simply display some or all of the information and leave it at that. A mailbox program might want to wait for all packets to be acknowledged before starting a timeout timer or before disconnecting. A mail forwarder would want to make sure all data was acknowledged before considering its task done. Any program might want to give up if throughput is too low. These functions are easy to implement with Host mode.

Chapter 8
Host Mode Error Recovery

What happens if something goes wrong on the computer-TNC link? Maybe there is a loose wire, or maybe RF got into the RS-232 lines. Maybe it's just Murphy. For whatever reason, the data on the line got corrupted. RS-232 links are very reliable, so you may never see an error, but the application program should be ready to cope with one.

The error recovery program must assume that the TNC is in a completely unknown state. It might be expecting more bytes to fill a byte count (if bytes coming from the computer were lost). It might be simply waiting for a transmission, but the computer thought there was an error since data was lost or corrupted on its way from the TNC. What we want to do is hit the TNC with a stream of data that will eventually bring it into a known state. Preferably, this process should generate little or no spurious data on the AX.25 link.

The first thing to do is dump any further data that may be coming from the TNC. We don't know what it is, and so we can't use it. The possibility exists that this is connected information we're dropping on the floor. This is just too bad. There is nothing we can do.

Next, we just send ^A's one at a time until the TNC responds. The first few ^A's may go to fulfill a byte count that the TNC is waiting for. If this happens to be a connected information transmission, these ^A's will go out on the AX.25 link. Tough. When any outstanding byte count is fulfilled (no more than 256 should be required), the following five ^A's will be seen as a command on channel 1. This is why ^A is chosen: to make the TNC interpret the extra transmission as a command.

```
01 01 01 01 01
!  !  !  ^A ^A
!  !  !
!  !  +-- Count of two bytes.
!  +----- This is a command.
+-------- Channel #1.
```

A command of "^A^A" doesn't mean anything, so the TNC should respond with a failure message:

```
01 02 49 4E 56 41 4C 49 44 20 43 4F 4D 4D 41 4E 44 00
```

```
! ! I N V A L I D    C O M M A N D !
! !                                 !
! +---- Failure message.       Null termination --+
+------- Same channel as the command specified
```

We don't really care what the response is, though. It may not be exactly what we expect. For instance, the first ^A might fulfill the byte count of a garbled command. This would usually generate an error message of some other type. All we need is to get the first response. Then we know that the TNC is waiting for a command.

This implies that we have to wait between ^A's long enough to see if there is a response yet. This can make the recovery process a slow one. For instance, if the TNC spuriously receives 00 00 FF, it expects to see 256 data bytes following, so we will have to send 256 ^A's before we get a response (which will probably be a success message, since this is just a data transmission to the TNC). Fortunately, this error recovery process is not required very often.

-EOF-