Terminal Manager Description, Installation, Startup and Operation

[Description: 2](#_Toc455213446)

[Components: 2](#_Toc455213447)

[Installation: 2](#_Toc455213448)

[Startup: 3](#_Toc455213449)

[Terminal Manager Statistics: 5](#_Toc455213450)

[Terminal Manager Monitoring: 6](#_Toc455213451)

[Authentication: 7](#_Toc455213452)

[Testing: 8](#_Toc455213453)

[TMDriver 8](#_Toc455213454)

[X3270 9](#_Toc455213455)

[Running a test script: 9](#_Toc455213456)

[GLINK 10](#_Toc455213457)

# Description:

The Terminal Manager is a C application designed to accept connections from any number of 3270 terminals and forward user input to servlets resident on a JBOSS container. The servlet forwards the input to remote processes (TPRs) which will perform CICS operations, collect the responses and return the results to Terminal Manager and back to the terminals originating the requests.

## Components:

Terminal Manager (TM ) is composed of a terminal processor, sessions (JBOSS interface) manager, queue manager, poll (deferred processing) manager, authentication manager, hash manager, remote processors (TPR interface) manager, and a statistics/monitor manager.

* Terminal Dispatch – composed of a Listener Thread to accept terminal connections and add them to an *Epoll* interface set and execute a 3270 handshake to establish a 3270 connection. The *Epoll* interface thread will loop through the established 3270 terminals and process them through a state engine.
* Sessions Dispatch - composed of a pool of threads built on the *ibcurl* API that represent *JBoss* interfaces. The pool is configurable at startup and can also grow dynamically. Input from the terminal processor state engine will be directed through these threads. Terminal/*JBoss* connections are initially established for each terminal, then subsequent user enquiries will be processed in these. Connection phases are processed via the queue and poll managers to allow simultaneous multiple terminal processing.
* Queue manager – supports deferred operations, e.g. *JBoss* connections and terminal authentications.
* Poll manager - supports deferred operations, e.g. *JBoss* connections and terminal authentications.
* Authentication manager – User credentials are processed here via a PAM interface. These are processed via the queue and poll managers to allow simultaneous multiple terminal processing.
* Hash manager – responsible for mapping of terminals to their assigned ID’s during the terminal phase and steering TPR responses back to their terminals.
* Remote Dispatch - composed of a Listener Thread to accept TPR connections and add them to an *Epoll* interface set . A thread is created for each remote connection to provide concurrent processing of TPRs. The main *Epoll* interface thread will loop through the TPRs selects and signal its thread to process its data back to the target terminal.
* Statistics/Monitor manager – Statistics are collected at various points throughout TM processing. A separate thread processes these statistics on a configurable interval and writes them to syslog. A monitor interface will be built here to send this data across a *JBoss* interface as well to be processed in a Libertp monitor .

# Installation:

TBD. It is assumed this will be accomplished via RPM and a spec file. A script should be developed to process user options and generate the configuration file to be read by libertm startup.

# Startup:

Operational parameters must be supplied at startup either via command line arguments or via a configuration file pointed to by a command line argument or both, *in which case putting the ‘-f’ option (see below) first will enable the command line arguments to override any configuration file settings, allowing for ad-hoc changes.*

Example : [singha] (drusak) ~> */data/ltpz/latest/bin/libertm -o 3307 -j 141.112.13.98:15080 –s 200 –d*

*These settings set the TM terminal port to ‘3377’ (by default), JBOSS host/port to 141.112.13.98:15080, the TPR return port to 3307, populate the session interface pool with 200 threads, set a debug option to capture extra diagnostic information, use the uppercase first character of the TM host as the terminal ID prefix (by default), the monitor interval to 5 (by default), authentication is off (by default) and logs to stdout (by default) .*

Command Line Parameters:

 -h Display this help and exit

 -v Display this version and exit

-d Debug mode .. log detailed JBOSS and other data

 -a Authenticate y/n .. default is n .. this informs TM whether to invoke PAM authentication on the User/Password entries on the login screen. See ‘Authentication’ below.

 -L LogType specify logging type .. LogType = 0 (default) => stdout, LogType > 0 => logfiles ‘libertmlog.0’ and ‘libertmlog.1’ . Note .. the file ‘log4crc’ must be in the execution directory to invoke file logging OR set the ‘*LOG4C\_RCPATH*’variable to point to it.

 -t TermID Prefix specify the TermID Prefix .. default is the uppercase first character of the TM host name. The rest of the termid is generated internally in TM to steer the terminal transactions to the TPR processing and back to the terminal.

-p TMPort specify the TM Port e.g. ‘-p 5555’ .. Default is 3377 .. this is the TM server port that 3270 terminals should connect to.

-o PortTpr specify the Tpr to TM Port. e.g. ‘-o 3307’ Default is 3278 .. this is the port that TPRs will communicate over back to the TM.

-j LTPHost:Port specify Jboss host/port. e.g. ‘-j 141.112.13.98:15080’ default singha.usrnd:9080 .. this is the host/port that TM executes its JBOSS calls against.

-b TM instance specify this TM’s id number .. e.g. ‘–b 2’ .. default ‘1’ ..

-k MonitorHost:Port specify the host/port to send monitor data to, e.g. ‘-j 141.112.13.98:15080/hoox/ltpz’ .. default default singha.usrnd:15080/hoox/ltpz’ ..

-s Number of sessions to preload .. Default 200 .. this will preconfigure this number of potential concurrent sessions to communicate with JBOSS over. Preconfiguring prevents the need for TM to dynamically allocate them if the need arises under a high terminal load.

-m Monitor interval default 5 seconds .. TM gathers statistical data (see Statistical Data below) and writes to syslog under ‘/var/log/messages’. This should be processed by a monitor mechanism for ease of us. (See ‘Monitor’ below).

Instead of providing command line arguments the above can be stored in a configuration file.

-f configuration file key/value format .. e.g. '-f config.ini'

 example config file

 tpr\_tm\_port=3077

 tm\_port=5555

 jboss\_host= 141.112.13.98:15080

 tm\_port=3377

 authenticate=y

# Terminal Manager Statistics:

TM maintains the following counts directly:

*int terminalQdConnect=0;// number of terminals queued for connect*

*int sessionCurCount=0;// number of current sessions , active and available*

*int sessionExCount=0; //number of exhausted sessions states found during connect process*

*long sessionJbossConnects=0;// total number of connect requests to jboss*

*long sessionJbossInvokes=0;//transaction invocations to JBOSS*

*long sessionJbossDisconnects=0;// terminal disconnects issued to JBOSS*

*long sessionCompletedRequests=0;//total responses from JBOSS*

*long sessionTDeltaSeconds=0;//seconds the last JBOSS session took to respond*

*long sessiontotalTDeltaSeconds=0;//seconds ALL JBOSS sessions took to respond*

*long sessionTDeltaNSeconds=0;//last response time from jboss in nanoseconds*

*long sessionTDeltaNAverage=0;//average response time from jboss in nanoseconds*

*long hashStrNSeconds=0;//last string (termid) hash response in nanoseconds*

*long hashIntNSeconds=0;//last fd ( terminal socket number) hash response in nanoseconds*

*long tprCompletedRequests=0;//count of completed tprs.*

*long tprDeltaNSeconds=0;// last response time from TPR process*

*long tprTotalDeltaNSeconds=0;//total of reponse times from TPR processes.*

*long tprDeltaNAverage=0;//average response time from TPR process*

*long termTOTprByteCount=0;// total bytes TMreceived from terminals*

*long tprTOTermByteCount=0;// tptmbc .. total bytes received from TPRs*

*long tprs=0;//current sockets supporting TPRs*

*int sessionThrdCount=0;//current count of session threads*

*int sessionRejects=0;// fsr .. count of failed session requests*

*int validConnects=0;//terminals connected without issues*

*int authSuccess=0;// number of successful authentications*

*int authRejects=0;//number of authentication rejects*

*long authTDeltaNSeconds=0;//total authentication time*

*long authTDeltaNAverage=0;// average response time to authenticate*

The above values are used to generate or display statistics.

In addition to those direct counters , some statistics are derived from other artifacts:

*Current terminals connected – derived from terminals hash count.*

*Current active TPRs – derived from TPRs hash count.*

*Current active sessions – derived from Sessions active queue*

These are generated to Syslog and can be derived via inspection of /var/log/messages on the TM host system , e.g. ‘tail –n 7 /var/log/messages’

#  Terminal Manager Monitoring:

Terminal Manager has a mechanism built in to send metrics via its exclusive libcurl interface over a session. **However the actual data interface needs to be defined prior to enabling this feature**. In the meantime the statistics are accessible via syslog .

# Authentication:

A PAM-based authentication mechanism has been implemented inside the TM utilizing the *pam-userpass* api. Refer to PAM documentation for setup and usage of PAM modules. If enabled via TM’ *‘-a y’* startup option the current implementation utilizes the standard linux ‘user/password’ PAM login driven through the PAM “service” *‘/etc/pam.d/libertm’*.The user logging in to TM inputs his linux user name and password to move to the CICS enquiry phase. The actual authentication mechanism is configurable via altering *‘/etc/pam.d/libertm’*. Some additional code and PAM services may need to be developed in the Authentication Manager to support various scenarios. The mechanism in use thus may need to become a configuration option for TM as well.

# Testing:

## TMDriver

To fulfill load-generation and demonstration requirements a driver ,TMDriver, has been developed.

It will start a configurable number of emulated 3270 terminals and proceed to send enquiries thru each terminal repeatedly using different customer identifiers. Each terminal will have a visual representation on the user screen and display a color appropriate to its current state, in-startup-handshake, enquiry-sent, waiting-response, error.

Startup parameters:

-host xxx.xxx.xx.x. – ip of TM host .. e.g. –host 141.112.17.7

-port xxxx- port on the TM host to connect to. This was the port configured in the TM by its ‘-p xxxx’ startup parameter.

-ssl false/true – set whether sockets are secure.

-terminals xx – number of terminals to start and drive.

 -mode demo/live – Use demo mode to start a large number of terminals (typically more than 10). Use live mode to start 10 or less and have full terminals cascade on the screen. This is to effectively accommodate the user screen’s real estate.

- loglevel trace/debug/info/warn/error/fatal – control what level of logging is required during the test run.

Example :

java -jar /data/ltpz/latest/TMDriver /tmdriver.jar -host singha.usrnd.lan -port 3377 -ssl false -terminals 25 -mode demo -loglevel error &



## X3270

X3270 is a 3270 terminal emulator useful to test the system on a manual interactive basis. In lieu of the driver it can be used to step through single interactions while inspecting traces and logs. It also has tracing and scripting capabilities.



The **x3270** scripting facilities allow the interactive 3270 emulators **x3270** and **c3270** to be operated under the control of another program, and forms the basis for the script-only emulator **s3270**.

### There are four basic scripting methods. The first is the peer script facility, invoked by the x3270 -script switch, and the default mode for s3270. This runs the emulator as a child of another process. Typically this would be a script using *expect*(1), *perl*(1), or the co-process facility of the Korn Shell *ksh*(1). In this mode, the emulator process looks for commands on its standard input, and places the responses on standard output

### Running a test peer script:

These are instructions to get familiar with the x3270 scripting system. This should be useful in developing automated test and verification.

On Sinha run :

*‘/data/ltpz/latest/bin/libertm -o 3307 -j singha.usrnd.lan:15080 -d -m 60’* to start an instance of TM.

Run

*‘/home/drusak/bull1/mur1/murach/scripts/runterm.sh’*

This will interact with the TM instance to generate an enquiry against customer number ‘123456’ over two separate “terminals”.

Scripting environment:

The above bash script calls the python script runterm.py, which in turn calls other python scripts that interact with the x3270 scripting facility to talk to TM. Note that runterm.sh sets up some environment variables so the subsequent commands can find these required components

The runtime stack can generally be represented by

Runterm.sh⬄runterm.py⬄Util3270⬄\_\_init\_\_.py⬄s3270.exe⬄TM⬄JBoss⬄Tprs

 Ltpz3270.py

Inspecting the output of the traces (look under ‘/tmp/xtrc..’) from these runs and the TM log should be helpful in developing more sophisticated automated tests.

## GLINK

And there is always glink. Someone apparently has an operable 3270 configuration.

Perhaps it also has a scripting facility that can be utilized for acceptance test in lieu of x3270??

