Test Work Bench Toolset Reference Version 1

CML00032-01

Code Magus Limited (England reg. no. 4024745)
Number 6, 69 Woodstock Road
Oxford, OX2 6EY, United Kingdom
www.codemagus.com

December 15, 2020

Contents

1 Introduction **2**

2 Log file structure **4**

3 Plan files **8**
3.1 Plan file format 9

4 Programs **16**
4.1 `twbarep`—Report on activities in a log file 16
4.2 `twbbview`—Viewer a session being proxied 18
4.3 `twbcloses`—Insert missing closes into a log file 18
4.4 `twbcounts`—Count log file session activity 18
4.5 `twbdrive`—Replay log files 20
4.6 `twbgrep`—Search a log file 37
4.7 `twblogon`—Locate log file logon activity 37
4.8 `twbplan`—Build a log file from a plan 37
4.9 `twbprint`—Print a log file 38
4.10 `twbproxy`—Proxy and log circuits 38
4.11 `twbprt3270`—Print a log file of 3270 data streams 45
4.12 `twbrrep`—Create replay report from log files 45
4.13 `twbsplit`—Split a log file on a pivot record 45
4.14 `twbstrip`—Strip sessions out of a log file 46
4.15 `twbvi3270`—View log file containing 3270 data streams 46
4.16 `twbrep3270`—Report on 3270 activities in a session 46
4.17 `twblnav`—Navigate log files 46

5 Examples **56**

6 Extensions **56**

List of Figures

1 Network configuration without a proxy 3
2 Network configuration with proxy in place 3
3 Network configuration with replay driver and proxy in place 4

1 Introduction

The `twb` toolset comprises a collection of program. The main program, `twbproxy` is a proxy program which operates at the TCP circuit level and behaves much like any proxy program. The purpose of the this program is not so much to behave as a proxy, but to produce a log file of all circuit traffic. The purpose of recording this traffic is primarily for testing and diagnostics. This testing or diagnosis could either be of the protocol and data being traced for debugging purposes, or it could be for purposes of replaying the data back as part of a test system driving some system under test. This redriving of the data is achieved by the program `twbdrive`.

A single instance of the proxy program `twbproxy` can, for all practical purposes proxy an arbitrary number of *sessions*. The originators of these sessions can be from anywhere on the network, and the destination of these sessions can be directed to any machine. Since a single instance of the proxy creates a single sequential log file, the proxy serialises all the sessions being proxied. Thus, for purposes of replay, it is possible to redrive session messages in the same order as they were observed by the proxy. This is also useful for diagnostics.

The replay program `twbdrive` mimics the sessions seen by the proxy program. This mimicry can be perturbed by changing the pace at which messages are replayed. This allows the toolset to perform stress testing of the system under test. Stress testing is possible either by ensuring the same serialisation of the circuits and their messages (by running a single copy of the `twbdrive` program) or by driving each session and its messages independently (by running a separate copy of the `twbdrive` program for each session in the log file).

When sessions are replayed, it is usual for the replay to drive the sessions back through the proxy program. This has the advantage of building a log file of the sessions as they being replayed. This file is useful in comparing the captured and playback sessions.

The most important program in the toolset is the proxy program `twbproxy`. This program creates original log files captured from TCP sessions. It is debatable whether `twbdrive` is the second most important program in the suite, but this is certainly the case if the tools are being used for replay style testing. The rest of the programs in the toolset are programs that are used to inspect and manipulate log files.

Figure 1 shows represents a network in which the three *clients* (white filled circles)

require services from the two *application servers* (lack filled circles). Originating from each of the clients are two circuits.

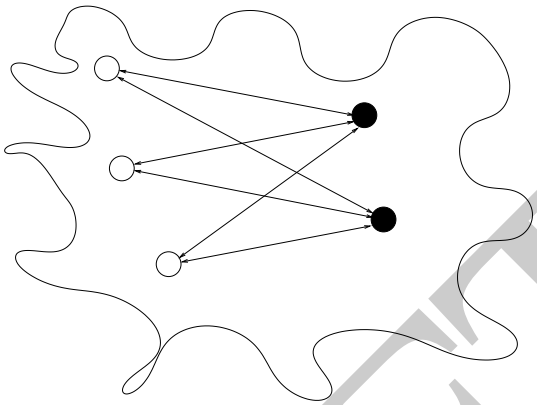


Figure 1: Network configuration without a proxy

In order to place a proxy between the clients and their application servers (for purposes of recording), the clients have to be reconfigured so that the IP addresses they have for the application servers are replaced by the IP address of the proxy server (the proxy server is the host which runs `twbproxy` and may be one of the existing clients or servers). This new arrangement is shown in Figure 2. The proxy in turn is configured to recognise connections on the ports the clients use and to forward these to connections to the application servers (the ports on the proxy server and the application servers need not be the same). It is the responsibility of the proxy to match the resultant circuits and to forward data between the clients and the application servers. As part of this process, the proxy writes all data to a log file if one is open (if a log file is not open, the proxy still forwards data).

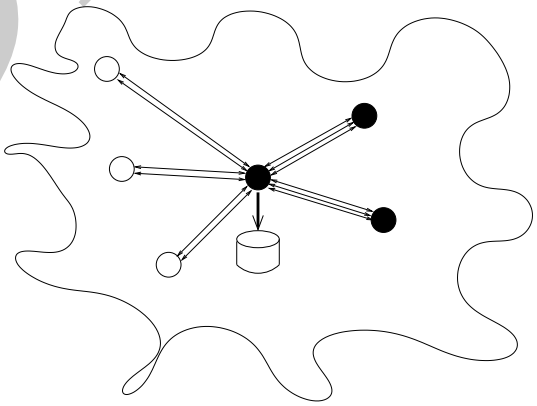


Figure 2: Network configuration with proxy in place

For purposes of performing a replay, a driver server is introduced. This is shown in Figure 3. The driver program `twbdrive` which runs on the driver server, takes over

responsibility of the clients and recreates the circuits as indicated in a log file previously recorded by the proxy (or recorded by the proxy and edited into a replay log file). Data messages in the log file that originated at the clients are replayed on the circuits to the application servers. In the figure, this is done via another instance of the proxy so that a new log file can be recorded. Both the replayed messages and the responses from the clients are passed to a state machine. The particular state machine is nominated for that class of circuit by the configuration of the driver program `twbdrive`.

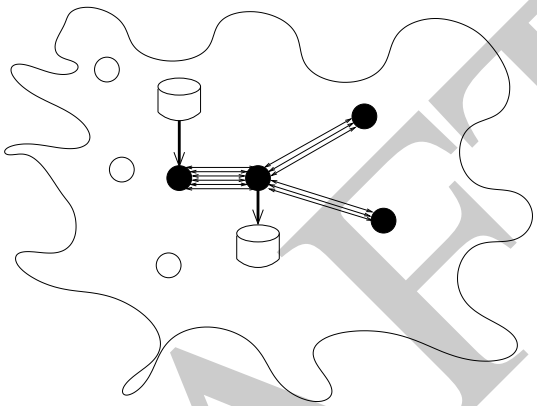


Figure 3: Network configuration with replay driver and proxy in place

The distinction between clients and servers in the above example, has no bearing on the toolset. What is important (for configuration purposes), is the originator of the circuits. For example, the a server (in the figure) may connect to the client and it is possible to replay such circuits as well. A warning though, the type of open from the 'client' (passive or active) dictates which directions will be labelled `inbound` and `outbound`. This means that when one views the log file in which a server initiated an active open, data messages going *to* the server will be labelled `outbound data`.

2 Log file structure

The toolset can be seen as a collection of programs that create and manipulate log files. The description of a session is derived from the *proxy* class defined to the `twbproxy` program instance that created the session. The session can be viewed as an object instance of the sort described by the proxy class or element. Because the `twbproxy` program offers a command shell which can be used to change the attributes of a proxy class a any time during the execution of the proxy program, it is feasible that some changes can result in an inconsistent state of the sessions. The attributes that can result in an inconsistent state are hence copied over to the session when the session is created. An example of such an attribute is the method of identifying the session's record boundaries.

Apart of the obvious attributes such as record length, type and data direction, a number of the session attributes are also copied to a log record header and are repeated for every record of the session in the log file.

A number of programs create log files. Programs that 'modify' log files do so by creating a *new* log file derived from the original log file. Examples of such programs are `twbsplit` which 'splits' a given log file using a record sequence number as a pivot record, and `twbstrip` which creates a log file from a session or sessions stripped from an input log file. It is not the preserve of programs in the toolset to manipulate log files and there is a header file `twblog.h` which contains prototype for functions in the file `twblog.o` which can be used to read and write log files. These functions are used by the tools themselves and are designed so that the toolset cannot overwrite an existing log file.

The structure `logdata_t` describes the log record header and has the following format:

```
typedef struct loghdr loghdr_t;   /* log file record header structure *
struct loghdr
   {
   int logflen;                   /* length of following logged data */
   int logplen;                   /* length of previous logged data */
   int logseq;                    /* sequence number */
   time_t logtime;                /* time record was logged */
   struct timeval logtod;         /* time of day record was logged */
   int logsid;                    /* session id */
   int logmode;                   /* session mode */
   int logtype;                   /* record type */
   int logdir;                    /* data direction */
   struct sockaddr_in logsrc;     /* source address */
   struct sockaddr_in logdst;     /* destination address */
   int logrecf;                   /* record format */
   int loglfmt;                   /* record length format */
   int logloff;                   /* record length offset */
   };
```

The header file `twblog.h` also contains prototypes for functions defined in `printbuf.o` which can be used to format log file headers and data in a standard format:

```
Seq=6, Mon Jul 24 10:02:14 2000: Session=0, inbound data, length=18
        src=10.62.129.60:1474, dst=192.168.207.1:23, tod=1004305335.761
        00..__..__05..__..__10..__..__15..__..__20..__..__25..__..__30.
  0000: FFFA180049424D2D333237382D322D45FFF0
  0000: ........I.B.M(-.3.2.7.8.-.2.-.E....0
```

Not the entire log record header is formatted, but all important fields for diagnostics are printed. The `Seq` item is the log records position in the *log* file (not within the

session) and starts at zero. The date and time formatted is the date and time that the record was written to the log file. The `Session` number identifies the TCP/IP circuit that the data belongs to. `Session` numbers are assigned serially from zero as sessions are created. The indication of whether the data on the circuit was `inbound data` (i.e. from the host that created the circuit using an active open—`connect`–to the host that made the passive open—`listen`) or `outbound data` (i.e. from the passive open host to the active open host). The destination and source IP addresses and port numbers, `src=10.62.129.60:1474`, `dst=192.168.207.1:23`, indicate the originating host and port number for this message and destination host and port number for this message. Messages for `outbound data` have these IP address and port number pairs exchanged. The `length` field indicates the number of bytes in the payload.

The data is printed in a dump format with each line containing up to thirty two bytes. Each byte is printed as two hexadecimal digits. On the line below this hex dump, each byte that is represented as a ASCII character is printed below the first hex digit and each byte that is represented by an EBCDIC character is printed below the second hex digit. If the byte is not represented by an ASCII or EBCDIC character then a period is printed in the corresponding position.

See the header file `twblog.h` for details of these functions. The program `twbprint` reproduced here, is an example of a small program which demonstrates the use of some of these functions.

```
#include "copynote.h"

  /*
   * Program twbprint.c formats log file formatted accroding to the
   * formats and APIs defined in twblog.h.
   *
   * Author: Stephen Donaldson.
   */

  /*
   * $Author: stephen $
   * $Date: 2010/02/16 15:38:17 $
   * $Id: twbprint.c,v 1.6 2010/02/16 15:38:17 stephen Exp $
   * $Name:  $
   * $Revision: 1.6 $
   * $State: Exp $
   *
   * $Log: twbprint.c,v $
   * Revision 1.6  2010/02/16 15:38:17  stephen
   * Change printing of large numerics to unsigned
   *
   * Revision 1.5  2007/12/28 11:39:36  stephen
   * Add support for active open on original circuit
   *
   * Revision 1.4  2000/09/07 13:40:47  cvs
   * Docs plus SA chnages july/aug
   *
```

```
   * Revision 1.3  2000/06/23 14:34:51  cvs
   * changes and additions for screen comp
   *
   * Revision 1.2  2000/06/19 20:47:41  cvs
   * Activity report, env var and log usage
   *
   */

static char *cvs =
   "$Id: twbprint.c,v 1.6 2010/02/16 15:38:17 stephen Exp $";

  /*
   * Large file support required for various environments:
   */

#define _LARGEFILE_SOURCE
#define _LARGEFILE64_SOURCE

#define _LARGE_FILES
#define _FILE_OFFSET_BITS 64

#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

  /*
   * Constants and options:
   */

#define MAXSESSIONS    200000       /* max sessions can select on */

#include "twblog.h"                 /* log file formats */
#include "translate.h"              /* ASCII/EBCDIC translation tables */
#define SET_MAX_ORD MAXSESSIONS
#include "setops.h"                 /* set operations */


int main(int argc, char *argv[])
   {
   LOG logfile;                 /* file to be formatted */
   loghdr_t loghdr;             /* log file record header */
   unsigned char buffer[BUFSZ]; /* log record data */
   set_t sessions;              /* sessions selected */
   int i, j;

   COPYNOTE;

   if (argc < 2)
      {
      printf("Usage: %s <log file name> [<session> ... ]\n",argv[0]);
      exit(1);
```

```
      }

   if (argc == 2)
      set_universal(sessions);
   else
      {
      set_empty(sessions);
      for (j = 2; j < argc; j++)
         if (sscanf(argv[j],"%d",&i) != 1 || i < 0 || i >= MAXSESSIONS)
            fprintf(stderr,"Error %s should be integer in [0..%d]\n",
               argv[j],MAXSESSIONS-1);
         else
            set_add(sessions,i);
      }

   if (log_open(&logfile,argv[1],LOG_INPUT) < 0)
      {
      fprintf(stderr,"%s\n",log_error(&logfile));
      exit(1);
      }

   while(log_read(&logfile,&loghdr,buffer,0) >= 0)
      {
      if (loghdr.logsid >= MAXSESSIONS || loghdr.logsid < 0
         || set_ismember(sessions,loghdr.logsid))
            printbuf(stdout,loghdr,buffer);

      } /* while */

   fprintf(stderr,"%s\n",log_error(&logfile));
   log_close(&logfile);
   exit(0);
}
```

3 Plan files

Plan files describe how a log file should be put together from the contents of other log files. Plan files are ASCII text and contain statements according to a specific syntax. These files can be hand-coded, but are also generated from some of the tools themselves.

The advantage of the plan files is that they can be edited without copying the files. For example, a plan file proposed for a replay can be created using the `twbarep` program. Before the replay, it may be necessary to omit certain activities or sessions from the replay log. Instead of providing a tool for navigating and editing the log file, a simple text editor can be used to edit the plan file. This means that the plan can be prepared on a system different from which the software is installed or available. Plan files are also a lot more portable than the log files they describe and can easily be ftp'ed or e-mailed.

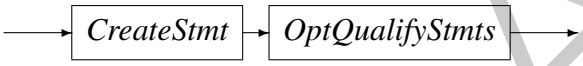
The program `twbplan` processes the plan file and creates a new log file from a set of input log files. The names of these files are contained in the plan file.

3.1 Plan file format

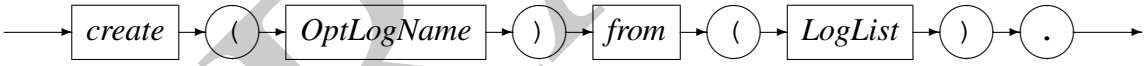
Plan files are ASCII files and describe the required log file in a free-format syntax. White space and newline characters are ignored except in comments. The comments in plan files are indicated using a leading hash character (`#`) and continue up to the end of the line containing the hash. Comments may be interspersed among the statements describing the log file plan. Programs such as `twbarep` generate comments in the plan file describing the activities associated with the included log file records.

The significant portion of the plan file is the free format statements that describe a log file to be created from a set of input log files and the records from these log files which should be included or excluded from the resultant file. Each of these statements is terminated with a period.

Plan

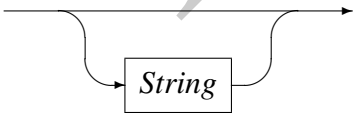


CreateStmt

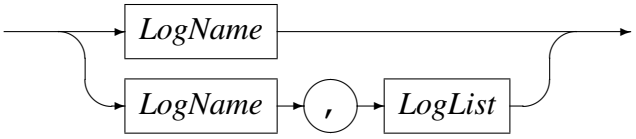


The first statement in the plan file must be the `create` statement. The *OptLogName* in parenthesis following the `create` keyword is the name of the log file that `twbplan` will create when the plan file is processed. This log file name is optional in the plan file, but is mandatory when the plan file is processed `twbplan`. The log files named in the `from` clause will be processed in the given order by the `twbplan` program to create the new log file.

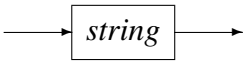
OptLogName



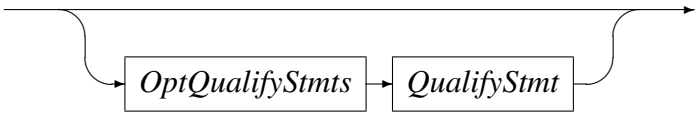
LogList



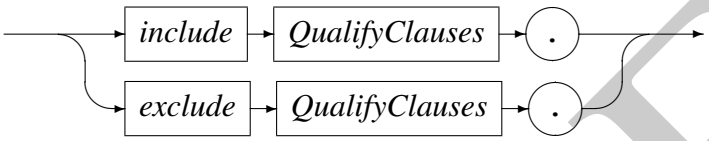
LogName



OptQualifyStmts



QualifyStmt



The `create` statement is optionally followed by a number of qualifying statements that modify the selection of records from the input log files to be included in the output log file. By default, that is without any qualifying statements, all the records of each of the input log files are copied to the output log file. Any number of qualification statements of any type can be specified.

The `include` statement is a means of overriding the default of including all log records in the output file and restricts the selection to those records that satisfy the statement.

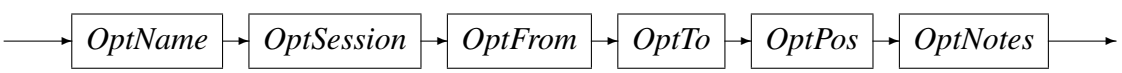
The `exclude` statements can be used to prevent certain records from being included in the output log file.

All the clauses on the `include` and `exclude` statements are optional and for the clauses whose values are used to select records to be included or excluded from the log file being created, default values are supplied.

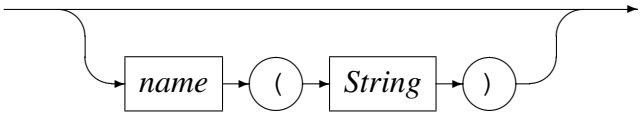
The clauses `session`, `from` and `to` specify record inclusion or exclusion. The default values for these clauses is to include all records when used in the `include` statement or to exclude all records when used in the `exclude` statement.

The optional clauses `name`, `pos` and `notes` are not used by the program `twbplan` to create the requested log file. The `name` clause simply provides a name to which the record range can be associated. The `notes` clause is used to provide documentation suitable for identifying the the records when the plan file is edited. The `pos` clause is used for positioning the log file at the first record in the range to be included or excluded. This field is used by programs which might want to navigate the log file (examples include `twbrep3270` and `twblnav`).

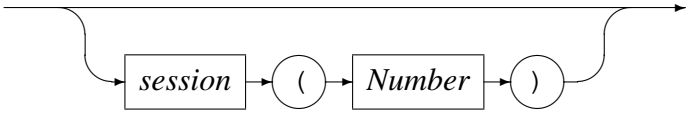
QualifyClauses



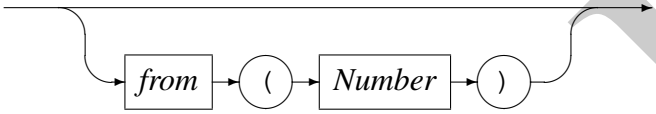
OptName



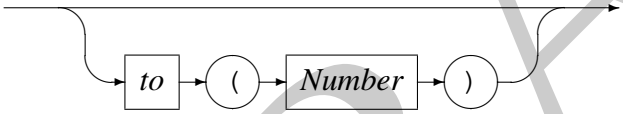
OptSession



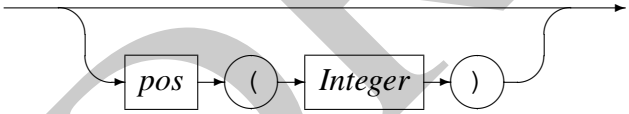
OptFrom



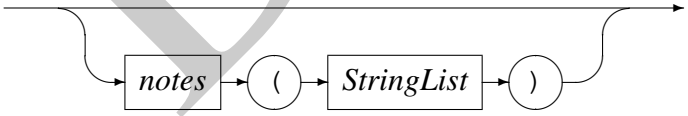
OptTo



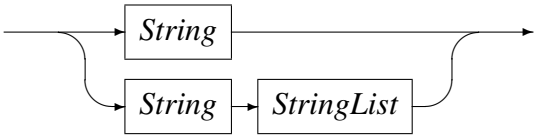
OptPos



OptNotes



StringList



Both `include` and `exclude` statements take optional clauses which indicate the range of log records over which the qualifiers operate. By default, without specifying any qualifying clauses, the `exclude` and `include` statements operate over all sessions and all records in the log file. Thus, for example, a single `exclude` state-

ment without any qualifying clauses produces an empty log file. On the other hand a single `include` statement without any qualifying clauses includes all log file records in the input log files in the new log file created. This latter case behaviour is the same behaviour achieved when no `include` or `exclude` statements are coded.

The `session` clause restricts the scope of the `include` or `exclude` statement to a particular session. Without any additional clauses the scope of the `include` or `exclude` statement is restricted to the entire indicated session. Independent of any qualification using the `session` clause, the first qualifying log record can be changed using the `from` clause. Similarly, the last qualifying log record can be changed using the `to` clause. By default the first log record encountered is the first qualifying record and the last log record encountered is the last log record encountered.

Environment variables can be substituted for each of the terminals `number` or `string`.

The following example shows the plan file `/tmp/sample.plan` created by the program `twbarep` using the command:

```
$ twbarep --plan Logs/monday_2_capture_testers.00000000
```

```
#
# Log name(s):
#    0. Logs/monday_2_capture_testers.00000000
#
create ( $CREATE_LOG ) from( Logs/monday_2_capture_testers.00000000 ) .
#
include name("LOGON") session(0) from(0) to(25) pos(0)
      notes("LOGON:\n"
            "Start=0, end=25, end by=ACTIVITY ENDED\n"
            "Sess=0, src=10.20.1.16:2648, dst=192.168.207.1:23\n"
            "Start time=Tue Apr 17 07:31:59 2001") .
#
include name("0173A") session(0) from(26) to(45) pos(3265)
      notes("0173A:C0TST01 :Correct a rejected merchant deposit.\n"
            "Start=26, end=45, end by=ACTIVITY ENDED\n"
            "Sess=0, src=10.20.1.16:2648, dst=192.168.207.1:23\n"
            "Start time=Tue Apr 17 07:41:14 2001") .
#
include name("0272A") session(0) from(46) to(75) pos(10718)
      notes("0272A:C0TST01 :Set up a new account within a new hierarchical structure.\n"
            "Start=46, end=75, end by=ACTIVITY ENDED\n"
            "Sess=0, src=10.20.1.16:2648, dst=192.168.207.1:23\n"
            "Start time=Tue Apr 17 07:42:27 2001") .
#
include name("0173A") session(0) from(76) to(91) pos(33889)
      notes("0173A:C0TST01 :Correct a rejected merchant deposit.\n"
            "Start=76, end=91, end by=ACTIVITY ENDED\n"
            "Sess=0, src=10.20.1.16:2648, dst=192.168.207.1:23\n"
            "Start time=Tue Apr 17 07:51:47 2001") .
```

```
#
include name("0272A") session(0) from(92) to(129) pos(39233)
      notes("0272A:C0TST01 :Set up a new account within a new hierarchical structure.\n"
            "Start=92, end=129, end by=ZZEND\n"
            "Sess=0, src=10.20.1.16:2648, dst=192.168.207.1:23\n"
            "Start time=Tue Apr 17 07:52:32 2001") .
#
include name("LOGON") session(1) from(0) to(162) pos(0)
      notes("LOGON:\n"
            "Start=0, end=162, end by=ACTIVITY ENDED\n"
            "Sess=1, src=10.20.1.16:2661, dst=192.168.207.1:23\n"
            "Start time=Tue Apr 17 08:26:06 2001") .
#
include name("0173A") session(1) from(163) to(178) pos(70665)
      notes("0173A:N112074 :Correct a rejected merchant deposit.\n"
            "Start=163, end=178, end by=ACTIVITY ENDED\n"
            "Sess=1, src=10.20.1.16:2661, dst=192.168.207.1:23\n"
            "Start time=Tue Apr 17 08:26:58 2001") .
#
include name("0272A") session(1) from(179) to(220) pos(76013)
      notes("0272A:N112074 :Set up a new account within a new hierarchical structure.\n"
            "Start=179, end=220, end by=ACTIVITY ENDED\n"
            "Sess=1, src=10.20.1.16:2661, dst=192.168.207.1:23\n"
            "Start time=Tue Apr 17 08:28:46 2001") .
#
include name("0272A") session(1) from(221) to(304) pos(109218)
      notes("0272A:N112074 :Set up a new account within a new hierarchical structure.\n"
            "Start=221, end=304, end by=ACTIVITY ENDED\n"
            "Sess=1, src=10.20.1.16:2661, dst=192.168.207.1:23\n"
```

```
            "Start time=Tue Apr 17 08:45:58 2001") .
#
include name("0174A") session(1) from(310) to(321) pos(176544)
      notes("0174A:N112074 :Delete a rejected merchant deposit.\n"
            "Start=310, end=321, end by=ACTIVITY ENDED\n"
            "Sess=1, src=10.20.1.16:2661, dst=192.168.207.1:23\n"
            "Start time=Tue Apr 17 09:05:48 2001") .
#
#
```

4 Programs

Each of the programs attempts to give assistance of its usage. For example, those that support a command shell implement a help command, and those programs that are driven via command line arguments include a usage option (which is also invoked if a command line error is detected) and those programs developed more recently include a command line help option which gives an expanded description of the command line arguments. In time, the earlier programs will be changed to include the same help options. The `popt` package (`ftp://ftp.redhat.com/pub/redhat/code/popt`) is used for this command line processing.

4.1 `twbarep`—Report on activities in a log file

Program `twbarep` reports on all activities found in a log file. The reported list of activities can optionally be placed in a file in a format suitable processing by `twbplan` to create a log file of the selected activities. The names of the last activities found on each session can also be placed in such a file.

Usage:

```
$ twbarep [-p?] [-f {stdout|<file>}] [-t {0|<count>}] [--usage] <log fi
```

Long forms of the options are supported and the program offers the following brief help if the options `--help` or `-?` is used:

```
Usage: twbarep <log file> ...
  -p, --plan                    Produce a plan suitable for twbaplan
  -f, --file={stdout|<file>}    File name for report or plan
  -t, --tail={0|<count>}        Restrict report and plan to last entries
                                sessions

Help options
  -?, --help                    Show this help message
  --usage                       Display brief usage message
```

Following is an example of an activity report for a log file created from the plan file in shown in Section 3.1:

```
$ CREATE_LOG=/tmp/sample_log.00000000 twbplan /tmp/day2c_capture_replay
$ twbarep /tmp/sample_log.00000000
```

```
-------------------------------------------------------------------------------------------------------------
Log name(s): /tmp/sample_log.00000000
   sid    source IP and port          dest ip and port              max seq   bytes in  bytes out    msgs in
-------------------------------------------------------------------------------------------------------------
    15    10.59.31.80:1617            192.168.207.1:23                 2890       3820      70656         76
            0       521    initial sequence                                                      ACTIVITY ENDE
          522       527    C0TST13       START                                                   ACTIVITY ENDE
          528       753    C0TST13 3041V Set up an application for a single cardholder.          ACTIVITY ENDE
         1646      1833    C0TST13 2838V Set up a classic business detail account.               ACTIVITY ENDE
         2228      2255    C0TST13 1099V Inquire on a purchase transaction.                      ACTIVITY ENDE
         2720      2774    C0TST13 1074V Change the automatic payment option on an account.      ACTIVITY ENDE
         2776      2779    C0TST13 1924  ** NO ACTIVITY DESC **                                  ACTIVITY ENDE
         2853      2886    C0TST13 1640V Close a plastic on an account                           ACTIVITY ENDE
         2887      2890    C0TST13                                                               Auto-end
    16    10.59.31.80:1619            192.168.207.1:23                 3341       8445     146297        125
            0       567    initial sequence                                                      ACTIVITY ENDE
          568       625    C0TST08       START                                                   ACTIVITY ENDE
          626       887    C0TST08 5825M Set up a business control debit card account.           ACTIVITY ENDE
          889      1045    C0TST08 5826M Set up a business detail debit card account.            ACTIVITY ENDE
         1046      1327    C0TST08 5829M Set up a classic business control account.              ACTIVITY ENDE
         1330      1523    C0TST08 5830M Set up a classic business detail account.               ACTIVITY ENDE
         3071      3226    C0TST08 5837M Set up a classic business detail account.               ACTIVITY ENDE
         3336      3341    C0TST08       START                                                   Auto-end
-------------------------------------------------------------------------------------------------------------
                                                                      12265     216953         201
-------------------------------------------------------------------------------------------------------------
```

4.2 twbbview—Viewer a session being proxied

Program twbbview formats a log file record fed from a port. An example of such a feeding program is the proxy program through the viewer port.

4.3 twbcloses—Insert missing closes into a log file

Program twbcloses reads a log file and writes a new log file out. For sessions missing a close record, one is generated after the last record for the session so that if the log file is used for a redrive, then resources can be reused once sessions become inactive.

In order to insert these closes at the correct places, the entire log file sets must be processed in two passes. The first pass notes the last activity for each session and whether a close record is missing from the log and the second pass writes a new log file set inserting the close records at appropriate points.

4.4 twbcounts—Count log file session activity

Program twbcounts counts analyses message traffic in a log file by session.

Usage:

```
$ twbcounts <log file name>
```

The report below was created from the log file created using the example plan file from Section 4.1 using the command:

```
$ twbcounts /tmp/sample_log.00000000
```

```
-------------------------------------------------------------------------------------------------
Log name: /tmp/sample_log.00000000
   sid    source IP and port        dest ip and port            max seq   bytes in  bytes out    msgs in
-------------------------------------------------------------------------------------------------
    15    10.59.31.80:1617          192.168.207.1:23               2890       3820      70656         76
    16    10.59.31.80:1619          192.168.207.1:23               3341       8445     146297        125
-------------------------------------------------------------------------------------------------
                                                                  12265     216953                  201
-------------------------------------------------------------------------------------------------
```

4.5 twbdrive—Replay log files

This program processes files created by the twbproxy program. and redrives the network sessions according to user parameters indicated by commands.

Usage:

```
$ twbdrive [<command> ...]
```

Each of the arguments entered on the command line of program twbdrive is interpreted as a full command. If the commands contain spaces, then they should be quoted to prevent the shell from splitting the command across multiple arguments.

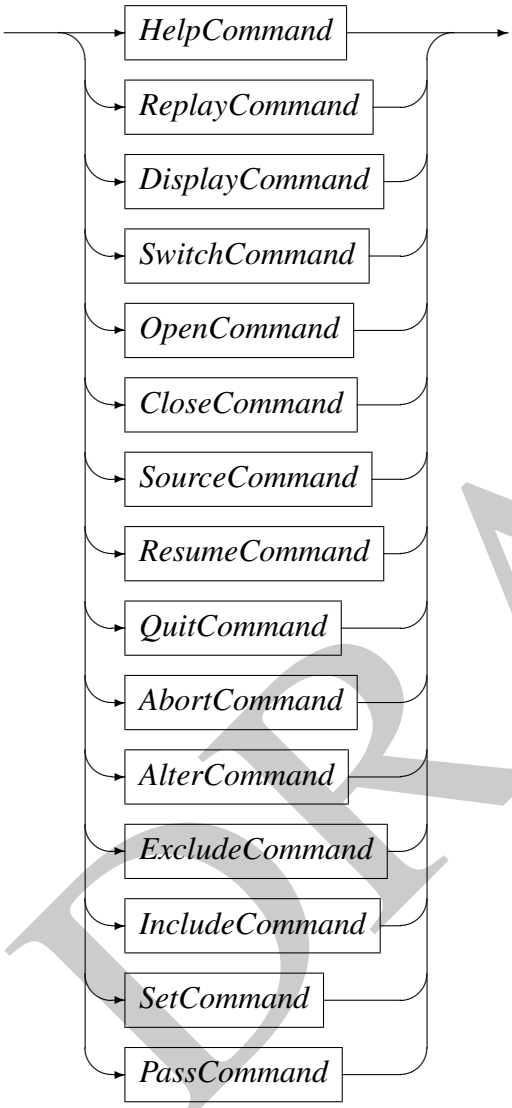
Once twbdrive has been started it provides a shell from which additional commands can be entered. The prompt for this shell is drive>. The same command syntax supported on the command line is supported by the shell. Furthermore, commands can also be sourced from files using the same syntax. This *scripting* can be very useful as configurations are often re-used.

There has to be a mechanism to re-establish the circuits for the sessions in the log file that are to participate in the replay. The creation of such sessions is triggered by the occurrence of connect records in the log file. Whether or not these circuits are re-enacted and how they are re-driven is determined by the configuration of twbdrive. Instead of insisting on an explicit configuration for each session found in the log file, only classes of configuration are necessary. This is facilitated by the TCP/IP protocol and the nature of most Internet applications. For example, all telnet sessions typically connect to the telnet server on port 23.

The commands provide the means for configuring twbdrive. There are commands for creating *replay* elements. Each replay element describes a class of *sessions*. When a replay element is successfully used to create a new circuit a new session element is created to describe that circuit. Using the commands, replay elements can be given certain attributes which describe how the circuit is established and how the state of the circuit is to be maintained. Probably the most important attribute assigned to the session is the state machine that controls the state of the session. Additional attributes might be used by this state machine in determining how to maintain additional state information. Another class of attributes is only used outside of the state machines by the main processing loop of twbdrive. Included in the first class of these attributes might be the *state machine* elected to control the session. The state machine, in turn, might use the record length field to allocated additional internal buffers. It is potentially catastrophic if such attributes would could be changed in an un-synchronised manner. Consequently, such attributes are copied from the replay element to the session element when the session is created. An example of an attribute used only by the main processing loop and not for maintaining any session state within any session elements are the think-time distribution and the think-time distribution parameters. These are quite safe to change at any time (modulo any application timing implications) and are not copied

to the session element. Instead, whenever such attributes are required by the session element, the current settings in the originating replay element is used.

Command

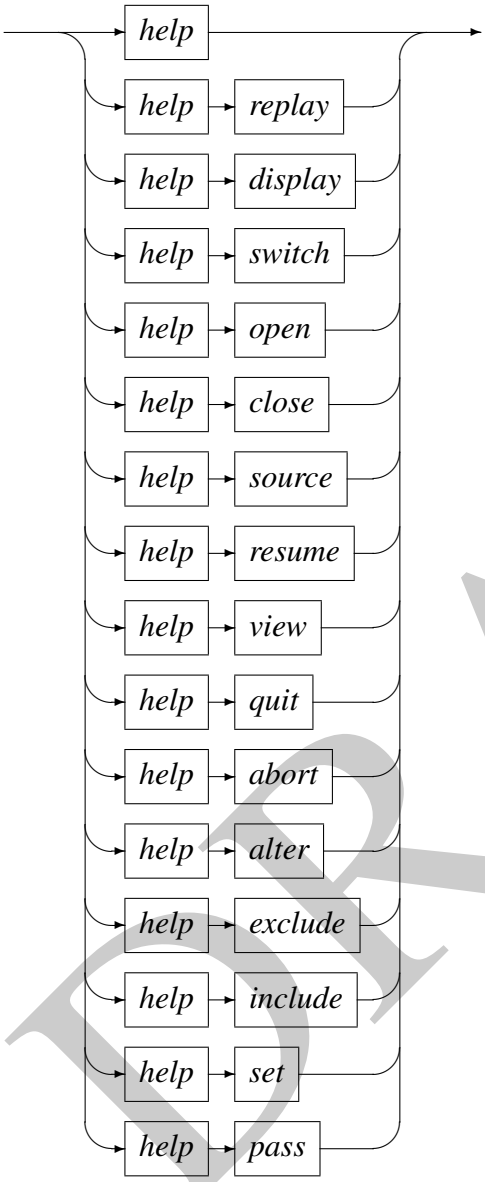


The above indicate the classes of command. Not that a blank line is a valid command (which does nothing). Comment lines are also allowed and are indicated by a hash (`#`) in the first position of the command line. Comments extend to the end of the line containing the hash character.

The shell can also be used to start `twbdrive` direct from a script using the usual `#!` signature:

```
#!/usr/local/bin/twbdrive
```

HelpCommand



The first class of commands is the help command. The help command gives a brief description of the syntax of the available commands. When used without a command name, `help` offers:

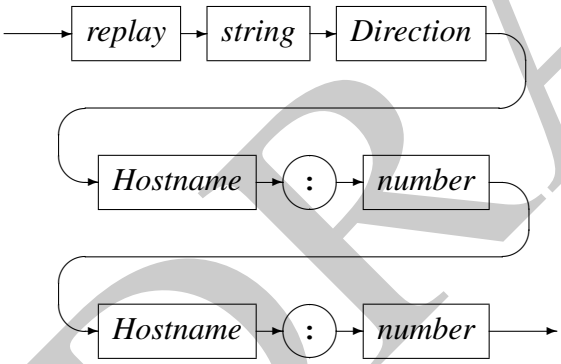
```
drive> help
   Try: help    {replay|display|switch|open|close
                |source|resume|view|quit|alter|exclude
                |include|set|pass}
   Try: - <navigation command>
+ACK
```

And when, for example, asking for help on the `replay` command, help offers:

```
drive> help replay
        Syntax: replay <name> {inbound|outbound} <mhost>:<mport> <rhost
        Where   <rhost> ::= {<ipaddress>|<hostname>}
                name    - name of the replay element
                mhost   - match IP address of connection
                mport   - match port of IP connect
                rhost   - replay IP address of connection
                rport   - replay port of IP connect
                inbound - the replay will be for inbound data
                outbound- the replay will be for outbound data
+ACK
```

The string `+ACK` indicates that the program `twbdrive` considered that the user's command was executed successfully. If any errors are detected whilst attempting to execute a command the final response (before the command prompt `drive>` appears) is `-ACK`. This feedback makes is possible for the driver program to be be driven by another program.

ReplayCommand



The `replay` command creates a replay element describing a class of sessions to be replayed. The default replay class is stateless. The replay class is given a name which can be used in further commands to change its state and attributes.

As replays are created they are also given a number. The replay element is known both by its name and its number. If the name given to the replay element is not unique, then a unique name is generated. In this case, any further commands referencing the replay element will have to use the number.

The *Direction* parameter indicates whether the `inbound` or `outbound` data for the associated sessions will be replayed on the circuit. If `inbound` data is to be replayed, then an active (`connect`) call is made when the log record is processed. When `outbound` data is to be replayed, `twbdrive` will accept incoming connections provided the listener is opened (see the `alter` command). If a listener has not be opened and an incoming connect has not been accepted by the time the connect log record for the session is processed, the replay suspends until a listener has been opened for the session

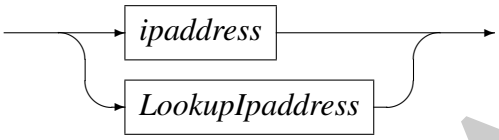
and an connection has been accepted for the connection.

The first *Hostname* and *number* pair are the IP address or DNS name and port number to match on `connect` records in the log file in order to create a session under this replay class. This address matches the destination IP and port number made in the original connection. This destination IP address and port number pair is `logdst` field found in the log record header.

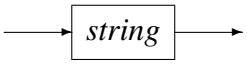
The second *Hostname* and *number* pair are the IP address or DNS name and port number to which the session must be replayed.

As sessions are created they are assigned a unique number. This number can be used in further commands to manipulate the session.

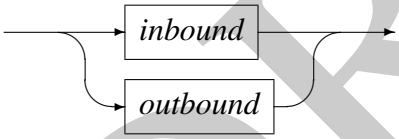
Hostname



LookupIpaddress

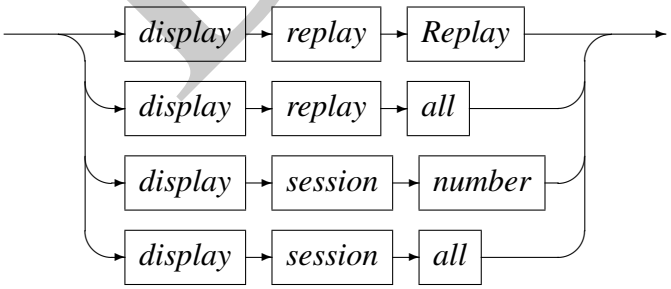


Direction



The `inbound` direction indicates traffic flow from the active open (`connect`) to the passive open (`listen`).

DisplayCommand

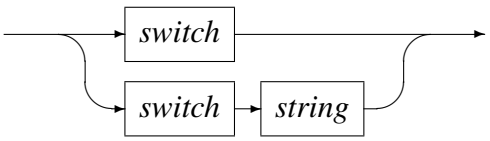


Display commands are provided for displaying the state of replay elements and sessions. Either all replay elements or sessions can be viewed using the `display` command:

```
drive> display replay tn3270
```

```
Replay: name = tn3270, number = 0, active = 1
Match     IP: 192.168.20.20. port: 23
Replay    IP: 10.59.31.80. port: 3023
State machine = 3270, reclen = 8000, recfm = iaceor
Thinktime distr = constant, distr parms = [0,0]
```

SwitchCommand

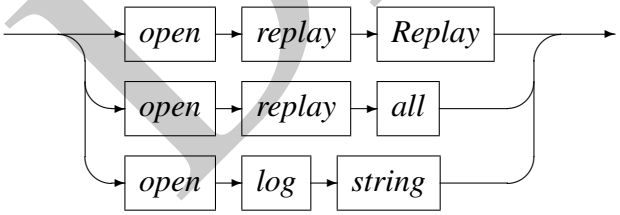


When specifying a log file from which to replay, only the base name is provided. Every time a new base name is introduced the name is completed by appending "`.00000000`" to the end of the name. The switch command without a new name string, switches the replay to the next numbered suffix. For example:

```
Now replaying from log /home/stephen/Logs/tuesday_test.00000000
Source command file ./stream-a-driver closed.
drive> switch
Now replaying from log /home/stephen/Logs/tuesday_test.00000001
drive>
```

This causes the current log file from reply is being performed to be closed (if it is not already closed and the next log file of the sequence to be opened for continuing the replay. The switch command supplying a new base name string, closes the current log file if one is open and opens the a new log file for replay using the given base name and a suffix of "`.00000000`".

OpenCommand



Inactive replays (i.e. those showing an indicator of `active = 0` when displayed) cannot create sessions and can be re-activated again using the `open` command:

```
Replay: name = tn3270, number = 0, active = 0
Match     IP: 127.0.0.1. port: 23
Replay    IP: 10.59.31.80. port: 3023
State machine = 3270, reclen = 8000, recfm = iaceor
Thinktime distr = constant, distr parms = [0,0]

drive> open replay tn3270
```

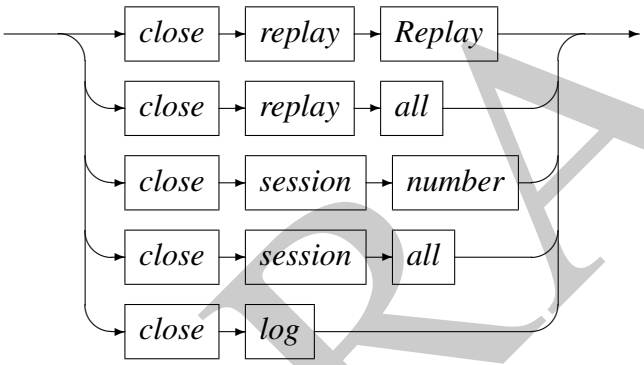
```
drive> display replay tn3270

        Replay: name = tn3270, number = 0, active = 1
        Match     IP: 127.0.0.1. port: 23
        Replay    IP: 10.59.31.80. port: 3023
        State machine = 3270, reclen = 8000, recfm = iaceor
        Thinktime distr = constant, distr parms = [0,0]
```

Replay classes can be inactivated using the `close` command. All inactive replays classes can be re-activated using `all` keyword:

```
drive> open replay all
```

A new log file base name can also be supplied using the open command. This command has the same meaning as the `switch` command when a new base name is provided.

CloseCommand



The `close` command can be used to set a particular replay class inactive by supplying the number of the replay class in the command:

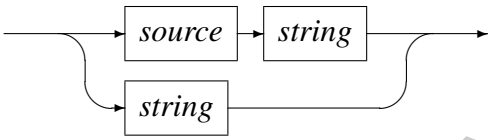
```
drive> display replay tn3270

        Replay: name = tn3270, number = 0, active = 1
        Match     IP: 127.0.0.1. port: 23
        Replay    IP: 10.59.31.80. port: 3023
        State machine = 3270, reclen = 8000, recfm = iaceor
        Thinktime distr = constant, distr parms = [0,0]

drive> close replay tn3270
drive> display replay tn3270

        Replay: name = tn3270, number = 0, active = 0
        Match     IP: 127.0.0.1. port: 23
        Replay    IP: 10.59.31.80. port: 3023
        State machine = 3270, reclen = 8000, recfm = iaceor
        Thinktime distr = constant, distr parms = [0,0]
```

All replay classes can be inactivated using the `all` keyword. Inactivated replay classes cannot be used to create sessions.

Sessions can be closed using the close session command. When a session is closed using the `close` command the local socket file descriptor is closed. This permanently disrupts the session and hence the session cannot be re-activated again. The session can also be in-activated by the peer end of the circuit performing a close operation on the socket. Locally `twbdrive` also closes the circuit when a close log record for the circuit is found on the log file.

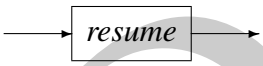
The final use for the `close` command is to close the log file which is currently being used for replay.

SourceCommand



The source command can be used for 'executing' a script file containing additional canned commands. If a single string entered at the command prompt or as a command line argument to `twbdrive` is not a reserved word, then it is assumed to be a script name. This is the mechanism which, together with hash comment lines allows the command files to be executed as shell scripts. Command scripts may also include these two forms of the source command. Source files can be stacked to a depth of twenty files.

ResumeCommand

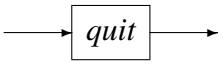


Under certain conditions, the replay enters a state of suspension in which the replay is inactive. This state exists to allow the controller (person operating the `twbdrive` command prompt) to change the state of the system in a controlled manner. When `twbdrive` initially gains control, replay is suspended allowed the controller to define any replay classes and to make appropriate adjustments. Once the controller is satisfied with the configuration, the `resume` command (re-)starts the replaying of messages from the currently open log file. An example of a condition which causes the replay to be suspended is the encountering of the end of the current log file:

```
./stream-a-driver: open log $DRIVER_LOG
Now replaying from log /home/stephen/Logs/sample.00000000
Source command file ./stream-a-driver closed.
drive> switch
Now replaying from log /home/stephen/Logs/sample.00000001
drive> resume
drive> End of log file /home/stephen/Logs/sample.00000001 encountered
```

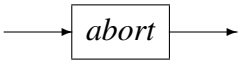
```
Replay has been suspended allowing the controller to attend to the
condition.
Enter the quit command to end the replay or resume once the situation h
been
corrected.
```

QuitCommand



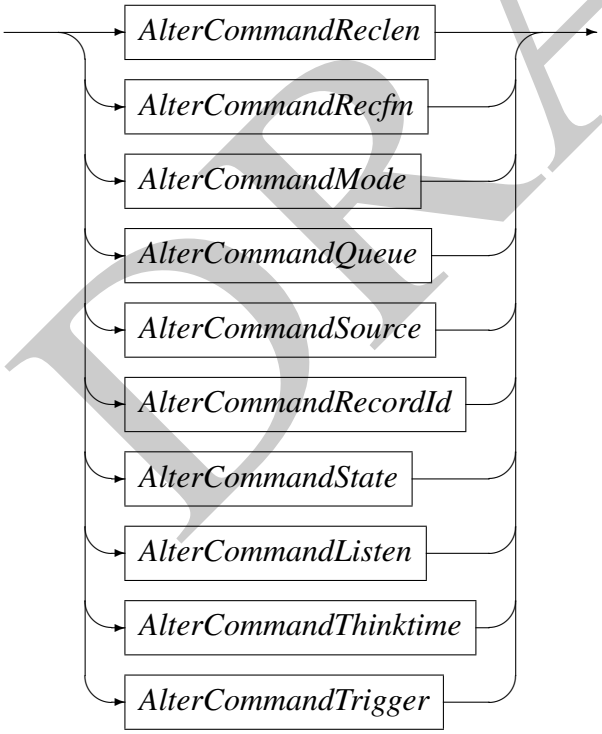
The `quit` command exits the driver. Caution should be used with this command as it is carried out without confirming with the controller.

AbortCommand



The `abort` command is similar to the `quit` command. It causes the driver to exit without confirmation. This command produces a core dump for diagnostic purposes.

AlterCommand



The `replay` command creates a replay class with default attributes. For example, the following replay command

```
$ replay tn3270 inbound 127.0.0.1:23 10.59.31.80:3023
```

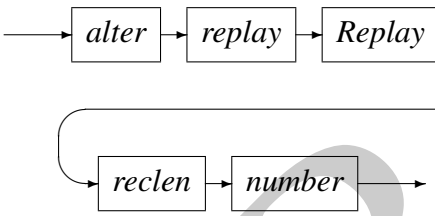
results in a replay class with the following attributes.

```
Replay: name = tn3270, number = 0, active = 1
Match     IP: 127.0.0.1. port: 23
Replay    IP: 10.59.31.80. port: 3023
State machine = stateless, reclen = 32768, recfm = none
Thinktime distr = constant, distr parms = [5000,0]
```

By default the replay class is active. However, any sessions created from this replay class will not be associated with any state machine, will log data in a stream mode without any message or record boundaries. The default think-time is $5000ms$ and which will be drawn as a constant.

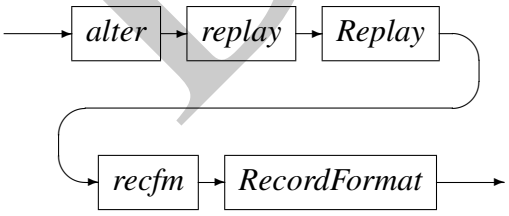
These attributes can be overridden using the `alter` command. The Thinktime distribution and the parameters of the distribution can be changed at any time and will immediately be applicable to all existing sessions that have been created under that class. Changes to the other attributes will only be applicable to new sessions created under the replay class.

AlterCommandReclen

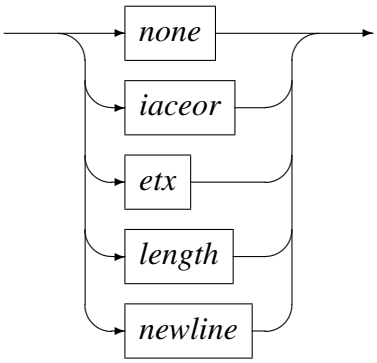


The `reclen` parameter (default value 32768 bytes) is used to allocate a buffer for each session created under the class. The session cannot process log records larger than this buffer size. Also, any data received on the circuit during the replay, will split over more than one buffer should the buffer fill during reception. If a record format is define, then the number of bytes in a records cannot exceed the `reclen` value.

AlterCommandRecfm



RecordFormat



The `recfm` parameter indicates to the `twbdrive` process whether the stream operates in a stream or record mode. The default (`none`) is a stream mode. In this mode, no record identification is made. All the other modes are record modes. In this case the mode indicates how records are identified.

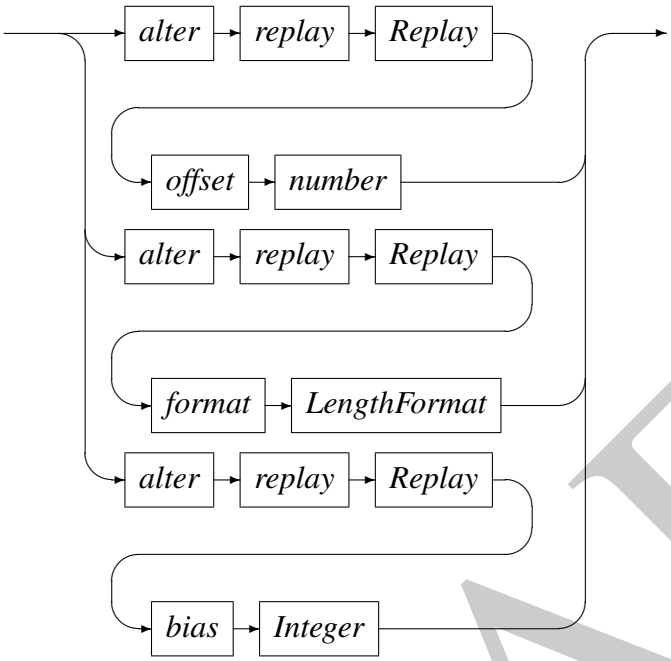
A `recfm` of `iaceor` indicates that the end of the records in a stream can be identified by the trailing TN3270 characters IAC (`0xff`) and EOR (`0xfe`).

A `recfm` of `etx` indicates that the end of the records in a stream can be identified by the trailing ASCII and EBCDIC ETX character (`0x03`).

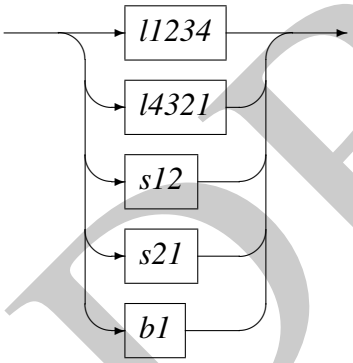
A `recfm` of `newline` indicates that the end of the records in a stream can be identified by the trailing ASCII line feed character (`0x0d`).

A `recfm` of `length` indicates that the end of the records in a stream can be identified by an embedded length field. Since such lengths can be arbitrary, a mechanism is supplied for describing binary lengths. This specification of the length is described by the `offset`, `format` and `bias` attributes.

AlterCommandRecordId

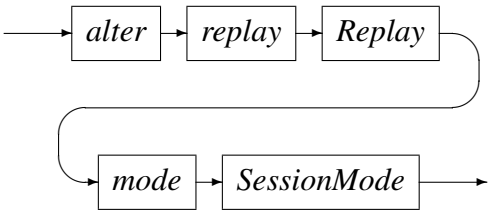


LengthFormat

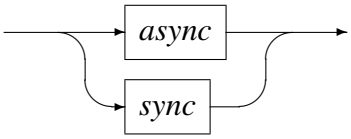


When the `recfm` attribute is set to `length`, `twbdrive` uses the value of the `offset` attribute to locate the embedded length field. The `format` attribute describes the binary format of the embedded length. Four byte length formats are `l1234` for big-endian and `l4321` for little endian. Similarly, two byte formats are described by `s12` and `s21`. A single byte format is also available `b1`. To complete the computation of the expected length, `twbdrive` needs to know the whether the length found in the record is biased. For example, records with fixed headers often contain a length field which describe the number of bytes that follow the header. In this case, the embedded length will need to be biased in order to derive the expected number of bytes in the record. The `bias` attribute supplies this bias value.

AlterCommandMode

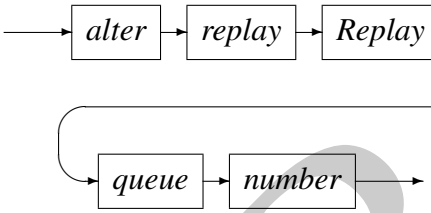


SessionMode



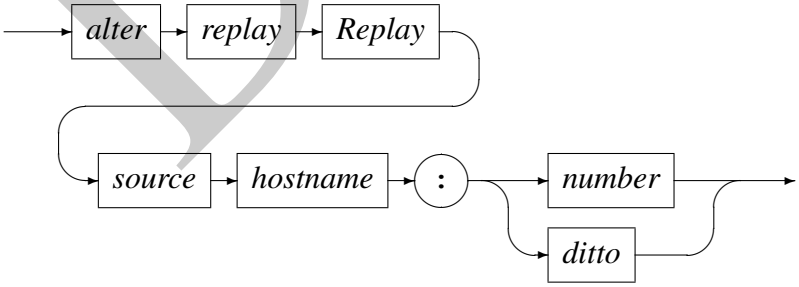
Whether sessions are synchronous or asynchronous is indicated using the `mode` parameter. Presently, this parameter is ignored.

AlterCommandQueue



The `queue` parameter supplies the queue depth for socket listeners. The value of this parameter is only used at the time of the `listen` socket call. See the description of the *AlterCommandListen* command. A listener is only used when replaying `outbound` data.

AlterCommandSource

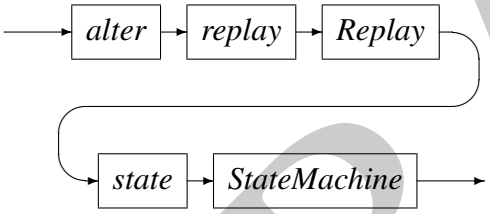


In certain distributed applications, the passive open peer (`listen`) recognises the active open peer (`connect`) by the address to which the active opener has bound. The default binding is to any local adaptor (`INADDR_ANY`) and a dynamically assigned port. In order that a specific adaptor and port number be used, the replay class can be assigned a source port to which the session should bind. If the keyword `ditto` is used instead of a port number, then the actual port number used in the bind is taken from the source socket address of the connection record for the session found in the log file.

This use of the source address only makes sense when `twbdrive` performs the active open (`connect`). If the replay class is defined as replaying `outbound` data then `twbdrive` performs a passive (`listen`) open for the session. In this case the `twbdrive` end of the session is not created when the corresponding session connection log record is found, but rather when the listener is opened (also done via an `alter` command). This allows the session connection to succeed even if it is attempted before the corresponding connect record has been processed.

As connections are accepted for open listeners, the unmatched sessions are queued until the corresponding log record is processed. At this point the destination address from the original session recorded in the log record header as `logdst` is compared to the replay class match IP address and port number. If they are the same then the session linked to the circuit and replay processing continues. If the `ditto` keyword is used, the matching has the extra condition that the source port (as seen by `twbdrive`) on the replay circuit must also match the source port found in the log record header in the field `logsrc.sin_port`.

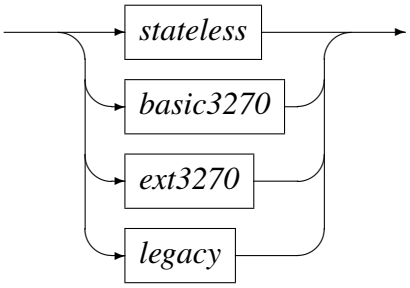
AlterCommandState



Nominating a state machine for the replay class allows the replay to proceed under an additional protocol layer. The protocol state machine has the ability of delaying the forwarding of messages until a certain state is reached or can prevent the forwarding of certain messages. In the latter case, the replay continues by considering the next record on the log file. The state machine also has the ability of altering the messages before the message is replayed.

The state machine can also send the message to other destinations. For example, the state machine has access to the triggered sessions associated to the session owning the log record and can send data on those circuits.

StateMachine



Presently there are three state machines in addition to the default stateless processing

of the replay on the circuit. These state machines are embedded inside twbdrive, but in the future they will be removed and loaded as shared objects as they are named and associated with a replay class.

The basic3270 replay engine observes a minimal set of the 3270 data stream protocol. TN3270E is not supported. This state machine allows 3270 sessions to behave in a manner that observes input inhibit. The code for this state machine is shown in Section 6.

The ext3270 replay engine observes the full 3270 data stream protocol. TN3270E is not supported. The state machine maintains a shadow of the 3270 protocol entity being replayed. This state is the state of the session as seen when the log records were captured. The shadow and replay screens are checked on outbound update for divergence between the two sessions. An arbitrary list of regular expressions can be supplied which to mask the data before comparing screen images.

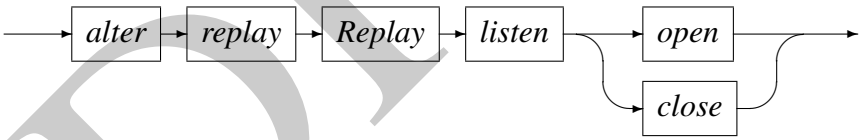
The regular expressions are passed to the ext3270 command function using the pass command as illustrated in the following example:

```
pass exttn3270 "r/TRAN/[0-9][0-9]:[0-9][0-9]:[0-9][0-9]/"
```

On detected of divergence a message is printed inviting a specially designed 3270 terminal emulator to take over the session in order that remediation can be performed. At a suitable point in time, the emulator can disconnect to allow the replay to continue.

The legacy state machine is the proprietary Nedcor legacy MCS state machine.

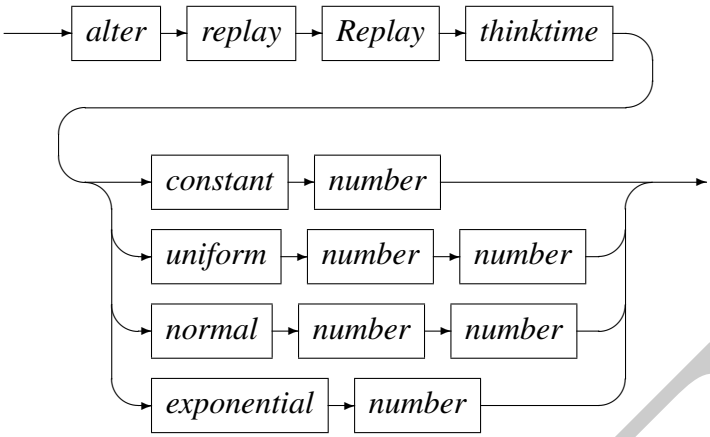
AlterCommandListen



For replay classes used to replay outbound data, the replay IP address and port number pair (the second pair specified in the replay command) is used to perform the passive open (listen). The listener associated with this replay class can be closed and opened with this alter command.

When the replay command is issued, the passive open is not performed until the first time it is opened using the alter command.

AlterCommandThinktime



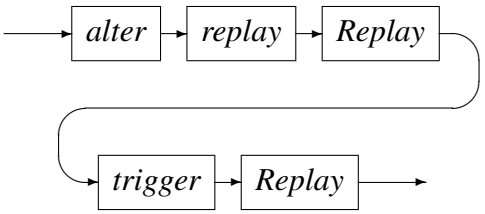
When replaying data on a circuit, each time a message is received from the peer, a random number is drawn from the indicated parameterised distribution. This number is interpreted as the number of milliseconds that have to elapse before a message from the log can be sent to the peer by `twbdrive`. When multiple sessions are being replayed concurrently, the actual delay before a message is resent along a session may be longer than the drawn random number. This is because the order of processing records forwarded by `twbdrive` from the log file have to be in the order recorded in the log file. Delays can be caused by the state machines associated with the sessions. A common case is when the peer of a session has a high response time and prevents the state machine from allowing additional data from the log file to be forwarded on that session. This also means that data for other sessions after this record on the log file could also be delayed.

The supported distributions of the `thinktime` and the meaning of the parameters are listed in Table 1.

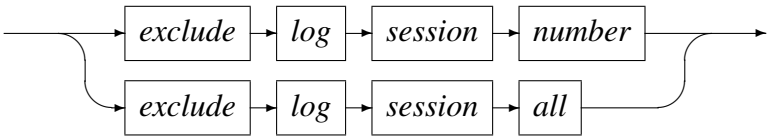
Distribution	Think time parameters
`constant`	In milliseconds
`uniform`	Minimum and maximum
`normal`	Mean and standard deviation
`exponential`	Mean

Table 1: Supported thinktime distributions and their parameters

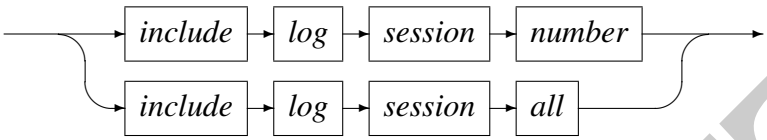
AlterCommandTrigger



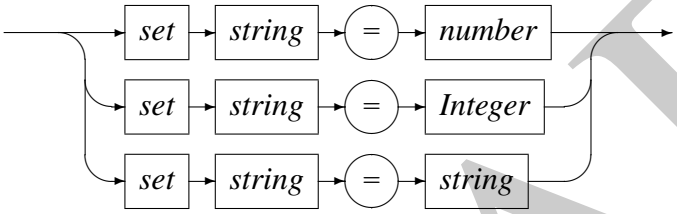
ExcludeCommand



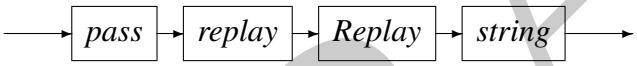
IncludeCommand



SetCommand



PassCommand

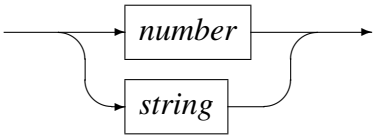


The `pass` command is used to pass a command to the command method of the replay state machine object. Currently only the `ext3270` object supports the command interface. The following examples illustrates the use of this interface where a masking regular expression command is being passed to the replay `tn3270`:

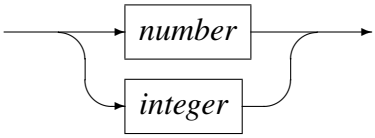
```
drive> replay tn3270 inbound stream-a:23 stream-b:2323

        Replay: name = tn3270, number = 0, active = 1
        Match     IP: 10.59.31.80. port: 23
        Replay    IP: 165.180.227.217. port: 2323
        State machine = stateless, reclen = 32768, recfm = none
        Thinktime distr = constant, distr parms = [5000,0]

+ACK
drive> alter replay tn3270 state ext3270

        Replay: name = tn3270, number = 0, active = 1
        Match     IP: 10.59.31.80. port: 23
        Replay    IP: 165.180.227.217. port: 2323
        State machine = x3270, reclen = 32768, recfm = none
        Thinktime distr = constant, distr parms = [5000,0]
```

```
+ACK
drive> pass replay tn3270 "r/TRAN/DATE: [0-9][0-9]:[0-9][0-9]:[0-9][0-9
+ACK
```

Replay



Integer



Environment variables can be substituted for each of the terminals `integer`, `number` or `string`.

4.6 twbgrep—Search a log file

Program `twbgrep` makes a new log file from an existing log file but includes only the records that match a given regular expression.

4.7 twblogon—Locate log file logon activity

Program `twblogin` filters a log file locating the connect records for the sessions indicated on the command line. Each entry on the the command line is a pair, the first entry of which is a character indicating the session type: '`c`' for CICS sessions and '`m`' for MCS sessions. The session type is used to tell when the end of the session login has been detected. For the MCS type this detection is immediately, but for the CICS sessions this is the first appearance of the EBCDIC string `"DFHCE3549 Sign-on is complete"`.

4.8 twbplan—Build a log file from a plan

Program `twbplan` creates a log file from given log file according to the spec contained in a plan file. This plan file is read either as stdin or from a file named on the command line.

4.9 twbprint—Print a log file

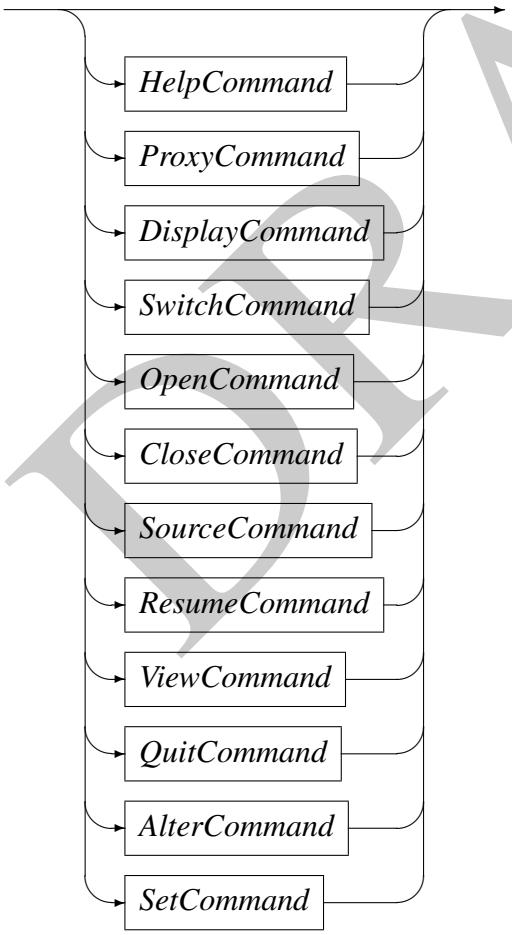
Program twbprint formats log file formatted according to the formats and APIs defined in twblog.h.

4.10 twbproxy—Proxy and log circuits

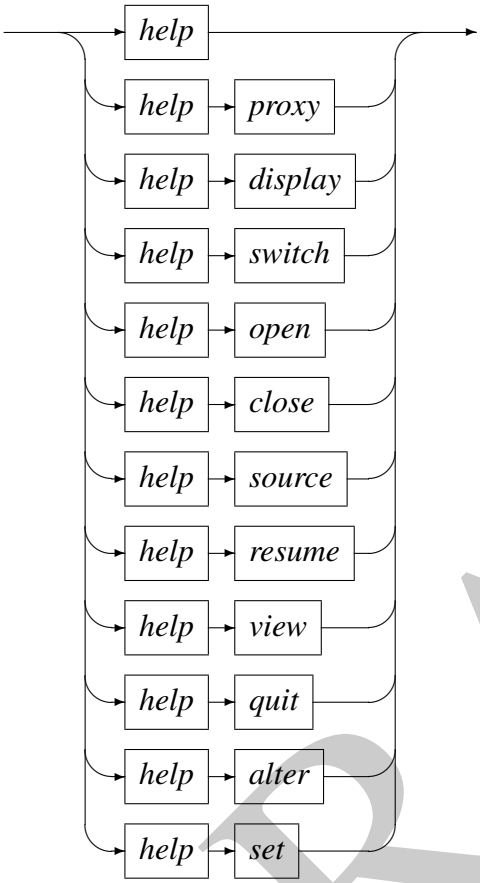
This program listens on a series of ports and for each incoming request it request establishes a connection to a another port. These two ports then make up a pair. Data arriving on one port is written to the other port in the pair and is copied to a trace file.

During processing stdin is also monitored for commands which change the state and report on the status of any sessions.

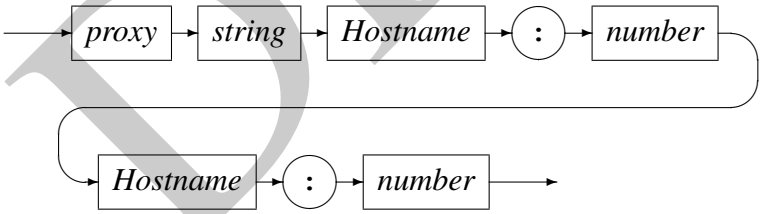
Command



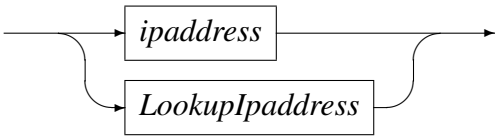
HelpCommand



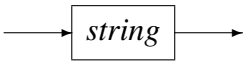
ProxyCommand



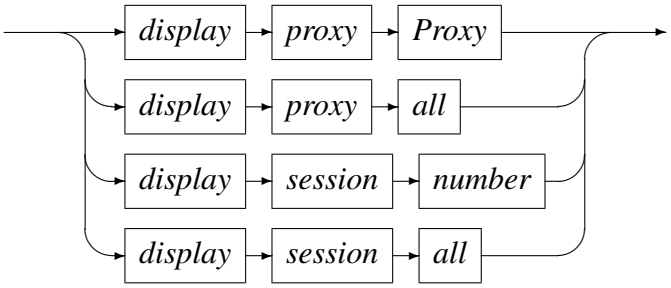
Hostname



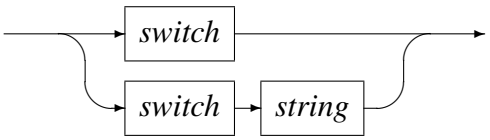
LookupIpaddress



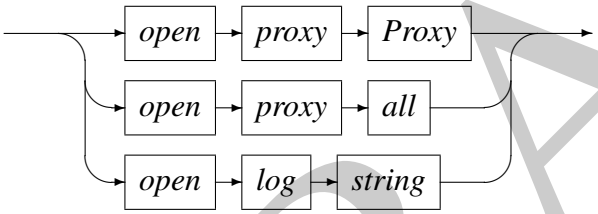
DisplayCommand



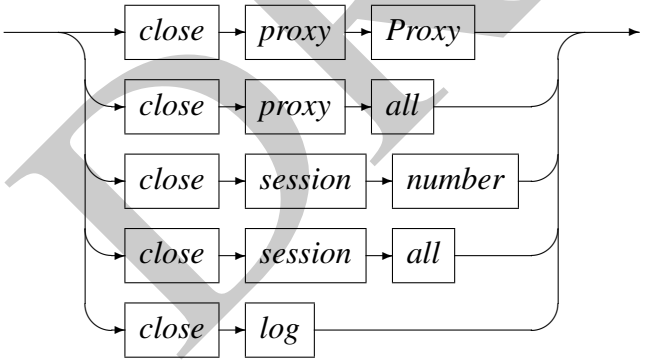
SwitchCommand



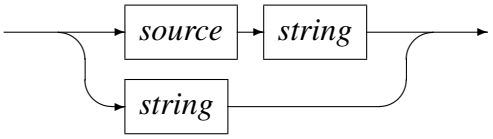
OpenCommand



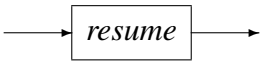
CloseCommand



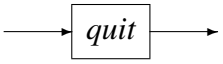
SourceCommand



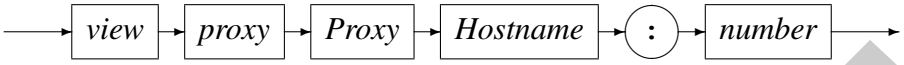
ResumeCommand



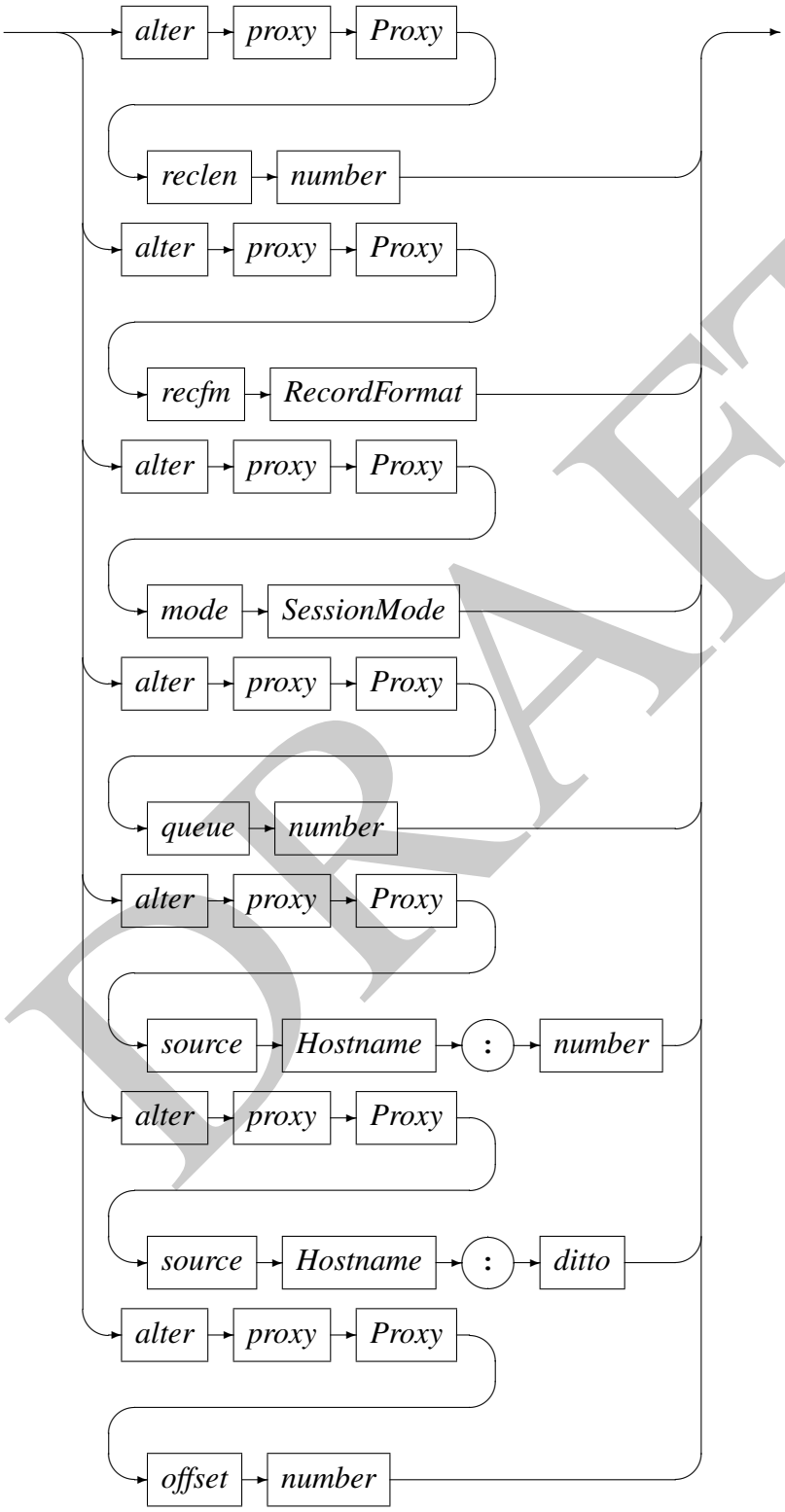
QuitCommand

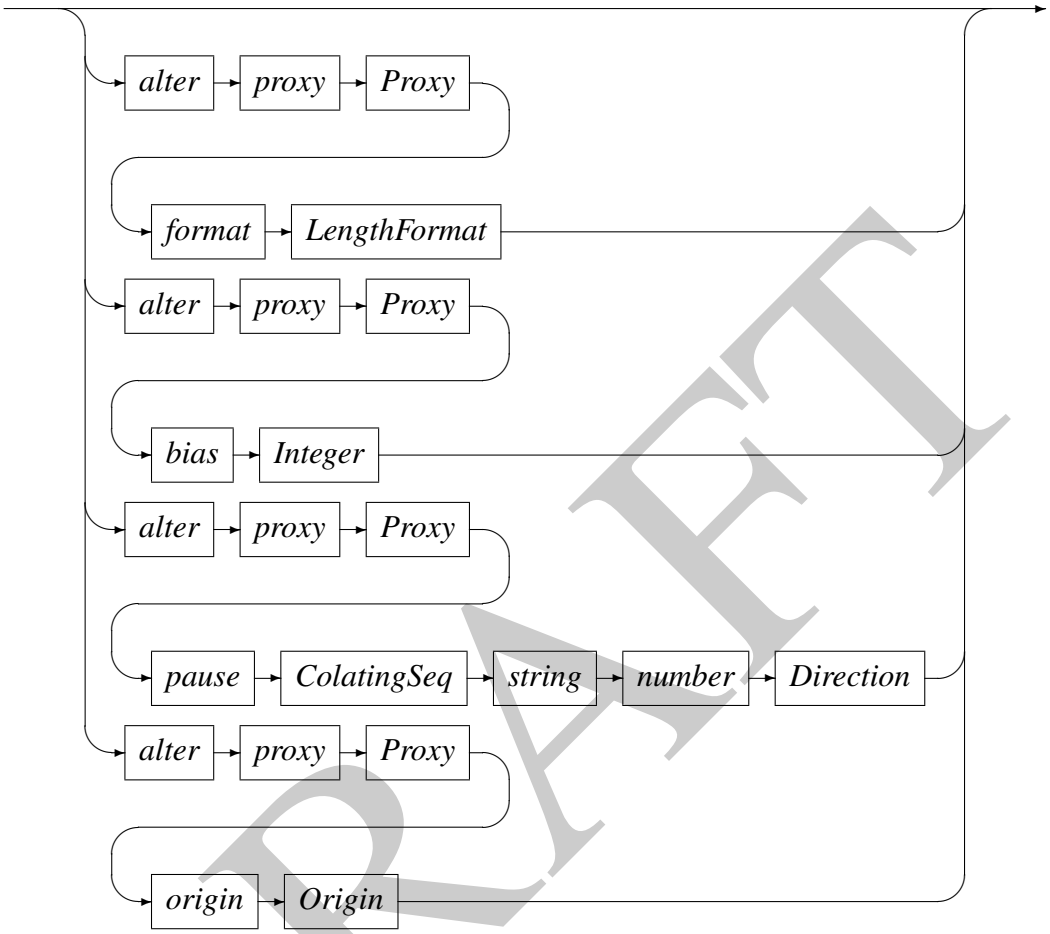


ViewCommand

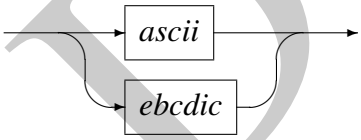


AlterCommand

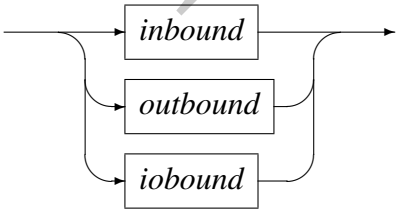




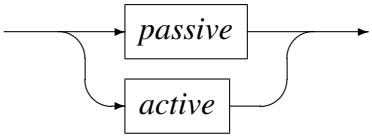
ColatingSeq



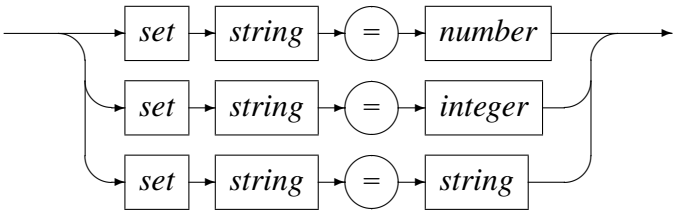
Direction



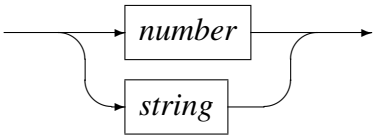
Origin



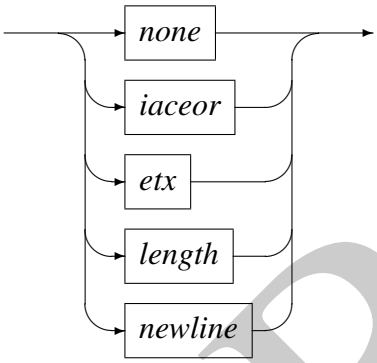
SetCommand



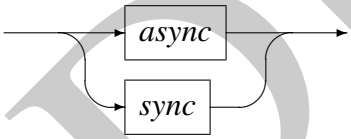
Proxy



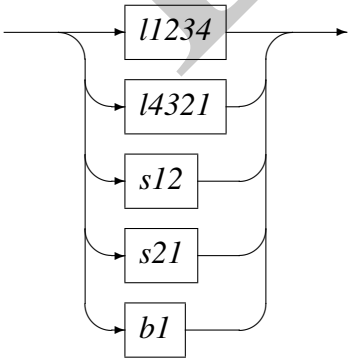
RecordFormat



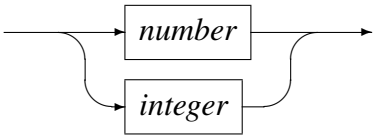
SessionMode



LengthFormat



Integer



Environment variables can be substituted for each of the terminals `integer`, `number` or `string`.

The log file navigation library is also embedded in the `twbdrive` program. Log file navigation commands can be entered in the driver program by prefixing the commands with the "–" character. See section 4.17 for details on the log file navigation commands.

4.11 twbprt3270—Print a log file of 3270 data streams

Program `twbprt3270` uses sks3270 to format the screens of a given session or all sessions if no session is indicated.

4.12 twbrrep—Create replay report from log files

Program `twbrrep` reports on the differences between a log file created by the proxy as when driven by the driver and the log file used by the driver. This log file used by the driver could be a log file created by the proxy as a result of capturing transactions; or some edited version of such a file. It could also in turn be a log file created by the proxy whilst being controlled by the driver.

The program is passed both file names on the command line as arguments. The first stage is the production of activity level inventories of both files. These inventories are checked against each other and any mismatches are reported. A plan for selecting these activities can be produced as an option.

Each of the matching activities are then compared logically with masking. Each activity in which a difference is detected is marked as such. On a final pass through the file, these activities are formatted and printed on a report. A plan for selecting these activities can be produced as an option.

4.13 twbsplit—Split a log file on a pivot record

Program `twbsplit` makes two new log file from an existing log file. The two new log files are the prefix and suffix of the input log file where the first record of the second new log file is the sequence number (or first sequence number greater than or equal to that) passed as the first The resultant log files are more compact and efficient to process when only few sessions from the original file are to be processed in the same drive session.

4.14 twbstrip—Strip sessions out of a log file

Program twbstrip makes a new log file from an existing log file but includes only the sessions indicated on the command line. The resultant log files are more compact and efficient to process when only few sessions from the original file are to be processed in the same drive session.

4.15 twbvi3270—View log file containing 3270 data streams

This program reads a log file and replays back a a selected session on a logged in terminal.

4.16 twbrep3270—Report on 3270 activities in a session

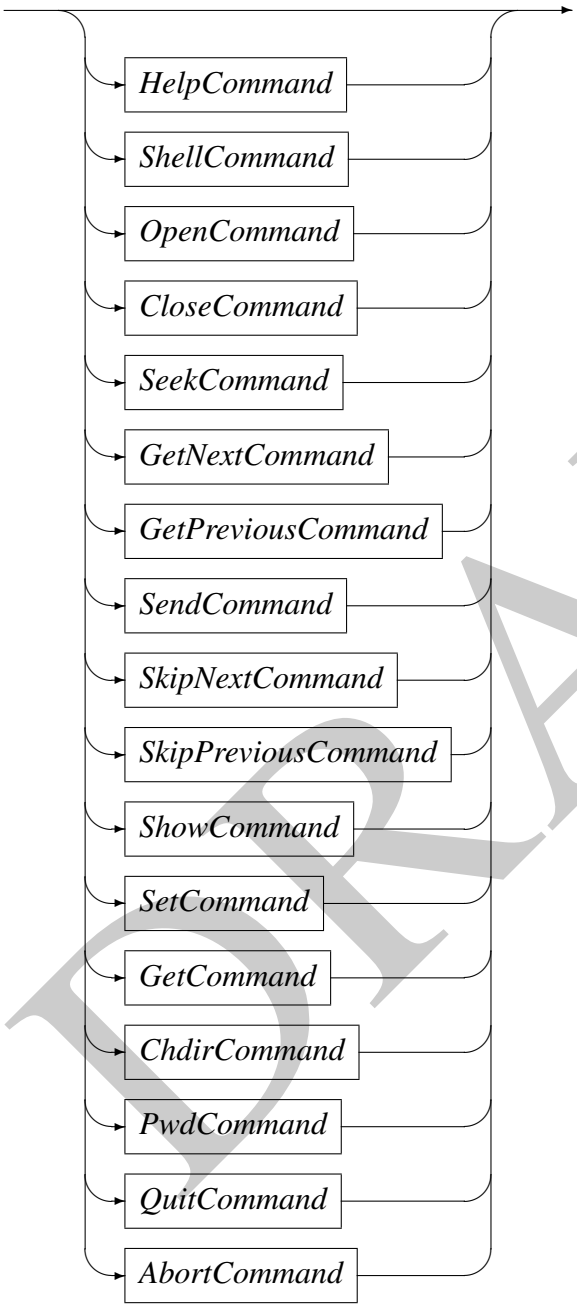
4.17 twblnav—Navigate log files

This program allows a log file to be navigated from a command line interface. Because twblnav also provides +ACK and -ACK feedback on commands entered, it is suitable for being driven by an application domain specific log file navigation program.

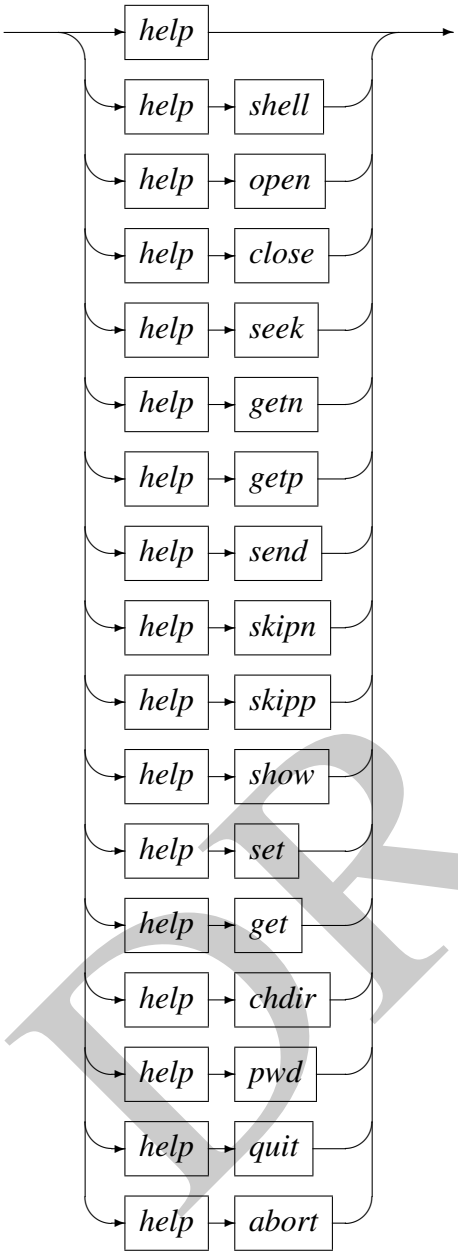
The log file navigation commands are actually provided by the library twbnlib.o. It is this library that is embedded in the twbdrive program to make the log file navigation commands available to the driver.

The log file navigation program (actually the library) provides a means of opening and closing log files; opening and closing sockets for data; reporting and setting the position within the log file; and sending selected data on one of the open sockets. An open log file can be repositioned anywhere in the log file based on the record sequence number or by the position within the log file. Care should be taken that the positions used actually correspond to the start positions of log file records. The current log record position can be sent on a data socket; reading forwards and backwards and with or without sending the corresponding data on a socket is also supported. The current working directory can be changed as well as queried.

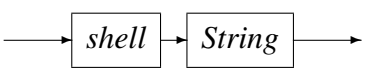
Command



HelpCommand



ShellCommand



The *String* is interpreted as shell command using the `system()` function. The process exit status is used to determine with the command was successful or not. The command status is reported as `+ACK` if the exit status from the shell command is zero and `-ACK` otherwise.

The shell command processes standard out and standard error files are echoed back to

the command interface shell.

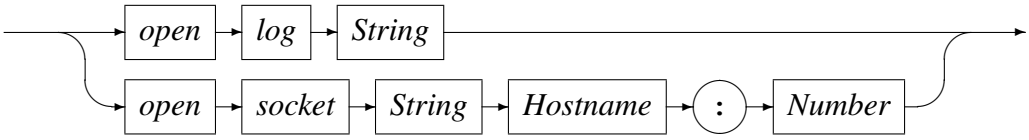
This is one way in which a driving application can obtain, for example, a plan file of a particular log file as the following example illustrates. A program issuing the `shell` command can read the response of the command up but excluding the feedback string `+ACK` or `-ACK`:

```
lnav> shell "twbarep --plan Logs/monday_trevor_2_capture_testers.00000000"
[twbarep] $Id: twbarep.c,v 1.13 2010/10/25 14:51:55 hayward Exp $
Copyright (c) 1996-2001 by Stephen Donaldson [stephen@codemagus.com].
End of log file Logs/monday_trevor_2_capture_testers.00000000 encountered
#
# Log name(s):
#    0. Logs/monday_trevor_2_capture_testers.00000000
#
create ( $CREATE_LOG ) from( Logs/monday_trevor_2_capture_testers.00000000 ) .
#
include name("LOGON") session(0) from(0) to(25) pos(0)
      notes("LOGON:\n"
            "Start=0, end=25, end by=ACTIVITY ENDED\n"
            "Sess=0, src=10.20.1.16:2648, dst=192.168.207.1:23\n"
            "Start time=Tue Apr 17 07:31:59 2001") .
#
include name("0173A") session(0) from(26) to(45) pos(3265)
      notes("0173A:C0TST01 :Correct a rejected merchant deposit.\n"
            "Start=26, end=45, end by=ACTIVITY ENDED\n"
            "Sess=0, src=10.20.1.16:2648, dst=192.168.207.1:23\n"
            "Start time=Tue Apr 17 07:41:14 2001") .
#
include name("0272A") session(0) from(46) to(75) pos(10718)
      notes("0272A:C0TST01 :Set up a new account within a new hierarchical structure.\n"
            "Start=46, end=75, end by=ACTIVITY ENDED\n"
            "Sess=0, src=10.20.1.16:2648, dst=192.168.207.1:23\n"
            "Start time=Tue Apr 17 07:42:27 2001") .
#
include name("0173A") session(0) from(76) to(91) pos(33889)
```

```
      notes("0173A:C0TST01 :Correct a rejected merchant deposit.\n"
            "Start=76, end=91, end by=ACTIVITY ENDED\n"
            "Sess=0, src=10.20.1.16:2648, dst=192.168.207.1:23\n"
            "Start time=Tue Apr 17 07:51:47 2001") .
#
include name("0272A") session(0) from(92) to(129) pos(39233)
      notes("0272A:C0TST01 :Set up a new account within a new hierarchical structure.\n"
            "Start=92, end=129, end by=ZZEND\n"
            "Sess=0, src=10.20.1.16:2648, dst=192.168.207.1:23\n"
            "Start time=Tue Apr 17 07:52:32 2001") .
#
include name("LOGON") session(1) from(0) to(162) pos(0)
      notes("LOGON:\n"
            "Start=0, end=162, end by=ACTIVITY ENDED\n"
            "Sess=1, src=10.20.1.16:2661, dst=192.168.207.1:23\n"
            "Start time=Tue Apr 17 08:26:06 2001") .
#
include name("0173A") session(1) from(163) to(178) pos(70665)
      notes("0173A:N112074 :Correct a rejected merchant deposit.\n"
            "Start=163, end=178, end by=ACTIVITY ENDED\n"
            "Sess=1, src=10.20.1.16:2661, dst=192.168.207.1:23\n"
            "Start time=Tue Apr 17 08:26:58 2001") .
#
include name("0272A") session(1) from(179) to(220) pos(76013)
      notes("0272A:N112074 :Set up a new account within a new hierarchical structure.\n"
            "Start=179, end=220, end by=ACTIVITY ENDED\n"
            "Sess=1, src=10.20.1.16:2661, dst=192.168.207.1:23\n"
            "Start time=Tue Apr 17 08:28:46 2001") .
#
```

```
include name("0272A") session(1) from(221) to(304) pos(109218)
      notes("0272A:N112074 :Set up a new account within a new hierarchical structure.\n"
            "Start=221, end=304, end by=ACTIVITY ENDED\n"
            "Sess=1, src=10.20.1.16:2661, dst=192.168.207.1:23\n"
            "Start time=Tue Apr 17 08:45:58 2001") .
#
include name("0174A") session(1) from(310) to(321) pos(176544)
      notes("0174A:N112074 :Delete a rejected merchant deposit.\n"
            "Start=310, end=321, end by=ACTIVITY ENDED\n"
            "Sess=1, src=10.20.1.16:2661, dst=192.168.207.1:23\n"
            "Start time=Tue Apr 17 09:05:48 2001") .
#
#
+ACK
```

OpenCommand

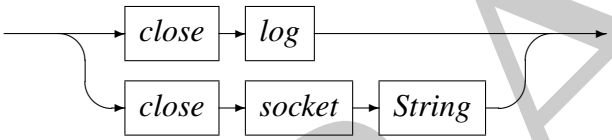


At any point in time only one log file can be open. This log file is opened using the `open log` command. If a log file is already open, then it closed before the new log file is opened.

Many data sockets can be open at the same time. Consequently, to distinguish amongst the open data sockets, the sockets are give names on the `open socket` command. Subsequent commands which require an open socket to send data use this name to refer to the socket.

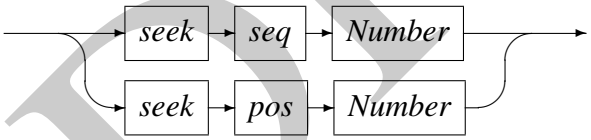
The open performed by `twblnav` is an active open (i.e. `twblnav` performs a `connect()` call) to the indicated adaptor and port.

CloseCommand



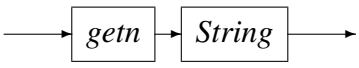
The `close log` command closes the current open log file and the `close socket` command closes the named data socket.

SeekCommand



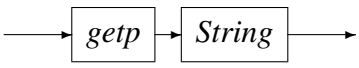
The `seek` command is used to re-position the current open log file to the indicated position or log record sequence number.

GetNextCommand



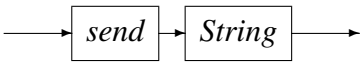
The `getn` command reads the next log record from the current open log file and sends the data on the named open data socket.

GetPreviousCommand

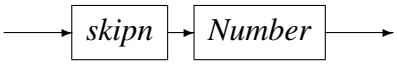


The `getp` command reads the previous log from the current open log file and sends the data on the named open data socket.

SendCommand

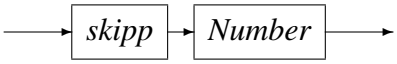


SkipNextCommand



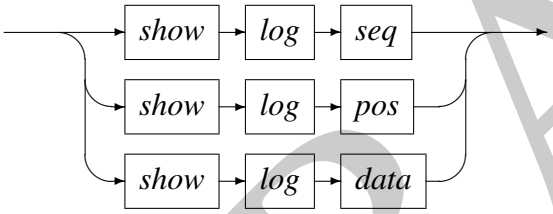
The `skipn` command skips the indicated number of records forward on the current open log file.

SkipPreviousCommand



The `skipp` command skips the indicated number of records backwards on the current open log file.

ShowCommand



The `show log seq` and `show log pos` commands report on the current open log file's position. The first form reports on the current log file's record sequence number and the second form reports on the current open log file's record position. The following example illustrates these commands:

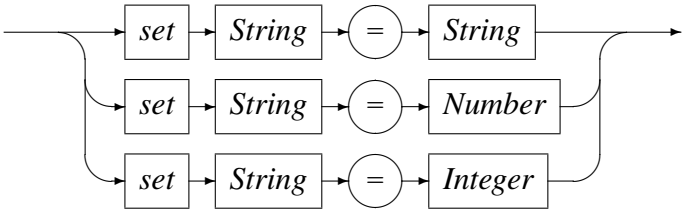
```
lnav> show log seq
AT SEQ 20
+ACK
lnav> show log pos
AT POS 1948
+ACK
```

The `show log data` command prints the current record of the current open log file in the standard hex-dump format:

```
lnav> show log data
Seq=20, Tue Apr 17 07:36:26 2001: Session=0, outbound data, length=4
src=192.168.207.1:23, dst=10.20.1.16:2648, tod=1004305335.761847
        00..__..__05..__..__10..__..__15..__..__20..__..__25..__..__30.
  0000: 01C2FFEF
  0000: ...B....
```

```
+ACK
```

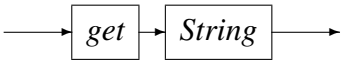
SetCommand



The `set` command is used to set environment variables to particular values.

```
lnav> set SUBWT_HOST_NAME = P390
+ACK
```

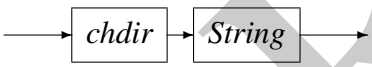
GetCommand



The `get` command is used to print the values of environment variables.

```
lnav> get SUBWT_HOST_NAME
set SUBWT_HOST_NAME = "P390"
+ACK
```

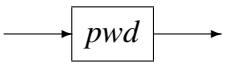
ChdirCommand



The `chdir` command is used to change the current working directory.

```
lnav> shell ls
Logs     body.aux    latexexits       report.dvi  report.tex   trace_pa
Scripts  body.tex    lmc_samples.zip  report.log  report.toc
TEMP     exitpoints  report.aux       report.ps   signoff.tex
+ACK
lnav> chdir Logs
+ACK
```

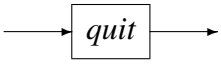
PwdCommand



The `pwd` command reports on the current working directory.

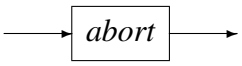
```
lnav> pwd
/home/stephen/acquirer/Logs
+ACK
```

QuitCommand



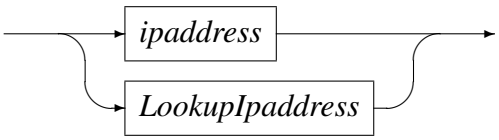
The `quit` command ends the navigator session and exits `twblnav` program.

AbortCommand

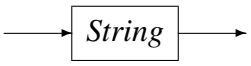


The `abort` command also exits the navigator session, but produces a core dump file for diagnostic purposes.

Hostname



LookupIpaddress



5 Examples

6 Extensions