DriveStat Users Guide

J. Fromm, K. Genser

December 28, 2001
version v1_3

Contents

Chapter 1

Overview

1.1 Introduction

`drivestat` is a Fermilab UPS[3] UNIX package designed to collect tape drive usage statistics into a centralized location. A C API is provided that allows applications to easily collect, compute, and send statistics to the centralized server.

In order for users to use the API `drivestat` and ftt[4] (used in many of the `drivestat` API calls) products should be installed and setup on the specific computer.

Chapter 2

Database

In the Fermilab installation the data is being collected into an ORACLE database which resides on a MISCOMP[1] computer and can be accessed using standard ORACLE methods including but not limited to the MISWEB[2] forms pointed to by the link on the `drivestat` WWW page:
`http://www-isd.fnal.gov/drivestat/`

2.1 Database Tables

There are four tables in the `drivestat` database, they are:

- `MASTER_DRIVESTAT`
- `DRIVESTAT_LOG`
- `DRIVE_MAINTENANCE_LOG`
- `DRIVE_LOCATION`

Brief table descriptions follow. The actual table definitions can be found in the appendix A.

2.1.1 `MASTER_DRIVESTAT`

This table holds the statistical information for each drive. It holds five basic sets of data:

- Absolute statistics as last reported for the drive. These columns are prefixed with an "a".
- Cumulative statistics for the life of the drive. These columns are prefixed with an "l".
- Cumulative statistics since the last install of the drive. These columns are prefixed with an "i".
- Cumulative statistics since the last cleaning of the drive. These columns are prefixed with a "c".
- Cumulative statistics since the last repair of the drive. These columns are prefixed with an "r".

There is one entry per drive in this table.

2.1.2 DRIVESTAT_LOG

This table contains a log of statistic transaction requests. Each time statistics are updated, an entry is made in this table.

2.1.3 DRIVE_LOCATION

This table has specific information about the location of a drive. There is one entry per drive per location.

Note that the STATUS column is used to determine if a drive is current or not. If a drive is moved, it's STATUS is changed but the row remains. This gives a history of locations for a particular drive.

2.1.4 DRIVE_MAINTENANCE_LOG

This table holds a log of maintenance transactions. Any time an install, repair, or clean is performed, an entry is made in this table.

It should be stressed that the information in the database is only as good as the data sent to it. Much depends on the diligence of tape drive users and their willingnes to use the tools provided.

Chapter 3

API

This chapter describes the API of drivestat.

3.1 Making an Executable

In order to use the drivestat API, one needs to include the follwing header file:

```
#include "ds_api.h"
```

The following is an example of a Makefile which can be used to link C code with the drivestat library. It assumes that Fermilab UPS/UPD environment is present at the machine and that ftt and drivestat products were set up.

```
CCFLAGS=-I $(DS_INC) -I $(FTT_INC)
LIB=-L $(DS_LIB) $(DS_LIBS) -L $(FTT_LIB) $(FTT_LIBS) ‘cat $(FTT_DIR)/lib/libs‘
DS_LIBS=-ldrivestat
FTT_LIBS=-lftt
FTT_INC=$(FTT_DIR)/include
FTT_LIB=$(FTT_DIR)/lib
DS_INC=$(DRIVESTAT_DIR)/src/inc
DS_LIB=$(DRIVESTAT_DIR)/lib

dsapi: dsapi.c
        $(CC) dsapi.c $(CCFLAGS) $(LIB) -o dsapi
```

3.2 Data Structures

The data structures and the API prototypes are available through the header file ds_api.h located in $(DRIVESTAT_DIR)/src/inc which should be consulted for the details. The following sections describe its main elements.

3.2.1 DS_DESCRIPTOR

The DS_DESCRIPTOR (see Appendix B.1) is a structure created via the ds_alloc() call, and is used in many of the drivestat API routines. The DS_DESCRIPTOR is used to store statistic information about a tape drive together with the information identifying the drive. There are four main sections of the DS_DESCRIPTOR:

- init_stats - This is where the initial statistics are stored. After computing deltas (see sec. 3.9 below) they are replaced with the most recent statistics.
- delta - This is where the calculation of the delta is stored. The delta is computed using the current (recent) and initial drive statistics.
- sum_of_deltas - This is used to hold the sum of delta statistics.
- recent_stats - This is used to hold the recent but not initial statistics. The recent statistics are then copied to init_stats once the deltas are computed.

The above sections have a common structure template defined as DS_STATS.

3.2.2 DS_STATS

DS_STATS is the place where the collected drive statistics are stored. It is defined as follows:

```
typedef struct ds_stats {
  int init_flag;
  int power_hrs;
  int motion_hrs;
  int mb_user_read;
  int mb_user_write;
  int mb_dev_read;
  int mb_dev_write;
  int read_errors;
  int write_errors;
  int track_retries;
  int underrun;
  int mount_count;
} DS_STATS;
```

with the field names meant to be self explanatory and correspond to the information obtained from ftt.

3.3 ds_alloc

Prototype:

```
DS_DESCRIPTOR *ds_alloc()
```

Allocates a DS_DESCRIPTOR and returns a pointer to it which can be used in subsequent calls. NULL is returned if ds_alloc fails.

3.4 ds_free

Prototype:

```
void ds_free(DS_DESCRIPTOR *ds_desc)
```

Frees the storage allocated via the ds_alloc.

3.5 ds_translate_ftt_drive_id

Prototype:

```
int ds_translate_ftt_drive_id(DS_DESCRIPTOR* ds_desc,
                              ftt_stat_buf ftt_stat_buffer);
```

Uses `ftt` provided information to set fields uniquely identifying the drive:

- `drive_serial_number`,
- `vendor` and
- `product_type`

in the `DS_DESCRIPTOR`. (ftt_stat_buf is described in the `ftt` documentation.) The function returns 0 on success, -1 on error.

3.6 ds_translate_ftt_stats

Prototype:

```
int ds_translate_ftt_stats(DS_DESCRIPTOR* ds_desc,
                           ftt_stat_buf ftt_stat_buffer,int flag);
```

Uses `ftt` provided information to set corresponding `DS_STATS` section of the `DS_DESCRIPTOR`. The flag determines which of the sections is set and can take the values of: INIT and RECENT. A typical use it to first set the `init_stats` section and subsequently keep updating the `recent_stats` one. The function returns 0 on success, -1 on error.

3.7 ds_init

Prototype:

```
int ds_init(DS_DESCRIPTOR* ds_desc, ftt_descriptor ftt_d)
```

This function is used to get the initial drive statistics using `ftt`. These initial statistics are used to calculate deltas in later function calls. This function calls `ds_translate_ftt_drive_id` and `ds_translate_ftt_stats`. It returns 0 on success, -1 on error.

Example (using `ftt`):

```
DS_DESCRIPTOR *ds_desc;
ftt_descriptor ftt_d;
int rc;
ftt_d = ftt_open("/dev/rmt/tps0d1",FTT_RDWR);
ds_desc = ds_alloc();
rc = ds_init(ds_desc,ftt_d);
```

In the above case drive parameters are obtained by `ftt` based on the specified device name and the drive query.

3.8 ds_set_character_field

Prototype:

```
int ds_set_character_field(DS_DESCRIPTOR* ds_desc,
                           char* string, int flag)
```

This function is used to set the host, logical drive name and tape volume serial number fields in DS_DESCRIPTOR. The field to be set is selected based on the value of the flag which can take values of: TAPE_VOLSER, HOST, LOGICAL_DRIVE_NAME.

3.9 ds_compute_delta

Prototype:

```
int ds_compute_delta(DS_DESCRIPTOR *ds_desc)
```

This function computes the delata of elements in the statistics sections of the DS_DESCRIPTOR as follows:

$$\texttt{delta} = \texttt{init\_stats} - \texttt{recent\_stats}$$

and stores it in the delta section. It then adds the result to the sum_of_deltas section. After the calculations are done, the init_stats portion is assigned the values of the recent_stats one while the recent one remians unchaged. It is an error to call this function without first setting the init and the recent portions of the DS_DESCRIPTOR.

3.10 ds_print

Prototype:

```
int ds_print(DS_DESCRIPTOR* ds_desc, char* file);
```

This function prints out all but ftt_descriptor data members of the drivestat structure. If file is not NULL, then the output is redirected to the file, otherwise the output is redirected to stdout.

3.11 ds_set_stats

Prototype:

```
int ds_set_stats(DS_DESCRIPTOR* ds_desc,
                 DS_STATS* ds_stat_buff, int flag)
```

This function can be used as an explicit method of setting a specific DS_STATS portions of DS_DESCRIPTOR. The values used to set this are taken from the storage pointed to by ds_stat_buff. The flag can be one of the following: DELTA, SUM_OF_DELTAS, INIT, RECENT.

3.12 ds_alloc_stats

Prototype:

```
DS_STATS *ds_alloc_stats();
```

Returns a pointer to DS_STATS to be used in subsequent calls. NULL is returned if ds_alloc_stats fails.

3.13 ds_free_stats

Prototype:

```
void ds_free_stats(DS_STATS* dss);
```

Frees the storage allocated via `ds_alloc_stats()`.

3.14 ds_extract_stats

Prototype:

```
int ds_extract_stats(DS_DESCRIPTOR* ds_desc,
                     DS_STATS* ds_stat_buff, int flag);
```

This function is used to obtain a specific `DS_STATS` portion of `DS_DESCRIPTOR`. The values are copied into location pointed to by `ds_stat_buff`.

3.15 ds_bump_deltasum

Prototype:

```
int ds_bump_deltasum(DS_DESCRIPTOR* ds_desc,
                     DS_STATS* ds_stat_buff);
```

This function can be used as an explicit method of increasing sum of deltas `DS_STATS` portion of `DS_DESCRIPTOR`. The values used to do this are taken from storage pointed to by `ds_stat_buff`.

3.16 ds_send_stats

Prototype:

```
int ds_send_stats(DS_DESCRIPTOR* ds_desc, int tout, int flag)
```

This function sends the statistics in the `drivestat` descriptor `ds_desc` to the `drivestat` server. The `flag` can be one of the following:

- `DELTA` - it instructs `drivestat` to send the computed `delta` statistics and add them into the cumulative portions of the `MASTER_DRIVESTAT` table as well as make an entry in the `DRIVESTAT_LOG` table.
- `SUM_OF_DELTAS` - it instructs `drivestat` to send the `sum_of_deltas` and add them into the cumulative portionsof the `MASTER_DRIVESTAT` table as well as make an entry in the `DRIVESTAT_LOG` table.
- `ABSOLUTE` - it instructs `drivestat` to send the `recent_stats` and if not present, the `init_stats` statistics and enter them into the absolute portion of the `MASTER_DRIVESTAT` table as well as make an entry in the `DRIVESTAT_LOG` table.
- `BUMP_MOUNTS` - it causes `drivestat` to increment the mount count for the drive by one in the cumulative portions of the `MASTER_DRIVESTAT` table. This flag can be or'ed with the other flags.

The `tout` int parameter as of version v1_x of `drivestat` is reserved for future use.

3.17 ds_drive_maintenance

Prototype:

```
int ds_drive_maintenance(DS_DESCRIPTOR* ds_desc,
                         int flag,char* host,
                         char* logical_drive_name)
```

This function sends information to the drivestat database server about maintanace operation performed on the drive. It does not actually perform the maintanace itself but merely records the fact that it had been done. The maintanace operation is specified using the flag which can be one of three values:

- INSTALL - This will create a new location entry in the drivestat database in the DRIVE_LOCATION table and set all statistics in the MASTER_DRIVE-STAT table to 0. An entry in DRIVE_MAINTENANCE_LOG table is created as well. If the drive is associated with a logical name, the logical_drive_name field should be set accordingly.
- REPAIR - This will reset the since repaired statistics in the MASTER_DRIVE-STAT table and create an entry in DRIVE_MAINTENANCE_LOG table.
- CLEAN - This will reset the since cleaned statistics in the MASTER_DRIVE-STAT table and create an entry in DRIVE_MAINTENANCE_LOG table.

3.18 Database Queries

The following interface is provided for short queries (limited to 99 rows) of the DRIVESTAT_LOG table). Users are encouraged to use MISWEB (see chapter 2 or other ORACLE interfaces for longer querries.

3.18.1 ds_prepare_list

Prototype:

```
int ds_prepare_list(char* drive,char* vendor,char* prod_type,
                    char* host, char* vsn,char* bdate,
                    char* edate,int* n)
```

This function is used to query the drivestat database DRIVESTAT_LOG . It establishes a database cursor on the server that is used to fetch rows (done by using the ds_getnext() function described below). The search criteria used to establish the cursor is determined by the arguments to the function. An argument with a non-NULL value will cause the cursor to include only rows with those values. If more than one field is non-NULL, a logical AND will be performed. The following example will setup a cursor for all drivestat log entries where host = "apple" *and* vsn = "abc":

```
list_descriptor = dc_prepare_list(NULL,NULL,NULL,
                   ''apple'',''abc'',NULL,NULL,NULL);
if (rc != 0)
do_error_processing();
```

On success, a list descriptor will be returned that is used in subsequent calls to get_next(). -1 is returned on failure.

3.18.2 ds_getnext

Prototype:

```
DS_REPORT* ds_getnext(int list_descriptor)
```

This function returns the next row from the database, storing it in the `report` structure. The `list_descriptor` must have been obtained in a previous call to `ds_prepare_list`.

3.18.3 ds_close_list

Prototype:

```
int ds_close_list(int list_sd)
```

The function releases the list descriptor. It returns 0 on success and -1 when encounering an error.

Appendix A

Database Table Layouts

A.1 MASTER_DRIVESTAT

```
DRIVE_SERIAL_NUMBER         VARCHAR2(80)  NOT NULL,
VENDOR                      VARCHAR2(80)  NOT NULL,
PRODUCT_TYPE                VARCHAR2(80)  NOT NULL,
TIME_STAMP                  DATE,
POWER_HRS                   NUMBER(9)     DEFAULT -1,
MOTION_HRS                  NUMBER(9)     DEFAULT -1,
CLEANING_BIT                NUMBER(1)
       CHECK (Cleaning_Bit = 0 OR Cleaning_Bit = 1),
A_MB_USER_READ              NUMBER(9)     DEFAULT 0,
A_MB_USER_WRITE             NUMBER(9)     DEFAULT 0,
A_MB_DEV_READ               NUMBER(9)     DEFAULT 0,
A_MB_DEV_WRITE              NUMBER(9)     DEFAULT 0,
A_READ_ERRORS               NUMBER(9)     DEFAULT 0,
A_WRITE_ERRORS              NUMBER(9)     DEFAULT 0,
A_TRACK_RETRIES             NUMBER(9)     DEFAULT 0,
A_UNDERRUN                  NUMBER(9)     DEFAULT 0,
A_MOUNT_COUNT               NUMBER(9)     DEFAULT 0,
L_MB_USER_READ              NUMBER(9)     DEFAULT 0,
L_MB_USER_WRITE             NUMBER(9)     DEFAULT 0,
L_MB_DEV_READ               NUMBER(9)     DEFAULT 0,
L_MB_DEV_WRITE              NUMBER(9)     DEFAULT 0,
L_READ_ERRORS               NUMBER(9)     DEFAULT 0,
L_WRITE_ERRORS              NUMBER(9)     DEFAULT 0,
L_TRACK_RETRIES             NUMBER(9)     DEFAULT 0,
L_UNDERRUN                  NUMBER(9)     DEFAULT 0,
L_MOUNT_COUNT               NUMBER(9)     DEFAULT 0,
I_MB_USER_READ              NUMBER(9)     DEFAULT 0,
I_MB_USER_WRITE             NUMBER(9)     DEFAULT 0,
I_MB_DEV_READ               NUMBER(9)     DEFAULT 0,
I_MB_DEV_WRITE              NUMBER(9)     DEFAULT 0,
I_READ_ERRORS               NUMBER(9)     DEFAULT 0,
I_WRITE_ERRORS              NUMBER(9)     DEFAULT 0,
I_TRACK_RETRIES             NUMBER(9)     DEFAULT 0,
I_UNDERRUN                  NUMBER(9)     DEFAULT 0,
I_MOUNT_COUNT               NUMBER(9)     DEFAULT 0,
C_MB_USER_READ              NUMBER(9)     DEFAULT 0,
C_MB_USER_WRITE             NUMBER(9)     DEFAULT 0,
```

```
C_MB_DEV_READ               NUMBER(9)     DEFAULT 0,
C_MB_DEV_WRITE              NUMBER(9)     DEFAULT 0,
C_READ_ERRORS               NUMBER(9)     DEFAULT 0,
C_WRITE_ERRORS              NUMBER(9)     DEFAULT 0,
C_TRACK_RETRIES             NUMBER(9)     DEFAULT 0,
C_UNDERRUN                  NUMBER(9)     DEFAULT 0,
C_MOUNT_COUNT               NUMBER(9)     DEFAULT 0,
R_MB_USER_READ              NUMBER(9)     DEFAULT 0,
R_MB_USER_WRITE             NUMBER(9)     DEFAULT 0,
R_MB_DEV_READ               NUMBER(9)     DEFAULT 0,
R_MB_DEV_WRITE              NUMBER(9)     DEFAULT 0,
R_READ_ERRORS               NUMBER(9)     DEFAULT 0,
R_WRITE_ERRORS              NUMBER(9)     DEFAULT 0,
R_TRACK_RETRIES             NUMBER(9)     DEFAULT 0,
R_UNDERRUN                  NUMBER(9)     DEFAULT 0,
R_MOUNT_COUNT               NUMBER(9)     DEFAULT 0,
PRIMARY KEY(DRIVE_SERIAL_NUMBER,VENDOR,PRODUCT_TYPE,TIME_STAMP)
```

A.2 DRIVESTAT_LOG

```
DRIVE_SERIAL_NUMBER         VARCHAR2(80)  NOT NULL,
VENDOR                      VARCHAR2(80)  NOT NULL,
PRODUCT_TYPE                VARCHAR2(80)  NOT NULL,
TIME_STAMP                  DATE,
TAPE_VOLSER                 VARCHAR2(20),
OPERATION                   VARCHAR2(20),
POWER_HRS                   NUMBER(9),
MOTION_HRS                  NUMBER(9),
CLEANING_BIT                NUMBER(1)
       CHECK (Cleaning_Bit = 0 OR Cleaning_Bit = 1),
MB_USER_READ                NUMBER(9)     DEFAULT 0,
MB_USER_WRITE               NUMBER(9)     DEFAULT 0,
MB_DEV_READ                 NUMBER(9)     DEFAULT 0,
MB_DEV_WRITE                NUMBER(9)     DEFAULT 0,
READ_ERRORS                 NUMBER(9)     DEFAULT 0,
WRITE_ERRORS                NUMBER(9)     DEFAULT 0,
TRACK_RETRIES               NUMBER(9)     DEFAULT 0,
UNDERRUN                    NUMBER(9)     DEFAULT 0,
HOST                        VARCHAR2(80),
PRIMARY KEY(DRIVE_SERIAL_NUMBER,VENDOR,PRODUCT_TYPE,TIME_STAMP)
```

A.3 DRIVE_LOCATION

```
DRIVE_SERIAL_NUMBER         VARCHAR2(80)  NOT NULL,
VENDOR                      VARCHAR2(80)  NOT NULL,
PRODUCT_TYPE                VARCHAR2(80)  NOT NULL,
TIME_STAMP                  DATE,
ACTION                      CHAR(1),
NOTES                       VARCHAR2(255),
PRIMARY KEY(DRIVE_SERIAL_NUMBER, VENDOR, PRODUCT_TYPE,TIME_STAMP)
```

A.4 DRIVE_MAINTENANCE_LOG

```
DRIVE_SERIAL_NUMBER         VARCHAR2(80)  NOT NULL,
VENDOR                      VARCHAR2(80)  NOT NULL,
PRODUCT_TYPE                VARCHAR2(80)  NOT NULL,
TIME_STAMP                  DATE,
HOST                        VARCHAR2(80),
LOGICAL_DRIVE_NAME          VARCHAR2(80),
STATUS                      CHAR(1),
PRIMARY KEY(DRIVE_SERIAL_NUMBER,VENDOR,PRODUCT_TYPE,TIME_STAMP)
```

Appendix B

Data Structures

B.1 DS_DESCRIPTOR

```
typedef struct ds_descriptor {

  int  ds_init_flag;
  char drive_serial_number[MAX_DRIVE_SERIAL_NUMBER_LEN + 1];
  char vendor[MAX_VENDOR_LEN + 1];
  char product_type[MAX_PRODUCT_TYPE_LEN + 1];
  char logical_drive_name[MAX_LOGICAL_DRIVE_NAME_LEN + 1];
  char host[MAX_HOST_LEN + 1];
  char tape_volser[MAX_VOLSER_LEN + 1];
  char operation[MAX_OPERATION_SIZE + 1];
  int  cleaning_bit;
  ftt_descriptor ftt_d;

  DS_STATS init_stats;
  DS_STATS delta;
  DS_STATS sum_of_deltas;
  DS_STATS recent_stats;
} DS_DESCRIPTOR;
```

B.2 DS_STATS

```
typedef struct ds_stats {
  int init_flag;
  int power_hrs;
  int motion_hrs;
  int mb_user_read;
  int mb_user_write;
  int mb_dev_read;
  int mb_dev_write;
  int read_errors;
  int write_errors;
  int track_retries;
  int underrun;
  int mount_count;
} DS_STATS;
```

Appendix C

Installing and (re-)Building the `drivestat` product

Installation of the `drivestat` client involves installing the ups `drivestat` product, and rebuilding it if necessary.

The following is an example of installing and declaring the `drivestat` product as a test instance of the client on a Linux node:

```
> upd install -f Linux -q client drivestat v1_3
> ups declare -t -f Linux drivestat v1_3
```

Subsequently the product may need to be rebuild depending e.g. on the ftt version of the product installed on the specific node.

```
> ups make -t drivestat
```

Bibliography

[1] Fermilab Computing Division Library Document WP0067: MISCOMP Project

[2] Fermilab Computing Division Library Document PU0397: MISWEB

[3] Fermilab Computing Division Library Document PU0117: UPS UNIX Product Support

[4] Fermilab Computing Division Library Document PU0236: Fermi Tape Tools.