PANDORA PRODUCTS

Pandora Products Manual
TERMS Terminal Server

Jim Schimpf

Document Number: PAM-200512002
Revision Number: 0.4
18 December 2005

Pandora Products.
215 Uschak Road
Derry, PA 15627

Pandora Products.

215 Uschak Road

Derry, PA 15627

Phone: 412.829.8145

Fax: 724.539.1276

Web: pandora.dyn-o-saur.com:8080/~jim/

Email: vze35xda@verizon.net

Document Revision History

Version	Author	Description	Date
0.1	js	Initial Version	25-Jun-2004
0.2	js	Convert to TR Format	18-Dec-2005
0.3	js	Add telnet use section	22-Dec-2005
0.4	js	Update telent use with bash script	13-Apr-2006

Contents

1 Introduction

TERMS is a simple terminal server running in a UNIX/Linux system. It connects a single serial port to a single TCP/IP port. In addition it is extendible via a Lua [1, 2] script. The Lua script will have functions with defined names (see 4) and these functions will be called by events in TERMS.

A terminal server is simply a device that connects a logical TCP/IP socket to a physical serial port. This allows use of normally serial only devices over a TCP/IP network. The terminal server has two major states, idle and connected. Idle means it owns the serial port and has an available socket on the TCP/IP network but no one is connected to the socket. It enters the connected state when a it accepts a TCP/IP socket connection. When the TCP/IP socket connection is dropped it then returns to idle.

Run with no options TERMS does this but when used with a Lua script, functions in that script can be called on these changes of state. This means for example it can toggle the serial DTR line on a connect or send keep alive messages to the serial port while connected.

This program makes it possible to access a serial port on a machine from anywhere on a network. Running it with existing programs like **telnet** a free VT100 serial terminal can be created.

2 Syntax

```
TERMS [-p <port #/Name>] [-baud <baud rate>] [-tcp_p <Port #>]
      [-c <Lua Script>] [-v]
```

	Option	Default
TERMS	-p <serial port #/Path	1
	-baud <Baud Rate>	38,400
	-tcp_p <TCP/IP Port #>	6000
	-c <Lua Script>	NONE
	-v	NONE

TERMS is run from the command line and with no options will be a terminal server connecting Serial port 1 to TCP/IP socket 6000. The serial port will be set at 38,400 baud and none of the control lines (DTR,RTS...) will be asserted.

2.1 Option Details

2.1.1 -p <Serial port #/Path>

This specifies the serial port number or path. If the -v option returns a list of ports you can use the slot number from that list as the port #. If not you may use the device path i.e. /dev/ttyS0

DEFAULT = “1”

2.1.2 -baud <Baud Rate>

This would set the baud rate for the port. Use a valid rate since there is no checking at the program level of this value, the underlying OS might check but try to use a *standard* value.

DEFAULT = 38,400 baud

2.1.3 -tcp_p <TCP/IP Port #>

This is the TCP/IP side of the connection. It specifies the port TERMS will open on its machine. TCP/IP connections would be made to <TERMS machine>:port#. This can be any value from 1-65535 but don't use ports already in use by your machine or in the case of UNIX non-admin users, they cannot use any port less than 1024.

DEFAULT = 6000

2.1.4 -c <Lua script>

The name following this is a Lua script that can modify the behavior of TERMS. See 4

2.1.5 -v

When present TERMS will print the following and NOT run

```
TERMS TCP/IP <=> SER SERVER [Dec  2 2005 10:29:20]
TERMS - TCP -> SERIAL SERVER Version Dec  2 2005 10:29:20
SYNTAX: TERMS   [-p <serial port #>]
                [-baud <Baud rate>]
                [-tcp_p <Port #>]
                [-c <Lua connect Script>]
                [-e]
                [-v]
    -p     <Serial port #>     DEFAULT = 1
    -baud  <Rate>          DEFAULT = 38400
    -tcp_p <TCP Port #>    DEFAULT = 6000
    -c     <Lua script>    DEFAULT = NONE
    -e Eat echo characters DEFAULT = Allow echo
    -v     Show this message and return
----- PORT LIST ----
[1] [cu.usbserial0] [/dev/cu.usbserial0]
[2] [cu.Bluetooth-PDA-Sync][/dev/cu.Bluetooth-PDA-Sync]
--------------------
```

3 Terminal Server Use

3.1 Introduction

TERMS attaches a serial port to a TCP/IP port and allows communication between them. This has numerous uses allowing the local serial port to be used by any machine on the network (or on the Internet). The connecting program can be a special purpose system designed to talk to the hardware attached to the serial port or it can be **telnet** used for user communication over the serial port.

3.2 telnet Use

This is one of the more interesting uses of TERMS. You can connect to the TERMS serviced serial port with telnet and in effect have serial terminal. **telnet** is a good client for this since it supports the VT100 command set.

For very nice VT100 emulation you can use a script like this run TERMS and telnet in the same window.

```
#!/bin/bash
#----- Start up TERMINAL session in a window
# Syntax: myterm <ser port> <tcpip port>
# Uses Default baud rate 38400# You can add more parameters or set them
# to fixed values
# The TCP/IP port set value is present so I can run
# multiple terminals on multiple ports and not have them
# try for the same IP port
TERMS -p $1 -tcp_p $2 &
# Wait for TERMS to run
sleep 2
# Now run TELNET to that port
telnet localhost $2
# Kill that running TERMS service
kill %+
```

The script starts TERMS in the background with the default baud rate on a user specified IP port using a user specified serial port. This script is only a suggestion, you can put in any sorts of values you need for your particular use. The result of this is a single window with TERMS running the background with **telnet** acting as a VT100 terminal. When you exit telnet the kill%+ acts to kill the TERMS which is still running.

3.3 telnet Initialization

In some cases to get correct appearence when typing or receiving characters you will need to set **telnet** for binary transfer. In that case you can modify the .telnetrc file in your local directory to be as follows:

```
#
#----- TELNET STARTUP FILE ----
# Binary/Character mode for DMU's
# Line/non-binary for LC servers
#
DEFAULT mode character
DEFAULT set binary
```

You can also issue these commands to **telnet** by escaping the terminal mode and getting to the > prompt then typing:

```
mode character
set binary
```

4 Lua Script Use

4.1 Scripting

Terminal servers by their nature need some customization so their behavior can change to suit the wide variety of serial devices that can be connected. Lua the scripting language allows TERMS controled by the script, then the actions it takes can be customized.

The mechanism for customization is to have TERMS call specific Lua functions on events. That is when TERMS starts up it calls the Lua function ON_START(), when a client connnects, it calls ON_CONN() and so on. If these functions are not present the program just keeps on going. Thus to customize the program you only need write functions to handle actions of importance any unneeded functions can just be skipped.

Information on the Lua language can be found here: Lua language site www.Lua.org and this book [2].

4.2 Lua Event Functions

The state leading to the function call will be detailed below, see 5 for further information on what functions are available.

4.2.1 ON_START()

This function is called as TERMS is starting but before it has begun monitoring for TCP/IP connection.

4.2.2 ON_CONN()

This function is called when a new TCP/IP connection is accepted but before TERMS enters into the terminal server loop, or the connected state.

4.2.3 ON_DROP()

This function is called after the TCP/IP connection is dropped and before TERMS enters the idle loop.

4.2.4 ON_CTS()

This function is called when CTS changes state (there are functions to see the current state see 5).

4.2.5 ON_PERIODIC()

This function is called every second TERMS is running (in both idle and connected states)

5 Lua Extensions

There are libraries added to this embedded Lua to supply the specialized functions for TERMS operation. There are three libraries added, timer, serial and net.

5.1 timer - Library

5.1.1 Open a timer timer.topen()

This function is used to open a millisecond timer for delay loops or other timing operations. There can be any number of these timers created during a run of TERMS.

```
t = timer.topen()
```

INPUT	NAME	USE
	-	
OUTPUT	t	Handle to created timer struct or nil if failure

5.1.2 Close a timer timer.tclose(t)

This function is used to free the memory used by a timer after use.

timer.tclose(t)

INPUT	NAME	USE
	t	This is the handle of a timer, the memory will be freed
OUTPUT	NONE	

5.1.3 Restart a timer timer.tstart(t)

This function is used to zero a timer. For example after a delay has elapsed then start can be called and the delay re-run.

timer.tstart(t)

INPUT	NAME	USE
	t	This timer will be re-started after this call
OUTPUT	NONE	

5.1.4 Check a timer for done timer.tdone(t,delay)

This function will return 1 when the timer passes the delay time. Up to that time it will return nil.

done = timer.tdone(t,delay)

INPUT	NAME	USE
	t	This is the handle of a timer
	delay	Delay time in ms
OUTPUT	done	nil till the timer passes the delay time and 1 after

5.2 ser - Library

This libary is used to access the open serial port of TERMS. It will be used to send or recieve data. In addition it can monitor and change the other serial control lines (DTR,CTS)

5.2.1 Write data to serial port ser.swrite(data)

This function will send data to the open serial port.

count = ser.swrite(data)

INPUT	NAME	USE
	data	This string of data will be sent to the serial port
OUTPUT	count	# of bytes sent

5.2.2 Read data from serial port ser.sread(len)

This function will read data from the serial port. This is a blocking read.

```
data = ser.sread(len)
```

INPUT	NAME	USE
	len	# Bytes of data to read
OUTPUT	data	Data as read from the port

5.2.3 Serial data status ser.sstat()

This function will return the count of available serial data bytes.

```
count = ser.sstat()
```

INPUT	NAME	USE
	-	
OUTPUT	count	# Bytes waiting to be read from serial port

5.2.4 Read serial control lines ser.sgetstate()

This function will return the status of the serial port status lines (CTS,RTS....). At present it only returns status of CTS

```
cts[,rts....] = ser.sgetstate()
```

INPUT	NAME	USE
	-	
OUTPUT	cts	1 if CTS asserted, 0 if not

5.2.5 Set serial control lines ser.ssetstate(dtr)

This function will set the state of the serial port status lines (DTR,RTS...). At present it only sets DTR

```
ser.ssetstate(dtr)
```

INPUT	NAME	USE
	dtr	Desired state of DTR, 1 = asserted, 0 = not
OUTPUT	NONE	-

5.3 net - Library

This libary is used to access the open TCP/IP port of TERMS. It will be used to send or recieve data. In addition it can monitor the connection status.

5.3.1 Write data to TCP/IP port net.swrite(data)

This function will send data to the open TCP/IP port.

```
count = net.swrite(data)
```

INPUT	NAME	USE
	data	This string of data will be sent to the TCP/IP port
OUTPUT	count	# of bytes sent

5.3.2 Read data from TCP/IP port net.sread(len)

This function will read data from the TCP/IP port. It will read up to len bytes and returns.

```
data = net.sread(len)
```

INPUT	NAME	USE
	len	# Bytes of data to read
OUTPUT	data	Data as read from the port

5.3.3 Control TCP/IP connection net.drop()

When called this will drop the TCP/IP connection

```
net.drop()
```

INPUT	NAME	USE
	-	
OUTPUT	NONE	

5.3.4 Read TCP/IP connection status net.cstat()

This function will return the status of the TCP/IP connection, 1 if connected and nil if idle

```
status = net.cstat()
```

INPUT	NAME	USE
	-	
OUTPUT	status	1 if connection up, nil if idle

6 Example Script

This is a simple example script that uses all the TERMS Lua functions. It will show a message on start, toggle DTR on connect, drop DTR on drop, show the state of CTS and put out periodic messages while a client is connected to the port.

```
-------------------------------------------------------------------------
--
-- function ON_START()    - Run at TERMS startup
--
-- INPUT:   NONE
--
-- OUTPUT:  NONE
--
-------------------------------------------------------------------------
function ON_START()
   local out
   print()
   print("--------------------------------------")
   out = string.format("**** TERMS SERVER UP *****")
   print(out)
   out = string.format("[%s]\n[%s]",
                       "Lua 5.0.2",
            "R. Ierusalimschy, L. H. de Figueiredo & W. Celes")
   print(out)
   print("--------------------------------------")
   print()
end
-------------------------------------------------------------------------
--
-- function ON_CONN()    - Run at network connection startup
--
-- INPUT:   NONE
--
-- OUTPUT:  NONE
--          This version will toggle DTR on connection and
--          send a crcr out of the serial port
--
-------------------------------------------------------------------------
function ON_CONN()
   local out
   local t
   local count
   out = string.format("Connection UP\n")
   print(out)
   net.swrite(out)
   -- Now Lower DTR hold for 2 seconds then raise
   print("Togging DTR")
   ser.ssetstate( 0 )
   t = timer.topen()
   if( t ~= nil )
   then
       while( timer.tdone(t,2000) == nil )
       do
          out = nil
       end
       timer.tclose(t)
   end
   print("DTR ASSERTED")
   ser.ssetstate( 1 )
   ser.swrite("\r\n")
end
-------------------------------------------------------------------------
--
-- function ON_DROP()    - Run at network drop
--
-- INPUT:   NONE
--
-- OUTPUT:  NONE
```

```
--          This will lower DTR on exit which will cause the connected
--          unit to logout
--
-----------------------------------------------------------------------
function ON_DROP()
   local out
   out = string.format("Connection DOWN")
   print(out)
   print("Lower DTR")
   ser.ssetstate( 0 )
end
-----------------------------------------------------------------------
--
-- function ON_CTS()    - Run at CTS change
--
-- INPUT:   NONE
--
-- OUTPUT:  NONE
--
-----------------------------------------------------------------------
function ON_CTS()
   local out
   local val
   val = ser.sgetstate()
   out = string.format("CTS = [%d]",val)
   print(out)
end
-- These globals used by ON_PERIODIC
count = 0
nettime = 0
-----------------------------------------------------------------------
--
-- function ON_PERIODIC()    - Run every second
--
-- INPUT:   NONE
--
-- OUTPUT:  NONE
--          Print connection status and time
--
-----------------------------------------------------------------------
function ON_PERIODIC()
   local out
   count = count + 1
   if( count > 5 )
   then
       if( net.cstat() )
       then
           out = string.format("Periodic Message [NET UP %d sec]",                                                    ne
           nettime = nettime + 5
       else
           out = string.format("Periodic Message")
           nettime = 0
       end

       print(out)
       count = 0
   end
end
```

Index

References

[1] W. C. R. Ierusalimschy, L. H. de Figueiredo, “Lua 5.0 reference manual.”

[2] R. Ierusalimschy, *Programming in Lua*. CIP - Bibioteca do Departmento de Informatica, PUC-Rio, 2003.