TTY-Connect Firmware

TTY-CONNECT TELETYPE INTERFACE SYSTEM

FIRMWARE 1.Ø

SEPTEMBER 8, 2ØØ4

Gil Smith
480-354-5556
www.baudot.net

Table of Contents

TTY-Connect Overview

TTY-Connect provides everything needed for connecting most teletype machines in up to three local current loops, and for connecting a computer (PC) and/or a radio terminal-unit (TU) via RS-232 serial ports. Power is provided for the TTY loops (and current-limiting), as well as full opto-isolation. Note: this unit is NOT designed to connect to externally-powered loops.

You can simply use the unit as a loop supply for connecting multiple TTYs on a loop. Or, you can manually patch a loop to a computer's 232-serial (com) port, allowing a program on the computer to talk to the tty gear. Or, you can manually patch a loop to a radio terminal-unit's 232-serial port, and have the tty gear print the received rtty broadcast. Or, you can include the optional microcontroller to provide programmable signal connections between the loops/TU/PC, data-regeneration, and customizable features like Auto-CR-LF-Insertion, Autostart (motor powering), Selective-Calling, Who-Are-You (WRU), Ascii/Baudot-Conversion, Speed-Conversion, etc.

With either the PC232 or TU232 port connected to a tty loop in Full-Duplex mode, the keyboard (or tape-reader) contacts are sensed, and the characters are sent to the 232 port RX line only. Characters from the 232 port TX line are sent to printers (and/or punches). In Half-Duplex mode, typed characters will echo locally on the tty (as well as all other ttys in the loop). Characters sent from the 232 TX line are also echoed on the RX line, since they are returned from the loop sense.

Firmware Features

When the optional PIC microcontroller is included on the TTY-Connect board, it provides a number of programmable features, and allows "connections" between various "ports" -- the ports are PC232, TU232, HV1 loop, HV2 loop, and LV loop. These features are accessible using commands sent to the PC232 port.

Connection Modes

Once a connection is made, the unit will power-up in that mode until it is changed (since the connection configuration is stored in non-volatile memory, and is established at power-up). The system provides eight connection modes:

1) Mode-1: Connect the PC232 port in Ascii, to talk Baudot to one of three powered tty loops, or to the TU232 port, at any of the standard tty speeds (60, 66, 75, or 100-wpm). The PC232 port is fixed Ascii at 38400-baud, and uses xon/xoff for data throttling -- the unit automatically converts to/from baudot. The data stream from the PC to the TTY/TU is piped through the baudot "Receive-Stream-Options," for programmable Auto-CRLF-Insertion, AutoStart or SelCal motor powering, Unshift-on-Space, etc.

   Example: PC connected to M15 on HV1 loop, or Dovetron on TU port...

2) Mode-2: Connect the PC232 port to talk Ascii directly to one of three powered tty loops, or the TU232 port, at a fixed 110-baud. The PC232 port is fixed at 38400-baud, and uses xon/xoff for data throttling.

   Example: PC connected to M33 on LV loop, or M43 on TU port...

3) Mode-3: Connect the PC232 port to talk raw 5-, 6-, 7-, or 8-bit data to one of three powered tty loops, or the TU232 port, at an adjustable speed (16 to 1000-baud, with 250us resolution). The PC232 port is fixed at 38400-baud, and uses xon/xoff for data throttling. Using 8-bits allows connection to Ascii ttys (M33/35...). Using 5/6/7 bits allows the PC to send/receive custom codes, such as non-ustty 5-level, or 6-level TTS.

   Example: PC connected to 6-level M20 on HV1 loop...

4) Mode-4: Connect the TU232 port to talk Baudot directly to one of the three powered tty loops, at any of the standard tty speeds (60, 66, 75, or 100-wpm). The TU and TTY must be on the same speed. The data stream from the TU to the TTY is piped through the baudot "Receive-Stream-Options," for programmable Auto-CRLF-Insertion, AutoStart or SelCal motor powering, Unshift-on-Space, etc.

   Example: Dovetron TU connected to M15 on HV1 loop, or M28 on HV2 loop...

5) Mode-5: Connect the baudot TU232 port to talk Ascii directly to one of three powered tty loops at a fixed 110-baud. The data stream from the TU to the TTY is piped through the ascii "Receive-Stream-Options," for programmable Auto-CRLF-Insertion, AutoStart motor powering, etc.

   Example: TU connected to M33 on LV loop...

6) Mode-6: Connect Baudot ports at DIFFERENT SPEEDS (60, 66, 75, or 100-wpm). A PC must be connected, and running a buffering program, for connecting the high-to-low speed stream. The data stream from Port-A to Port-B is piped through the baudot "Receive-Stream-Options," for programmable Auto-CRLF-Insertion, AutoStart or SelCal motor powering, Unshift-on-Space, etc.

    Example: TU at 60-wpm connected to M28 on HV1 loop at 100-wpm

    Example: TU at 75-wpm connected to M15 on HV2 loop at 60-wpm

7) Mode-10: No connection, but place tty loops in space state (for system testing).

8) Mode-11: No connection, but place tty loops in mark state (for system testing).

Data Regeneration

Since the ports are connected via sampling software, data-regeneration is inherent in the design, That is, incoming codes are sampled and interpreted, and outgoing codes are generated with perfect framing and bit times (within the 250-us resolution of the software's sampling mechanism).

Programmable Features

All customizable features in TTY-Connect are stored in non-volatile memory (eeprom). This includes the current connection (TU to HV1 loop, PC to LV loop...), current port parameters (60-wpm...), selcal sequences, which, if any, selcals are enabled, etc. In addition to the primary connection features of TTY-Connect, there are two categories of programmable options: General-Options and RX-Stream-Options.

General-Options

These general options are applied regardless of the baudot-specific RX-Stream-Options (discussed next). These features may be independently enabled/disabled. The factory setting is disabled.

1: Unconnected-TTY-Loops-Mark: If enabled, tty loops NOT involved in the current connection will be set to the mark state (loop current flowing). Enable this if you need to provide loop current for an attached tty machine; disable this to minimize power/heat in unused loop circuitry.

2: TU-Data-Inversion: If enabled, will invert the TU data for connection to TUs having MIL-188 ports. Standard RS-232 signals use mark < -3V, and space > +3V. Some TUs use MIL-188 signals, in which case mark = +5V, and space = -5V.

3: Auto-CRLF-RX-Insertion, if enabled:

   a- After a CR is detected, reaching a max char trigger count (default: 72) forces return/linefeed string.

   b- For a baudot connection (modes 1,4,6), the default string to be inserted is CR CR LF LTRS LTRS

   c- For an ascii connection (modes 2,5), the default string to be inserted is CR LF NUL NUL NUL

4: Motor-Power-Mode provides Autostart or Selcal motor powering. All modes are valid for baudot operation, but only a subset of modes are valid for ascii use. The modes are discussed in the following section.

5: TX-Diddle Insertion: If enabled, constantly send LTRS chars when TX stream is idle (modes 1,4,6).

RX-Stream-Options

The RX-Stream options are applied to the character stream flowing TO the TTY (tty-rx) – this is a tty selector-magnets/typing-unit/tape-punch, which is plugged into a loop output jack (eg: HV1-OUT). All of these features may be independently enabled/disabled. The factory setting is disabled. All programmable baudot strings are 0 to 8-chars long. If an option does not list a baudot/ascii variation, the option only applies to baudot connections (modes 1,4,6).

1: RX-Unshift-on-Space: If enabled, when in FIGS mode will shift to LTRS mode after receipt of SPACE char.

2: RX-Unshift-on-CR: If enabled, when in FIGS mode will shift to LTRS mode after receipt of CR char.

3: RX-Diddle-Filter: If enabled, discards two or more successive LTRS or FIGS chars.

4: Motor-Power-Mode: AutoStart-Char – baudot connection modes (1,4,6), and ascii connection modes (2,5):

a- Upon receiving a valid character, a signal is activated for powering the motor of the connected tty. Note that the motor relay circuitry is optional.

b- Upon inactivity for 2 minutes, the motor is turned off.

5: Motor-Power-Mode: AutoStart-String -- baudot connection modes (1,4,6):

a- Upon receiving the "Start-of-Message" string, SP SP SP SP SP CR CR LF by default, a signal is activated for powering the motor of the connected tty. Note that the default string is the standard Conditioning-Code used on HoffNet. Also note that the motor relay circuitry is optional.

b- Upon receiving the "End-of-Message" string, LF N N N N by default, the motor is turned off.

6: Motor-Power-Mode: AutoStart-Special -- baudot connection modes (1,4,6):

(reserved)

7: Motor-Power-Mode: SelCal-Station -- baudot connection modes (1,4,6):

a- Upon receiving the "Conditioning-Code" string, SP SP SP SP SP CR CR LF by default, the system waits for a valid SelCal string. Note that the default string is the standard Conditioning-Code used on HoffNet.

b- Upon receiving the "SelCal-Station" string, T T Y C O N N by default, a signal is activated for powering the motor of the connected tty. Note that the default string is arbitrary, as it will normally be programmed to the last four characters of the staion callsign. Also note that the motor relay circuitry is optional.

c- Upon receiving the "End-of-Message" string, LF N N N N by default, the motor is turned off

8: Motor-Power-Mode: SelCal-All -- baudot connection modes (1,4,6):

a- Upon receiving the "Conditioning-Code" string, SP SP SP SP SP CR CR LF by default, the system waits for a valid SelCal string. Note that the default string is the standard Conditioning-Code used on HoffNet.

b- Upon receiving a "SelCal-All" string, Z C Z C by default, a signal is activated for powering the motor of the connected tty. Note that the motor relay circuitry is optional.

c- Upon receiving the "End-of-Message" string, LF N N N N by default, the motor is turned off.

9: Motor-Power-Mode: SelCal-Both -- baudot connection modes (1,4,6):

10: WRU-Reply (Who-Are-You) (not yet implemented):

a- Upon receiving the "Conditioning-Code" string, SP SP SP SP SP CR CR LF by default, the system waits for a valid SelCal string. Note that the default string is the standard Conditioning-Code used on HoffNet.

b- Upon receiving the "SelCal-Station" string plus a FIGS-H char, the PTT signal is activated for enabling a transmitter, the WRU-Reply string is sent, and then PTT is deactivated.

TTY-Connect Commands & Messages (PC232 Control)

Connection of a computer to the TTY-Connect's PC232 serial port allows you to fully control the system using a selection of "Commands," and to obtain immediate or polled feedback on system status by listening to "Messages."

Send Commands, and Receive Messages

The simplest method for configuring TTY-Connect is to run a terminal-emulation application on a PC (eg: HyperTerminal), and type in the commands directly. Indeed, for many uses this is quite adequate. You can setup the TTY-Connect unit for a specific mode (connected ports, port parameters...) -- this configuration will be saved in non-volatile memory, and restored every power-up. If you don't need to change your configuration very often, typing in a couple of commands is a simple way to control the unit.

Alternatively, a PC application can be created with a serial port driver that implements all or part of the TTY-Connect protocol. A basic application would merely send Commands, and ignore any received Messages. The next level of complexity involves interpreting the Messages that are returned in response to control operations. You may watch for an expected message immediately after sending a Command (single-threaded program), or you may have a separate section of code that parses the messages independently of the command processing code, and then updates internal variables and displays (multi-threaded program).

TTY-Connect Commands are in ASCII, and begin with a slash and a period (/.), followed by two letters defining the type of command, two or more comma-delimited decimal-number parameters, and finally a carriage-return and/or line-feed character (CR/LF). The CR/LF characters are identified in this documentation as <cr>.

An example Command:

/.TW,1,2,4,66 <cr>

Similarly, TTY-Connect Messages begin with a hyphen and a period (-.), followed by two letters defining the type of message, two or more comma-delimited decimal number parameters, and a terminating CR/LF.

An example Message:

-.TC,1,2,4,66 <cr>

The PC232 serial port requires 38400 baud, 8 data bits, no parity, 1 stop bit, with Xon/Xoff flow control enabled. The code is usually ascii, but special codes can be used in raw mode.

Command Definition

TTY-Connect Commands are in ASCII, and begin with a slash and a period (/.), followed by two letters defining the type of command, two or more comma-delimited decimal-number parameters, and finally a carriage-return and/or line-feed character (CR/LF).

Letters in the TTY-Connect Command header may be upper or lowercase (but not mixed upper/lower).

The decimal number parameters in Commands are ascii-encoded-decimal number strings that normally range from 0 to 255. There may be any number of digits in a parameter (eg: 5, 21, 114), and a parameter may optionally include any number of leading zeroes (ie: 002, 02, and 2 are all equivalent). Parameters normally should not exceed 255, although larger numbers are allowed. Numbers greater than 255 are truncated to 1-byte internally (this is actually handy for the optional checksum parameter, discussed later).

A comma ',' is used to delimit parameter fields. A null parameter (nothing between the commas) will evaluate to 0. An 'X' character may be placed in a parameter position, and will evaluate to 255.

The first number parameter is a "command-id" number, defining the specific action for this type of command, and which parameters follow to complete the command. The second number parameter is a "number-of-params-to-follow" number, defining how many parameters follow to complete the command. If a Command (optionally) includes an extra parameter, the last parameter is presumed to be a checksum. If the checksum is included, it is not counted in the “num” parameter.

Commands are terminated with a carriage-return (CR) and/or line-feed (LF) character (CR = 0x0d = 13; LF = 0x0a = 10), to work with string terminations found in various programming environments:

- linux/unix systems use LF (\n aka newline)
- macintosh systems use CR (\r aka return)
- windows/dos systems use CR/LF (\r\n)

This documentation uses the symbol <cr> to indicate termination of Commands (CR and/or LF accepted).

Two TTY-Connect Command formats are defined:

Write-Command: /.TW,id,num(,param1)(,param2)...(,checksum) <cr>

Read-Command: /.TR,id,num(,param1)(,param2)...(,checksum) <cr>

A Command will be quietly discarded if not properly/fully-formed:

1) if it is truncated with a premature CR, LF, or /
2) if the delimit character is anything other than comma
3) if non-digit chars are embedded in the numerical parameters (even spaces)
   (except 'X' may be used in place of a parameter, evaluating to 255).

An error message will result if:

1) the command-type is invalid (ie: not TW or TR)
2) the command-id number is invalid (undefined command)
3) there is an improper number of parameters for the command
4) the (optional) checksum is incorrect

Optional Command Checksum

If a Command includes an extra parameter, the last parameter is presumed to be a checksum of all preceding parameters. The checksum can be truncated to 1-byte before being sent (overflow discarded) or it can simply be the total sum (which, if over 255, gets truncated to 1-byte internally). If a Command includes the optional checksum, the checksum is not counted in the "num" parameter.

Example: identical commands to connect TU to HV1 at 75-wpm:

/.TW,4,2,1,75	this is normal command with no checksum
/.TW,4,2,1,75,82	this is command with checksum (4+2+1+75=82)

Example: identical commands (note that checksum can be >255, or truncated):

/.TW,3,3,4,248,8	this is normal command with no checksum
/.TW,3,3,4,248,8,265	this is command with total checksum (3+3+4+248+8=266)
/.TW,3,3,4,248,8,10	this is command with truncated (1-byte) checksum (266-256=10)

Message Definition

TTY-Connect Messages begin with a hyphen and a period (-.), followed by two letters defining the type of message, two or more comma-delimited decimal number parameters, and a terminating CR/LF.

A carriage-return (CR) and line-feed (LF) character (CR = 0x0d = 13; LF = 0x0a = 10) is also sent before each message.

Letters in the TTY-Connect Message header are uppercase.

The first number parameter is a "message-id" number, defining the content of the message. The second parameter is the "number-of-parameters" that follow, and, finally, the parameters (if any) complete the message.

The decimal number parameters in Messages are ascii-encoded-decimal number strings that range from 0 to 255.

A comma ',' is used to delimit parameter fields.

Messages are terminated with a carriage-return (CR) and line-feed (LF) character (CR = 0x0d = 13; LF = 0x0a = 10). This documentation uses the symbol <cr> to indicate termination of Messages (CR and LF are sent).

One TTY-Connect Message format is defined:

-.TC,id,num(,param1)(,param2)...<cr>

Connect Commands & Messages

In response to TTY-Connect Write Commands (/.TW), Status-Messages (-.TC) are returned automatically. System status may also be determined at any time by polling using the optional Status Read Commands (/.TR). The staus messages all have the same format and number of parameters, allowing the parsing program to simply buffer the message, identify the message by the first id param, and then use the next five params for updating system variables and displays. The Connect Commands set the primary configuration of the machine.

Connect Command Description	*Commands*	*Status-Message*
Get Current Connection Configuration	**/.TR,1,0 <cr>**	**-.TC,1,2,ptu,spd <cr>**
	or	**-.TC,2,1,ptu <cr>**
	or	**-.TC,3,3,ptu,spx,bit <cr>**
	or	**-.TC,4,2,pty,spd <cr>**
	or	**-.TC,5,2,spd,pty <cr>**
	or	**-.TC,6,4,pta,spa,ptb,spb <cr>**
	or	**-.TC,10,0 <cr>**
	or	**-.TC,11,0 <cr>**
Connect Ascii PC232 to Baudot TTY/TU (Mode-1 -- note 1)	**/.TW,1,2,ptu,spd <cr>**	**-.TC,1,2,ptu,spd <cr>**
Connect Ascii PC232 to 110-baud Ascii TTY/TU (Mode-2 -- note 2)	**/.TW,2,1,ptu <cr>**	**-.TC,2,1,ptu <cr>**
Connect Raw PC232 to Raw TTY/TU (Mode-3 -- note 3)	**/.TW,3,3,ptu,spx,bit <cr>**	**-.TC,3,3,ptu,spx,bit <cr>**
Connect Baudot TU232 to Baudot TTY (Mode-4 -- note 4)	**/.TW,4,2,pty,spd <cr>**	**-.TC,4,2,pty,spd <cr>**
Connect Baudot TU232 to 110-baud Ascii TTY (Mode-5 -- note 5)	**/.TW,5,2,spd,pty <cr>**	**-.TC,5,2,spd,pty <cr>**
Connect Baudot ports at Different-Speeds (Mode-6 -- note 6)	**/.TW,6,4,pta,spa,ptb,spb <cr>**	**-.TC,6,4,pta,spa,ptb,spb <cr>**
No connection, but set all tty loops to Space	**/.TW,10,0 <cr>**	**-.TC,10,0 <cr>**
No connection, but set all tty loops to Mark (Mode-10/11 -- note 7)	**/.TW,11,0 <cr>**	**-.TC,11,0 <cr>**

bit is number of bits for raw tty/tu port	**5, 6, 7, or 8**
pta/ptb is tty/tu port A/B	**1=HV1, 2=HV2, 3=LV, 4=TU**
ptu is tty/tu port	**1=HV1, 2=HV2, 3=LV, 4=TU**
pty is tty (only) port	**1=HV1, 2=HV2, 3=LV**
spa/spb is baudot port A/B speed in wpm	**60, 66, 75, or 100**
spd is baudot port speed in wpm	**60, 66, 75, or 100**

spx is raw port speed in 4x bit-period-ms 4 to 255

(4 = 1.00-ms bits = 1000-baud = not-too-useful)
(13 = 3.25-ms bits = 307.7-baud ~ 300-baud)
(36 = 9.00-ms bits = 111.1-baud ~ 110-baud)
(54 = 13.5-ms bits = 74.07-baud = 100-wpm/5-bit)
(72 = 18.0-ms bits = 55.55-baud = 75-wpm/5-bit)
(78 = 19.5-ms bits = 51.28-baud ~ 51-baud = 60-wpm/6-bits)
(80 = 20.0-ms bits = 50.00-baud = 66-wpm/5-bit)
(88 = 22.0-ms bits = 45.45-baud = 60-wpm/5-bit)
(255 = 63.75-ms bits = 15.70-baud = not-too-useful)

Connection Notes:

Note 1 -- Connect Mode 1: Ascii PC232 port to Baudot TTY/TU port (/.TW,1...):
- For connection of PC to baudot TTY machines or TU port (eg: M15 on HV1 port, Dovetron on TU port...)
- Ascii PC232 port is fixed at 38400-baud, 8-bits, no-parity, 1-stop, with xon/xoff flow control enabled
- Baudot TTY or TU port is at one of four standard speeds (60, 66, 75, or 100 wpm)
- Ascii-to-Baudot (ustty) conversion is automatic in both directions
- RX-Options are applied on the PC-to-TTY/TU baudot stream for programmable Auto-CRLF-Insertion, Autostart or Selcal motor powering, etc.
- Note that the connection is saved in non-volatile configuration memory, and the unit will power-up in this configuration until it is changed.

Note 2 -- Connect Mode 2: Ascii PC232 port Ascii TTY/TU port at 110-baud (/.TW,2...):
- For connection of PC to ascii TTY machines or TU (eg: M33 on LV port, M43 on TU port...)
- Ascii PC232 port is fixed at 38400-baud, 8-bits, no-parity, 1-stop, with xon/xoff flow control enabled
- Ascii TTY or TU port is fixed at 110-baud
- RX-Options are applied on the PC-to-TTY/TU ascii stream for programmable Auto-CRLF-Insertion, Autostart motor powering, etc.
- Note that the connection is saved in non-volatile configuration memory, and the unit will power-up in this configuration until it is changed.

Note 3 -- Connect Mode 3: Raw PC232 port to Raw TTY/TU port (/.TW,3...):
- For connecting to ascii or special-code machines (PC handles code conversion, if needed)
- 8-bit mode used for ascii machines (mode 2 is also available for 110-baud only)
- 7-bit mode used for ?
- 6-bit mode used for TTS or other 6-bit machines (eg: M20 on HV1 port...)
- 5-bit mode used for non-ustty baudot (eg: European machines with different 5-level char sets)
- Raw PC232 port is fixed at 38400-baud, 8-bits, no-parity, 1-stop, with xon/xoff flow control enabled
- Raw TTY/TU port can use 5, 6, 7, or 8 (lower) bits from the PC232 port
- Port speed spx is defined as 4x the bit period in milliseconds, and can range from 4 (1-ms bits) to 255 (63.75-ms bits).
- Note that the connection is saved in non-volatile configuration memory, and the unit will power-up in this configuration until it is changed.

Note 4 -- Connect Mode 4: Baudot TU232 port to Baudot TTY port (/.TW,4...):

- For connection of TU232 port to baudot TTY machines at same speed (eg: M15 on HV1 port, M28 on HV2 port...)
- Baudot TU and TTY ports are at one of four standard speeds (60, 66, 75, or 100 wpm)
- RX-Options are applied on the TU-to-TTY baudot stream for programmable Auto-CRLF-Insertion, Autostart or Selcal motor powering, etc.
- Note that the connection is saved in non-volatile configuration memory, and the unit will power-up in this configuration until it is changed.

Note 5 -- Connect Mode 5: Baudot TU232 port to Ascii TTY port at 110-baud (/.TW,5...):

- For connection of TU232 port to ascii TTY machines (eg: M33 on LV port)
- Ascii PC232 port is fixed at 38400-baud, 8-bits, no-parity, 1-stop, with xon/xoff flow control enabled
- Ascii TTY port is fixed at 110-baud
- Note that for the TTY-to-TU direction, with the TTY at low wpm, some char loss will occur when streaming from the TTY (eg: reading a paper tape), since only small internal buffers are available for handling the speed conversion. This will not be noticed at typical typing speeds. This mode most likely to be used for the TU-to-TTY direction anyway.
- RX-Options are applied on the PC-to-TTY/TU ascii stream for programmable Auto-CRLF-Insertion, Autostart motor powering, etc.
- Note that the connection is saved in non-volatile configuration memory, and the unit will power-up in this configuration until it is changed.

Note 6 -- Connect Mode 6: Baudot Ports at Different Speeds (/.TW,6...):

- For connecting two baudot TTY/TU ports at different speeds
- PC must be connected, and running a special buffering program, for high-to-low-speed direction
- Baudot Port-A/Port-B ports are at two of four standard speeds (60, 66, 75, or 100 wpm)
- Port-A (pta) and Port-B (ptb) cannot be the same port
- Port-A-Speed (spa) and Port-B-Speed (spb) cannot be the same speed
- RX-Options are applied on the PortA-to-PortB baudot stream for programmable Auto-CRLF-Insertion, Autostart or Selcal motor powering, etc.
- Note that the connection is saved in non-volatile configuration memory, and the unit will power-up in this configuration until it is changed.

Note 7 -- Connect Modes 10 or 11: Set all TTY loops to Space or Mark:

- No connection – these modes are for testing or special use
- Mode 10 sets all TTY Loops (HV1, HV2, and LV) to Space state (no loop current flowing)
- Mode 11 sets all TTY Loops (HV1, HV2, and LV) to Mark state (loop current flowing)

Programming Commands & Messages

The Programming Commands change customizable settings.

General Programming	*Command*	*Status-Message*
Unconnected-TTY-Loops-Mark (note 1)	**/.TW,40,1,enb <cr> /.TR,40,0 <cr>**	**-.TC,40,1,enb <cr>**
TU-Data-Inversion (note 2)	**/.TW,41,1,enb <cr> /.TR,41,0 <cr>**	**-.TC,41,1,enb <cr>**
Auto-CRLF-RX-Insertion (note 3)	**/.TW,49,1,enb <cr> /.TR,49,0 <cr>**	**-.TC,49,1,enb <cr>**
Auto-CRLF-Max-Chars/Line (note 3)	**/.TW,50,1,cmx <cr> /.TR,50,0 <cr>**	**-.TC,50,1,cmx <cr>**
Future: Auto-CRLF-NUL-pads (note 3)	**/.TW,52,1,nul <cr> /.TR,52,0 <cr>**	**-.TC,52,1,nul <cr>**
Motor-Power-Mode (note 4)	**/.TW,53,1,mpm <cr> /.TR,53,0 <cr>**	**-.TC,53,1,mpm <cr>**
Restore-Factory-Settings (note 5)	**/.TW,250,0 <cr>**	**-.TC,250,0 <cr>**

cmx is max number chars per line **10 to 80 (factory setting = 72)**

enb is enable or disable **1=enable, 0=disable (factory setting = 0)**

mpm is is motor power mode **0 to 6 (factory setting = 0)**

0 = Off
1 = Autostart-Char **(motor-on with any char, off after ???)**
2 = Autostart-String **(motor-on/off with programmable baudot strings)**
3 = Autostart-Special **(reserved)**
4 = Selcal-Station **(motor-on/off with programmable baudot strings)**
5 = Selcal-All **(motor-on/off with programmable baudot strings)**
6 = Selcal-Both **(responds to both Selcal-Station and Selcal-All strings)**

nul is number of NULS **1 to 20 (factory setting = 3)**

Baudot-Specific Programming	*Command*	*Status-Message*
RX-Unshift-on-Space (note 10)	/.TW,70,1,enb <cr> /.TR,70,0 <cr>	-.TC,70,1,enb <cr>
RX-Unshift-on-CR (note 11)	/.TW,71,1,enb <cr> /.TR,71,0 <cr>	-.TC,71,1,enb <cr>
RX-Diddle-Filter (note 12)	/.TW,72,1,enb <cr> /.TR,72,0 <cr>	-.TC,72,1,enb <cr>
TX-Diddle-Insertion (note 13)	/.TW,73,1,enb <cr> /.TR,73,0 <cr>	-.TC,73,1,enb <cr>
SI/SO-for-LTRS/FIGS (note 14)	/.TW,74,1,enb <cr> /.TR,74,0 <cr>	-.TC,74,1,enb <cr>
Future: WRU-Reply (note 15)	/.TW,75,1,enb <cr> /.TR,75,0 <cr>	-.TC,75,1,enb <cr>
Auto-CRLF String (notes 3,20,21)	/.TW,90,8,p1,p2,p3,p4,p5,p6,p7,p8 <cr> /.TR,90,0 <cr>	-.TC,90,8,p1,p2,p3,p4,p5,p6,p7,p8 <cr>
Start-of-Message String (notes 20,22)	/.TW,91,8,p1,p2,p3,p4,p5,p6,p7,p8 <cr> /.TR,91,0 <cr>	-.TC,91,8,p1,p2,p3,p4,p5,p6,p7,p8 <cr>
End-of-Message String (notes 20,23)	/.TW,92,8,p1,p2,p3,p4,p5,p6,p7,p8 <cr> /.TR,92,0 <cr>	-.TC,92,8,p1,p2,p3,p4,p5,p6,p7,p8 <cr>
Conditioning-Code String (notes 20,24)	/.TW,93,8,p1,p2,p3,p4,p5,p6,p7,p8 <cr> /.TR,93,0 <cr>	-.TC,93,8,p1,p2,p3,p4,p5,p6,p7,p8 <cr>
Selcal-Station String (notes 20,25)	/.TW,94,8,p1,p2,p3,p4,p5,p6,p7,p8 <cr> /.TR,94,0 <cr>	-.TC,94,8,p1,p2,p3,p4,p5,p6,p7,p8 <cr>
Selcal-All String (notes 20,26)	/.TW,95,8,p1,p2,p3,p4,p5,p6,p7,p8 <cr> /.TR,95,0 <cr>	-.TC,95,8,p1,p2,p3,p4,p5,p6,p7,p8 <cr>

enb is enable-or-disable **1=enable, 0=disable (factory setting = 0)**

p1,p2...px are command-specific parameters **0 to 255**

Programming Notes:

Note 1 – Unconnected TTY loops Mark (/.TW,40...):

If enabled, tty loops that are not connected (by a connect command) will assume the mark state. This will cause loop currents to flow and prevent tty machines from running open (if powered). However, this will also result in extra power and heat dissipation in the loop circuitry. Factory-setting is disabled, to turn off unused loops. This setting won't take effect until reboot or a new connection command is issued.

Note 2 – TU-Data-Inversion (/.TW,41...):

If enabled, will invert the TU data for connection to TUs having MIL-188 ports. Standard RS-232 signals use mark < -3V, and space > +3V. Some TUs use MIL-188 signals, in which case mark = +5V, and space = -5V.

Note 3 – Auto-CRLF-RX-Insertion (/.TW,49...,/.TW,50...,/.TW,52.../.TW,90...):

When Auto-CRLF-RX-Insertion is enabled, and a line exceeds Auto-CRLF-Max-Chars/Line, one of two things happen: For a baudot rx port, the programmable Auto-CRLF string is inserted. For an ascii rx port, ascii CR and LF are followed by a number of NULs defined by Auto-CRLF-NUL-Padding.

Note 4 – Motor-Power-Mode (/.TW,53...):

The Motor-Power-Mode provides various options for Autostart or Selcal control of tty motors.

Note 5 – Restore-Factory-Settings (/.TW,250...):

All programmable parameters (in non-volatile memory) are restored to factory settings.

Note 10 – RX-Unshift-on-Space (/.TW,70...):

If enabled, when in FIGS mode will shift to LTRS mode after receipt of SPACE char.

Note 11 – RX-Unshift-on-CR (/.TW,71...):

If enabled, when in FIGS mode will shift to LTRS mode after receipt of CR char.

Note 12 – RX-Diddle-Filter (/.TW,72...):

If enabled, removes duplicate FIGS or LTRS chars from rx stream.

Note 13 – TX-Diddle-Insertion (/.TW,73...):

If enabled, inserts LTRS chars when tx stream is idle.

Note 14 – SI/SO-for-LTRS/FIGS (/.TW,74...):

If enabled, baudot-to-ascii conversions will provide SI/SO chars when LTRS/FIGS chars are received.

This does not need to be enabled for the opposite ascii-to-baudot conversion. Sending SI (^O) will send an explicit LTRS char, and sending SO (^N) will send an explicit FIGS char -- these are only needed for special cases, since LTRS and FIGS are provided as needed by the ascii-to-baudot conversion.

Note 15 – WRU-Reply (/.TW,75...):

If enabled, system will send WRU string... (future option).

Note 20 – Baudot-USTTY code for programmable strings:

In programmable baudot strings, use the decimal values of desired characters. Use the BLANK char (0) to pad the right side of the string out to 8 characters (ie: zeroes are needed to pad unused character positions, and effectively shorten the string) – only the non-BLANK baudot chars are valid in the string.

Example: 8,8,2,31,31,0,0,0 results in a 5-char string: CR CR LF LTRS LTRS

Example: 31,16,16,21,27,23,0,0 results in a 6-char string: LTRS T T Y FIGS 1 (TTY1)

decimal	LTRS	FIGS	decimal	LTRS	FIGS
0	BLANK	BLANK	16	T	5
1	E	3	17	Z	"
2	LF	LF	18	L	)
3	A	-	19	W	2
4	SP	SP	20	H	#
5	S	BELL	21	Y	6
6	I	8	22	P	0
7	U	7	23	Q	1
8	CR	CR	24	O	9
9	D	$	25	B	?
10	R	4	26	G	&
11	J	'	27	FIGS	FIGS
12	N	,	28	M	.
13	F	!	29	X	/
14	C	:	30	V	;
15	K	(	31	LTRS	LTRS

Note 21 – Auto-CRLF String (/.TW,90...):

This baudot string has a factory-setting of CR CR LF LTRS LTRS (8,8,2,31,31,0,0,0).

Note 22 – Start-of-Message String (/.TW,91...):

This baudot string has a factory-setting of SP SP SP SP SP CR CR LF (4,4,4,4,4,8,8,2).

Note 23 – End-of-Message String (/.TW,92...):

This baudot string has a factory-setting of LF N N N N (2,12,12,12,12,0,0,0).

Note 24 – Conditioning-Code String (/.TW,93...):

This baudot string has a factory-setting of SP SP SP SP SP CR CR LF (4,4,4,4,4,8,8,2).

Note 25 – Selcal-Station String (/.TW,94...):

This baudot string has a factory-setting of T T Y C O N N (16,16,21,14,24,12,12,0).

Note 26 – Selcal-All String (/.TW,95...):

This baudot string has a factory-setting of Z C Z C (17,14,17,14,0,0,0,0).

Misc Commands & Messages

Misc Command Description	*Command*	*Status-Message*
Get System-Info	**/.TR,0,0 <cr>**	**-.TC,0,4,0,0,svh,svl <cr>**
Change State of an Output Signal	**/.TW,20,2,out,sta <cr>**	**-.TC,20,2,out,sta <cr>**
Read State of an Output Signal	**/.TR,20,1,out <cr>**	**-.TC,20,2,out,sta <cr>**

out is output signal — **1=HV1_MOTOR, 2=HV2_MOTOR, 3=LV_MOTOR, 4=PTT**

sta is output signal state — **0=off, 1=on**

svh is software version high digit — **0 to 255**
svl is software version low digit — **0 to 255**
(eg: svh=1,svl=0, software version is 1.0)

Error Message Description	*Status-Message*
Error Message (general)	**-.TC,240,1,err_code <cr>**
Command-Type error (not TW or TR)	**-.TC,240,1,1 <cr>**
Command-ID error (undefined command)	**-.TC,240,1,2 <cr>**
Wrong number of params, or bad checksum	**-.TC,240,1,3 <cr>**
Ports or speeds cannot have same value	**-.TC,240,1,4 <cr>**
Invalid out param	**-.TC,240,1,5 <cr>**
Internal-Diagnostic Message (general)	**-.TC,249,1,diag_code <cr>**
PC232 input buffer overflow	**-.TC,249,1,1 <cr>**
PA output buffer overflow	**-.TC,249,1,2 <cr>**
PA input buffer overflow	**-.TC,249,1,3 <cr>**
PB output buffer overflow	**-.TC,249,1,4 <cr>**
PB input buffer overflow	**-.TC,249,1,5 <cr>**
Invalid mode param -- Message	**-.TC,249,1,6 <cr>**
Invalid mode param -- Connect	**-.TC,249,1,7 <cr>**

Technical Notes

Asynchronous Serial Communication

Asynchronous serial communication was initially developed for electromechanical teleprinters, and evolved into various electronic data communication and computer interface standards still in use today. In order for serial communication to occur, both transmiter and receiver must be compatible with the physical transmission line, character format (number of bits per char, most-significant or least-significant bit first, etc.), and rate at which the characters are to be transmitted.

A serial transmission line has two logical states (called "Mark" and "Space"), which are typically implemented as a current-on/current-off design (eg: TTY loops), or a voltage-1/voltage-2 design (eg: RS-232 ports). Specific time intervals during a character transmission are called bits; during a bit time, the line is set to one of its two states, to indicate whether the bit is to be interpreted as a logical 1 or 0.

Each character is framed by "Start" and "Stop" bits that indicate the beginning and end of the group of bits that define each character. The line is considered to be in the Mark state when it is idle (just "marking time"). When a Space state first occurs (the opposite line condition), it is the beginning of the Start bit – the receiver uses this line transition as a reference for timing the midpoints of each bit, in order to interpret the bit pattern, and hence the character. After the character transmission is complete, the line returns to the Mark state.

Since the Start bit is a Space state, and the Stop bit is a Mark state, characters may be transmitted with no additional delays between them, as the beginning of the Start bit is always recognizable by the receiver. Mechanical teleprinters may, for practical reasons, impose additional constraints on this scheme, but the theory is the same for both mechanical and electronic versions of a serial data interface.

The communication is considered "asynchronous," since there is no clock information sent along with the data. The receiver uses the Start bit leading edge as a reference, and then depends on accuracy of its own timing system for sampling the bits at the appropriate times during the character transmission. The receiver only needs to maintain enough accuracy for the duration of each character, since it is re-synchronizing at the start of the next character.

After the Start bit, the format usually defines the next bit to be the Least Significant Bit (LSB) of the charcter, continuing to the Most Significant Bit (MSB), and finally the Stop bit. A character may be defined to be any number of bits, corresponding to any of a number of codes. The most common formats are 5-bit baudot for teleprinters, and 7- or 8-bit ascii for computer systems.

Some formats include a "Parity" bit just before the Stop bit. This bit is used for simple error checking to catch a bit drop out. If "Even" parity is used, the parity bit is set to 1 or 0 as needed to make the total number of 1s in the character even. "Odd" parity will make the total number of 1s in the character odd. Parity may not catch multiple bit errors.

When the "Break" key is depressed (if available), a line spacing condition is forced for at least one character length. This is a physical line state, and not a defined character. The Break condition can be sensed and used for simple signalling.

5-Level Codes

There were a few variations in character codes for five-level teletypewriter machines. The two most-common character codes were ITA2 and USTTY (a variation of ITA2).

The USTTY and ITA2 5-level teletypewriter codes are commonly referred to as "Baudot" codes. While this is technically incorrect, these popular 5-level codes evolved from the work of Jean Maurice Emile Baudot of France -- it seems fitting to accept the defacto reference to "Baudot" as implying USTTY or ITA2 codes, since they were the 5-level codes that saw practical use in teletypewriter systems. However, the true Baudot code dates to around 1874, when Baudot designed the "Baudot Multiplex System," a printing telegraph. The system used a 5-level code generated by a device with five keys, operated with two left-hand fingers, and three right-hand fingers -- this required great skill on the part of the operator who entered the code directly. However, it was still a major improvement in communications -- prior to Baudot's design, communication was carried out using Morse code with a telegraph key. The 5-level "Baudot" code was actually designed by Johann Gauss and Wilhelm Weber. Used primarily in France, the Baudot Multiplex System, also made inroads in Britain. The original Baudot code defined the familiar structure of a 5-level code set, using LTRS and FIGS case shifting, and became known as the International Telegraph Alphabet 1 (ITA1). Another recognition of Baudot's contribution to data communications is the term "baud," which refers to bits-per-second speed of serial data.

Around 1901, Donald Murray of New Zealand developed an automatic telegraphy system, using a typewriter-like keyboard mechanism, and a variation on the original Baudot 5-level code. While Baudot's code was designed with finger-actuation in mind, Murray's code was designed for mechanization, to minimize machine wear for frequently-occuring charaters. The Murray system employed a keyboard perforator, which allowed an operator to manually punch a paper tape, and a tape transmitter for sending the message from the punched tape. At the receiving end of the line, a printing mechanism would print on a paper tape, and/or a reperforator could be used to make a perforated copy of the message. Early British Creed machines used the Murray system.

Around 1930, the CCITT (International Telegraph and Telephone Consultative Committee) introduced the International Telegraph Alphabet 2 (ITA2). The United States standardized on a variation of ITA2 called the American Teletypewriter code or USTTY. ITA2 and the USTTY variant became the basis for 5-level teletypewriter codes until 7-level ASCII code debuted (in an upper-case-only form) in 1963, and finally matured in 1967 to the form still used today.

Common to all of these 5-level codes is the shifting of keys using a FIGS or LTRS code. The same 5-level (5-bit) code is used to represent a lowercase symbol (LTRS case) or an uppercase symbol (FIGS case). When the LTRS or FIGS character is transmitted, it defines whether subsequent characters are to be interpreted as lowercase or uppercase. The receiving machine must remember the case until it is next changed.

Some teletypewriter machines "unshift-on-space," which is to say they revert to LTRS state after a space character is received. Some machines unshift at the the end of a line; this machine-specific behavior is not presumed by the code. Transmitted messages typically used a CR LF LTRS LTRS sequence at the end of a line (eg: at 72 chars) -- the CR LF positioned printing at the start of the next line, and the LTRS LTRS not only provided an explicit unshift, but it allowed the carriage time to fully return to the home position.

Baudot 5-Level Codes: USTTY, ITA2, and Other Variations

The ITA2 code was an international standard, used in applications such as Telex service. The shifted-D character could be interpreted as WRU (Who-Are-You). The shifted-F, G, and H characters were technically undefined, but had commonly-used symbols. USTTY was a variation of ITA2, and was the prevalent code used in the United States. The letters D, F, G, H, J, S, V, and Z had different FIGS symbols than ITA2, and, for some reason, BELL and ' were opposite of ITA2. There were also variations for specific applications -- that is, machines that had primarily a USTTY or ITA2 code set, with just a few changed keys. J and H were said to have the most commonly-changed FIGS characters. USTTY was also the base code used for conversion of five-level to seven-level codes for later dial TWX service.

The shifted-H symbol (#/STOP/blank) was sometimes used in USTTY machines as a "motor off" control code; the next character would turn the motor back on. Since some characters are missed until the motor is up to speed, a BLANK (NULL) character would be sent to start the motor, with perhaps a dozen additional BLANKs sent as filler to allow the motor speed to stabilize before reception of the message.

LTRS	USTTY FIGS	ITA2 FIGS	Fractions FIGS	other FIGS variations
A	-	-	-	
B	?	?	5/8	
C	:	:	1/8	WRU (degree?)
D	$	# (WRU)	$	
E	3	3	3	
F	!	@ (undef)	1/4	# % (blank)
G	&	* (undef)	&	@
H	#	$ (undef)	#	STOP (blank) (British-pound)
I	8	8	8	
J	'	BELL	'	,
K	(	(	1/2	
L	)	)	3/4	
M	.	.	?	
N	,	,	7/8	\
O	9	9	9	
P	0	0	0	
Q	1	1	1	
R	4	4	4	
S	BELL	'	BELL	(blank)
T	5	5	5	
U	7	7	7	
V	;	=	3/8	
W	2	2	2	
X	/	/	/	
Y	6	6	6	
Z	"	+	"	

Baudot USTTY and ITA2 Codes

Note that the USTTY and ITA2 codes are identical except for eight places in the FIGS case, and indicated below by <<<.

USTTY					ITA2					
decimal	hex	binary	LTRS	FIGS	decimal	hex	binary	LTRS	FIGS	
0	0x00	00000	BLANK	BLANK	0	0x00	00000	BLANK	BLANK	
1	0x01	00001	E	3	1	0x01	00001	E	3	
2	0x02	00010	LF	LF	2	0x02	00010	LF	LF	
3	0x03	00011	A	-	3	0x03	00011	A	-	
4	0x04	00100	SP	SP	4	0x04	00100	SP	SP	
5	0x05	00101	S	BELL	5	0x05	00101	S	'	<<<
6	0x06	00110	I	8	6	0x06	00110	I	8	
7	0x07	00111	U	7	7	0x07	00111	U	7	
8	0x08	01000	CR	CR	8	0x08	01000	CR	CR	
9	0x09	01001	D	$	9	0x09	01001	D	#	<<<
10	0x0A	01010	R	4	10	0x0A	01010	R	4	
11	0x0B	01011	J	'	11	0x0B	01011	J	BELL	<<<
12	0x0C	01100	N	,	12	0x0C	01100	N	,	
13	0x0D	01101	F	!	13	0x0D	01101	F	@	<<<
14	0x0E	01110	C	:	14	0x0E	01110	C	:	
15	0x0F	01111	K	(	15	0x0F	01111	K	(	
16	0x10	10000	T	5	16	0x10	10000	T	5	
17	0x11	10001	Z	"	17	0x11	10001	Z	+	<<<
18	0x12	10010	L	)	18	0x12	10010	L	)	
19	0x13	10011	W	2	19	0x13	10011	W	2	
20	0x14	10100	H	#	20	0x14	10100	H	$	<<<
21	0x15	10101	Y	6	21	0x15	10101	Y	6	
22	0x16	10110	P	0	22	0x16	10110	P	0	
23	0x17	10111	Q	1	23	0x17	10111	Q	1	
24	0x18	11000	O	9	24	0x18	11000	O	9	
25	0x19	11001	B	?	25	0x19	11001	B	?	
26	0x1A	11010	G	&	26	0x1A	11010	G	*	<<<
27	0x1B	11011	FIGS	FIGS	27	0x1B	11011	FIGS	FIGS	
28	0x1C	11100	M	.	28	0x1C	11100	M	.	
29	0x1D	11101	X	/	29	0x1D	11101	X	/	
30	0x1E	11110	V	;	30	0x1E	11110	V	=	<<<
31	0x1F	11111	LTRS	LTRS	31	0x1F	11111	LTRS	LTRS	

5-Level Weather Code

Starting in the 1940s, weather reports were sent hourly on landline circuits at 60 WPM. The weather, or "WX" symbols were used on model 15 and 19 machines with a special WX type basket. The WX code had special FIGS symbols representing cloud cover and wind direction

The WX service did not set their machines to unshift-on-space, which would have required a FIGS before each group of numerals (some weather codes were in groups of five numerals). The end-of-line sequence was typically LTRS CR LF LTRS, or, if the next line began with numerals, LTRS CR LF FIGS.

From a model 15 manual, the weather symbol set is defined as:

Lowercase (ltrs)	Uppercase Weather symbols (figs)	Definition
A	(up arrow)	North wind
B	(circle w/ vertical & horiz line)	Overcast -- More than 9/10 covered
C	(circle)	Sky clear -- less than 1/10 cloud cover
D	(upper-right arrow)	Northeast wind
F	(right arrow)	East wind
G	(lower-right arrow)	Southeast wind
H	(down arrow)	South wind
J	(lower-left arrow)	Southwest wind
K	(left arrow)	West wind
L	(upper-left arrow)	Northwest wind
N	(circle w/ two vertical lines)	Broken clouds -- More than 1/2 covered
V	(circle w/ one vertical line)	Scattered clouds -- less than 1/2 covered
Z	+	

The remainder of the upper case symbols are the same as USTTY.

6-Level TTS (Teletypesetting) Code

The six-level TTS (Teletypesetting) code was introduced by Walter Morey, to address the need for full upper and lowercase letters for newspaper and other publishing applications. The Teletype Model 20 is a six-level teletypewriter based on a Model 15 machine.

(need table)

7- and 8-Level Codes: ASCII-63 to ASCII-67 (ITA5)

The ASCII code (American Standard Code for Information Interchange) is fourty years old, but is still widely used today. ASCII-63, introduced in 1963, was a 7-level (7-bit) code which did away with the LTRS/FIGS case shifting of the original 5-level codes.

ASCII-63 only had uppercase letters -- upper and lowercase letters were both defined in the ASCII-67 (1967) version, which is still in use today. ASCII-67 was accepted as an international version of ASCII, and called ITA5 (International Telegraph Alphabet No. 5).

ASCII is technically a 7-bit code, but it is often referred to as an 8-bit code, since it is usually transmitted as 8-bits, with the 8th bit either space, mark, or even-parity. "8-level" ASCII teletype machines usually sent even parity ascii from the keyboard, but some (especially early units) sent bit-8 always mark. Model 33 and 35 machines were uppercase-only, and, except for DEL, characters from col-4 printed as col-3 (see table below).

ASCII-63 was mostly identical to the current ASCII-67 version. The definitions of the control characters (col-1 above) varied between the two versions, as defined below. Also, in ASCII-63, the upper 32 positions (col-4) were undefined, except for three: RUB (0x7F), ACK (0x7C), and ESC (0x7E). There are inconsistent references to an ALT-MODE char (0x7D) in ASCII-63. In the 1967 version, RUB became DEL and stayed in the same position, but ACK and ESC moved into the control character area (col-1). In ASCII-67, ^ replaced the up-arrow symbol, and _ replaced the left-arrow.

ASCII-63 and ASCII-67 are the common variants of ASCII, but there appear to have been some transitional versions as well: in the Teletype Model 33 manual, there are references to a 1965 version of ASCII, that had SS in place of SUB (0x1A), \ for @ (0x40), ~ for \ (0x5C), an odd character in place of | (0x7C), and | for ~ (0x7E). A Teletype code card for M33 and M35 machines indicates a 1966 version of ASCII, though the printable characters shown on the card were identical in all versions.

ASCII-67 (ITA5) Code

ASCII-67

col-1		col-2		col-3		col-4	
0x00	^@ NUL	0x20	SP	0x40	@	0x60	`
0x01	^A SOH	0x21	!	0x41	A	0x61	a
0x02	^B STX	0x22	"	0x42	B	0x62	b
0x03	^C ETX	0x23	#	0x43	C	0x63	c
0x04	^D EOT	0x24	$	0x44	D	0x64	d
0x05	^E ENQ	0x25	%	0x45	E	0x65	e
0x06	^F ACK	0x26	&	0x46	F	0x66	f
0x07	^G BEL	0x27	'	0x47	G	0x67	g
0x08	^H BS	0x28	(	0x48	H	0x68	h
0x09	^I HT	0x29	)	0x49	I	0x69	i
0x0A	^J LF	0x2A	*	0x4A	J	0x6A	j
0x0B	^K VT	0x2B	+	0x4B	K	0x6B	k
0x0C	^L FF	0x2C	,	0x4C	L	0x6C	l
0x0D	^M CR	0x2D	-	0x4D	M	0x6D	m
0x0E	^N SO	0x2E	.	0x4E	N	0x6E	n
0x0F	^O SI	0x2F	/	0x4F	O	0x6F	o
0x10	^P DLE	0x30	0	0x50	P	0x70	p
0x11	^Q DC1	0x31	1	0x51	Q	0x71	q
0x12	^R DC2	0x32	2	0x52	R	0x72	r
0x13	^S DC3	0x33	3	0x53	S	0x73	s
0x14	^T DC4	0x34	4	0x54	T	0x74	t
0x15	^U NAK	0x35	5	0x55	U	0x75	u
0x16	^V SYN	0x36	6	0x56	V	0x76	v
0x17	^W ETB	0x37	7	0x57	W	0x77	w
0x18	^X CAN	0x38	8	0x58	X	0x78	x
0x19	^Y EM	0x39	9	0x59	Y	0x79	y
0x1A	^Z SUB	0x3A	:	0x5A	Z	0x7A	z
0x1B	^[ESC	0x3B	;	0x5B	[	0x7B	{
0x1C	^\ FS	0x3C	<	0x5C	\	0x7C	|
0x1D	^] GS	0x3D	=	0x5D	]	0x7D	}
0x1E	^^ RS	0x3E	>	0x5E	^	0x7E	~
0x1F	^_ US	0x3F	?	0x5F	_	0x7F	DEL

^x is control key and character key together

ASCII Control Characters

The DC1 and DC3 control characters have been used as aux-device start/stop commands (eg: auto-tape-reader option in Teletype M33asr). DC2/DC4 have also been used as auto-tape-punch on/off commands in the M33asr.

DC1 and DC3 are also referred to as XON and XOFF respectively. XON and XOFF are widely used today as software-handshaking flow-control characters for throttling data transfer into a serial buffer.

ASCII-63 Control Characters	ASCII-67 Control Characters
NULL = Null char	NUL = Null char
SOM = Start of Message	SOH = Start of Heading
EOA = End of Address	STX = Start of Text
EOM = End of Message	ETX = End of Text
EOT = End of Transmission	EOT = End of Transmission
WRU = Who Are You	ENQ = Enquiry
RU = Are You	ACK = Acknowledge
BELL = Signal Bell	BEL = Signal Bell
FE0 = Format Effector 0	BS = Back Space
HT/SK = Horizontal Tab	HT = Horizontal Tab
LF = Line Feed	LF = Line Feed (aka NL = New Line)
VTAB = Vertical Tab	VT = Vertical Tab
FF = Form Feed	FF = Form Feed
CR = Carriage Return	CR = Carriage Return
SO = Shift Out	SO = Shift Out (special)
SI = Shift In	SI = Shift In (normal)
DC0 = Device Control 0	DLE = Data Link Escape
DC1 = Device Control 1	DC1 = Device Control 1 (aka XON) (M33 tape-reader-on)
DC2 = Device Control 2	DC2 = Device Control 2 (M33 tape-punch-on)
DC3 = Device Control 3	DC3 = Device Control 3 (aka XOFF) (M33 tape-reader-off)
DC4 = Device Control 4	DC4 = Device Control 4 (M33 tape-punch-off)
ERR = Error	NAK = Negative Acknowledge
SYNC = Synchronous	SYN = Synchronous Idle
LEM = Logical End of Medium	ETB = End of Transmission Block
S0 = Special Application 0	CAN = Cancel
S1 = Special Application 1	EM = End of Medium (Media?)
S2 = Special Application 2	SUB = Substitute
S3 = Special Application 3	ESC = Escape
S4 = Special Application 4	FS = Field Separator
S5 = Special Application 5	GS = Group Separator
S6 = Special Application 6	RS = Record Separator
S7 = Special Application 7	US = Unit Separator
RUB = Rubout	DEL = Delete
ACK = Acknowledge	(position 0x7C became |)
ALT-MODE = Alternate Mode	(position 0x7D became })
ESC = Escape	(position 0x7E became ~)

ASCII-67 to Baudot-USTTY Conversion

A translation of ASCII-67 to Baudot-USTTY, as implemented in TTY-Connect.

- automatically change case as needed by sending LTRS or FIGS: L=LTRS, F=FIGS, N=No-Change
- use SO to explicitly insert FIGS, SI to to explicitly insert LTRS (special uses)
- ascii lowercase chars converted to uppercase
- map control chars NUL, BEL, LF and CR
- no translation for other control chars, or ( ) < > [ ] { } ? % * + = @ ` _ \ ^ | ~

ASCII-67				USTTY translation			
col-1	col-2	col-3	col-4	col-1	col-2	col-3	col-4
^@ NUL	SP	@	`	N 0x00 BLANK	N 0x04 SP	--	--
^A SOH	!	A	a	--	F 0x0D !	L 0x03 A	L 0x03 A
^B STX	"	B	b	--	F 0x11 "	L 0x19 B	L 0x19 B
^C ETX	#	C	c	--	F 0x14 #	L 0x0E C	L 0x0E C
^D EOT	$	D	d	--	F 0x09 $	L 0x09 D	L 0x09 D
^E ENQ	%	E	e	--	--	L 0x01 E	L 0x01 E
^F ACK	&	F	f	--	F 0x1A &	L 0x0D F	L 0x0D F
^G BEL	'	G	g	F 0x05 BELL	F 0x0B '	L 0x1A G	L 0x1A G
^H BS	(	H	h	--	F 0x0F (	L 0x14 H	L 0x14 H
^I HT	)	I	i	--	F 0x12)	L 0x06 I	L 0x06 I
^J LF	*	J	j	N 0x02 LF	--	L 0x0B J	L 0x0B J
^K VT	+	K	k	--	--	L 0x0F K	L 0x0F K
^L FF	,	L	l	--	F 0x0C ,	L 0x12 L	L 0x12 L
^M CR	-	M	m	N 0x08 CR	F 0x03 -	L 0x1C M	L 0x1C M
^N SO	.	N	n	N 0x1B FIGS	F 0x1C .	L 0x0C N	L 0x0C N
^O SI	/	O	o	N 0x1F LTRS	F 0x1D /	L 0x18 O	L 0x18 O
^P DLE	0	P	p	--	F 0x16 0	L 0x16 P	L 0x16 P
^Q DC1	1	Q	q	--	F 0x17 1	L 0x17 Q	L 0x17 Q
^R DC2	2	R	r	--	F 0x13 2	L 0x0A R	L 0x0A R
^S DC3	3	S	s	--	F 0x01 3	L 0x05 S	L 0x05 S
^T DC4	4	T	t	--	F 0x0A 4	L 0x10 T	L 0x10 T
^U NAK	5	U	u	--	F 0x10 5	L 0x07 U	L 0x07 U
^V SYN	6	V	v	--	F 0x15 6	L 0x1E V	L 0x1E V
^W ETB	7	W	w	--	F 0x07 7	L 0x13 W	L 0x13 W
^X CAN	8	X	x	--	F 0x06 8	L 0x1D X	L 0x1D X
^Y EM	9	Y	y	--	F 0x18 9	L 0x15 Y	L 0x15 Y
^Z SUB	:	Z	z	--	F 0x0E :	L 0x11 Z	L 0x11 Z
^[ESC	;	[	{	--	F 0x1E ;	--	--
^\ FS	<	\	|	--	--	--	--
^] GS	=	]	}	--	--	--	--
^^ RS	>	^	~	--	--	--	--
^_ US	?	_	DEL	--	F 0x19 ?	--	--

Baudot-USTTY to ASCII-67 Conversion

A translation of Baudot-USTTY to ASCII-67, as implemented in TTY-Connect.

- current case (LTRS or FIGS) determines actual ASCII character mapped for specific baudot char

USTTY				ASCII translation				
				LTRS		FIGS		
hex	binary	LTRS	FIGS					
0x00	00000	BLANK	BLANK	0x00	NUL	0x00	NUL	
0x01	00001	E	3	0x45	E	0x33	3	
0x02	00010	LF	LF	0x0A	LF	0x0A	LF	
0x03	00011	A	-	0x41	A	0x2D	-	
0x04	00100	SP	SP	0x20	SP	0x20	SP	
0x05	00101	S	BELL	0x53	S	0x07	BEL	
0x06	00110	I	8	0x49	I	0x38	8	
0x07	00111	U	7	0x55	U	0x37	7	
0x08	01000	CR	CR	0x0D	CR	0x0D	CR	
0x09	01001	D	$	0x44	D	0x24	$	
0x0A	01010	R	4	0x52	R	0x34	4	
0x0B	01011	J	'	0x4A	J	0x27	'	
0x0C	01100	N	,	0x4E	N	0x2C	,	
0x0D	01101	F	!	0x46	F	0x21	!	
0x0E	01110	C	:	0x43	C	0x3A	:	
0x0F	01111	K	(	0x4B	K	0x28	(	
0x10	10000	T	5	0x54	T	0x35	5	
0x11	10001	Z	"	0x5A	Z	0x22	"	
0x12	10010	L	)	0x4C	L	0x29	)	
0x13	10011	W	2	0x57	W	0x32	2	
0x14	10100	H	#	0x48	H	0x23	#	
0x15	10101	Y	6	0x59	Y	0x36	6	
0x16	10110	P	0	0x50	P	0x30	0	
0x17	10111	Q	1	0x51	Q	0x31	1	
0x18	11000	O	9	0x4F	O	0x39	9	
0x19	11001	B	?	0x42	B	0x3F	?	
0x1A	11010	G	&	0x47	G	0x26	&	
0x1B	11011	FIGS	FIGS	0x0E	SO	0x0E	SO	(optional)
0x1C	11100	M	.	0x4D	M	0x2E	.	
0x1D	11101	X	/	0x58	X	0x2F	/	
0x1E	11110	V	;	0x56	V	0x3B	;	
0x1F	11111	LTRS	LTRS	0x0F	SI	0x0F	SI	(optional)

Appendix 1 – PC232 Serial Port (DB9-F)

The PC232 Port is designed to connect directly to any computer that has a standard RS-232 serial port. A DB9-F (female) connector allows a PC to connect using a standard straight 9-pin male-female cable. The 232 signals are converted to logic-level, and are available for either driving an optional on-board microcontroller, or for patching directly to one of the opto-isolated TTY loops. If the PC232 port is patched directly to a loop, the PC program must be able to send the correct word format required by the TTYs (eg: 5-bit, 60-wpm baudot).

Typically, only three wires are needed: TX, RX, and Ground. If your serial port expects handshake inputs (on CTS, DSR, and/or DCD), you may be able to either change your port configuration to ignore these signals, or you may provide the 5V signals from the PC232 connector to drive them active. FYI: the proper term for the connector is DE9, but DB9 is common terminology. Likewise, the proper term for the interface is EIA-232, but RS-232 is the defacto lingo.

The DTR and RTS signals on the PC232 jack are buffered and available on the AUX jack, an 8P8C modular connector. These buffered outputs may be used for radio keying, or other functions; the PC program can then control the state of these lines as needed. These lines drive open-collector transistors (not opto-isolated). When the RTS or DTR line is active (positive) the associated transistor will pull to ground and sink up to 100 mA (40V max). If you are driving a relay (or other inductive load), be sure to add a freewheel diode across the coil (anode to collector, cathode to v+) to protect the transistor from the di/dt spike.

The PC232 serial port requires 38400 baud, 8 data bits, no parity, 1 stop bit, with Xon/Xoff flow control enabled. The code is usually ascii, but special codes can be used in raw mode.

Pin	PC232 Port	Signal Direction	Standard Computer Port
1	5V *	→	DCD (Data Carrier Detect)
2	Data Out	→	RXD (Receive Data)
3	Data In	←	TXD (Transmit Data)
4	DTR	←	DTR (Data Terminal Ready)
5	Signal Ground	—	Signal Ground
6	5V *	→	DSR (Data Set Ready)
7	RTS	←	RTS (Request To Send)
8	5V *	→	CTS (Clear To Send)
9	5V *	→	RI (Ring Indicator)

- Typical Connections: Data-In (TXD), Data-Out (RXD), and Signal Ground.
- Minimal Connections: Data-In (TXD), and Signal Ground (for commands only)
- DTR and RTS signals drive open-collector transistors on the AUX port.
- The DB9 shield rim (the "D") is connected to signal ground.
- Signal Ground is common to the TU232 port, but is isolated from TTY loops.
- * If CTS, DSR, or DCD is needed by your serial port, connect to the 5V signal (1-mA max load).

Appendix 2 – TU232 Serial Port (6P6C)

The TU232 Port is a 6P6C modular jack for connection of a Terminal Unit (TU) with an RS-232 interface -- for receive and/or transmit use. The signals are available for either driving an optional on-board microcontroller, or for patching directly to one of the opto-isolated TTY loops.

If the TU232 port is patched directly to a loop, the TTYs must be able to receive the correct word format received by the radio/TU (eg: 5-bit, 75-wpm baudot). This interface is for standard RS-232 signals (mark < -3V, space > +3V). If the optional PIC microcontroller is installed, there is a software inversion available for connecting to a TU with MIL-188 signals (mark = +5V, space = -5V).

If the optional PIC micro is installed, there is also a push-to-talk (PTT) signal available on the TU232 jack – this is an open-collector transistor (not opto-isolated), for keying a transmitter. When the micro drives this line active, the transistor will pull to ground and sink up to 100 mA (40V max). If you are driving a relay (or other inductive load), be sure to add a freewheel diode across the coil (anode to collector, cathode to v+) to protect the transistor from the di/dt spike.

For connection to Tempest Dovetrons, a special cable with a couple of BNC plugs and a 6P6C modular plug will be needed, to connect to the Dovetron's POLAR (232 in) and OUTPUT (232 out) jacks. Or, you could tap into these points internally, and just bring a 6P6C cable out the back of the Dovetron.

Pin	TU232 Port	Signal Direction	TU Interface	Std Colors *
1	/PTT (open-collector)	→	(activate tx when low)	Wht
2	5V	→	-	Blk
3	Data In	←	TXD (Dovetron OUTPUT)	Red
4	Signal Ground	—	Signal Ground	Grn
5	Data Out	→	RXD (Dovetron POLAR)	Yel
6	Signal Ground	—	Signal Ground	Blu

- Typical Connections: Data-In (TXD), Data-Out (RXD), and Signal Ground.
- Minimal Connections: Data-In (TXD), and Signal Ground (for TU receive only)
- PTT (push-to-talk) signal is an open-collector transistor – transistor is on (sinks current) when active.
- 5V signal has 1-mA max load.
- Signal Ground is common to the PC232 port, but is isolated from TTY loops.
- * Standard colors apply when using flat 6-conductor modular cable, crimped to the plug in the appropriate direction

Appendix 3 – AUX Port (8P8C)

The DTR and RTS signals from the PC232 jack are buffered and available on the AUX Port, an 8P8C modular connector. These buffered outputs may be used for radio keying, or other functions; the PC program can then control the state of these lines as needed. These lines drive open-collector transistors (not opto-isolated). When the RTS or DTR line is active (positive) the associated transistor will pull to ground and sink up to 100 mA (40V max). If you are driving a relay (or other inductive load), be sure to add a freewheel diode across the coil (anode to collector, cathode to v+) to protect the transistor from the di/dt spike.

Pin	AUX Port	Signal Direction	Use	Std Colors *
1	TX-T	←	(reserved)	Wht/Orn
2	TX-R	←	(reserved)	Orn/Wht
3	RX-T	→	(reserved)	Wht/Grn
4	/DTR (open-collector)	→	Buffered DTR	Blu/Wht
5	Signal Ground	—	Signal Ground	Wht/Blu
6	RX-R	→	(reserved)	Grn/Wht
7	Signal Ground	—	Signal Ground	Wht/Brn
8	/RTS (open-collector)	→	Buffered RTS	Brn/Wht

- DTR and RTS signals are open-collector transistors – transistor is on (sinks current) when active.
- Signal Ground is common to the PC232 and TU232 ports, but is isolated from TTY loops.
- * Standard colors apply when using a CAT-5 UTP cable, crimped to the 8P8C modular plug according to the EIA-568B spec

Appendix 4 – INTERCONNECT header (20-pin)

The signals on the INTERCONNECT header are at logic levels (5V=mark, 0V=space). This connector can be used for manually patching sections together, when the on-board micro is not installed. Example: to connect the PC232 port directly to the HV1 loop, jumper PC232-IN to HV1-OUT, and jumper PC232-OUT to HV1-IN.

For a custom application, the board can be connected elsewhere via a ribbon cable. Note that grounds are interleaved with the signals to minimize coupling/radiation/susceptibility when using a flat parallel ribbon cable.

Pin	INTERCONNECT	Signal Direction
1	PC232-IN	→
2	Gnd	—
3	PC232-OUT	←
4	Gnd	—
5	HV1-IN	→
6	Gnd	—
7	HV1-OUT	←
8	Gnd	—
9	TU232-IN	→
10	Gnd	—
11	TU232-OUT	←
12	Gnd	—
13	LV-IN	→
14	Gnd	—
15	LV-OUT	←
16	Gnd	—
17	HV2-IN	→
18	Gnd	—
19	HV2-OUT	←
20	Gnd	—

Appendix 5 – EXP-1/MOT header (10-pin)

The signals on the EXP-1/MOT header are at logic levels (5V=active, 0V=inactive). This connector is used when the on-board micro is installed, and provides signals for driving motor-control relays or other circuitry. These signals are direct outputs from the micro, and must be properly buffered by the optional circuitry.

Note that RF grounds are interleaved with the signals to minimize coupling/radiation/susceptibility when using a flat parallel ribbon cable.

Pin	EXP-1/MOT	Signal Direction	Use
1	P1A4	→	(reserved)
2	+12V	→	(100mA max)
3	P1A3	→	(reserved)
4	Gnd	—	
5	HV2-MOT	→	HV2 Motor On when high
6	Gnd	—	
7	HV1-MOT	→	HV1 Motor On when high
8	+5V	→	(100mA max)
9	LV-MOT	→	LV Motor On when high
10	/RESET	←	(reserved)

Appendix 6 – EXP-2/LCD header (10-pin)

The signals on the EXP-2/LCD header are at 5V logic levels. This connector is used when the on-board micro is installed, and provides signals for driving an lcd or other circuitry. These signals are direct outputs from the micro, and must be properly buffered by the optional circuitry.

Pin	EXP-2/LCD	Signal Direction	Use
1	/CONTROL_PB	←	(optional – active-low contact closure)
2	+12V	→	(100mA max)
3	STATUS_LED	→	(optional – active-high) *
4	Gnd	—	
5	T1MON	→	(reserved)
6	Gnd	—	
7	SDA	←→	I2C-SDA (lcd)
8	+5V	→	(100mA max)
9	SCL	→	I2C-SCL (lcd)
10	SCIMON	→	(reserved)

- * Status led flashes when TC commands processed, and is on while PC232 port input is paused by XON throttling.

Appendix 7 – Updating TTY-Connect Firmware

- System uses flash-memory-based PIC controller, with programmed bootloader, allowing software download from windows PC (XP, 98...) using PIC-Downloader utility
- Obtain appropriate system software file and PIC-Downloader utility
- Connect serial cable from TTY-Connect PC232 port to com port on PC (straight-through DB9, pins 2,3,5 min)
- Start PIC-Downloader on PC
- Leave the BD (baud) selection at 19200 (this is the download speed, not the TTY-Connect operation speed)
- Set PORT to COM1 (or whichever com port you are using)
- Press Search/F2 button, select file to download (eg: tc_1_0.hex)
- Press Write/F4 button, and you will get a "Searching For Bootloader" message
- Power TTY-Connect unit off for a second, then back on, and you will get a "writing, please wait" message
- PIC-Downloader shows progress during download (about 45 secs to load)
- If file has been transferred correctly, you will get an "All OK" message from the downloader, and a boot message (-.TC,0...) sent by TTY-Connect
- Done

Appendix 8 – Using HyperTerminal with TTY-Connect

1) Start the HyperTerminal application on the Windows PC (XT or 98).
    1. Start/Programs/Accessories/Communications/HyperTerminal
    2. New-Connection dialog: Name = tty-connect
    3. Connect-To dialog: Connect Using Com 1 (or 2,3,4)
    4. Com-1 Properties/Port Settings: (in Win 98, this is in File/Properties/Configure)
        1. Bits-per-second: 38400
        2. Data Bits: 8
        3. Parity: None
        4. Stop Bits: 1
        5. Flow Control: Xon/Xoff
        6. Apply/OK
    5. File/PropertiesSettings/Ascii-Setup: enable or disable Echo-Charaters-Locally, as desired
    6. File/Save-As tty-connect.ht on Desktop (after this you will simply double-click the icon to start)
2) Connect the appropriate serial cable between the Computer and the PC232 port – this is usually an off-the-shelf 9-pin straight-through male-female cable.
3) Test the serial connection: Turn your TTY-Connect unit off and back on -- you should see a message similar to "TTY-Connect Ver: 1.0" displayed in your terminal window (the last numbers are the firmware version). There will be a couple of other lines displayed as well. Now type /.TR,0,0 and then press return – you should get a message like: -.TC,0,4,0,0,1,0 where the last two digits are the firmware version.

Appendix 9 – PC232 Control Examples

The Connect Commands set the primary configuration of the machine.

Connect Command Description	*Commands*	*Status-Message*
Get Current Connection: **> PC232 is connected to HV1 at 60-wpm**	**/.TR,1,0 <cr>**	**-.TC,1,2,1,60 <cr>**
Connect Ascii PC232 to Baudot HV2 at 100-wpm:	**/.TW,1,2,2,100 <cr>**	**-.TC,1,2,2,100 <cr>**
Connect Ascii PC232 to Ascii LV at 110-baud:	**/.TW,2,1,3 <cr>**	**-.TC,2,1,3 <cr>**
Connect Baudot TU232 to Baudot HV2 at 75-wpm:	**/.TW,4,2,2,75 <cr>**	**-.TC,4,2,2,75 <cr>**
Connect TU port at 75-wpm to HV1 at 60-wpm:	**/.TW,6,4,4,75,1,60 <cr>**	**-.TC,6,4,4,75,1,60 <cr>**

The Control Commands alter other aspects of the TTY-Connect system.

Control Command Description	*Command*	*Status-Message*
Turn on HV1 motor:	**/.TW,20,1,1 <cr>**	**-.TC,20,2,1,1<cr>**
Turn off LV motor:	**/.TW,20,3,0 <cr>**	**-.TC,20,2,3,0<cr>**
Activate PTT:	**/.TW,20,4,1 <cr>**	**-.TC,20,2,4,1<cr>**

The Programming Commands change customizable settings.

Programming Command Desc	*Command*	*Status-Message*
Enable RX-Unshift-on-Space	**/.TW,70,1,1 <cr>**	**-.TC,70,1,1 <cr>**
Disable TU-Data-Inversion	**/.TW,41,1,0 <cr>**	**-.TC,41,1,0 <cr>**
Enable Auto-CRLF-RX-Insertion	**/.TW,49,1,1 <cr>**	**-.TC,49,1,1 <cr>**
Load new Auto-CRLF-String	**/.TW,90,8,8,8,2,31,31,0,0,0 <cr>** **> string = CR CR LF LTRS LTRS**	**-.TC,90,8,8,8,2,31,31,0,0,0 <cr>**