NAME

devstat, **devstat_getnumdevs**, **devstat_getgeneration**, **devstat_getversion**, **devstat_checkversion**, **devstat_getdevs**, **devstat_selectdevs**, **devstat_buildmatch**, **devstat_compute_statistics**, **devstat_compute_etime** - device statistics utility library

LIBRARY

Device Statistics Library (libdevstat, -ldevstat)

SYNOPSIS

#include <devstat.h>

int
devstat_getnumdevs(*kvm_t *kd*);

long
devstat_getgeneration(*kvm_t *kd*);

int
devstat_getversion(*kvm_t *kd*);

int
devstat_checkversion(*kvm_t *kd*);

int
devstat_getdevs(*kvm_t *kd*, *struct statinfo *stats*);

int
devstat_selectdevs(*struct device_selection **dev_select*, *int *num_selected*, *int *num_selections*, *long *select_generation*, *long current_generation*, *struct devstat *devices*, *int numdevs*, *struct devstat_match *matches*, *int num_matches*, *char **dev_selections*, *int num_dev_selections*, *devstat_select_mode select_mode*, *int maxshowdevs*, *int perf_select*);

int
devstat_buildmatch(*char *match_str*, *struct devstat_match **matches*, *int *num_matches*);

int
devstat_compute_statistics(*struct devstat *current*, *struct devstat *previous*, *long double etime*, ...);

long double
devstat_compute_etime(*struct bintime *cur_time*, *struct bintime *prev_time*);

DESCRIPTION

The **devstat** library is a library of helper functions for dealing with the kernel devstat(9) interface, which is accessible to users via sysctl(3) and kvm(3). All functions that take a *kvm_t ** as first argument can be passed NULL instead of a kvm handle as this argument, which causes the data to be read via sysctl(3). Otherwise, it is read via kvm(3) using the supplied handle. The **devstat_checkversion**() function should be called with each kvm handle that is going to be used (or with NULL if sysctl(3) is going to be used).

The **devstat_getnumdevs**() function returns the number of devices registered with the **devstat** subsystem in the kernel.

The **devstat_getgeneration**() function returns the current generation of the **devstat** list of devices in the kernel.

The **devstat_getversion**() function returns the current kernel **devstat** version.

The **devstat_checkversion**() function checks the userland **devstat** version against the kernel **devstat** version. If the two are identical, it returns zero. Otherwise, it prints an appropriate error in *devstat_errbuf* and returns -1.

The **devstat_getdevs**() function fetches the current list of devices and statistics into the supplied *statinfo* structure. The *statinfo* structure can be found in <*devstat.h*>:

```
struct statinfo {
        long            cp_time[CPUSTATES];
        long            tk_nin;
        long            tk_nout;
        struct devinfo  *dinfo;
        long double     snap_time;
};
```

The **devstat_getdevs**() function expects the *statinfo* structure to be allocated, and it also expects the *dinfo* subelement to be allocated and zeroed prior to the first invocation of **devstat_getdevs**(). The *dinfo* subelement is used to store state between calls, and should not be modified after the first call to **devstat_getdevs**(). The *dinfo* subelement contains the following elements:

```
struct devinfo {
        struct devstat      *devices;
        uint8_t             *mem_ptr;
        long                generation;
```

```
        int             numdevs;
};
```

The *kern.devstat.all* sysctl(8) variable contains an array of **devstat** structures, but at the head of the array is the current **devstat** generation. The reason the generation is at the head of the buffer is so that userland software accessing the **devstat** statistics information can atomically get both the statistics information and the corresponding generation number. If client software were forced to get the generation number via a separate sysctl(8) variable (which is available for convenience), the list of devices could change between the time the client gets the generation and the time the client gets the device list.

The *mem_ptr* subelement of the *devinfo* structure is a pointer to memory that is allocated, and resized if necessary, by **devstat_getdevs**(). The devices subelement of the *devinfo* structure is basically a pointer to the beginning of the array of devstat structures from the *kern.devstat.all* sysctl(8) variable (or the corresponding values read via kvm(3)). The generation subelement of the *devinfo* structure contains the corresponding generation number. The *numdevs* subelement of the *devinfo* structure contains the current number of devices registered with the kernel **devstat** subsystem.

The **devstat_selectdevs**() function selects devices to display based upon a number of criteria:

specified devices
: Specified devices are the first selection priority. These are generally devices specified by name by the user e.g. da0, da1, cd0.

match patterns
: These are pattern matching expressions generated by **devstat_buildmatch**() from user input.

performance
: If performance mode is enabled, devices will be sorted based on the *bytes* field in the *device_selection* structure passed in to **devstat_selectdevs**(). The *bytes* value currently must be maintained by the user. In the future, this may be done for him in a **devstat** library routine. If no devices have been selected by name or by pattern, the performance tracking code will select every device in the system, and sort them by performance. If devices have been selected by name or pattern, the performance tracking code will honor those selections and will only sort among the selected devices.

order in the devstat list
: If the selection mode is set to DS_SELECT_ADD, and if there are still less than *maxshowdevs* devices selected, **devstat_selectdevs**() will automatically select up to *maxshowdevs* devices.

The **devstat_selectdevs**() function performs selections in four different modes:

DS_SELECT_ADD	In "add" mode, **devstat_selectdevs**() will select any unselected devices specified by name or matching pattern. It will also select more devices, in devstat list order, until the number of selected devices is equal to *maxshowdevs* or until all devices are selected.
DS_SELECT_ONLY	In "only" mode, **devstat_selectdevs**() will clear all current selections, and will only select devices specified by name or by matching pattern.
DS_SELECT_REMOVE	In "remove" mode, **devstat_selectdevs**() will remove devices specified by name or by matching pattern. It will not select any additional devices.
DS_SELECT_ADDONLY	In "add only" mode, **devstat_selectdevs**() will select any unselected devices specified by name or matching pattern. In this respect it is identical to "add" mode. It will not, however, select any devices other than those specified.

In all selection modes, **devstat_selectdevs**() will not select any more than *maxshowdevs* devices. One exception to this is when you are in "top" mode and no devices have been selected. In this case, **devstat_selectdevs**() will select every device in the system. Client programs must pay attention to selection order when deciding whether to pay attention to a particular device. This may be the wrong behavior, and probably requires additional thought.

The **devstat_selectdevs**() function handles allocation and resizing of the *dev_select* structure passed in by the client. The **devstat_selectdevs**() function uses the *numdevs* and *current_generation* fields to track the current **devstat** generation and number of devices. If *num_selections* is not the same as *numdevs* or if *select_generation* is not the same as *current_generation*, **devstat_selectdevs**() will resize the selection list as necessary, and re-initialize the selection array.

The **devstat_buildmatch**() function takes a comma separated match string and compiles it into a *devstat_match* structure that is understood by **devstat_selectdevs**(). Match strings have the following format:

device,type,if

The **devstat_buildmatch**() function takes care of allocating and reallocating the match list as necessary. Currently known match types include:

device type:

da	Direct Access devices
sa	Sequential Access devices
printer	Printers
proc	Processor devices
worm	Write Once Read Multiple devices
cd	CD devices
scanner	Scanner devices
optical	Optical Memory devices
changer	Medium Changer devices
comm	Communication devices
array	Storage Array devices
enclosure	Enclosure Services devices
floppy	Floppy devices

interface:

IDE	Integrated Drive Electronics devices
SCSI	Small Computer System Interface devices
other	Any other device interface

passthrough:

pass	Passthrough devices

The **devstat_compute_statistics**() function provides complete statistics calculation. There are four arguments for which values *must* be supplied: *current*, *previous*, *etime*, and the terminating argument for the varargs list, DSM_NONE. For most applications, the user will want to supply valid *devstat* structures for both *current* and *previous*. In some instances, for instance when calculating statistics since system boot, the user may pass in a NULL pointer for the *previous* argument. In that case, **devstat_compute_statistics**() will use the total stats in the *current* structure to calculate statistics over *etime*. For each statistics to be calculated, the user should supply the proper enumerated type (listed below), and a variable of the indicated type. All statistics are either integer values, for which a *uint64_t* is used, or floating point, for which a *long double* is used. The statistics that may be calculated are:

DSM_NONE	type: N/A This *must* be the last argument passed to **devstat_compute_statistics**(). It is an argument list terminator.
DSM_TOTAL_BYTES	type: *uint64_t* *

The total number of bytes transferred between the acquisition of *previous* and *current*.

DSM_TOTAL_BYTES_READ

DSM_TOTAL_BYTES_WRITE

DSM_TOTAL_BYTES_FREE — type: *uint64_t* *

The total number of bytes in transactions of the specified type between the acquisition of *previous* and *current*.

DSM_TOTAL_TRANSFERS — type: *uint64_t* *

The total number of transfers between the acquisition of *previous* and *current*.

DSM_TOTAL_TRANSFERS_OTHER

DSM_TOTAL_TRANSFERS_READ

DSM_TOTAL_TRANSFERS_WRITE

DSM_TOTAL_TRANSFERS_FREE — type: *uint64_t* *

The total number of transactions of the specified type between the acquisition of *previous* and *current*.

DSM_TOTAL_DURATION — type: *long double* *

The total duration of transactions, in seconds, between the acquisition of *previous* and *current*.

DSM_TOTAL_DURATION_OTHER

DSM_TOTAL_DURATION_READ

DSM_TOTAL_DURATION_WRITE

DSM_TOTAL_DURATION_FREE — type: *long double* *

The total duration of transactions of the specified type between the acquisition of *previous* and *current*.

DSM_TOTAL_BUSY_TIME — type: *long double* *

Total time the device had one or more transactions outstanding between the acquisition of *previous* and *current*.

DSM_TOTAL_BLOCKS — type: *uint64_t* *

The total number of blocks transferred between the acquisition of *previous* and *current*. This number is in terms of the blocksize reported by the device. If no blocksize has been reported (i.e., the block size is 0), a default blocksize of 512 bytes will be used in the calculation.

DSM_TOTAL_BLOCKS_READ

DSM_TOTAL_BLOCKS_WRITE

DSM_TOTAL_BLOCKS_FREE — type: *uint64_t* *

The total number of blocks of the specified type between the acquisition of *previous* and *current*. This number is in terms of the blocksize reported by the device. If no blocksize has been reported (i.e., the block size is 0), a default blocksize of 512 bytes will be used in the calculation.

DSM_KB_PER_TRANSFER — type: *long double* *

The average number of kilobytes per transfer between the acquisition of *previous* and *current*.

DSM_KB_PER_TRANSFER_READ

DSM_KB_PER_TRANSFER_WRITE

DSM_KB_PER_TRANSFER_FREE type: *long double* *

The average number of kilobytes in the specified type transaction between the acquisition of *previous* and *current*.

DSM_TRANSFERS_PER_SECOND type: *long double* *

The average number of transfers per second between the acquisition of *previous* and *current*.

DSM_TRANSFERS_PER_SECOND_OTHER

DSM_TRANSFERS_PER_SECOND_READ

DSM_TRANSFERS_PER_SECOND_WRITE

DSM_TRANSFERS_PER_SECOND_FREE type: *long double* *

The average number of transactions of the specified type per second between the acquisition of *previous* and *current*.

DSM_MB_PER_SECOND type: *long double* *

The average number of megabytes transferred per second between the acquisition of *previous* and *current*.

DSM_MB_PER_SECOND_READ

DSM_MB_PER_SECOND_WRITE

DSM_MB_PER_SECOND_FREE type: *long double* *

The average number of megabytes per second in the specified type of transaction between the acquisition of *previous* and *current*.

DSM_BLOCKS_PER_SECOND — type: *long double* *

The average number of blocks transferred per second between the acquisition of *previous* and *current*. This number is in terms of the blocksize reported by the device. If no blocksize has been reported (i.e., the block size is 0), a default blocksize of 512 bytes will be used in the calculation.

DSM_BLOCKS_PER_SECOND_READ

DSM_BLOCKS_PER_SECOND_WRITE

DSM_BLOCKS_PER_SECOND_FREE — type: *long double* *

The average number of blocks per second in the specified type of transaction between the acquisition of *previous* and *current*. This number is in terms of the blocksize reported by the device. If no blocksize has been reported (i.e., the block size is 0), a default blocksize of 512 bytes will be used in the calculation.

DSM_MS_PER_TRANSACTION — type: *long double* *

The average duration of transactions between the acquisition of *previous* and *current*.

DSM_MS_PER_TRANSACTION_OTHER

DSM_MS_PER_TRANSACTION_READ

DSM_MS_PER_TRANSACTION_WRITE

DSM_MS_PER_TRANSACTION_FREE — type: *long double* *

The average duration of transactions of the specified type between the acquisition of *previous* and *current*.

DSM_BUSY_PCT — type: *long double* *

The percentage of time the device had one or more transactions outstanding between the acquisition of *previous* and *current*.

DSM_QUEUE_LENGTH — type: *uint64_t* *

The number of not yet completed transactions at the time when *current* was acquired.

DSM_SKIP — type: N/A

If you do not need a result from **devstat_compute_statistics**(), just put DSM_SKIP as first (type) parameter and NULL as second parameter. This can be useful in scenarios where the statistics to be calculated are determined at run time.

The **devstat_compute_etime**() function provides an easy way to find the difference in seconds between two *bintime* structures. This is most commonly used in conjunction with the time recorded by the **devstat_getdevs**() function (in *struct statinfo*) each time it fetches the current **devstat** list.

RETURN VALUES

The **devstat_getnumdevs**(), **devstat_getgeneration**(), and **devstat_getversion**() function return the indicated sysctl variable, or -1 if there is an error fetching the variable.

The **devstat_checkversion**() function returns 0 if the kernel and userland **devstat** versions match. If they do not match, it returns -1.

The **devstat_getdevs**() and **devstat_selectdevs**() functions return -1 in case of an error, 0 if there is no error, and 1 if the device list or selected devices have changed. A return value of 1 from **devstat_getdevs**() is usually a hint to re-run **devstat_selectdevs**() because the device list has changed.

The **devstat_buildmatch**() function returns -1 for error, and 0 if there is no error.

The **devstat_compute_etime**() function returns the computed elapsed time.

The **devstat_compute_statistics**() function returns -1 for error, and 0 for success.

If an error is returned from one of the **devstat** library functions, the reason for the error is generally printed in the global string *devstat_errbuf* which is DEVSTAT_ERRBUF_SIZE characters long.

SEE ALSO

systat(1), kvm(3), sysctl(3), iostat(8), rpc.rstatd(8), sysctl(8), vmstat(8), devstat(9)

HISTORY

The **devstat** statistics system first appeared in FreeBSD 3.0. The new interface (the functions prefixed with devstat_) first appeared in FreeBSD 5.0.

AUTHORS

Kenneth Merry <*ken@FreeBSD.org*>

BUGS

There should probably be an interface to de-allocate memory allocated by **devstat_getdevs**(), **devstat_selectdevs**(), and **devstat_buildmatch**().

The **devstat_selectdevs**() function should probably not select more than *maxshowdevs* devices in "top" mode when no devices have been selected previously.

There should probably be functions to perform the statistics buffer swapping that goes on in most of the clients of this library.

The *statinfo* and *devinfo* structures should probably be cleaned up and thought out a little more.