NAME

stats - statistics gathering

LIBRARY

library "libstats"

SYNOPSIS

#include <sys/arb.h>
#include <sys/qmath.h>
#include <sys/stats.h>

Stats Blob Template Management Functions

int
stats_tpl_alloc(*const char *name, uint32_t flags*);

int
stats_tpl_fetch_allocid(*const char *name, uint32_t hash*);

int
stats_tpl_fetch(*int tpl_id, struct statsblob_tpl **tpl*);

int
stats_tpl_id2name(*uint32_t tpl_id, char *buf, size_t len*);

int
stats_tpl_sample_rates(*SYSCTL_HANDLER_ARGS*);

int
stats_tpl_sample_rollthedice(*struct stats_tpl_sample_rate *rates, int nrates, void *seed_bytes, size_t seed_len*);

struct voistatspec
STATS_VSS_SUM();

struct voistatspec
STATS_VSS_MAX();

struct voistatspec
STATS_VSS_MIN();

struct voistatspec
STATS_VSS_CRHIST<32|64>_LIN(*lb*, *ub*, *stepinc*, *vsdflags*);

struct voistatspec
STATS_VSS_CRHIST<32|64>_EXP(*lb*, *ub*, *stepbase*, *stepexp*, *vsdflags*);

struct voistatspec
STATS_VSS_CRHIST<32|64>_LINEXP(*lb*, *ub*, *nlinsteps*, *stepbase*, *vsdflags*);

struct voistatspec
STATS_VSS_CRHIST<32|64>_USR(**HBKTS**(**CRBKT**(*lb*), ...), *vsdflags*);

struct voistatspec
STATS_VSS_DRHIST<32|64>_USR(**HBKTS**(**DRBKT**(*lb*, *ub*), ...), *vsdflags*);

struct voistatspec
STATS_VSS_DVHIST<32|64>_USR(**HBKTS**(**DVBKT**(*val*), ...), *vsdflags*);

struct voistatspec
STATS_VSS_TDGSTCLUST<32|64>(*nctroids*, *prec*);

int
stats_tpl_add_voistats(*uint32_t tpl_id*, *int32_t voi_id*, *const char *voi_name*,
enum vsd_dtype voi_dtype, *uint32_t nvss*, *struct voistatspec *vss*, *uint32_t flags*);

Stats Blob Data Gathering Functions

int
stats_voi_update_<abs|rel>_<dtype>(*struct statsblob *sb*, *int32_t voi_id*, *<dtype> voival*);

Stats Blob Utility Functions

*struct statsblob **
stats_blob_alloc(*uint32_t tpl_id*, *uint32_t flags*);

int
stats_blob_init(*struct statsblob *sb*, *uint32_t tpl_id*, *uint32_t flags*);

int
stats_blob_clone(*struct statsblob **dst*, *size_t dstmaxsz*, *struct statsblob *src*, *uint32_t flags*);

void

stats_blob_destroy(*struct statsblob *sb*);

int
stats_voistat_fetch_dptr(*struct statsblob *sb, int32_t voi_id, enum voi_stype stype, enum vsd_dtype *retdtype, struct voistatdata **retvsd, size_t *retvsdsz*);

int
stats_voistat_fetch_<dtype>(*struct statsblob *sb, int32_t voi_id, enum voi_stype stype, <dtype> *ret*);

int
stats_blob_snapshot(*struct statsblob **dst, size_t dstmaxsz, struct statsblob *src, uint32_t flags*);

int
stats_blob_tostr(*struct statsblob *sb, struct sbuf *buf, enum sb_str_fmt fmt, uint32_t flags*);

int
stats_voistatdata_tostr(*const struct voistatdata *vsd, enum vsd_dtype dtype, enum sb_str_fmt fmt, struct sbuf *buf, int objdump*);

typedef int
(*stats_blob_visitcb_t)(*struct sb_visit *sbv, void *usrctx*);

int
stats_blob_visit(*struct statsblob *sb, stats_blob_visitcb_t func, void *usrctx*);

DESCRIPTION

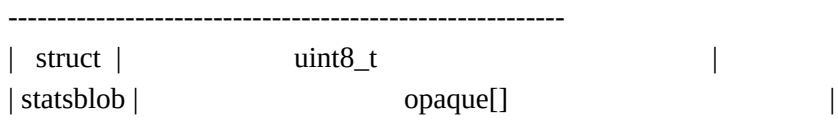
The **stats** framework facilitates real-time kernel and user space statistics gathering. The framework is built around the "statsblob", an object embedded within a contiguous memory allocation that is mostly opaque to consumers and stores all required state. A "statsblob" object can itself be embedded within other objects either directly or indirectly using a pointer.

Objects or subsystems for which statistics are to be gathered are initialized from a template "statsblob", which acts as the blueprint for an arbitrary set of Variables Of Interest (VOIs) and their associated statistics. Each template defines a schema plus associated metadata, which are kept separate to minimize the memory footprint of blobs.

Data gathering hook functions added at appropriate locations within the code base of interest feed VOI data into the framework for processing.

Each "statsblob", consists of a *struct statsblob* header and opaque internal blob structure per the

following diagram:



The publicly visible 8-byte header is defined as:

```
struct statsblob {
        uint8_t         abi;
        uint8_t         endian;
        uint16_t        flags;
        uint16_t        maxsz;
        uint16_t        cursz;
        uint8_t         opaque[];
};
```

abi specifies which API version the blob's *opaque* internals conform to (STATS_ABI_V1 is the only version currently defined). *endian* specifies the endianness of the blob's fields (SB_LE for little endian, SB_BE for big endian, or SB_UE for unknown endianness). *cursz* specifies the size of the blob, while *maxsz* specifies the size of the underlying memory allocation in which the blob is embedded. Both *cursz* and *maxsz* default to units of bytes, unless a flag is set in *flags* that dictates otherwise.

Templates are constructed by associating arbitrary VOI IDs with a set of statistics, where each statistic is specified using a *struct voistatspec* per the definition below:

```
struct voistatspec {
        vss_hlpr_fn                     hlpr;
        struct vss_hlpr_info*hlprinfo;
        struct voistatdata      *iv;
        size_t                          vsdsz;
        uint32_t        flags;
        enum vsd_dtype                  vs_dtype : 8;
        enum voi_stype                  stype : 8;
};
```

It is generally expected that consumers will not work with *struct voistatspec* directly, and instead use the **STATS_VSS_***() helper macros.

The **stats** framework offers the following statistics for association with VOIs:

VS_STYPE_SUM The sum of VOI values.

VS_STYPE_MAX The maximum VOI value.

VS_STYPE_MIN The minimum VOI value.

VS_STYPE_HIST A static bucket histogram of VOI values, including a count of "out-of-band/bucket" values which did not match any bucket. Histograms can be specified as "*C*ontinuous *R*ange" (CRHIST), "*D*iscrete *R*ange" (DRHIST) or "*D*iscrete *V*alue" (DVHIST), with 32 or 64 bit bucket counters, depending on the VOI semantics.

VS_STYPE_TDGST A dynamic bucket histogram of VOI values based on the t-digest method (refer to the t-digest paper in the *SEE ALSO* section below).

A "visitor software design pattern"-like scheme is employed to facilitate iterating over a blob's data without concern for the blob's structure. The data provided to visitor callback functions is encapsulated in *struct sb_visit* per the definition below:

```
struct sb_visit {
        struct voistatdata      *vs_data;
        uint32_t                tplhash;
        uint32_t                flags;
        int16_t                         voi_id;
        int16_t                         vs_dsz;
        enum vsd_dtype                  voi_dtype : 8;
        enum vsd_dtype                  vs_dtype : 8;
        int8_t                          vs_stype;
        uint16_t                vs_errs;
};
```

The **stats_tpl_sample_rates**() and **stats_tpl_sample_rollthedice**() functions utilize *struct stats_tpl_sample_rate* to encapsulate per-template sample rate information per the definition below:

```
struct stats_tpl_sample_rate {
        int32_t         tpl_slot_id;
        uint32_t  tpl_sample_pct;
};
```

The *tpl_slot_id* member holds the template's slot ID obtained from **stats_tpl_alloc**() or **stats_tpl_fetch_allocid**(). The *tpl_sample_pct* member holds the template's sample rate as an integer percentage in the range [0,100].

The *stats_tpl_sr_cb_t* conformant function pointer that is required as the *arg1* of **stats_tpl_sample_rates**() is defined as:

```
enum stats_tpl_sr_cb_action {
        TPL_SR_UNLOCKED_GET,
        TPL_SR_RLOCKED_GET,
        TPL_SR_RUNLOCK,
        TPL_SR_PUT
};
typedef int (*stats_tpl_sr_cb_t)(enum stats_tpl_sr_cb_action action,
    struct stats_tpl_sample_rate **rates, int *nrates, void *ctx);
```

It is required that a conformant function:

- Return an appropriate errno(2) on error, otherwise 0.

- When called with "action == TPL_SR_*_GET", return the subsystem's rates list ptr and count, locked or unlocked as requested.

- When called with "action == TPL_SR_RUNLOCK", unlock the subsystem's rates list ptr and count. Pair with a prior "action == TPL_SR_RLOCKED_GET" call.

- When called with "action == TPL_SR_PUT", update the subsystem's rates list ptr and count to the sysctl processed values and return the inactive list details in *rates* and *nrates* for garbage collection by **stats_tpl_sample_rates**().

Where templates need to be referenced via textual means, for example via a MIB variable, the following string based template spec formats can be used:

1. "<tplname>":<tplhash>, for example "TCP_DEFAULT":1731235399

2. "<tplname>", for example "TCP_DEFAULT"

3. :<tplhash>, for example :1731235399

The first form is the normative spec format generated by the framework, while the second and third

forms are convenience formats primarily for user input. The use of inverted commas around the template name is optional.

MIB Variables

The in-kernel **stats** framework exposes the following framework-specific variables in the *kern.stats* branch of the sysctl(3) MIB.

templates Read-only CSV list of registered templates in normative template spec form.

Template Management Functions

The **stats_tpl_alloc**() function allocates a new template with the specified unique name and returns its runtime-stable template slot ID for use with other API functions. The *flags* argument is currently unused.

The **stats_tpl_fetch_allocid**() function returns the runtime-stable template slot ID of any registered template matching the specified name and hash.

The **stats_tpl_fetch**() function returns the pointer to the registered template object at the specified template slot ID.

The **stats_tpl_id2name**() function returns the name of the registered template object at the specified template slot ID.

The **stats_tpl_sample_rates**() function provides a generic handler for template sample rates management and reporting via sysctl(3) MIB variables. Subsystems can use this function to create a subsystem-specific SYSCTL_PROC(9) MIB variable that manages and reports subsystem-specific template sampling rates. Subsystems must supply a *stats_tpl_sr_cb_t* conformant function pointer as the sysctl's *arg1*, which is a callback used to interact with the subsystem's stats template sample rates list. Subsystems can optionally specify the sysctl's *arg2* as non-zero, which causes a zero-initialized allocation of arg2-sized contextual memory to be heap-allocated and passed in to all subsystem callbacks made during the operation of **stats_tpl_sample_rates**().

The **stats_tpl_sample_rollthedice**() function makes a weighted random template selection from the supplied array of template sampling rates. The cumulative percentage of all sampling rates should not exceed 100. If no seed is supplied, a PRNG is used to generate a true random number so that every selection is independent. If a seed is supplied, selection will be made randomly across different seeds, but deterministically given the same seed.

The **stats_tpl_add_voistats**() function is used to add a VOI and associated set of statistics to the registered template object at the specified template slot ID. The set of statistics is passed as an array of

struct voistatspec which can be initialized using the **STATS_VSS_***() helper macros or manually for non-standard use cases. For static *vss* arrays, the *nvss* count of array elements can be determined by passing *vss* to the **NVSS**() macro. The SB_VOI_RELUPDATE flag can be passed to configure the VOI for use with **stats_voi_update_rel_<dtype>**(), which entails maintaining an extra 8 bytes of state in the blob at each update.

Data Gathering Functions

The **stats_voi_update_abs_<dtype>**() and **stats_voi_update_rel_<dtype>**() functions both update all the statistics associated with the VOI identified by *voi_id*. The "abs" call uses *voival* as an absolute value, whereas the "rel" call uses *voival* as a value relative to that of the previous update function call, by adding it to the previous value and using the result for the update. Relative updates are only possible for VOIs that were added to the template with the SB_VOI_RELUPDATE flag specified to **stats_tpl_add_voistats**().

Utility Functions

The **stats_blob_alloc**() function allocates and initializes a new blob based on the registered template object at the specified template slot ID.

The **stats_blob_init**() function initializes a new blob in an existing memory allocation based on the registered template object at the specified template slot ID.

The **stats_blob_clone**() function duplicates the *src* blob into *dst*, leaving only the *maxsz* field of *dst* untouched. The SB_CLONE_ALLOCDST flag can be passed to instruct the function to allocate a new blob of appropriate size into which to clone *src*, storing the new pointer in **dst*. The SB_CLONE_USRDSTNOFAULT or SB_CLONE_USRDST flags can be set to respectively signal that copyout_nofault(9) or copyout(9) should be used because **dst* is a user space address.

The **stats_blob_snapshot**() function calls **stats_blob_clone**() to obtain a copy of *src* and then performs any additional functions required to produce a coherent blob snapshot. The flags interpreted by **stats_blob_clone**() also apply to **stats_blob_snapshot**(). Additionally, the SB_CLONE_RSTSRC flag can be used to effect a reset of the *src* blob's statistics after a snapshot is successfully taken.

The **stats_blob_destroy**() function destroys a blob previously created with **stats_blob_alloc**(), **stats_blob_clone**() or **stats_blob_snapshot**().

The **stats_blob_visit**() function allows the caller to iterate over the contents of a blob. The callback function *func* is called for every VOI and statistic in the blob, passing a *struct sb_visit* and the user context argument *usrctx* to the callback function. The *sbv* passed to the callback function may have one or more of the following flags set in the *flags* struct member to provide useful metadata about the iteration: SB_IT_FIRST_CB, SB_IT_LAST_CB, SB_IT_FIRST_VOI, SB_IT_LAST_VOI,

SB_IT_FIRST_VOISTAT, SB_IT_LAST_VOISTAT, SB_IT_NULLVOI and SB_IT_NULLVOISTAT. Returning a non-zero value from the callback function terminates the iteration.

The **stats_blob_tostr**() renders a string representation of a blob into the sbuf(9) *buf*. Currently supported render formats are SB_STRFMT_FREEFORM and SB_STRFMT_JSON. The SB_TOSTR_OBJDUMP flag can be passed to render version specific opaque implementation detail for debugging or string-to-binary blob reconstruction purposes. The SB_TOSTR_META flag can be passed to render template metadata into the string representation, using the blob's template hash to lookup the corresponding template.

The **stats_voistatdata_tostr**() renders a string representation of an individual statistic's data into the sbuf(9) *buf*. The same render formats supported by the **stats_blob_tostr**() function can be specified, and the *objdump* boolean has the same meaning as the SB_TOSTR_OBJDUMP flag.

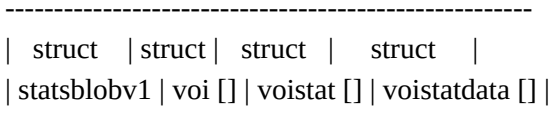
The **stats_voistat_fetch_dptr**() function returns an internal blob pointer to the specified *stype* statistic data for the VOI *voi_id*. The **stats_voistat_fetch_<dtype>**() functions are convenience wrappers around **stats_voistat_fetch_dptr**() to perform the extraction for simple data types.

IMPLEMENTATION NOTES

The following notes apply to STATS_ABI_V1 format statsblobs.

Space-Time Complexity

Blobs are laid out as three distinct memory regions following the header:



Blobs store VOI and statistic blob state (8 bytes for *struct voi* and 8 bytes for *struct voistat* respectively) in sparse arrays, using the *voi_id* and *enum voi_stype* as array indices. This allows O(1) access to any voi/voistat pair in the blob, at the expense of 8 bytes of wasted memory per vacant slot for templates which do not specify contiguously numbered VOIs and/or statistic types. Data storage for statistics is only allocated for non-vacant slot pairs.

To provide a concrete example, a blob with the following specification:

- Two VOIs; ID 0 and 2; added to the template in that order

- VOI 0 is of data type *int64_t*, is configured with SB_VOI_RELUPDATE to enable support for

relative updates using **stats_voi_update_rel_<dtype>**(), and has a VS_STYPE_MIN statistic associated with it.

- VOI 2 is of data type *uint32_t* with VS_STYPE_SUM and VS_STYPE_MAX statistics associated with it.

would have the following memory layout:

```
---------------------------------------
| header                              | struct statsblobv1, 32 bytes
|-------------------------------------|
| voi[0]                              | struct voi, 8 bytes
| voi[1] (vacant)                     | struct voi, 8 bytes
| voi[2]                              | struct voi, 8 bytes
|-------------------------------------|
| voi[2]voistat[VOISTATE] (vacant)    | struct voistat, 8 bytes
| voi[2]voistat[SUM]                  | struct voistat, 8 bytes
| voi[2]voistat[MAX]                  | struct voistat, 8 bytes
| voi[0]voistat[VOISTATE]             | struct voistat, 8 bytes
| voi[0]voistat[SUM] (vacant)         | struct voistat, 8 bytes
| voi[0]voistat[MAX] (vacant)         | struct voistat, 8 bytes
| voi[0]voistat[MIN]                  | struct voistat, 8 bytes
|-------------------------------------|
| voi[2]voistat[SUM]voistatdata       | struct voistatdata_int32, 4 bytes
| voi[2]voistat[MAX]voistatdata       | struct voistatdata_int32, 4 bytes
| voi[0]voistat[VOISTATE]voistatdata  | struct voistatdata_numeric, 8 bytes
| voi[0]voistat[MIN]voistatdata       | struct voistatdata_int64, 8 bytes
---------------------------------------
                                        TOTAL 136 bytes
```

When rendered to string format using **stats_blob_tostr**(), the SB_STRFMT_FREEFORM *fmt* and the SB_TOSTR_OBJDUMP flag, the rendered output is:

```
struct statsblobv1@0x8016250a0, abi=1, endian=1, maxsz=136, cursz=136, \
  created=6294158585626144, lastrst=6294158585626144, flags=0x0000, \
  stats_off=56, statsdata_off=112, tplhash=2994056564
    vois[0]: id=0, name="", flags=0x0001, dtype=INT_S64, voistatmaxid=3, \
      stats_off=80
        vois[0]stat[0]: stype=VOISTATE, flags=0x0000, dtype=VOISTATE, \
          dsz=8, data_off=120
```

```
        voistatdata: prev=0
      vois[0]stat[1]: stype=-1
      vois[0]stat[2]: stype=-1
      vois[0]stat[3]: stype=MIN, flags=0x0000, dtype=INT_S64, \
       dsz=8, data_off=128
        voistatdata: 9223372036854775807
    vois[1]: id=-1
    vois[2]: id=2, name="", flags=0x0000, dtype=INT_U32, voistatmaxid=2, \
     stats_off=56
      vois[2]stat[0]: stype=-1
      vois[2]stat[1]: stype=SUM, flags=0x0000, dtype=INT_U32, dsz=4, \
       data_off=112
        voistatdata: 0
      vois[2]stat[2]: stype=MAX, flags=0x0000, dtype=INT_U32, dsz=4, \
       data_off=116
        voistatdata: 0
```

Note: The "\" present in the rendered output above indicates a manual line break inserted to keep the man page within 80 columns and is not part of the actual output.

Locking

The **stats** framework does not provide any concurrency protection at the individual blob level, instead requiring that consumers guarantee mutual exclusion when calling API functions that reference a non-template blob.

The list of templates is protected with a rwlock(9) in-kernel, and pthread(3) rw lock in user space to support concurrency between template management and blob initialization operations.

RETURN VALUES

stats_tpl_alloc() returns a runtime-stable template slot ID on success, or a negative errno on failure. -EINVAL is returned if any problems are detected with the arguments. -EEXIST is returned if an existing template is registered with the same name. -ENOMEM is returned if a required memory allocation fails.

stats_tpl_fetch_allocid() returns a runtime-stable template slot ID, or negative errno on failure. -ESRCH is returned if no registered template matches the specified name and/or hash.

stats_tpl_fetch() returns 0 on success, or ENOENT if an invalid *tpl_id* is specified.

stats_tpl_id2name() returns 0 on success, or an errno on failure. EOVERFLOW is returned if the length

of *buf* specified by *len* is too short to hold the template's name. ENOENT is returned if an invalid *tpl_id* is specified.

stats_tpl_sample_rollthedice() returns a valid template slot id selected from *rates* or -1 if a NULL selection was made, that is no stats collection this roll.

stats_tpl_add_voistats() return 0 on success, or an errno on failure. EINVAL is returned if any problems are detected with the arguments. EFBIG is returned if the resulting blob would have exceeded the maximum size. EOPNOTSUPP is returned if an attempt is made to add more VOI stats to a previously configured VOI. ENOMEM is returned if a required memory allocation fails.

stats_voi_update_abs_<dtype>() and **stats_voi_update_rel_<dtype>**() return 0 on success, or EINVAL if any problems are detected with the arguments.

stats_blob_init() returns 0 on success, or an errno on failure. EINVAL is returned if any problems are detected with the arguments. EOVERFLOW is returned if the template blob's *cursz* is larger than the *maxsz* of the blob being initialized.

stats_blob_alloc() returns a pointer to a newly allocated and initialized blob based on the specified template with slot ID *tpl_id*, or NULL if the memory allocation failed.

stats_blob_clone() and **stats_blob_snapshot**() return 0 on success, or an errno on failure. EINVAL is returned if any problems are detected with the arguments. ENOMEM is returned if the SB_CLONE_ALLOCDST flag was specified and the memory allocation for *dst* fails. EOVERFLOW is returned if the src blob's *cursz* is larger than the *maxsz* of the *dst* blob.

stats_blob_visit() returns 0 on success, or EINVAL if any problems are detected with the arguments.

stats_blob_tostr() and **stats_voistatdata_tostr**() return 0 on success, or an errno on failure. EINVAL is returned if any problems are detected with the arguments, otherwise any error returned by **sbuf_error**() for *buf* is returned.

stats_voistat_fetch_dptr() returns 0 on success, or EINVAL if any problems are detected with the arguments.

stats_voistat_fetch_<dtype>() returns 0 on success, or an errno on failure. EINVAL is returned if any problems are detected with the arguments. EFTYPE is returned if the requested data type does not match the blob's data type for the specified *voi_id* and *stype*.

SEE ALSO

errno(2), arb(3), qmath(3), tcp(4), sbuf(9)

Ted Dunning and Otmar Ertl, *Computing Extremely Accurate Quantiles Using t-digests*, https://github.com/tdunning/t-digest/raw/master/docs/t-digest-paper/histo.pdf.

HISTORY

The **stats** framework first appeared in FreeBSD 13.0.

AUTHORS

The **stats** framework and this manual page were written by Lawrence Stewart <lstewart@FreeBSD.org> and sponsored by Netflix, Inc.

CAVEATS

Granularity of timing-dependent network statistics, in particular TCP_RTT, depends on the HZ timer. To minimize the measurement error avoid using HZ lower than 1000.