

powerd++

0.3.0

Generated by Doxygen 1.8.13

## Contents

<b>1</b>	<b>Main Page</b>	<b>2</b>
<b>2</b>	<b>LICENSE</b>	<b>4</b>
<b>3</b>	<b>loadplay(1)</b>	<b>4</b>
<b>4</b>	<b>loadrec(1)</b>	<b>7</b>
<b>5</b>	<b>powerd++(8)</b>	<b>9</b>
<b>6</b>	<b>Namespace Index</b>	<b>13</b>
6.1	Namespace List . . . . .	13
<b>7</b>	<b>Hierarchical Index</b>	<b>14</b>
7.1	Class Hierarchy . . . . .	14
<b>8</b>	<b>Class Index</b>	<b>16</b>
8.1	Class List . . . . .	16
<b>9</b>	<b>File Index</b>	<b>17</b>
9.1	File List . . . . .	17
<b>10</b>	<b>Namespace Documentation</b>	<b>18</b>
10.1	anonymous_namespace{clas.cpp} Namespace Reference . . . . .	18
10.1.1	Detailed Description . . . . .	18
10.1.2	Enumeration Type Documentation . . . . .	18
10.1.3	Function Documentation . . . . .	19
10.1.4	Variable Documentation . . . . .	19
10.2	anonymous_namespace{loadplay.cpp} Namespace Reference . . . . .	19
10.2.1	Detailed Description . . . . .	20
10.2.2	Function Documentation . . . . .	20
10.2.3	Variable Documentation . . . . .	22
10.3	anonymous_namespace{loadrec.cpp} Namespace Reference . . . . .	22
10.3.1	Detailed Description . . . . .	23

10.3.2	Enumeration Type Documentation	23
10.3.3	Function Documentation	24
10.3.4	Variable Documentation	24
10.4	anonymous_namespace{powerd++.cpp} Namespace Reference	25
10.4.1	Detailed Description	26
10.4.2	Class Documentation	26
10.4.3	Enumeration Type Documentation	29
10.4.4	Function Documentation	30
10.4.5	Variable Documentation	33
10.5	clas Namespace Reference	33
10.5.1	Detailed Description	34
10.5.2	Function Documentation	34
10.6	constants Namespace Reference	37
10.6.1	Detailed Description	37
10.6.2	Variable Documentation	37
10.7	errors Namespace Reference	38
10.7.1	Detailed Description	38
10.7.2	Class Documentation	39
10.7.3	Enumeration Type Documentation	39
10.7.4	Function Documentation	40
10.7.5	Variable Documentation	40
10.8	fixme Namespace Reference	41
10.8.1	Detailed Description	41
10.8.2	Function Documentation	41
10.9	nih Namespace Reference	42
10.9.1	Detailed Description	42
10.9.2	Class Documentation	42
10.9.3	Function Documentation	43
10.10	sys Namespace Reference	44
10.10.1	Detailed Description	44

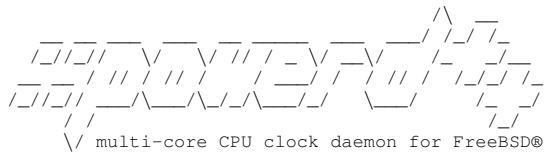
10.11sys::ctl Namespace Reference . . . . .	44
10.11.1 Detailed Description . . . . .	45
10.11.2 Class Documentation . . . . .	45
10.11.3 Typedef Documentation . . . . .	45
10.11.4 Function Documentation . . . . .	46
10.12sys::pid Namespace Reference . . . . .	49
10.12.1 Detailed Description . . . . .	49
10.12.2 Class Documentation . . . . .	49
10.13sys::sig Namespace Reference . . . . .	49
10.13.1 Detailed Description . . . . .	49
10.13.2 Class Documentation . . . . .	50
10.14timing Namespace Reference . . . . .	50
10.14.1 Detailed Description . . . . .	50
10.15types Namespace Reference . . . . .	50
10.15.1 Detailed Description . . . . .	50
10.15.2 Typedef Documentation . . . . .	50
10.16utility Namespace Reference . . . . .	51
10.16.1 Detailed Description . . . . .	51
10.16.2 Function Documentation . . . . .	51
10.17utility::literals Namespace Reference . . . . .	53
10.17.1 Detailed Description . . . . .	53
10.17.2 Function Documentation . . . . .	53

<b>11 Class Documentation</b>	<b>54</b>
11.1 anonymous_namespace{loadplay.cpp}::Callback< FunctionArgs > Class Template Reference . . .	54
11.1.1 Detailed Description . . . . .	55
11.1.2 Constructor & Destructor Documentation . . . . .	55
11.1.3 Member Function Documentation . . . . .	57
11.2 timing::Cycle Class Reference . . . . .	57
11.2.1 Detailed Description . . . . .	58
11.2.2 Member Function Documentation . . . . .	58
11.3 anonymous_namespace{loadplay.cpp}::Emulator Class Reference . . . . .	59
11.3.1 Detailed Description . . . . .	60
11.3.2 Constructor & Destructor Documentation . . . . .	60
11.3.3 Member Function Documentation . . . . .	61
11.3.4 Member Data Documentation . . . . .	61
11.4 nih::enum_has_members< Enum, class > Struct Template Reference . . . . .	62
11.4.1 Detailed Description . . . . .	62
11.5 utility::Formatter< BufSize > Class Template Reference . . . . .	63
11.5.1 Detailed Description . . . . .	63
11.5.2 Member Function Documentation . . . . .	64
11.6 anonymous_namespace{powerd++.cpp}::FreqGuard Class Reference . . . . .	64
11.6.1 Detailed Description . . . . .	65
11.7 anonymous_namespace{loadplay.cpp}::Hold< T > Class Template Reference . . . . .	66
11.7.1 Detailed Description . . . . .	66
11.7.2 Constructor & Destructor Documentation . . . . .	66
11.7.3 Member Data Documentation . . . . .	67
11.8 anonymous_namespace{loadplay.cpp}::Main Class Reference . . . . .	67
11.8.1 Detailed Description . . . . .	68
11.8.2 Constructor & Destructor Documentation . . . . .	68
11.9 anonymous_namespace{loadplay.cpp}::mib_t Struct Reference . . . . .	68
11.9.1 Detailed Description . . . . .	69
11.9.2 Constructor & Destructor Documentation . . . . .	69

11.9.3 Member Function Documentation . . . . .	69
11.10sys::ctl::Once< T, SysctlT > Class Template Reference . . . . .	71
11.10.1 Detailed Description . . . . .	72
11.10.2 Constructor & Destructor Documentation . . . . .	72
11.10.3 Member Function Documentation . . . . .	72
11.11nih::Options< Enum, DefCount > Class Template Reference . . . . .	73
11.11.1 Detailed Description . . . . .	74
11.11.2 Constructor & Destructor Documentation . . . . .	74
11.11.3 Member Function Documentation . . . . .	75
11.11.4 Member Data Documentation . . . . .	78
11.12sys::pid::Pidfile Class Reference . . . . .	78
11.12.1 Detailed Description . . . . .	79
11.12.2 Constructor & Destructor Documentation . . . . .	79
11.12.3 Member Function Documentation . . . . .	79
11.12.4 Member Data Documentation . . . . .	80
11.13sys::sc_error< Domain > Struct Template Reference . . . . .	80
11.13.1 Detailed Description . . . . .	80
11.13.2 Member Function Documentation . . . . .	81
11.14sys::sig::Signal Class Reference . . . . .	81
11.14.1 Detailed Description . . . . .	82
11.14.2 Constructor & Destructor Documentation . . . . .	82
11.15sys::ctl::Sync< T, SysctlT > Class Template Reference . . . . .	82
11.15.1 Detailed Description . . . . .	83
11.15.2 Constructor & Destructor Documentation . . . . .	83
11.15.3 Member Function Documentation . . . . .	84
11.16sys::ctl::Sysctl< MibDepth > Class Template Reference . . . . .	84
11.16.1 Detailed Description . . . . .	85
11.16.2 Constructor & Destructor Documentation . . . . .	86
11.16.3 Member Function Documentation . . . . .	86
11.17sys::ctl::Sysctl< 0 > Class Template Reference . . . . .	88
11.17.1 Detailed Description . . . . .	89
11.17.2 Constructor & Destructor Documentation . . . . .	89
11.17.3 Member Function Documentation . . . . .	90
11.18anonymous_namespace{loadplay.cpp}::Sysctls Class Reference . . . . .	92
11.18.1 Detailed Description . . . . .	93
11.18.2 Member Function Documentation . . . . .	93
11.18.3 Member Data Documentation . . . . .	94
11.19anonymous_namespace{loadplay.cpp}::SysctlValue Class Reference . . . . .	95
11.19.1 Detailed Description . . . . .	96
11.19.2 Constructor & Destructor Documentation . . . . .	97
11.19.3 Member Function Documentation . . . . .	97
11.19.4 Member Data Documentation . . . . .	102

<b>12 File Documentation</b>	<b>103</b>
12.1 clas.hpp File Reference	103
12.1.1 Detailed Description	104
12.2 constants.hpp File Reference	104
12.2.1 Detailed Description	106
12.3 Cycle.hpp File Reference	106
12.3.1 Detailed Description	107
12.4 errors.hpp File Reference	107
12.4.1 Detailed Description	108
12.4.2 Class Documentation	108
12.5 fixme.hpp File Reference	109
12.5.1 Detailed Description	110
12.6 loadplay.cpp File Reference	110
12.6.1 Detailed Description	112
12.6.2 Function Documentation	112
12.7 loadrec.cpp File Reference	115
12.7.1 Detailed Description	117
12.7.2 Function Documentation	117
12.8 Options.hpp File Reference	118
12.8.1 Detailed Description	119
12.8.2 Class Documentation	120
12.9 power++.cpp File Reference	121
12.9.1 Detailed Description	123
12.9.2 Class Documentation	123
12.9.3 Function Documentation	126
12.10sys/error.hpp File Reference	127
12.10.1 Detailed Description	128
12.11sys/pidfile.hpp File Reference	128
12.11.1 Detailed Description	129
12.11.2 Class Documentation	129
12.12sys/signal.hpp File Reference	129
12.12.1 Detailed Description	130
12.12.2 Class Documentation	130
12.13sys/sysctl.hpp File Reference	131
12.13.1 Detailed Description	132
12.13.2 Class Documentation	132
12.14types.hpp File Reference	133
12.14.1 Detailed Description	134
12.15utility.hpp File Reference	134
12.15.1 Detailed Description	136

## 1 Main Page



The `powerd++` daemon is a drop-in replacement for FreeBSD's native `powerd(8)`. It monitors CPU load and core temperatures to adjust the CPU clock, avoiding some of the pitfalls of `powerd`.

### What Pitfalls?

At the time `powerd++` was first created (February 2016), `powerd` exhibited some unhealthy behaviours on multi-core machines.

In order to make sure that single core loads do not suffer from the use of `powerd` it was designed to use the sum load of all cores as the current load rating. A side effect of this is that it causes `powerd` to never clock down on systems with even moderate numbers of cores. E.g. on a quad-core system with hyper threading a background load of 12.5% per core suffices to score a 100% load rating.

The more cores are added, the worse it gets. Even on a dual core machine (with HT) having a browser and an e-mail client open, suffices to keep the load rating above 100% for most of the time, even without user activity. Thus `powerd` never does its job of saving energy by reducing the clock frequency.

### Advantages of `powerd++`

The `powerd++` implementation addresses this issue and more:

- `powerd++` groups cores with a common clock frequency together and handles each group's load and target frequency separately. I.e. the moment FreeBSD starts offering individual clock settings on the CPU, core or thread level, `powerd++` already supports it.
- `powerd++` takes the highest load within a group of cores to rate the load. This approach responds well to single core loads as well as evenly distributed loads.
- `powerd++` sets the clock frequency according to a load target, i.e. it jumps right to the clock rate it will stay in if the load does not change.
- `powerd++` supports taking the average load over more than two samples, this makes it more robust against small load spikes, but sacrifices less responsiveness than just increasing the polling interval would. Because only the oldest and the newest sample are required for calculating the average, this approach does not even cause additional runtime cost!
- `powerd++` parses command line arguments as floating point numbers, allowing expressive commands like `powerd++ --batt 1.2ghz`.
- `powerd++` supports temperature based throttling.



## Building

Download the repository and run make:

```
> make
c++ -O2 -pipe -std=c++14 -Wall -Werror -pedantic -c src/powerd++.cpp -o powerd++.o
c++ -O2 -pipe -std=c++14 -Wall -Werror -pedantic -c src/clas.cpp -o clas.o
c++ -O2 -pipe -std=c++14 -Wall -Werror -pedantic powerd++.o clas.o -lutil -o powerd++
c++ -O2 -pipe -std=c++14 -Wall -Werror -pedantic -c src/loadrec.cpp -o loadrec.o
c++ -O2 -pipe -std=c++14 -Wall -Werror -pedantic loadrec.o clas.o -o loadrec
c++ -O2 -pipe -std=c++14 -Wall -Werror -pedantic -fPIC -c src/loadplay.cpp -o loadplay.o
c++ -O2 -pipe -std=c++14 -Wall -Werror -pedantic loadplay.o -lpthread -shared -o libloadplay.so
```

## Documentation

The manual pages can be read with the following commands:

```
> man ./powerd++.8 ./loadrec.1 ./loadplay.1
```

## Tooling

In addition to the `powerd++` daemon this repository also comes with the tools `loadrec` and `loadplay`. They can be used to record loads and test both `powerd` and `powerd++` under reproducible load conditions.

This is great for tuning, testing, bug reports and creating fancy plots.

## FAQ

- **Why C++?** The `powerd++` code is not object oriented, but it uses some *C++* and *\*C++11* features to avoid common pitfalls of writing *C* code. E.g. there is a small *RAII* wrapper around the `pidfile` facilities (`pidfile_open()`, `pidfile_write()`, `pidfile_remove()`), turning the use of `pidfiles` into a fire and forget affair. Templated wrappers around calls like `sysctl()` use array references to infer buffer sizes at compile time, taking the burden of safely passing these buffer sizes on to the command away from the programmer. The `std::unique_ptr<>` template obsoletes memory cleanup code, providing the liberty of using exceptions without worrying about memory leaks.
- **Why does `powerd++` show a high load when `top` shows a high idle time?** By default `top` shows the load percentage over all cores/threads, `powerd++` uses the load of a single core/thread (the one with the highest load). This keeps `powerd++` from starving single threaded processes, because they only have a small impact on overall load. An effect that increases with the number of cores/threads. E.g. 80% load on a quad core CPU with hyper threading only has an overall load impact of 10%. Use `top -P` to monitor idle times per core/thread.

## LICENSE

For those who care about this stuff, this project is available under the [ISC license](#).

## 2 LICENSE

Copyright © 2016 Dominic Fandrey [kami@freebsd.org](mailto:kami@freebsd.org)

Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

## 3 loadplay(1)

Manual page for `loadplay(1)`.

loadplay(1) FreeBSD General Commands Manual loadplay(1)

### NAME

**loadplay** --- CPU load player

### SYNOPSIS

```
loadplay -h
loadplay [ -i file ] [ -o file ] command ...
```

### DESCRIPTION

The **loadplay** command replays a load recording created with `loadrec(1)`. The *command* can either be `powerd(8)` or `powerd++(8)`, compatibility with other tools has not been tested.

### OPTIONS

The following options are supported:

```
-h, --help
    Show usage and exit.

-i, --input file
    Read load recording from file instead of stdin.

-o, --output file
    Output statistics to file instead of stdout.
```

### USAGE NOTES

The **loadplay** command is a shell script that injects `libloadplay.so` into *command*. This library simulates the load from the input and outputs load statistics. If *command* generates output on *stdout*, it will be mixed into the load statistics. So `powerd(8)` should be run without the `-v` flag and `powerd++(8)` without the `-f` flag.

**OUTPUT**

The first line of output contains column headings, columns are separated by a single space.

The Following columns are present, columns containing **d** occur for each core simulated:

**time[s]**

The simulation progress in seconds.

**cpu.d.freq[MHz]**

The current clock speed rating.

**cpu.d.reload**

The load as recorded, a value in the range [0, 1.0].

**cpu.d.load**

The load under the current clock speed. This value may go above 1.0, because the current speed rating may be too low to consume the load. In this case the unconsumed load (i.e. everything above 1.0) spills over to the next simulation frame.

**max(freqs) [MHz]**

The highest clock speed in the output line.

**sum(reloads)**

The sum of recorded loads in in the output line, a value in the range [0, ncpu].

**max(reloads)**

The highest recorded load in this output line, a value in the range [0, 1.0].

**sum(loads)**

The sum of loads according to the current clock speed in the output line.

**max(loads)**

The highest load according to the current clock speed in the output line.

**SAMPLING**

There is one sample for each recorded line. The duration of each frame depends on the recording, which defaults to 25 ms. At this sample rate loads are dominated by noise, so a gliding average should be applied to any load columns for further use, such as plotting. Note that the **max()** columns must be recreated in this case.

**IMPLEMENTATION**

The injected *libloadplay.so* works by intercepting system function calls and substituting the host environment with the recording. To achieve this the following function calls are intercepted:

- `sysctl(3)`, `sysctlnametomib(3)`, `sysctlbyname(3)`
- `daemon(3)`
- `geteuid(2)`
- `pidfile_open(3)`, `pidfile_write`, `pidfile_close(3)`, `pidfile_remove(3)`, `pidfile_fileno(3)`

## INITIALISATION

The **sysctl** family of functions is backed by a table that is initialised from the header of the load recording. If the heading is incomplete the setup routines print a message on `stderr`. All the following intercepted function calls will return failure, ensuring that the hijacked process is unable to operate and terminates.

Like `powerd++(8)` and `loadrec(1)` **loadplay** is core agnostic. Meaning that any core may have a **.freq** and **.freq\_levels** `sysctl` handle. Due to this flexibility load recordings may in part or wholly be fabricated to test artificial loads or systems and features that do not yet exist. E.g. it is possible to offer a **.freq** handle for each core or fabricate new **.freq\_levels**.

## SIMULATION

If setup succeeds a simulation thread is started that reads the remaining input lines, simulates the load and updates the **kern.cp\_times** entry in the thread safe `sysctl` table. For each frame a line of output with load statistics is produced.

Interaction with the hijacked process happens solely through the `sysctl` table. The simulation reads the recorded loads and the current core frequencies to update **kern.cp\_times**. The hijacked process reads this data and adjusts the clock frequencies, which in turn affects the next frame.

## FINALISATION

After reading the last line of input the simulation thread sends a **SIGINT** to the process to cause it to terminate.

## FILES

`%PREFIX%/lib/libloadplay.so` A library injected into `command` via the `LD_PRELOAD` environment variable.

## SEE ALSO

`loadrec(1)`, `powerd(8)`, `powerd++(8)`, `rtld(1)`, `signal(3)`

## AUTHORS

Implementation and manual by Dominic Fandrey [kami@freebsd.org](mailto:kami@freebsd.org)

## 4 loadrec(1)

Manual page for loadrec(1).

loadrec(1) FreeBSD General Commands Manual loadrec(1)

### NAME

**loadrec** --- CPU load recorder

### SYNOPSIS

```
loadrec -h  
loadrec [ -v] [ -d ival] [ -p ival] [ -o file]
```

### DESCRIPTION

The **loadrec** command performs a recording of the current load. The purpose is to reproduce this load to test different powerd(8) and powerd++(8) configurations under identical load conditions using loadplay(1).

### ARGUMENTS

The following argument types can be given:

*ival* A time interval can be given in seconds or milliseconds.  
s, ms  
An interval without a unit is treated as milliseconds.

*file* A file name.

### OPTIONS

The following options are supported:

**-h, --help**  
Show usage and exit.

**-v, --verbose**  
Be verbose and produce initial diagnostics on *stderr*.

**-d, --duration** *ival*  
The duration of the recording session, defaults to 30 seconds.

**-p, --poll** *ival*  
The polling interval to take load samples at, defaults to 25 milliseconds.

**-o, --output** *file*  
The output file to write the load to.

### USAGE NOTES

To create reproducible results stop any running CPU clock daemons like powerd(8) or powerd++(8). And set a fixed CPU frequency below the threshold at which the turbo mode is activated. E.g. an **Intel(R) Core(TM) i7-4500U CPU** supports the following frequency settings:

```
> sysctl dev.cpu.0.freq_levels
dev.cpu.0.freq_levels: 2401/15000 2400/15000 2300/14088 2200/13340 2000/11888 1900/11184 1800/10495 1700/9680
```

Supposedly the first mode, which is off by 1 MHz, invokes the turbo mode. However all modes down to 1800 MHz actually invoke the turbo mode for this model. The only way to determine this is by benchmarking the step-pings to find out that there is a huge performance step between 1700 and 1800 MHz and that all the modes above 1700 MHz show the exact same performance (given similar thermal conditions).

So in order to produce a usable measurement for this CPU the clock needs to be set to 1700 MHz or lower (higher is better to be able to record a wider range of loads):

```
# service powerd++ stop
Stopping powerdxx.
Waiting for PIDS: 63574.
# sysctl dev.cpu.0.freq=1700
dev.cpu.0.freq: 2401 -> 1700
```

Run **loadrec** for a brief time to test it:

```
> loadrec -d.25s
hw.machine=amd64
hw.model=Intel(R) Core(TM) i7-4500U CPU @ 1.80GHz
hw.ncpu=4
hw.acpi.acline=1
dev.cpu.0.freq=1700
dev.cpu.0.freq_levels=2401/15000 2400/15000 2300/14088 2200/13340 2000/11888 1900/11184 1800/10495 1700/9680 1
0 1658618 4728 563004 146277 7197047 1900009 5683 552503 10915 7100524 1849422 5838 528699 25364 7160311 18830
25 0 0 0 0 3 0 0 1 0 2 1 0 0 0 2 0 0 0 0 3
25 0 0 0 0 3 0 0 2 0 1 0 0 0 0 3 0 0 0 0 3
25 0 0 0 0 4 1 0 2 0 1 0 0 0 0 4 0 0 0 0 2
25 0 0 1 1 1 0 0 1 0 2 0 0 0 0 3 1 0 0 0 4
25 0 0 0 0 3 0 0 0 0 2 0 0 0 0 3 1 0 0 0 2
25 0 0 0 0 3 2 0 0 0 1 0 0 0 0 3 0 0 0 0 3
25 0 0 1 0 2 1 0 1 0 2 0 0 0 0 3 0 0 0 1 2
25 0 0 0 0 3 0 0 0 0 3 0 0 0 0 3 0 0 0 0 3
25 0 0 2 0 2 0 0 1 0 3 0 0 0 0 3 0 0 0 0 3
25 1 0 2 0 0 1 0 0 0 2 1 0 0 0 3 0 0 0 0 4
```

Printing the load creates significant load itself, so for the actual measurement the output should be written to a file. Create your workload and start your measurement:

```
> loadrec -o video-session.load
```

On the example setup **loadrec** produces a load of 0.001 (i.e. 0.1%), so its effect on the measurement is negligible.

#### SEE ALSO

cpufreq(4), loadplay(1), powerd(8), powerd++(8), sysctl(8)

#### AUTHORS

Implementation and manual by Dominic Fandrey [kami@freebsd.org](mailto:kami@freebsd.org)

## 5 powerd++(8)

Manual page for powerd++(8).

powerd++(8)

FreeBSD System Manager's Manual

powerd++(8)

### NAME

**powerd++** --- CPU clock speed daemon

### SYNOPSIS

```
powerd++ -h
powerd++ [ -vf] [ -a mode] [ -b mode] [ -n mode] [ -m freq] [ -M freq]
[ -F freq:freq] [ -A freq:freq] [ -B freq:freq] [ -H temp:temp]
[ -p ival] [ -s cnt] [ -P file]
```

### DESCRIPTION

The **powerd++** daemon monitors the system load and adjusts the CPU clock speed accordingly. It is a drop-in replacement for powerd(8) and supports two modes of operation, a load feedback control loop or fixed frequency operation.

### ARGUMENTS

The following argument types can be given:

*mode*     The mode is either a *load* target or a fixed *freq*. The powerd(8) modes are interpreted as follows:

- maximum, max  
          Use the highest clock frequency.
- minimum, min  
          Use the lowest clock frequency.
- adaptive, adp  
          A target load of 0.5 (50%).
- hiadaptive, hadp  
          A target load of 0.375 (37.5%).

If a scalar number is given, it is interpreted as a load.

*load*     A load is either a fraction in the range [0.0, 1.0] or a percentage in the range [0%, 100%].

*freq*     A clock frequency consists of a number and a frequency unit.  
          Hz, KHz, MHz, GHz, THz  
The unit is not case sensitive, if omitted MHz are assumed for compatibility with powerd(8).

*temp*     A temperature consisting of a number and a temperature unit. Supported units are:  
          C, K, F, R  
These units stand for deg. Celsius, Kelvin, deg. Fahrenheit and deg. Rankine. A value without a unit is treated as deg. Celsius.

*ival*     A time interval can be given in seconds or milliseconds.  
          s, ms  
          An interval without a unit is treated as milliseconds.

*cnt*     A positive integer.

*file*    A file name.

## OPTIONS

The following options are supported:

- h, --help**  
Show usage and exit
- v, --verbose**  
Be verbose and produce initial diagnostics on *stderr*.
- f, --foreground**  
Stay in foreground, produce an event log on *stdout*.
- a, --ac *mode***  
Mode to use while the AC power line is connected (default *hdp*).
- b, --batt *mode***  
Mode to use while battery powered (default *adp*).
- n, --unknown *mode***  
Mode to use while the power line state is unknown (default *hdp*).
- m, --min *freq***  
The lowest CPU clock frequency to use (default 0Hz).
- M, --max *freq***  
The highest CPU clock frequency to use (default 1THz).
- min-ac *freq***  
The lowest CPU clock frequency to use on AC power.
- max-ac *freq***  
The highest CPU clock frequency to use on AC power.
- min-batt *freq***  
The lowest CPU clock frequency to use on battery power.
- max-batt *freq***  
The highest CPU clock frequency to use on battery power.
- F, --freq-range *freq:freq***  
A pair of frequency values representing the minimum and maximum CPU clock frequency.



- A, --freq-range-ac** *freq:freq*  
A pair of frequency values representing the minimum and maximum CPU clock frequency on AC power.
- B, --freq-range-batt** *freq:freq*  
A pair of frequency values representing the minimum and maximum CPU clock frequency on battery power.
- H, --hitemp-range** *temp:temp*  
Set the high to critical temperature range, enables temperature based throttling.
- p, --poll** *ival*  
The polling interval that is used to take load samples and update the CPU clock (default 0.5s).
- s, --samples** *cnt*  
The number of load samples to use to calculate the current load. The default is 4.
- P, --pid** *file*  
Use an alternative pidfile, the default is `/var/run/powerd.pid`. The default ensures that `powerd(8)` and **powerd++** are not run simultaneously.
- i, -r** *load*  
Legacy arguments from `powerd(8)` not applicable to **powerd++** and thus ignored.

## SERVICE

The **powerd++** daemon can be run as an `rc(8)` service. Add the following line to `rc.conf(5)`:

```
powerdxx_enable="YES"
```

Command line arguments can be set via `powerdxx_flags`.

## TOOLS

The `loadrec(1)` and `loadplay(1)` tools offer the possibility to record system loads and replay them.

## IMPLEMENTATION NOTES

This section describes the operation of **powerd++**.

Both `powerd(8)` and **powerd++** have in common, that they work by polling `kern.cp_times` via `sysctl(3)`, which is an array of the accumulated loads of every core. By subtracting the last `cp_times` sample the loads over the polling interval can be determined. This information is used to set a new CPU clock frequency by updating `dev.cpu.0.freq`.

## Initialisation

After parsing command line arguments **powerd++** assigns a clock frequency controller to every core. I.e. cores are grouped by a common `dev.cpu.d.freq` handle that controls the clock for all of them. Due to limitations of `cpufreq(4)` `dev.cpu.0.freq` is the controlling handle for all cores, even across multiple CPUs. However **powerd++** is not built with that assumption and per CPU, core or thread controls will work as soon as the hardware and kernel support them.

In the next initialisation stage the available frequencies for every core group are determined to set appropriate lower and upper boundaries. This is a purely cosmetic measure and used to avoid unnecessary frequency updates. The controlling algorithm does not require this information, so failure to do so will only be reported (non-fatally) in verbose mode.

Unless the **-H** option is given, the initialisation checks for a critical temperature source. If one is found temperature throttling is implicitly turned on, causing throttling to start 10 deg. Celsius below the critical temperature.

So far the `sysctl(3) dev.cpu.d.coretemp.tjmax` is the only supported critical temperature source.

### Detaching From the Terminal

After the initialisation phase **powerd++** prepares to detach from the terminal. The first step is to acquire a lock on the pidfile. Afterwards all the frequencies are read and written as a last opportunity to fail. After detaching from the terminal the pidfile is written and the daemon goes into frequency controlling operation until killed by a signal.

### Load Control Loop

The original `powerd(8)` uses a hysteresis to control the CPU frequency. I.e. it determines the load over all cores since taking the last sample (the summary load during the last polling interval) and uses a lower and an upper load boundary to decide whether it should update the frequency or not.

**powerd++** has some core differences. It can take more than two samples (four by default), this makes it more robust against small spikes in load, while retaining much of its ability to quickly react to sudden surges in load. Changing the number of samples does not change the run-time cost of running **powerd++**.

Instead of taking the sum of all loads, the highest load within the core group is used to decide the next frequency target. Like with `powerd(8)` this means, that high load on a single core will cause an increase in the clock frequency. Unlike `powerd(8)` it also means that moderate load over all cores allows a decrease of the clock frequency.

The **powerd++** daemon steers the clock frequency to match a load target, e.g. if there was a 25% load on 2 GHz and the load target was 50%, the frequency would be set to 1 GHz.

### Temperature Based Throttling

If temperature based throttling is active and the temperature is above the high temperature boundary (the critical temperature minus 10 deg. Celsius by default), the core clock is limited to a value below the permitted maximum. The limit depends on the remaining distance to the critical temperature.

Thermal throttling ignores user-defined frequency limits, i.e. when using **-F**, **-B**, **-A** or **-m** to prevent the clock from going unreasonably low, sufficient thermal load may cause **powerd++** to select a clock frequency below the user provided minimum.

**Termination and Signals**

The signals HUP and TERM cause an orderly shutdown of **powerd++**. An orderly shutdown means the pidfile is removed and the clock frequencies are restored to their original values.

**FILES**

`/var/run/powerd.pid` Common pidfile with `powerd(8)`.  
`%PREFIX%/etc/rc.d/powerdxx` Service file, enable in `rc.conf(5)`.

**EXAMPLES**

Run in foreground, minimum clock frequency 800 MHz:  
`powerd++ -fm800`

Report configuration before detaching into the background:  
`powerd++ -v`

Target 75% load on battery power and run at 2.4 GHz on AC power:  
`powerd++ -b .75 -a 2.4ghz`

Target 25% load on AC power:  
`powerd++ -a 25%`

Use the same load sampling `powerd(8)` does:  
`powerd++ -s2 -p.25s`

Limit CPU clock frequencies to a range from 800 MHz to 1.8 GHz:  
`powerd++ -F800:1.8ghz`

**DIAGNOSTICS**

The **powerd++** daemon exits 0 on receiving an INT or TERM signal, and >0 if an error occurs.

**COMPATIBILITY**

So far **powerd++** requires ACPI to detect the current power line state.

**SEE ALSO**

`cpufreq(4)`, `powerd(8)`, `loadrec(1)`, `loadplay(1)`

**AUTHORS**

Implementation and manual by Dominic Fandrey [kami@freebsd.org](mailto:kami@freebsd.org)

FreeBSD 11.1

19 October, 2016

FreeBSD 11.1

## 6 Namespace Index

### 6.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

<a href="#">anonymous_namespace{clas.cpp}</a>	
File local scope	18
<a href="#">anonymous_namespace{loadplay.cpp}</a>	
File local scope	19
<a href="#">anonymous_namespace{loadrec.cpp}</a>	
File local scope	22
<a href="#">anonymous_namespace{powerd++.cpp}</a>	
File local scope	25
<a href="#">clas</a>	
A collection of functions to process command line arguments	33
<a href="#">constants</a>	
A collection of constants	37
<a href="#">errors</a>	
Common error handling types and functions	38
<a href="#">fixme</a>	
Workarounds for compiler/library bugs	41
<a href="#">nih</a>	
Not invented here namespace, for code that substitutes already commonly available functionality	42
<a href="#">sys</a>	
Wrappers around native system interfaces	44
<a href="#">sys::ctl</a>	
This namespace contains safer c++ wrappers for the <a href="#">sysctl()</a> interface	44
<a href="#">sys::pid</a>	
This namespace contains safer c++ wrappers for the <a href="#">pidfile_*</a> () interface	49
<a href="#">sys::sig</a>	
This namespace provides c++ wrappers for <a href="#">signal(3)</a>	49
<a href="#">timing</a>	
Namespace for time management related functionality	50
<a href="#">types</a>	
A collection of type aliases	50
<a href="#">utility</a>	
A collection of generally useful functions	51
<a href="#">utility::literals</a>	
Contains literals	53

## 7 Hierarchical Index

### 7.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

<code>anonymous_namespace{powerd++.cpp}::Global::ACSet</code>	25
<code>anonymous_namespace{loadplay.cpp}::Callback&lt; FunctionArgs &gt;</code>	54
<code>anonymous_namespace{loadplay.cpp}::Callback&lt; anonymous_namespace{loadplay.cpp}::SysctlValue &amp;&gt;</code>	54
<code>anonymous_namespace{powerd++.cpp}::Core</code>	25
<code>timing::Cycle</code>	57
<code>anonymous_namespace{loadplay.cpp}::Emulator</code>	59
<code>sys::pid::error</code>	49
<code>sys::sig::error</code>	49
<code>sys::ctl::error</code>	44
<code>errors::Exception</code>	38
<code>utility::Formatter&lt; BufSize &gt;</code>	63
<code>anonymous_namespace{powerd++.cpp}::FreqGuard</code>	64
<code>anonymous_namespace{powerd++.cpp}::Global</code>	25
<code>anonymous_namespace{loadplay.cpp}::Hold&lt; T &gt;</code>	66
<code>anonymous_namespace{loadplay.cpp}::Main</code>	67
<code>anonymous_namespace{loadplay.cpp}::mib_t</code>	68
<code>sys::ctl::Once&lt; T, SysctlT &gt;</code>	71
<code>sys::ctl::Once&lt; coreid_t, 2 &gt;</code>	71
<code>nih::Option&lt; Enum &gt;</code>	42
<code>nih::Options&lt; Enum, DefCount &gt;</code>	73
<code>sys::pid::Pidfile</code>	78
<code>sys::sc_error&lt; Domain &gt;</code>	80
<code>sys::sig::Signal</code>	81
<code>sys::ctl::Sync&lt; T, SysctlT &gt;</code>	82
<code>sys::ctl::Sync&lt; decikelvin_t &gt;</code>	82
<code>sys::ctl::Sync&lt; mhz_t &gt;</code>	82
<code>sys::ctl::Sysctl&lt; MibDepth &gt;</code>	84
<code>sys::ctl::Sysctl&lt; 0 &gt;</code>	88
<code>anonymous_namespace{loadplay.cpp}::Sysctls</code>	92
<code>anonymous_namespace{loadplay.cpp}::SysctlValue true_type</code>	95
<code>nih::enum_has_members&lt; Enum, class &gt;</code>	62

## 8 Class Index

### 8.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#"><code>anonymous_namespace{loadplay.cpp}::Callback&lt; FunctionArgs &gt;</code></a>	54
Implements a recursion safe <code>std::function</code> wrapper	
<a href="#"><code>timing::Cycle</code></a>	57
Implements an interruptible cyclic sleeping functor	
<a href="#"><code>anonymous_namespace{loadplay.cpp}::Emulator</code></a>	59
Instances of this class represent an emulator session	
<a href="#"><code>nih::enum_has_members&lt; Enum, class &gt;</code></a>	62
Tests whether the given enum provides all the required definitions	
<a href="#"><code>utility::Formatter&lt; BufSize &gt;</code></a>	63
A formatting wrapper around string literals	
<a href="#"><code>anonymous_namespace{powerd++.cpp}::FreqGuard</code></a>	64
A core frequency guard	
<a href="#"><code>anonymous_namespace{loadplay.cpp}::Hold&lt; T &gt;</code></a>	66
Sets a referenced variable to a given value and restores it when going out of context	
<a href="#"><code>anonymous_namespace{loadplay.cpp}::Main</code></a>	67
Singleton class representing the main execution environment	
<a href="#"><code>anonymous_namespace{loadplay.cpp}::mib_t</code></a>	68
Represents MIB, but wraps it to provide the necessary operators to use it as an <code>std::map</code> key	
<a href="#"><code>sys::ctl::Once&lt; T, SysctlT &gt;</code></a>	71
A read once representation of a <a href="#"><code>Sysctl</code></a>	
<a href="#"><code>nih::Options&lt; Enum, DefCount &gt;</code></a>	73
An instance of this class offers operators to retrieve command line options and arguments	
<a href="#"><code>sys::pid::Pidfile</code></a>	78
A wrapper around the <code>pidfile_*</code> family of commands implementing the RAIL pattern	
<a href="#"><code>sys::sc_error&lt; Domain &gt;</code></a>	80
Can be thrown by syscall function wrappers if the function returned with an error	
<a href="#"><code>sys::sig::Signal</code></a>	81
Sets up a given signal handler and restores the old handler when going out of scope	
<a href="#"><code>sys::ctl::Sync&lt; T, SysctlT &gt;</code></a>	82
This is a wrapper around <a href="#"><code>Sysctl</code></a> that allows semantically transparent use of a <code>sysctl</code>	
<a href="#"><code>sys::ctl::Sysctl&lt; MibDepth &gt;</code></a>	84
Represents a <code>sysctl</code> MIB address	
<a href="#"><code>sys::ctl::Sysctl&lt; 0 &gt;</code></a>	88
This is a specialisation of <a href="#"><code>Sysctl</code></a> for <code>sysctls</code> using symbolic names	
<a href="#"><code>anonymous_namespace{loadplay.cpp}::Sysctls</code></a>	92
Singleton class representing the <code>sysctl</code> table for this library	

<a href="#">anonymous_namespace{loadplay.cpp}::SysctlValue</a>	
Instances of this class represents a specific sysctl value	95

## 9 File Index

### 9.1 File List

Here is a list of all documented files with brief descriptions:

<a href="#">clas.hpp</a>	
Implements functions to process command line arguments	103
<a href="#">constants.hpp</a>	
Defines a collection of constants	104
<a href="#">Cycle.hpp</a>	
Implements <a href="#">timing::Cycle</a> , a cyclic sleep functor	106
<a href="#">errors.hpp</a>	
Common error handling code	107
<a href="#">fixme.hpp</a>	
Implementations in the fixme namespace	109
<a href="#">loadplay.cpp</a>	
Implements a library intended to be injected into a clock frequency daemon via LD_PRELOAD	110
<a href="#">loadrec.cpp</a>	
Implements a load recorder, useful for simulating loads to test CPU clock daemons and settings	115
<a href="#">Options.hpp</a>	
This file provides <a href="#">nih::Options&lt;&gt;</a> , a substitute for getopt (3)	118
<a href="#">powerd++.cpp</a>	
Implements powerd++ a drop in replacement for FreeBSD's powerd	121
<a href="#">types.hpp</a>	
A collection of type aliases	133
<a href="#">utility.hpp</a>	
Implements generally useful functions	134
<a href="#">sys/error.hpp</a>	
Provides system call error handling	127
<a href="#">sys/pidfile.hpp</a>	
Implements safer c++ wrappers for the pidfile_*( ) interface	128
<a href="#">sys/signal.hpp</a>	
Implements a c++ wrapper for the signal(3) call	129
<a href="#">sys/sysctl.hpp</a>	
Implements safer c++ wrappers for the <a href="#">sysctl()</a> interface	131

## 10 Namespace Documentation

### 10.1 anonymous\_namespace{clas.cpp} Namespace Reference

File local scope.

#### Enumerations

- enum [Unit](#) : size\_t {  
[Unit::SCALAR](#), [Unit::PERCENT](#), [Unit::SECOND](#), [Unit::MILLISECOND](#),  
[Unit::HZ](#), [Unit::KHZ](#), [Unit::MHZ](#), [Unit::GHZ](#),  
[Unit::THZ](#), [Unit::CELSIUS](#), [Unit::KELVIN](#), [Unit::FAHRENHEIT](#),  
[Unit::RANKINE](#), [Unit::UNKNOWN](#) }

*Command line argument units.*

#### Functions

- [Unit unit](#) (std::string const &str)  
*Determine the unit of a string encoded value.*

#### Variables

- char const \*const [UnitStr](#) []  
*The unit strings on the command line, for the respective Unit instances.*

#### 10.1.1 Detailed Description

File local scope.

#### 10.1.2 Enumeration Type Documentation

##### 10.1.2.1 Unit

```
enum anonymous_namespace{clas.cpp}::Unit : size_t [strong]
```

Command line argument units.

These units are supported for command line arguments, for SCALAR arguments the behaviour of powerd is to be imitated.

#### Enumerator

SCALAR	Values without a unit.
PERCENT	%
SECOND	s
MILLISECOND	ms
HZ	hz
KHZ	khz
MHZ	mhz
GHZ	ghz



### 10.1.3 Function Documentation

#### 10.1.3.1 unit()

```
Unit anonymous_namespace{clas.cpp}::unit (
    std::string const & str )
```

Determine the unit of a string encoded value.

#### Parameters

<i>str</i>	The string to determine the unit of
------------	-------------------------------------

#### Returns

A unit

### 10.1.4 Variable Documentation

#### 10.1.4.1 UnitStr

```
char const* const anonymous_namespace{clas.cpp}::UnitStr[ ]
```

#### Initial value:

```
{
    "", "%", "s", "ms", "hz", "khz", "mhz", "ghz", "thz", "C", "K", "F", "R"
}
```

The unit strings on the command line, for the respective Unit instances.

## 10.2 anonymous\_namespace{loadplay.cpp} Namespace Reference

File local scope.

#### Classes

- class [Callback](#)  
*Implements a recursion safe std::function wrapper.*
- class [Emulator](#)  
*Instances of this class represent an emulator session.*
- class [Hold](#)  
*Sets a referenced variable to a given value and restores it when going out of context.*
- class [Main](#)  
*Singleton class representing the main execution environment.*
- struct [mib\\_t](#)  
*Represents MIB, but wraps it to provide the necessary operators to use it as an std::map key.*
- class [Sysctls](#)  
*Singleton class representing the sysctl table for this library.*
- class [SysctlValue](#)  
*Instances of this class represents a specific sysctl value.*

## Functions

- `template<size_t Size>`  
`int strcmp (char const *const s1, char const (&s2)[Size])`  
*Safe wrapper around strcmp, which automatically determines the buffer size of s2.*
- `std::regex operator"" _r (char const *const str, size_t const len)`  
*User defined literal for regular expressions.*
- `template<>`  
`std::string SysctlValue::get< std::string > () const`  
*Returns a copy of the value string.*
- `void warn (std::string const &msg)`  
*Print a warning.*
- `void fail (std::string const &msg)`  
*This prints an error message and sets sys\_results to make the hijacked process fail.*

## Variables

- `int sys_results = 0`  
*The success return value of intercepted functions.*
- `class anonymous_namespace{loadplay.cpp}::Sysctls sysctls`  
*Sole instance of Sysctls.*
- `class anonymous_namespace{loadplay.cpp}::Main main`  
*Sole instance of Main.*
- `bool sysctl_fallback = false`  
*Set to activate fallback to the original sysctl functions.*

### 10.2.1 Detailed Description

File local scope.

### 10.2.2 Function Documentation

#### 10.2.2.1 fail()

```
void anonymous_namespace{loadplay.cpp}::fail (
    std::string const & msg ) [inline]
```

This prints an error message and sets sys\_results to make the hijacked process fail.

#### Parameters

<code>msg</code>	The error message
------------------	-------------------

## 10.2.2.2 operator""\_r()

```
std::regex anonymous_namespace{loadplay.cpp}::operator""_r (
    char const *const str,
    size_t const len ) [inline]
```

User defined literal for regular expressions.

## Parameters

<i>str, len</i>	The literal string and its length
-----------------	-----------------------------------

## Returns

A regular expression

## 10.2.2.3 strcmp()

```
template<size_t Size>
int anonymous_namespace{loadplay.cpp}::strcmp (
    char const *const s1,
    char const (&) s2[Size] ) [inline]
```

Safe wrapper around strncmp, which automatically determines the buffer size of s2.

## Template Parameters

<i>Size</i>	The size of the buffer s2
-------------	---------------------------

## Parameters

<i>s1, s2</i>	The strings to compare
---------------	------------------------

## Return values

0	Strings are equal
!0	Strings are not equal

## 10.2.2.4 SysctlValue::get&lt; std::string &gt;()

```
template<>
std::string anonymous_namespace{loadplay.cpp}::SysctlValue::get< std::string > ( ) const
```

Returns a copy of the value string.

## Returns

The value

### 10.2.2.5 warn()

```
void anonymous_namespace{loadplay.cpp}::warn (
    std::string const & msg ) [inline]
```

Print a warning.

#### Parameters

<i>msg</i>	The warning message
------------	---------------------

## 10.2.3 Variable Documentation

### 10.2.3.1 main

```
class anonymous_namespace{loadplay.cpp}::Main anonymous_namespace{loadplay.cpp}::main
```

Sole instance of [Main](#).

### 10.2.3.2 sysctls

```
class anonymous_namespace{loadplay.cpp}::Sysctls anonymous_namespace{loadplay.cpp}::sysctls
```

Sole instance of [Sysctls](#).

## 10.3 anonymous\_namespace{loadrec.cpp} Namespace Reference

File local scope.

#### Enumerations

- enum [OE](#) {  
[OE::USAGE](#), [OE::IVAL\\_DURATION](#), [OE::IVAL\\_POLL](#), [OE::FILE\\_OUTPUT](#),  
[OE::FILE\\_PID](#), [OE::FLAG\\_VERBOSE](#), [OE::OPT\\_UNKNOWN](#), [OE::OPT\\_NOOPT](#),  
[OE::OPT\\_DASH](#), [OE::OPT\\_LDASH](#), [OE::OPT\\_DONE](#) }

*An enum for command line parsing.*

#### Functions

- void [verbose](#) (std::string const &msg)  
*Outputs the given message on stderr if g.verbose is set.*
- void [init](#) ()  
*Set up output to the given file.*
- void [read\\_args](#) (int const argc, char const \*const argv[])  
*Parse command line arguments.*
- void [print\\_sysctls](#) ()  
*Print the sysctls.*
- void [run](#) ()  
*Report the load frames.*

## Variables

- - struct {
  - bool **verbose** {false}
  - Verbosity flag.*
  - ms **duration** {30000}
  - Recording duration in ms.*
  - ms **interval** {25}
  - Recording sample interval in ms.*
  - std::ofstream **outfile** {}
  - The output file stream to use if an outfilename is provided on the CLI.*
  - std::ostream \* **out** = &std::cout
  - A pointer to the stream to use for output, either std::cout or outfile.*
  - char const \* **outfilename** {nullptr}
  - The user provided output file name.*
  - char const \* **pidfilename** {POWERD\_PIDFILE}
  - The PID file location for clock frequency daemons.*
  - [sys::ctl::SysctlOnce](#)< coreid\_t, 2 > const **ncpu** {1U, {CTL\_HW, HW\_NCPU}}
  - The number of CPU cores/threads.*
  - } [g](#)
  - The global state.*
  - char const \*const **USAGE** = "[-hv] [-d ival] [-p ival] [-o file]"
  - The short usage string.*
  - [Option](#)< [OE](#) > const **OPTIONS** []
  - Definitions of command line options.*

## 10.3.1 Detailed Description

File local scope.

## 10.3.2 Enumeration Type Documentation

## 10.3.2.1 OE

```
enum anonymous_namespace{loadrec.cpp}::OE [strong]
```

An enum for command line parsing.

## Enumerator

USAGE	Print help.
IVAL_DURATION	Set the duration of the recording.
IVAL_POLL	Set polling interval.
FILE_OUTPUT	Set output file.
FILE_PID	Set PID file.
FLAG_VERBOSE	Verbose output on stderr.
OPT_UNKNOWN	Obligatory.
OPT_NOOPT	Obligatory.
OPT_DASH	Obligatory.
OPT_LDASH	Obligatory.
OPT_DONE	Obligatory.

### 10.3.3 Function Documentation

#### 10.3.3.1 read\_args()

```
void anonymous_namespace{loadrec.cpp}::read_args (
    int const argc,
    char const *const argv[ ] )
```

Parse command line arguments.

##### Parameters

<i>argc,argv</i>	The command line arguments
------------------	----------------------------

#### 10.3.3.2 run()

```
void anonymous_namespace{loadrec.cpp}::run ( )
```

Report the load frames.

This prints the time in ms since the last frame and the `cp_times` growth as a space separated list.

#### 10.3.3.3 verbose()

```
void anonymous_namespace{loadrec.cpp}::verbose (
    std::string const & msg ) [inline]
```

Outputs the given message on stderr if `g.verbose` is set.

##### Parameters

<i>msg</i>	The message to output
------------	-----------------------

### 10.3.4 Variable Documentation

#### 10.3.4.1 OPTIONS

```
Option<OE> const anonymous_namespace{loadrec.cpp}::OPTIONS[ ]
```

**Initial value:**

```
{
  {OE::USAGE,      'h', "help",      "",      "Show usage and exit"},
  {OE::FLAG_VERBOSE, 'v', "verbose", "",      "Be verbose"},
  {OE::IVAL_DURATION, 'd', "duration", "ival", "The duration of the recording"},
  {OE::IVAL_POLL,    'p', "poll",     "ival", "The polling interval"},
  {OE::FILE_OUTPUT,  'o', "output",   "file", "Output to file"},
  {OE::FILE_PID,     'P', "pid",      "file", "PID file of the local clock frequency daemon"},
}
```

Definitions of command line options.

## 10.4 anonymous\_namespace{powerd++.cpp} Namespace Reference

File local scope.

### Classes

- struct [Core](#)  
*Contains the management information for a single CPU core. [More...](#)*
- class [FreqGuard](#)  
*A core frequency guard.*
- struct [Global](#)  
*A collection of all the gloabl, mutable states. [More...](#)*

### Enumerations

- enum [AcLineState](#) : unsigned int { [AcLineState::BATTERY](#), [AcLineState::ONLINE](#), [AcLineState::UNKNOWN](#), [AcLineState::LENGTH](#) }  
*The available AC line states.*
- enum [OE](#) {  
[OE::USAGE](#), [OE::MODE\\_AC](#), [OE::MODE\\_BATT](#), [OE::FREQ\\_MIN](#),  
[OE::FREQ\\_MAX](#), [OE::FREQ\\_MIN\\_AC](#), [OE::FREQ\\_MAX\\_AC](#), [OE::FREQ\\_MIN\\_BATT](#),  
[OE::FREQ\\_MAX\\_BATT](#), [OE::FREQ\\_RANGE](#), [OE::FREQ\\_RANGE\\_AC](#), [OE::FREQ\\_RANGE\\_BATT](#),  
[OE::HITEMP\\_RANGE](#), [OE::MODE\\_UNKNOWN](#), [OE::IVAL\\_POLL](#), [OE::FILE\\_PID](#),  
[OE::FLAG\\_VERBOSE](#), [OE::FLAG\\_FOREGROUND](#), [OE::CNT\\_SAMPLES](#), [OE::IGNORE](#),  
[OE::OPT\\_UNKNOWN](#), [OE::OPT\\_NOOPT](#), [OE::OPT\\_DASH](#), [OE::OPT\\_LDASH](#),  
[OE::OPT\\_DONE](#) }  
*An enum for command line parsing.*

### Functions

- void [verbose](#) (std::string const &msg)  
*Outputs the given message on stderr if g.verbose is set.*
- void [sysctl\\_fail](#) (sys::sc\_error< [sys::ctl::error](#) > const err)  
*Treat sysctl errors.*
- void [init](#) ()  
*Perform initial tasks.*
- template<bool Load = 1, bool Temperature = 0>  
void [update\\_loads](#) ()  
*Updates the cp\_times ring buffer and computes the load average for each core.*
- template<>  
void [update\\_loads](#)< 0, 0 > ()

- Do nada if neither load nor temperature are to be updated.*

  - `template<bool Foreground, bool Temperature, bool Fixed>`  
`void update\_freq (Global::ACSet const &acstate)`

*Update the CPU clocks depending on the AC line state and targets.*
  - `void update\_freq ()`

*Dispatch [update\\_freq](#)<>().*
  - `void init\_loads ()`

*Fill the loads buffers with n samples.*
  - `void set\_mode (AcLineState const line, char const *const str)`

*Sets a load target or fixed frequency for the given AC line state.*
  - `void read\_args (int const argc, char const *const argv[])`

*Parse command line arguments.*
  - `void show\_settings ()`

*Prints the configuration on stderr in verbose mode.*
  - `void signal\_recv (int signal)`

*Sets g.signal, terminating the main loop.*
  - `void run\_daemon ()`

*Daemonise and run the main loop.*

## Variables

- `struct anonymous_namespace{powerd++.cpp}::Global g`

*The gobal state.*
- `char const *const USAGE = "[ -hvf] [ -abn mode] [ -mM freq] [ -FAB freq:freq] [ -H temp:temp] [ -p ival] [ -s cnt] [ -P file]"`

*The short usage string.*
- `Option< OE > const OPTIONS []`

*Definitions of command line options.*

### 10.4.1 Detailed Description

File local scope.

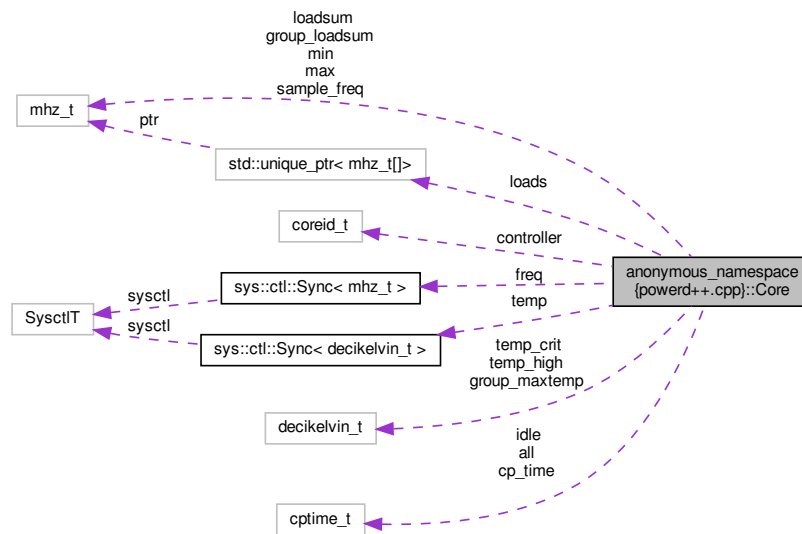
### 10.4.2 Class Documentation

#### 10.4.2.1 `struct anonymous_namespace{powerd++.cpp}::Core`

Contains the management information for a single CPU core.



Collaboration diagram for anonymous\_namespace{power++.cpp}::Core:



#### Class Members

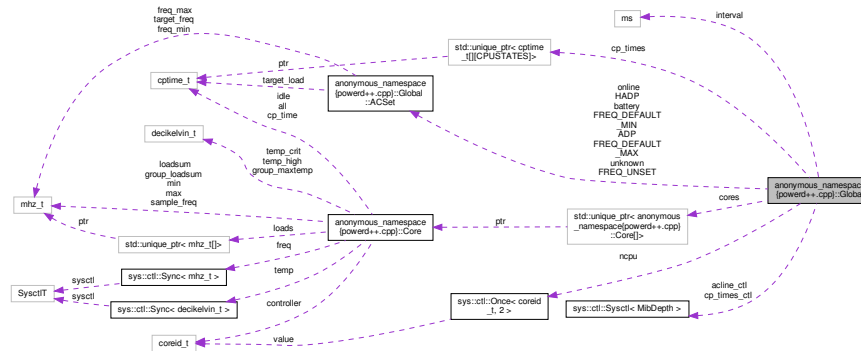
cptime_t	all	Count of all ticks.
coreid_t	controller	The core that controls the frequency for this core.
cptime_t const *	cp_time	A pointer to the kern.cp_times section for this core.
SysctlSync<mhz_t>	freq	The sysctl kern.cpu.N.freq, if present.
mhz_t	group_loadsum	For the controlling core this is set to the group loadsum. This is updated by <a href="#">update_loads()</a> .
decikelvin_t	group_maxtemp	For the controlling core this is set to the maximum temperature measurement taken in the group.
cptime_t	idle	The idle ticks count.
unique_ptr<mhz_t[]>	loads	A ring buffer of load samples for this core. Each load sample is weighted with the core frequency at which it was taken. This is updated by <a href="#">update_loads()</a> .
mhz_t	loadsum	The sum of all load samples. This is updated by <a href="#">update_loads()</a> .
mhz_t	max	The maximum core clock rate.
mhz_t	min	The minimum core clock rate.
mhz_t	sample_freq	The kern.cpu.N.freq value for the current load sample. This is updated by <a href="#">update_loads()</a> .
SysctlSync<decikelvin_t>	temp	The dev.cpu. d.temperature sysctl, if present.
decikelvin_t	temp_crit	Critical core temperature in dK.
decikelvin_t	temp_high	High core temperature in dK.

#### 10.4.2.2 struct anonymous\_namespace{power++.cpp}::Global

A collection of all the global, mutable states.

This is mostly for semantic clarity.

Collaboration diagram for anonymous\_namespace{powerd++.cpp}::Global:



### Class Members

	Sysctl	ac_line_ctl	The hw.acpi.ac_line ctl.
struct	ADP[3]		
anonymous_namespace{powerd++}			
struct	battery[3]		
anonymous_namespace{powerd++}			
unique_ptr< Core[] >	cores		This buffer is to be allocated with ncpu instances of the <a href="#">Core</a> struct to store the management information of every core.
unique_ptr< cp_time_t[] >[CPUSTATES]	cp_times		The kern.cp_times buffer for all cores.
Sysctl	cp_times_ctl		The kern.cp_times sysctl.
bool	foreground		Foreground mode.
struct	FREQ_DEFAULT_MAX[3]		
anonymous_namespace{powerd++}			
struct	FREQ_DEFAULT_MIN[3]		
anonymous_namespace{powerd++}			
struct	FREQ_UNSET[3]		
anonymous_namespace{powerd++}			
struct	HADP[3]		
anonymous_namespace{powerd++}			
ms	interval		The polling interval.
SysctlOnce< coreid_t, 2 > const	ncpu		The number of CPU cores or threads.
struct	online[3]		
anonymous_namespace{powerd++}			
char const *	pidfilename		Name of an alternative pidfile. If not given <a href="#">pidfile_open()</a> uses a default name.
size_t	sample		The current sample.
size_t	samples		The number of load samples to take.
volatile sig_atomic_t	signal		The last signal received, used for terminating.
bool	temp_throttling		Temperature throttling mode.
struct	unknown[3]		The power states.
anonymous_namespace{powerd++}			
bool	verbose		Verbose mode.

## 10.4.3 Enumeration Type Documentation

## 10.4.3.1 AcLineState

```
enum anonymous_namespace{power++.cpp}::AcLineState : unsigned int [strong]
```

The available AC line states.

## Enumerator

BATTERY	Battery is power source.
ONLINE	External power source.
UNKNOWN	Unknown power source.
LENGTH	Enum length.

## 10.4.3.2 OE

```
enum anonymous_namespace{power++.cpp}::OE [strong]
```

An enum for command line parsing.

## Enumerator

USAGE	Print help.
MODE_AC	Set AC power mode.
MODE_BATT	Set battery power mode.
FREQ_MIN	Set minimum clock frequency.
FREQ_MAX	Set maximum clock frequency.
FREQ_MIN_AC	Set minimum clock frequency on AC power.
FREQ_MAX_AC	Set maximum clock frequency on AC power.
FREQ_MIN_BATT	Set minimum clock frequency on battery power.
FREQ_MAX_BATT	Set maximum clock frequency on battery power.
FREQ_RANGE	Set clock frequency range.
FREQ_RANGE_AC	Set clock frequency range on AC power.
FREQ_RANGE_BATT	Set clock frequency range on battery power.
HITEMP_RANGE	Set a high temperature range.
MODE_UNKNOWN	Set unknown power source mode.
IVAL_POLL	Set polling interval.
FILE_PID	Set pidfile.
FLAG_VERBOSE	Activate verbose output on stderr.
FLAG_FOREGROUND	Stay in foreground, log events to stdout.
CNT_SAMPLES	Set number of load samples.
IGNORE	Legacy settings.
OPT_UNKNOWN	Obligatory.
OPT_NOOPT	Obligatory.
OPT_DASH	Obligatory.
OPT_LDASH	Obligatory.
OPT_DONE	Obligatory.

## 10.4.4 Function Documentation

### 10.4.4.1 `init()`

```
void anonymous_namespace{powerd++.cpp}::init ( )
```

Perform initial tasks.

- Get number of CPU cores/threads
- Determine the clock controlling core for each core
- Set the MIBs of hw.acpi.acline and kern.cp\_times

### 10.4.4.2 `init_loads()`

```
void anonymous_namespace{powerd++.cpp}::init_loads ( )
```

Fill the loads buffers with n samples.

The samples are filled with the target load, this creates a bias to stay at the initial frequency until sufficient real measurements come in to flush these initial samples out.

### 10.4.4.3 `read_args()`

```
void anonymous_namespace{powerd++.cpp}::read_args (
    int const argc,
    char const *const argv[] )
```

Parse command line arguments.

#### Parameters

<i>argc,argv</i>	The command line arguments
------------------	----------------------------

### 10.4.4.4 `set_mode()`

```
void anonymous_namespace{powerd++.cpp}::set_mode (
    AcLineState const line,
    char const *const str )
```

Sets a load target or fixed frequency for the given AC line state.

The string must be in the following format:

```

mode_predefined = "minimum" | "min" | "maximum" | "max" |
                  "adaptive" | "adp" | "hiadaptive" | "hadp";
mode =            mode_predefined | load | freq;

```

Scalar values are treated as loads.

The predefined values have the following meaning:

Symbol	Meaning
minimum	The minimum clock rate (default 0 MHz)
min	
maximum	The maximum clock rate (default 1000000 MHz)
max	
adaptive	A target load of 50%
adp	
hiadaptive	A target load of 37.5%
hadp	

#### Parameters

<i>line</i>	The power line state to set the mode for
<i>str</i>	A mode string

#### 10.4.4.5 signal\_recv()

```

void anonymous_namespace{powerd++.cpp}::signal_recv (
    int signal )

```

Sets g.signal, terminating the main loop.

#### Parameters

<i>signal</i>	The signal number received
---------------	----------------------------

#### 10.4.4.6 sysctl\_fail()

```

void anonymous_namespace{powerd++.cpp}::sysctl_fail (
    sys::sc_error< sys::ctl::error > const err ) [inline]

```

Treat sysctl errors.

Fails appropriately for the given error.

#### Parameters

<i>err</i>	The errno value after calling sysctl
------------	--------------------------------------

#### 10.4.4.7 update\_freq()

```
template<bool Foreground, bool Temperature, bool Fixed>
void anonymous_namespace{powerd++.cpp}::update_freq (
    Global::ACSet const & acstate )
```

Update the CPU clocks depending on the AC line state and targets.

##### Template Parameters

<i>Foreground</i>	Set for foreground operation (reporting on std::cout)
<i>Temperature</i>	Set for temperature based throttling
<i>Fixed</i>	Set for fixed frequency mode

##### Parameters

<i>acstate</i>	The set of acline dependent variables
----------------	---------------------------------------

#### 10.4.4.8 update\_loads()

```
template<bool Load = 1, bool Temperature = 0>
void anonymous_namespace{powerd++.cpp}::update_loads ( )
```

Updates the cp\_times ring buffer and computes the load average for each core.

##### Template Parameters

<i>Load</i>	Determines whether <code>group_loadsum</code> is updated
<i>Temperature</i>	Determines whether <code>group_maxtemp</code> is updated

#### 10.4.4.9 verbose()

```
void anonymous_namespace{powerd++.cpp}::verbose (
    std::string const & msg ) [inline]
```

Outputs the given message on stderr if g.verbose is set.

##### Parameters

<i>msg</i>	The message to output
------------	-----------------------

## 10.4.5 Variable Documentation

## 10.4.5.1 g

```
struct anonymous_namespace{powerd++.cpp}::Global anonymous_namespace{powerd++.cpp}::g
```

The gobal state.

## 10.4.5.2 OPTIONS

```
Option<OE> const anonymous_namespace{powerd++.cpp}::OPTIONS[ ]
```

Initial value:

```
{
  {OE::USAGE,          'h', "help",          "",          "Show usage and exit"},
  {OE::FLAG_VERBOSE,   'v', "verbose",        "",          "Be verbose"},
  {OE::FLAG_FOREGROUND, 'f', "foreground",    "",          "Stay in foreground"},
  {OE::MODE_AC,        'a', "ac",             "mode",      "Mode while on AC power"},
  {OE::MODE_BATT,      'b', "batt",           "mode",      "Mode while on battery power"},
  {OE::MODE_UNKNOWN,   'n', "unknown",        "mode",      "Mode while power source is unknown"},
  {OE::FREQ_MIN,       'm', "min",            "freq",      "Minimum CPU frequency"},
  {OE::FREQ_MAX,       'M', "max",            "freq",      "Maximum CPU frequency"},
  {OE::FREQ_MIN_AC,    0, "min-ac",           "freq",      "Minimum CPU frequency on AC power"},
  {OE::FREQ_MAX_AC,    0, "max-ac",           "freq",      "Maximum CPU frequency on AC power"},
  {OE::FREQ_MIN_BATT,  0, "min-batt",         "freq",      "Minimum CPU frequency on battery power"},
  {OE::FREQ_MAX_BATT,  0, "max-batt",         "freq",      "Maximum CPU frequency on battery power"},
  {OE::FREQ_RANGE,     'F', "freq-range",     "freq:freq", "CPU frequency range (min:max)"},
  {OE::FREQ_RANGE_AC, 'A', "freq-range-ac",   "freq:freq", "CPU frequency range on AC power"},
  {OE::FREQ_RANGE_BATT, 'B', "freq-range-batt", "freq:freq", "CPU frequency range on battery power"},
  {OE::HITEMP_RANGE,   'H', "hitemp-range",   "temp:temp", "High temperature range (high:critical)"},
  {OE::IVAL_POLL,      'P', "poll",           "ival",      "The polling interval"},
  {OE::CNT_SAMPLES,    's', "samples",        "cnt",       "The number of samples to use"},
  {OE::FILE_PID,       'P', "pid",            "file",      "Alternative PID file"},
  {OE::IGNORE,         'i', "",              "load",      "Ignored"},
  {OE::IGNORE,         'r', "",              "load",      "Ignored"}
}
```

Definitions of command line options.

## 10.5 clas Namespace Reference

A collection of functions to process command line arguments.

## Functions

- [types::cptime\\_t load](#) (char const \*const str)  
*Convert string to load in the range [0, 1024].*
- [types::mhz\\_t freq](#) (char const \*const str)  
*Convert string to frequency in MHz.*
- [types::ms ival](#) (char const \*const str)  
*Convert string to time interval in milliseconds.*
- [size\\_t samples](#) (char const \*const str)  
*A string encoded number of samples.*
- [types::decikelvin\\_t temperature](#) (char const \*const str)  
*Convert string to temperature in dK.*
- [int celsius](#) ([types::decikelvin\\_t](#) const val)  
*Converts dK into °C for display purposes.*
- [template<typename T>](#)  
[std::pair< T, T > range](#) (char const \*const str, T(\*func)(char const \*const))  
*Takes a string encoded range of values and returns them.*

### 10.5.1 Detailed Description

A collection of functions to process command line arguments.

### 10.5.2 Function Documentation

#### 10.5.2.1 celsius()

```
int clas::celsius (
    types::decikelvin_t const val ) [inline]
```

Converts dK into °C for display purposes.

##### Parameters

<i>val</i>	A temperature in dK
------------	---------------------

##### Returns

The temperature in °C

#### 10.5.2.2 freq()

```
types::mhz_t clas::freq (
    char const *const str )
```

Convert string to frequency in MHz.

The given string must have the following format:

```
freq = <float>, [ "hz" | "khz" | "mhz" | "ghz" | "thz" ];
```

For compatibility with powerd MHz are assumed, if no unit string is given.

The resulting frequency must be in the range [0Hz, 1THz].

##### Parameters

<i>str</i>	A string encoded frequency
------------	----------------------------

##### Returns

The frequency given by str



## 10.5.2.3 ival()

```
types::ms clas::ival (
    char const *const str )
```

Convert string to time interval in milliseconds.

The given string must have the following format:

```
ival = <float>, [ "s" | "ms" ];
```

For compatibility with powerd scalar values are assumed to represent milliseconds.

## Parameters

<i>str</i>	A string encoded time interval
------------	--------------------------------

## Returns

The interval in milliseconds

## 10.5.2.4 load()

```
types::cptime_t clas::load (
    char const *const str )
```

Convert string to load in the range [0, 1024].

The given string must have the following format:

```
load = <float>, [ "%" ];
```

The input value must be in the range [0.0, 1.0] or [0%, 100%].

## Parameters

<i>str</i>	A string encoded load
------------	-----------------------

## Return values

[0,1024]	The load given by str
>	1024 The given string is not a load

## 10.5.2.5 range()

```
template<typename T >
```

```
std::pair<T, T> clas::range (
    char const *const str,
    T(*) (char const *const) func )
```

Takes a string encoded range of values and returns them.

A range has the format `from:to`.

#### Template Parameters

<i>T</i>	The return type of the conversion function
----------	--

#### Parameters

<i>str</i>	The string containing the range
<i>func</i>	The function that converts the values from the string

#### Returns

A pair with the `from` and `to` values

#### 10.5.2.6 samples()

```
size_t clas::samples (
    char const *const str )
```

A string encoded number of samples.

The string is expected to contain a scalar integer.

#### Parameters

<i>str</i>	The string containing the number of samples
------------	---

#### Returns

The number of samples

#### 10.5.2.7 temperature()

```
types::decikelvin_t clas::temperature (
    char const *const str )
```

Convert string to temperature in dK.

The given string must have the following format:

```
temperature = <float>, [ "C" | "K" | "F" | "R" ];
```

In absence of a unit °C is assumed.

## Parameters

<i>str</i>	A string encoded temperature
------------	------------------------------

## Returns

The temperature given by *str*

## 10.6 constants Namespace Reference

A collection of constants.

## Variables

- char const \*const [CP\\_TIMES](#) = "kern.cp\_times"  
*The MIB name for per-CPU time statistics.*
- char const \*const [ACLIN](#) = "hw.acpi.acline"  
*The MIB name for the AC line state.*
- char const \*const [FREQ](#) = "dev.cpu.%d.freq"  
*The MIB name for CPU frequencies.*
- char const \*const [FREQ\\_LEVELS](#) = "dev.cpu.%d.freq\_levels"  
*The MIB name for CPU frequency levels.*
- char const \*const [TEMPERATURE](#) = "dev.cpu.%d.temperature"  
*The MIB name for CPU temperatures.*
- char const \*const [TJMAX\\_SOURCES](#) []  
*An array of maximum temperature sources.*
- [types::mhz\\_t](#) const [FREQ\\_DEFAULT\\_MAX](#) {1000000}  
*Default maximum clock frequency value.*
- [types::mhz\\_t](#) const [FREQ\\_DEFAULT\\_MIN](#) {0}  
*Default minimum clock frequency value.*
- [types::mhz\\_t](#) const [FREQ\\_UNSET](#) {1000001}  
*Clock frequency representing an uninitialised value.*
- char const \*const [POWERD\\_PIDFILE](#) = "/var/run/powerd.pid"  
*The default pidfile name of powerd.*
- [types::cptime\\_t](#) const [ADP](#) {512}  
*The load target for adaptive mode, equals 50% load.*
- [types::cptime\\_t](#) const [HADP](#) {384}  
*The load target for hiadaptive mode, equals 37.5% load.*
- [types::decikelvin\\_t](#) const [HITEMP\\_OFFSET](#) {100}  
*The default temperautre offset between high and critical temperature.*

## 10.6.1 Detailed Description

A collection of constants.

## 10.6.2 Variable Documentation

### 10.6.2.1 TJMAX\_SOURCES

```
char const* const constants::TJMAX_SOURCES[ ]
```

#### Initial value:

```
= {  
    "dev.cpu.%d.coretemp.tjmax"  
}
```

An array of maximum temperature sources.

## 10.7 errors Namespace Reference

Common error handling types and functions.

### Classes

- struct [Exception](#)  
*Exceptions bundle an exit code, errno value and message. [More...](#)*

### Enumerations

- enum [Exit](#) : int {  
    [Exit::OK](#), [Exit::ECLARG](#), [Exit::EOUTOFRANGE](#), [Exit::ELOAD](#),  
    [Exit::EFREQ](#), [Exit::EMODE](#), [Exit::EIVAL](#), [Exit::ESAMPLES](#),  
    [Exit::ESYSCTL](#), [Exit::ENOFREQ](#), [Exit::ECONFLICT](#), [Exit::EPID](#),  
    [Exit::EFORBIDDEN](#), [Exit::EDAEMON](#), [Exit::EWOPEN](#), [Exit::ESIGNAL](#),  
    [Exit::ERANGEFMT](#), [Exit::ETEMPERATURE](#), [Exit::LENGTH](#) }  
*Exit codes.*

### Functions

- void [fail](#) ([Exit](#) const exitcode, int const err, std::string const &msg)  
*Throws an [Exception](#) instance with the given message.*

### Variables

- const char \*const [ExitStr](#) []  
*Printable strings for exit codes.*

### 10.7.1 Detailed Description

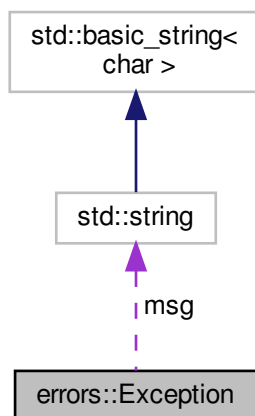
Common error handling types and functions.

## 10.7.2 Class Documentation

## 10.7.2.1 struct errors::Exception

Exceptions bundle an exit code, errno value and message.

Collaboration diagram for errors::Exception:



## Class Members

int	err	The errno value at the time of creation.
<a href="#">Exit</a>	exitcode	The code to exit with.
string	msg	An error message.

## 10.7.3 Enumeration Type Documentation

## 10.7.3.1 Exit

```
enum errors::Exit : int [strong]
```

Exit codes.

## Enumerator

OK	Regular termination.
ECLARG	Unexpected command line argument.
EOUTOFRANGE	A user provided value is out of range.
ELOAD	The provided value is not a valid load.

## Enumerator

EFREQ	The provided value is not a valid frequency.
EMODE	The provided value is not a valid mode.
EIVAL	The provided value is not a valid interval.
ESAMPLES	The provided value is not a valid sample count.
ESYSCTL	A sysctl operation failed.
ENOFREQ	System does not support changing core frequencies.
ECONFLICT	Another frequency daemon instance is running.
EPID	A pidfile could not be created.
EFORBIDDEN	Insufficient privileges to change sysctl.
EDAEMON	Unable to detach from terminal.
EWOPEN	Could not open file for writing.
ESIGNAL	Failed to install signal handler.
ERANGEFMT	A user provided range is missing the separator.
ETEMPERATURE	The provided value is not a valid temperature.
LENGTH	Enum length.

## 10.7.4 Function Documentation

## 10.7.4.1 fail()

```
void errors::fail (
    Exit const exitcode,
    int const err,
    std::string const & msg ) [inline]
```

Throws an [Exception](#) instance with the given message.

## Parameters

<i>exitcode</i>	The exit code to return on termination
<i>err</i>	The errno value at the time the exception was created
<i>msg</i>	The message to show

## 10.7.5 Variable Documentation

## 10.7.5.1 ExitStr

```
const char* const errors::ExitStr[]
```

## Initial value:

```
{
    "OK", "ECLARG", "EOUTOFRANGE", "ELOAD", "EFREQ", "EMODE", "EIVAL",
    "ESAMPLES", "ESYSCTL", "ENOFREQ", "ECONFLICT", "EPID", "EFORBIDDEN",
    "EDAEMON", "EWOPEN", "ESIGNAL", "ERANGEFMT", "ETEMPERATURE"
}
```

Printable strings for exit codes.

## 10.8 fixme Namespace Reference

Workarounds for compiler/library bugs.

### Functions

- `template<typename T >`  
`std::string to_string (T const &op)`  
*G++ 5.3 does not believe in `std::to_string()`.*

### 10.8.1 Detailed Description

Workarounds for compiler/library bugs.

### 10.8.2 Function Documentation

#### 10.8.2.1 to\_string()

```
template<typename T >
std::string fixme::to_string (
    T const & op ) [inline]
```

G++ 5.3 does not believe in `std::to_string()`.

#### Template Parameters

<i>T</i>	The argument type to convert
----------	------------------------------

#### Parameters

<i>op</i>	The argument to convert
-----------	-------------------------

#### Returns

A string of the given argument

## 10.9 nih Namespace Reference

Not invented here namespace, for code that substitutes already commonly available functionality.

### Classes

- struct [enum\\_has\\_members](#)  
*Tests whether the given enum provides all the required definitions.*
- struct [Option](#)  
*Container for an option definition. [More...](#)*
- class [Options](#)  
*An instance of this class offers operators to retrieve command line options and arguments.*

### Typedefs

- `template<class... >`  
`using void\_t = void`  
*See `std::void_t` in C++17 `<type_traits>`.*

### Functions

- `template<class Enum >`  
`size_t argCount (Option< Enum > const &def)`  
*Retrieves the count of arguments in an option definition.*
- `template<class Enum , size_t DefCount>`  
`constexpr Options< Enum, DefCount > make\_Options (int const argc, char const *const argv[], char const *const usage, Option< Enum > const (&defs)[DefCount])`  
*Wrapper around the `Options<>` constructor, that uses function template matching to deduce template arguments.*

#### 10.9.1 Detailed Description

Not invented here namespace, for code that substitutes already commonly available functionality.

#### 10.9.2 Class Documentation

##### 10.9.2.1 struct nih::Option

```
template<class Enum>
struct nih::Option< Enum >
```

Container for an option definition.

Aliases can be defined by creating definitions with the same enumval member.

The lopt, args and usage members have to be 0 terminated, using string literals is safe.



## Template Parameters

<i>Enum</i>	An enum or enum class representing the available options
-------------	--

## Class Members

char const *	args	A comma separated list of arguments. Set to nullptr or "" if no argument is available.
Enum	enumval	The enum value to return for this option.
char const *	lopt	The long version of this option. Set to nullptr or "" if no long option is available.
char	sopt	The short version of this option. Set to 0 if no short option is available.
char const *	usage	A usage string.

## 10.9.3 Function Documentation

## 10.9.3.1 argCount()

```
template<class Enum >
size_t nih::argCount (
    Option< Enum > const & def )
```

Retrieves the count of arguments in an option definition.

## Template Parameters

<i>Enum</i>	An enum or enum class representing the available options
-------------	--

## Parameters

<i>def</i>	The option definition
------------	-----------------------

## Returns

The number of arguments specified in the given definition

## 10.9.3.2 make\_Options()

```
template<class Enum , size_t DefCount>
constexpr Options<Enum, DefCount> nih::make_Options (
    int const argc,
    char const *const argv[],
    char const *const usage,
    Option< Enum > const (&) defs[DefCount] )
```

Wrapper around the Options<> constructor, that uses function template matching to deduce template arguments.

### Template Parameters

<i>Enum</i>	An enum for all the available options
<i>DefCount</i>	The number of option definitions

### Parameters

<i>argc,argv</i>	The command line arguments
<i>usage</i>	A usage string that is used in the header of the usage output
<i>defs</i>	An array of option definitions

## 10.10 sys Namespace Reference

Wrappers around native system interfaces.

### Namespaces

- [ctl](#)  
*This namespace contains safer c++ wrappers for the [sysctl\(\)](#) interface.*
- [pid](#)  
*This namespace contains safer c++ wrappers for the [pidfile\\_\\*](#)() interface.*
- [sig](#)  
*This namespace provides c++ wrappers for [signal\(3\)](#).*

### Classes

- struct [sc\\_error](#)  
*Can be thrown by [syscall](#) function wrappers if the function returned with an error.*

#### 10.10.1 Detailed Description

Wrappers around native system interfaces.

## 10.11 sys::ctl Namespace Reference

This namespace contains safer c++ wrappers for the [sysctl\(\)](#) interface.

### Classes

- struct [error](#)  
*The domain error type. [More...](#)*
- class [Once](#)  
*A read once representation of a [Sysctl](#).*
- class [Sync](#)  
*This is a wrapper around [Sysctl](#) that allows semantically transparent use of a [sysctl](#).*
- class [Sysctl](#)  
*Represents a [sysctl](#) MIB address.*
- class [Sysctl< 0 >](#)  
*This is a specialisation of [Sysctl](#) for [sysctls](#) using symbolic names.*

## Typedefs

- typedef int [mib\\_t](#)  
*Management Information Base identifier type (see sysctl(3)).*
- template<typename T, size\_t MibDepth = 0>  
using [SysctlSync](#) = [Sync](#)< T, [Sysctl](#)< MibDepth > >  
*A convenience alias around [Sync](#).*
- template<typename T, size\_t MibDepth>  
using [SysctlOnce](#) = [Once](#)< T, [Sysctl](#)< MibDepth > >  
*A convenience alias around [Once](#).*

## Functions

- void [sysctl\\_raw](#) ([mib\\_t](#) const \*name, u\_int const namelen, void \*const oldp, size\_t \*const oldlenp, void const \*const newp, size\_t const newlen)  
*A wrapper around the [sysctl\(\)](#) function.*
- template<size\_t MibDepth>  
void [sysctl\\_get](#) ([mib\\_t](#) const (&mib)[MibDepth], void \*const oldp, size\_t &oldlen)  
*Returns a [sysctl\(\)](#) value to a buffer.*
- template<size\_t MibDepth>  
void [sysctl\\_set](#) ([mib\\_t](#) const (&mib)[MibDepth], void const \*const newp, size\_t const newlen)  
*Sets a [sysctl\(\)](#) value.*
- template<typename... Args>  
constexpr [Sysctl](#)< sizeof...(Args)> [make\\_Sysctl](#) (Args const ... args)  
*Create a [Sysctl](#) instances.*
- template<typename T, class SysctlT>  
constexpr [Once](#)< T, [SysctlT](#) > [make\\_Once](#) (T const &value, [SysctlT](#) const &[sysctl](#)) noexcept  
*This creates a [Once](#) instance.*

### 10.11.1 Detailed Description

This namespace contains safer c++ wrappers for the [sysctl\(\)](#) interface.

The template class [Sysctl](#) represents a sysctl address and offers handles to retrieve or set the stored value.

The template class [Sync](#) represents a sysctl value that is read and written synchronously.

The template class [Once](#) represents a read once value.

### 10.11.2 Class Documentation

#### 10.11.2.1 struct sys::ctl::error

The domain error type.

### 10.11.3 Typedef Documentation

#### 10.11.3.1 SysctlOnce

```
template<typename T, size_t MibDepth>
using sys::ctl::SysctlOnce = typedef Once<T, Sysctl<MibDepth> >
```

A convenience alias around [Once](#).

```
// Once<coreid_t, Sysctl<2>> ncpu{0, {CTL_HW, HW_NCPU}};
SysctlOnce<coreid_t, 2> ncpu{1, {CTL_HW, HW_NCPU}};
```

### Template Parameters

<i>T</i>	The type to represent the sysctl as
<i>MibDepth</i>	The maximum allowed MIB depth

#### 10.11.3.2 SysctlSync

```
template<typename T , size_t MibDepth = 0>
using sys::ctl::SysctlSync = typedef Sync<T, Sysctl<MibDepth> >
```

A convenience alias around [Sync](#).

```
// Sync<int, Sysctl<>> sndUnit({"hw.snd.default_unit"});
SysctlSync<int> sndUnit({"hw.snd.default_unit"});
if (sndUnit != 3) { // read from sysctl
    sndUnit = 3;    // assign to sysctl
}
```

### Template Parameters

<i>T</i>	The type to represent the sysctl as
<i>MibDepth</i>	The MIB depth, provide only for compile time initialisation

#### 10.11.4 Function Documentation

##### 10.11.4.1 make\_Once()

```
template<typename T , class SysctlT >
constexpr Once<T, SysctlT> sys::ctl::make_Once (
    T const & value,
    SysctlT const & sysctl ) [noexcept]
```

This creates a [Once](#) instance.

This is intended for cases when a [Once](#) instance is created as a temporary to retrieve a value, using it's fallback to a default mechanism.

### Template Parameters

<i>T</i>	The value type
<i>SysctlT</i>	The <a href="#">Sysctl</a> type

### Parameters

<i>value</i>	The default value to fall back to
<i>sysctl</i>	The sysctl to try and read from

## 10.11.4.2 make\_Sysctl()

```
template<typename... Args>
constexpr Sysctl<sizeof...(Args)> sys::ctl::make_Sysctl (
    Args const ... args )
```

Create a [Sysctl](#) instances.

This is only compatible with creating sysctls from predefined MIBs.

## Template Parameters

<i>Args</i>	List of argument types, should all be <code>pid_t</code>
-------------	--

## Parameters

<i>args</i>	List of initialising arguments
-------------	--------------------------------

## Returns

A [Sysctl](#) instance with the depth matching the number of arguments

## 10.11.4.3 sysctl\_get()

```
template<size_t MibDepth>
void sys::ctl::sysctl_get (
    mib_t const (&) mib[MibDepth],
    void *const oldp,
    size_t & oldlen )
```

Returns a [sysctl\(\)](#) value to a buffer.

## Template Parameters

<i>MibDepth</i>	The length of the MIB buffer
-----------------	------------------------------

## Parameters

<i>mib</i>	The MIB buffer
<i>oldp, oldlen</i>	A pointers to the return buffer and a reference to its length

## Exceptions

<code>sys::sc_error&lt;error&gt;</code>	Throws if <a href="#">sysctl()</a> fails for any reason
---	---

#### 10.11.4.4 sysctl\_raw()

```
void sys::ctl::sysctl_raw (
    mib_t const * name,
    u_int const namelen,
    void *const oldp,
    size_t *const oldlenp,
    void const *const newp,
    size_t const newlen ) [inline]
```

A wrapper around the [sysctl\(\)](#) function.

All it does is throw an exception if [sysctl\(\)](#) fails.

##### Parameters

<i>name,namelen</i>	The MIB buffer and its length
<i>oldp,oldlenp</i>	Pointers to the return buffer and its length
<i>newp,newlen</i>	A pointer to the buffer with the new value and the buffer length

##### Exceptions

<code>sys::sc_error&lt;error&gt;</code>	Throws if <a href="#">sysctl()</a> fails for any reason
---	---

#### 10.11.4.5 sysctl\_set()

```
template<size_t MibDepth>
void sys::ctl::sysctl_set (
    mib_t const (&) mib[MibDepth],
    void const *const newp,
    size_t const newlen )
```

Sets a [sysctl\(\)](#) value.

##### Template Parameters

<i>MibDepth</i>	The length of the MIB buffer
-----------------	------------------------------

##### Parameters

<i>mib</i>	The MIB buffer
<i>newp,newlen</i>	A pointer to the buffer with the new value and the buffer length

##### Exceptions

<code>sys::sc_error&lt;error&gt;</code>	Throws if <a href="#">sysctl()</a> fails for any reason
---	---

## 10.12 sys::pid Namespace Reference

This namespace contains safer c++ wrappers for the `pidfile_*`() interface.

### Classes

- struct [error](#)  
*The domain error type. [More...](#)*
- class [Pidfile](#)  
*A wrapper around the `pidfile_*` family of commands implementing the RAII pattern.*

### 10.12.1 Detailed Description

This namespace contains safer c++ wrappers for the `pidfile_*`() interface.

The class [Pidfile](#) implements the RAII pattern for holding a pidfile.

### 10.12.2 Class Documentation

#### 10.12.2.1 struct sys::pid::error

The domain error type.

## 10.13 sys::sig Namespace Reference

This namespace provides c++ wrappers for `signal(3)`.

### Classes

- struct [error](#)  
*The domain error type. [More...](#)*
- class [Signal](#)  
*Sets up a given signal handler and restores the old handler when going out of scope.*

### Typedefs

- using [sig\\_t](#) = void(\*)(int)  
*Convenience type for signal handlers.*

### 10.13.1 Detailed Description

This namespace provides c++ wrappers for `signal(3)`.

### 10.13.2 Class Documentation

#### 10.13.2.1 struct sys::sig::error

The domain error type.

## 10.14 timing Namespace Reference

Namespace for time management related functionality.

### Classes

- class [Cycle](#)  
*Implements an interruptible cyclic sleeping functor.*

#### 10.14.1 Detailed Description

Namespace for time management related functionality.

## 10.15 types Namespace Reference

A collection of type aliases.

### Typedefs

- typedef std::chrono::milliseconds [ms](#)  
*Millisecond type for polling intervals.*
- typedef int [coreid\\_t](#)  
*Type for CPU core indexing.*
- typedef unsigned long [cptime\\_t](#)  
*Type for load counting.*
- typedef unsigned int [mhz\\_t](#)  
*Type for CPU frequencies in MHz.*
- typedef int [decikelvin\\_t](#)  
*Type for temperatures in dK.*

#### 10.15.1 Detailed Description

A collection of type aliases.

#### 10.15.2 Typedef Documentation



## 10.15.2.1 cptime\_t

```
typedef unsigned long types::cptime_t
```

Type for load counting.

According to src/sys/kern/kern\_clock.c the type is `long` (an array of loads `long[CPUSTATES]` is defined). But in order to have defined wrapping characteristics `unsigned long` will be used here.

## 10.16 utility Namespace Reference

A collection of generally useful functions.

## Namespaces

- [literals](#)

*Contains literals.*

## Classes

- class [Formatter](#)

*A formatting wrapper around string literals.*

## Functions

- `template<typename T, size_t Count>`  
`constexpr size_t countof (T(&)[Count])`  
*Like `sizeof()`, but it returns the number of elements an array consists of instead of the number of bytes.*
- `template<typename... Args>`  
`void sprintf (Args...)`  
*This is a safeguard against accidentally using `sprintf()`.*
- `template<size_t Size, typename... Args>`  
`int sprintf\_safe (char(&dst)[Size], char const *const format, Args const ... args)`  
*A wrapper around `snprintf()` that automatically pulls in the destination buffer size.*
- `template<class ET, typename VT = typename std::underlying_type<ET>::type>`  
`constexpr VT to\_value (ET const op)`  
*Casts an enum to its underlying value.*

## 10.16.1 Detailed Description

A collection of generally useful functions.

## 10.16.2 Function Documentation

## 10.16.2.1 countof()

```
template<typename T, size_t Count>
constexpr size_t utility::countof (
    T(&) [Count] )
```

Like `sizeof()`, but it returns the number of elements an array consists of instead of the number of bytes.

**Template Parameters**

<i>T, Count</i>	The type and number of array elements
-----------------	---------------------------------------

**Returns**

The number of array entries

**10.16.2.2 `sprintf()`**

```
template<typename... Args>
void utility::sprintf (
    Args...    ) [inline]
```

This is a safeguard against accidentally using `sprintf()`.

Using it triggers a `static_assert()`, preventing compilation.

**Template Parameters**

<i>Args</i>	Catch all arguments
-------------	---------------------

**10.16.2.3 `sprintf_safe()`**

```
template<size_t Size, typename... Args>
int utility::sprintf_safe (
    char(&) dst[Size],
    char const *const format,
    Args const ... args ) [inline]
```

A wrapper around `snprintf()` that automatically pulls in the destination buffer size.

**Template Parameters**

<i>Size</i>	The destination buffer size
<i>Args</i>	The types of the arguments

**Parameters**

<i>dst</i>	A reference to the destination buffer
<i>format</i>	A printf style formatting string
<i>args</i>	The printf arguments

**Returns**

The number of characters in the resulting string, regardless of the available space

**10.16.2.4 to\_value()**

```
template<class ET , typename VT = typename std::underlying_type<ET>::type>
constexpr VT utility::to_value (
    ET const op )
```

Casts an enum to its underlying value.

**Template Parameters**

<i>ET, VT</i>	The enum and value type
---------------	-------------------------

**Parameters**

<i>op</i>	The operand to convert
-----------	------------------------

**Returns**

The integer representation of the operand

**10.17 utility::literals Namespace Reference**

Contains literals.

**Functions**

- `std::string operator""_s` (char const \*const op, size\_t const size)  
*A string literal operator equivalent to the `operator "" s` literal provided by C++14 in `<string>`.*
- `constexpr Formatter< 16384 > operator""_fmt` (char const \*const fmt, size\_t const)  
*Literal to convert a string literal to a `Formatter` instance.*

**10.17.1 Detailed Description**

Contains literals.

**10.17.2 Function Documentation****10.17.2.1 operator""\_fmt()**

```
constexpr Formatter<16384> utility::literals::operator""_fmt (
    char const *const fmt,
    size_t const )
```

Literal to convert a string literal to a `Formatter` instance.

**Parameters**

<i>fmt</i>	A printf style format string
<i>const</i>	Unused

**Returns**

A [Formatter](#) instance

**10.17.2.2 `operator""_s()`**

```
std::string utility::literals::operator""_s (
    char const *const op,
    size_t const size ) [inline]
```

A string literal operator equivalent to the `operator ""_s` literal provided by C++14 in `<string>`.

**Parameters**

<i>op</i>	The raw string to turn into an <code>std::string</code> object
<i>size</i>	The size of the raw string

**Returns**

An `std::string` instance

**11 Class Documentation****11.1 `anonymous_namespace{loadplay.cpp}::Callback< FunctionArgs >` Class Template Reference**

Implements a recursion safe `std::function` wrapper.

**Public Types**

- `typedef std::function< void(FunctionArgs...)>` [function\\_t](#)  
The callback function type.

**Public Member Functions**

- [Callback](#) ()  
Default constructor, creates a non-callable handle.
- [Callback](#) ([function\\_t](#) const &[callback](#))  
Construct from function.
- [Callback](#) ([function\\_t](#) &&[callback](#))  
Construct from temporary function.
- void [operator](#)() (FunctionArgs... args)  
Forward call to callback functions.

## Private Attributes

- `function_t callback`  
*Storage for the callback function.*
- `bool called {false}`  
*Set if this handle is currently in use.*

## 11.1.1 Detailed Description

```
template<typename... FunctionArgs>
class anonymous_namespace{loadplay.cpp}::Callback< FunctionArgs >
```

Implements a recursion safe std::function wrapper.

The purpose is to prevent recursive calls of a callback function handle, in cases when a callback function performs actions that cause a successive call of the callback function.

To avoid having to return a value when a successive function call occurs only functions returning void are valid callback functions.

This is not thread safe.

## Template Parameters

<i>FunctionArgs</i>	The argument types of the callback function
---------------------	---

## 11.1.2 Constructor &amp; Destructor Documentation

## 11.1.2.1 Callback() [1/2]

```
template<typename... FunctionArgs>
anonymous_namespace{loadplay.cpp}::Callback< FunctionArgs >::Callback (
    function_t const & callback ) [inline]
```

Construct from function.

## Parameters

<i>callback</i>	The callback function
-----------------	-----------------------

## 11.1.2.2 Callback() [2/2]

```
template<typename... FunctionArgs>
anonymous_namespace{loadplay.cpp}::Callback< FunctionArgs >::Callback (
    function_t && callback ) [inline]
```

Construct from temporary function.

## Parameters

<i>callback</i>	The callback function
-----------------	-----------------------

## 11.1.3 Member Function Documentation

## 11.1.3.1 operator()

```
template<typename... FunctionArgs>
void anonymous_namespace{loadplay.cpp}::Callback< FunctionArgs >::operator() (
    FunctionArgs... args ) [inline]
```

Forward call to callback functions.

## Parameters

<i>args</i>	The arguments to the callback function
-------------	--

## Exceptions

<i>std::bad_function_call</i>	In case this handler was default constructed or constructed from a nullptr
-------------------------------	--

The documentation for this class was generated from the following file:

- [loadplay.cpp](#)

## 11.2 timing::Cycle Class Reference

Implements an interruptible cyclic sleeping functor.

```
#include <Cycle.hpp>
```

## Public Member Functions

- bool [operator\(\)](#) () const  
*Completes an interrupted sleep cycle.*
- template<class... DurTraits>  
bool [operator\(\)](#) (std::chrono::duration< DurTraits... > const &cycleTime)  
*Sleep for the time required to complete the given cycle time.*

## Private Types

- using [clock](#) = std::chrono::steady\_clock  
*Use steady\_clock, avoid time jumps.*
- using [us](#) = std::chrono::microseconds  
*Shorthand for microseconds.*

## Private Attributes

- `std::chrono::time_point< clock > clk = clock::now()`  
The current time clock.

### 11.2.1 Detailed Description

Implements an interruptible cyclic sleeping functor.

Cyclic sleeping means that instead of having a fixed sleeping time, each sleep is timed to meet a fixed wakeup time. I.e. the waking rhythm does not drift with changing system loads.

The canonical way to do this in C++ is like this:

```
#include <chrono>
#include <thread>

int main() {
    std::chrono::milliseconds const ival{500};
    auto time = std::chrono::steady_clock::now();
    while (...something...) {
        std::this_thread::sleep_until(time += ival);
        ...do stuff...
    }
    return 0;
}
```

The issue is that you might want to install a signal handler to guarantee stack unwinding and `sleep_until()` will resume its wait after the signal handler completes.

The `Cycle` class offers you an interruptible sleep:

```
#include "Cycle.hpp"
#include <csignal>
...signal handlers...

int main() {
    std::chrono::milliseconds const ival{500};
    ...setup some signal handlers...
    timing::Cycle sleep;
    while (...something... && sleep(ival)) {
        ...do stuff...
    }
    return 0;
}
```

In the example the while loop is terminated if the `sleep()` is interrupted by a signal. Optionally the sleep cycle can be resumed:

```
timing::Cycle sleep;
while (...something...) {
    if (!sleep(ival)) {
        ...interrupted...
        while (!sleep());
    }
    ...do stuff...
}
```

Note there was a design decision between providing a cycle time to the constructor or providing it every cycle. The latter was chosen so the cycle time can be adjusted.

### 11.2.2 Member Function Documentation

#### 11.2.2.1 `operator()()` [1/2]

```
bool timing::Cycle::operator() ( ) const [inline]
```

Completes an interrupted sleep cycle.

I.e. if the last sleep cycle was 500 ms and the sleep was interrupted 300 ms into the cycle, this would sleep for the remaining 200 ms unless interrupted.



## Return values

<i>true</i>	Sleep completed uninterrupted
<i>false</i>	Sleep was interrupted

## 11.2.2.2 operator() [ 2 / 2 ]

```
template<class... DurTraits>
bool timing::Cycle::operator() (
    std::chrono::duration< DurTraits... > const & cycleTime ) [inline]
```

Sleep for the time required to complete the given cycle time.

I.e. if the time since the last sleep cycle was 12 ms and the given cycleTime was 500 ms, the actual sleeping time would be 488 ms.

## Template Parameters

<i>DurTraits</i>	The traits of the duration type
------------------	---------------------------------

## Parameters

<i>cycleTime</i>	The duration of the cycle to complete
------------------	---------------------------------------

## Return values

<i>true</i>	Command completed uninterrupted
<i>false</i>	Command was interrupted

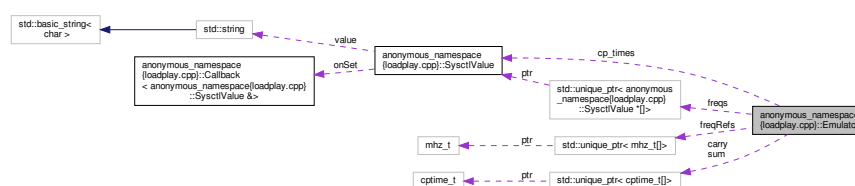
The documentation for this class was generated from the following file:

- [Cycle.hpp](#)

## 11.3 anonymous\_namespace{loadplay.cpp}::Emulator Class Reference

Instances of this class represent an emulator session.

Collaboration diagram for anonymous\_namespace{loadplay.cpp}::Emulator:



## Public Member Functions

- **Emulator** (bool const &die)  
*The constructor initialises all the members necessary for emulation.*
- void **operator()** ()  
*Performs load emulation and prints statistics std::cout.*

## Private Attributes

- bool const & **die**  
*A reference to a bool that tells the emulator to die.*
- int const **ncpu** = sysctls[{CTL\_HW, HW\_NCPU}].get<int>()  
*The hw.ncpu value.*
- std::unique\_ptr< **SysctlValue** \*[]> **freqs** {new **SysctlValue** \*[ncpu]}
- std::unique\_ptr< mhz\_t[]> **freqRefs** {new mhz\_t[ncpu]}
- **SysctlValue** & **cp\_times** = sysctls[sysctls.getMib(CP\_TIMES)]  
*The kern.cp\_times sysctl handler.*
- std::unique\_ptr< cptime\_t[]> **sum** {new cptime\_t[CPUSTATES \* ncpu]}
- std::unique\_ptr< cptime\_t[]> **carry** {new cptime\_t[ncpu]}
- size\_t const **size** = CPUSTATES \* ncpu \* sizeof(cptime\_t)  
*The size of the kern.cp\_times buffer.*

### 11.3.1 Detailed Description

Instances of this class represent an emulator session.

This should be run in its own thread and expects the sysctl table to be complete.

### 11.3.2 Constructor & Destructor Documentation

#### 11.3.2.1 Emulator()

```
anonymous_namespace{loadplay.cpp}::Emulator::Emulator (
    bool const & die ) [inline]
```

The constructor initialises all the members necessary for emulation.

It also prints the column headers on stdout.

#### Exceptions

<code>std::out_of_range</code>	In case one of the required sysctls is missing
--------------------------------	--

## Parameters

<i>die</i>	If the referenced bool is true, emulation is terminated prematurely
------------	---

## 11.3.3 Member Function Documentation

## 11.3.3.1 operator()

```
void anonymous_namespace{loadplay.cpp}::Emulator::operator() ( ) [inline]
```

Performs load emulation and prints statistics `std::cout`.

Reads `std::cin` to pull in load changes and updates the `kern.cp_times sysctl` to represent the current state.

When it runs out of load changes it terminates emulation and sends a SIGINT to the process.

## 11.3.4 Member Data Documentation

## 11.3.4.1 freqs

```
std::unique_ptr<SysctlValue * []> anonymous_namespace{loadplay.cpp}::Emulator::freqs {new  
SysctlValue *[ncpu]{} } [private]
```

Pointers to the `dev.cpu`.

d.freq handlers.

The documentation for this class was generated from the following file:

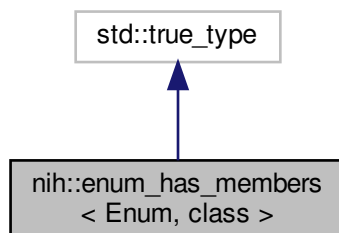
- [loadplay.cpp](#)

## 11.4 nih::enum\_has\_members< Enum, class > Struct Template Reference

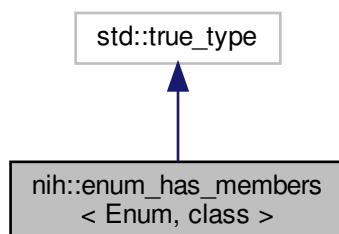
Tests whether the given enum provides all the required definitions.

```
#include <Options.hpp>
```

Inheritance diagram for nih::enum\_has\_members< Enum, class >:



Collaboration diagram for nih::enum\_has\_members< Enum, class >:



### 11.4.1 Detailed Description

```
template<class Enum, class = void>
struct nih::enum_has_members< Enum, class >
```

Tests whether the given enum provides all the required definitions.

The Options<> template expects the provided enum to provide the following members:

Member	Description
OPT_UNKNOWN	An undefined option (long or short) was encountered
OPT_NOOPT	The encountered command line argument is not an option
OPT_DASH	A single dash "-" was encountered
OPT_LDASH	Double dashes "--" were encountered
OPT_DONE	All command line arguments have been processed

## Template Parameters

<i>Enum</i>	An enum or enum class representing the available options
-------------	--

The documentation for this struct was generated from the following file:

- [Options.hpp](#)

## 11.5 utility::Formatter&lt; BufSize &gt; Class Template Reference

A formatting wrapper around string literals.

```
#include <utility.hpp>
```

## Public Member Functions

- constexpr [Formatter](#) (char const \*const [fmt](#))  
*Construct from string literal.*
- template<typename... ArgTs>  
std::string [operator\(\)](#) (ArgTs const &... args) const  
*Returns a formatted string.*

## Private Attributes

- char const \*const [fmt](#)  
*Pointer to the string literal.*

## 11.5.1 Detailed Description

```
template<size_t BufSize>
class utility::Formatter< BufSize >
```

A formatting wrapper around string literals.

Overloads operator (), which treats the string as a printf formatting string, the arguments represent the data to format.

In combination with the literal `_fmt`, it can be used like this:

```
std::cout << "%-15.15s %#018p\n"_fmt ("Address:", this);
```

## Template Parameters

<i>BufSize</i>	The buffer size for formatting, resulting strings cannot grow beyond <code>BufSize - 1</code>
----------------	---

## 11.5.2 Member Function Documentation

### 11.5.2.1 operator()

```
template<size_t BufSize>
template<typename... ArgTs>
std::string utility::Formatter< BufSize >::operator() (
    ArgTs const &... args ) const [inline]
```

Returns a formatted string.

#### Template Parameters

<i>ArgTs</i>	Variadic argument types
--------------	-------------------------

#### Parameters

<i>args</i>	Variadic arguments
-------------	--------------------

#### Returns

An std::string formatted according to fmt

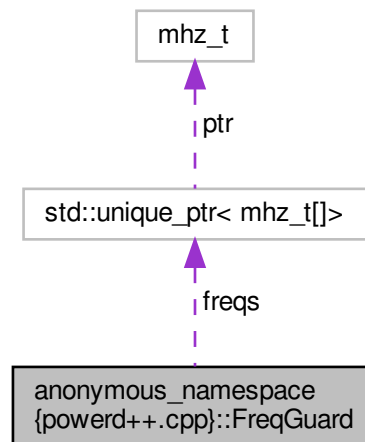
The documentation for this class was generated from the following file:

- [utility.hpp](#)

## 11.6 anonymous\_namespace{powerd++.cpp}::FreqGuard Class Reference

A core frequency guard.

Collaboration diagram for anonymous\_namespace{powerd++.cpp}::FreqGuard:



#### Public Member Functions

- [FreqGuard \(\)](#)  
*Read and write all core frequencies, may throw.*
- [~FreqGuard \(\)](#)  
*Restore all core frequencies.*

#### Private Attributes

- `std::unique_ptr< mhz_t[]>` [freqs](#)  
*The list of initial frequencies.*

#### 11.6.1 Detailed Description

A core frequency guard.

This uses the RAII pattern to achieve two things:

- Upon creation it reads and writes all controlling cores
- Upon destruction it sets all cores to the maximum frequencies

The documentation for this class was generated from the following file:

- [powerd++.cpp](#)

## 11.7 anonymous\_namespace{loadplay.cpp}::Hold< T > Class Template Reference

Sets a referenced variable to a given value and restores it when going out of context.

### Public Member Functions

- [Hold](#) (T &[ref](#), T const value)  
*The constructor sets the referenced variable to the given value.*
- [~Hold](#) ()  
*Restores the original value.*

### Private Attributes

- T const [restore](#)  
*The original value.*
- T & [ref](#)  
*Reference to the variable.*

#### 11.7.1 Detailed Description

```
template<typename T>
class anonymous_namespace{loadplay.cpp}::Hold< T >
```

Sets a referenced variable to a given value and restores it when going out of context.

#### Template Parameters

<i>T</i>	The type of the value to hold
----------	-------------------------------

#### 11.7.2 Constructor & Destructor Documentation

##### 11.7.2.1 Hold()

```
template<typename T >
anonymous_namespace{loadplay.cpp}::Hold< T >::Hold (
    T & ref,
    T const value ) [inline]
```

The constructor sets the referenced variable to the given value.

#### Parameters

<i>ref</i>	The variable to hold and restore
<i>value</i>	The value to set the variable to



## 11.7.3 Member Data Documentation

## 11.7.3.1 ref

```
template<typename T >
T& anonymous_namespace{loadplay.cpp}::Hold< T >::ref [private]
```

Reference to the variable.

## 11.7.3.2 restore

```
template<typename T >
T const anonymous_namespace{loadplay.cpp}::Hold< T >::restore [private]
```

The original value.

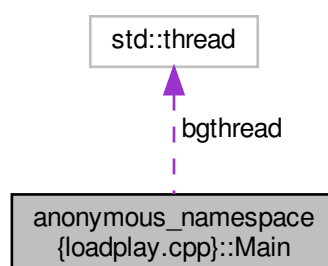
The documentation for this class was generated from the following file:

- [loadplay.cpp](#)

## 11.8 anonymous\_namespace{loadplay.cpp}::Main Class Reference

Singleton class representing the main execution environment.

Collaboration diagram for anonymous\_namespace{loadplay.cpp}::Main:



## Public Member Functions

- [Main](#) ()  
The constructor starts up the emulation.
- [~Main](#) ()  
Clean up the background emulation thread.

## Private Attributes

- `std::thread` [bgthread](#)  
*The background emulation thread.*
- `bool` [die](#) {false}  
*Used to request premature death from the emulation thread.*

### 11.8.1 Detailed Description

Singleton class representing the main execution environment.

### 11.8.2 Constructor & Destructor Documentation

#### 11.8.2.1 Main()

```
anonymous_namespace{loadplay.cpp}::Main::Main ( ) [inline]
```

The constructor starts up the emulation.

- Read the headers from `std::cin` and populate `sysctls`
- Ensure the existence of all required `sysctls`
- Spawn an [Emulator](#) instance in its own thread

The documentation for this class was generated from the following file:

- [loadplay.cpp](#)

## 11.9 anonymous\_namespace{loadplay.cpp}::mib\_t Struct Reference

Represents MIB, but wraps it to provide the necessary operators to use it as an `std::map` key.

## Public Member Functions

- `template<typename... Ints>`  
`constexpr` [mib\\_t](#) (Ints const ... ints)  
*Construct a mib with the given number of arguments.*
- [mib\\_t](#) (int const \*const [mibs](#), `u_int` const len)  
*Initialise from a pointer to an int array.*
- `bool` [operator==](#) ([mib\\_t](#) const &op) const  
*Equality operator required by std::map.*
- `bool` [operator<](#) ([mib\\_t](#) const &op) const  
*Less than operator required by std::map.*
- [operator int \\*](#) ()  
*Cast to int \* for value access.*
- [operator int const \\*](#) () const  
*Cast to int const \* for value access.*

## Public Attributes

- int [mibs](#) [CTL\_MAXNAME]  
*The mib values.*

## 11.9.1 Detailed Description

Represents MIB, but wraps it to provide the necessary operators to use it as an std::map key.

## 11.9.2 Constructor &amp; Destructor Documentation

## 11.9.2.1 mib\_t() [1/2]

```
template<typename...  Ints>
constexpr anonymous_namespace{loadplay.cpp}::mib_t::mib_t (
    Ints const ...  ints ) [inline]
```

Construct a mib with the given number of arguments.

## Template Parameters

<i>Ints</i>	A list of integer types
-------------	-------------------------

## Parameters

<i>ints</i>	A list of integers to create a mib from
-------------	---

## 11.9.2.2 mib\_t() [2/2]

```
anonymous_namespace{loadplay.cpp}::mib_t::mib_t (
    int const *const mibs,
    u_int const len ) [inline]
```

Initialise from a pointer to an int array.

## Parameters

<i>mibs, len</i>	The array and its length
------------------	--------------------------

## 11.9.3 Member Function Documentation

**11.9.3.1 operator int \*()**

```
anonymous_namespace{loadplay.cpp}::mib_t::operator int * ( ) [inline]
```

Cast to `int *` for value access.

**Returns**

A pointer to mibs

**11.9.3.2 operator int const \*()**

```
anonymous_namespace{loadplay.cpp}::mib_t::operator int const * ( ) const [inline]
```

Cast to `int const *` for value access.

**Returns**

A pointer to mibs

**11.9.3.3 operator<()**

```
bool anonymous_namespace{loadplay.cpp}::mib_t::operator< (
    mib_t const & op ) const [inline]
```

Less than operator required by `std::map`.

**Parameters**

<i>op</i>	Another <code>mib_t</code> instance
-----------	-------------------------------------

**Returns**

Whether this mib is less than the given one

**11.9.3.4 operator==( )**

```
bool anonymous_namespace{loadplay.cpp}::mib_t::operator==(
    mib_t const & op ) const [inline]
```

Equality operator required by `std::map`.

**Parameters**

<i>op</i>	Another <code>mib_t</code> instance
-----------	-------------------------------------

### Returns

Whether all values in this and the given mib are equal

The documentation for this struct was generated from the following file:

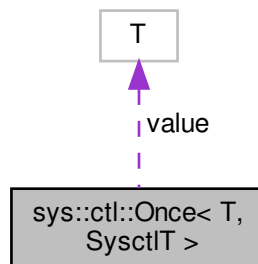
- [loadplay.cpp](#)

## 11.10 sys::ctl::Once< T, SysctlT > Class Template Reference

A read once representation of a [Sysctl](#).

```
#include <sysctl.hpp>
```

Collaboration diagram for sys::ctl::Once< T, SysctlT >:



### Public Member Functions

- [Once](#) (T const &[value](#), SysctlT const &[sysctl](#)) noexcept  
*The constructor tries to read and store the requested sysctl.*
- [operator T const & \(\) const](#)  
*Return a const reference to the value.*

### Private Attributes

- T [value](#)  
*The sysctl value read upon construction.*

### 11.10.1 Detailed Description

```
template<typename T, class SysctlT>
class sys::ctl::Once< T, SysctlT >
```

A read once representation of a [Sysctl](#).

This reads a sysctl once upon construction and always returns that value. It does not support assignment.

This class is intended for sysctls that are not expected to change, such as `hw.ncpu`. A special property of this class is that the constructor does not throw and takes a default value in case reading the sysctl fails.

```
// Read number of CPU cores, assume 1 on failure:
Once<coreid_t, Sysctl<2>> ncpu{1, {CTL_HW, HW_NCPU}};
// Equivalent:
int hw_ncpu;
try {
    Sysctl<2>{CTL_HW, HW_NCPU}.get(hw_ncpu);
} catch (sys::sc_error<error>) {
    hw_ncpu = 1;
}
```

#### Template Parameters

<i>T</i>	The type to represent the sysctl as
<i>SysctlT</i>	The <a href="#">Sysctl</a> type

### 11.10.2 Constructor & Destructor Documentation

#### 11.10.2.1 Once()

```
template<typename T, class SysctlT>
sys::ctl::Once< T, SysctlT >::Once (
    T const & value,
    SysctlT const & sysctl ) [inline], [noexcept]
```

The constructor tries to read and store the requested sysctl.

If reading the requested sysctl fails for any reason, the given value is stored instead.

#### Parameters

<i>value</i>	The fallback value
<i>sysctl</i>	The sysctl to represent

### 11.10.3 Member Function Documentation

## 11.10.3.1 operator T const &amp;()

```
template<typename T, class SysctlT>
sys::ctl::Once< T, SysctlT >::operator T const & ( ) const [inline]
```

Return a const reference to the value.

## Returns

A const reference to the value

The documentation for this class was generated from the following file:

- [sys/sysctl.hpp](#)

## 11.11 nih::Options&lt; Enum, DefCount &gt; Class Template Reference

An instance of this class offers operators to retrieve command line options and arguments.

```
#include <Options.hpp>
```

## Public Member Functions

- [Options](#) (int const [argc](#), char const \*const [argv](#)[], char const \*const [usage](#), [Option](#)< Enum > const (&defs)[DefCount])  
*Construct an options functor.*
- Enum [operator](#)() ()  
*Returns the next option from the command line arguments.*
- char const \* [operator](#)[] (int const i) const  
*Retrieve arguments to the current option.*
- std::string [usage](#) () const  
*Returns a string for usage output, created from the option definitions.*

## Private Member Functions

- Enum [get](#) (char const ch)  
*Finds the short option matching the given character.*
- Enum [get](#) (char const \*const str)  
*Finds the long option matching the given string.*

## Static Private Member Functions

- static char const \* [removePath](#) (char const \*const file)  
*Returns a pointer to the file name portion of the given string.*
- static bool [match](#) (char const \*const lstr, char const \*const rstr)  
*Returns true if the given strings match.*
- static bool [bmatch](#) (char const \*const lstr, char const \*const rstr)  
*Returns true if one of the given strings matches the beginning of the other.*

## Private Attributes

- `int const argc`  
*The number of command line arguments.*
- `char const *const *const argv`  
*The command line arguments.*
- `char const *const usageStr`  
*A string literal for the [usage\(\)](#) output.*
- `Option< Enum > const (& defs ) [DefCount]`  
*A reference to the option definitions.*
- `Option< Enum > const expose`  
*The option definition to use for special options that are exposed by the [] operator.*
- `int argi`  
*The index of the command line argument containing the current option.*
- `char const * argp`  
*Points to the current short option character.*
- `Option< Enum > const * current`  
*Points to the current option definition.*

### 11.11.1 Detailed Description

```
template<class Enum, size_t DefCount>
class nih::Options< Enum, DefCount >
```

An instance of this class offers operators to retrieve command line options and arguments.

Instantiate with [make\\_Options\(\)](#) to infer template parameters automatically.

Check the `operator ()` and `operator []` for use.

#### Template Parameters

<i>Enum</i>	An enum or enum class matching the requirements set by <a href="#">enum_has_members</a>
<i>DefCount</i>	The number of option definitions

### 11.11.2 Constructor & Destructor Documentation

#### 11.11.2.1 Options()

```
template<class Enum , size_t DefCount>
nih::Options< Enum, DefCount >::Options (
    int const argc,
    char const *const argv[],
    char const *const usage,
    Option< Enum > const (&) defs[DefCount] ) [inline]
```

Construct an options functor.



## Parameters

<i>argc,argv</i>	The command line arguments
<i>usage</i>	A usage string following "usage: progname "
<i>defs</i>	An array of option definitions

## 11.11.3 Member Function Documentation

## 11.11.3.1 bmatch()

```
template<class Enum , size_t DefCount>
static bool nih::Options< Enum, DefCount >::bmatch (
    char const *const lstr,
    char const *const rstr ) [inline], [static], [private]
```

Returns true if one of the given strings matches the beginning of the other.

## Parameters

<i>lstr,rstr</i>	Two 0 terminated strings
------------------	--------------------------

## Return values

<i>true</i>	The shorter string matches the beginning of the other string
<i>false</i>	The strings do not match

## 11.11.3.2 get() [1/2]

```
template<class Enum , size_t DefCount>
Enum nih::Options< Enum, DefCount >::get (
    char const ch ) [inline], [private]
```

Finds the short option matching the given character.

## Parameters

<i>ch</i>	The short option to find
-----------	--------------------------

## Returns

The option or OPT\_UNKNOWN

### 11.11.3.3 `get()` [2/2]

```
template<class Enum , size_t DefCount>
Enum nih::Options< Enum, DefCount >::get (
    char const *const str ) [inline], [private]
```

Finds the long option matching the given string.

#### Parameters

<i>str</i>	The long option to find
------------	-------------------------

#### Returns

The option or OPT\_UNKNOWN

### 11.11.3.4 `match()`

```
template<class Enum , size_t DefCount>
static bool nih::Options< Enum, DefCount >::match (
    char const *const lstr,
    char const *const rstr ) [inline], [static], [private]
```

Returns true if the given strings match.

#### Parameters

<i>lstr,rstr</i>	Two 0 terminated strings
------------------	--------------------------

#### Return values

<i>true</i>	The given strings match
<i>false</i>	The strings do not match

### 11.11.3.5 `operator()()`

```
template<class Enum , size_t DefCount>
Enum nih::Options< Enum, DefCount >::operator() ( ) [inline]
```

Returns the next option from the command line arguments.

#### Returns

An Enum member representing the current option

## Return values

<i>OPT_UNKNOWN</i>	An option that was not in the list of option definitions was encountered
<i>OPT_NOOPT</i>	An argument that is not an option was encountered
<i>OPT_DASH</i>	A lone dash "-" was encountered
<i>OPT_LDASH</i>	A lone long dash "--" was encountered
<i>OPT_DONE</i>	All arguments have been processed

## 11.11.3.6 operator[]()

```
template<class Enum , size_t DefCount>
char const* nih::Options< Enum, DefCount >::operator[] (
    int const i ) const [inline]
```

Retrieve arguments to the current option.

The string containing the current option is returned with  $i = 0$ , the arguments following the option with greater values of  $i$ .

When no more arguments are left the empty string is returned.

## Parameters

<i>i</i>	The index of the argument to retrieve
----------	---------------------------------------

## Returns

The option or one of its arguments

## 11.11.3.7 removePath()

```
template<class Enum , size_t DefCount>
static char const* nih::Options< Enum, DefCount >::removePath (
    char const *const file ) [inline], [static], [private]
```

Returns a pointer to the file name portion of the given string.

## Parameters

<i>file</i>	The string containing the path to the file
-------------	--

## Returns

A pointer to the file name portion of the path

### 11.11.3.8 usage()

```
template<class Enum , size_t DefCount>
std::string nih::Options< Enum, DefCount >::usage ( ) const [inline]
```

Returns a string for usage output, created from the option definitions.

#### Returns

A usage string for printing on the CLI

## 11.11.4 Member Data Documentation

### 11.11.4.1 expose

```
template<class Enum , size_t DefCount>
Option<Enum> const nih::Options< Enum, DefCount >::expose [private]
```

#### Initial value:

```
{
    Enum::OPT_NOOPT, 0, nullptr, nullptr, nullptr
}
```

The option definition to use for special options that are exposed by the [] operator.

The documentation for this class was generated from the following file:

- [Options.hpp](#)

## 11.12 sys::pid::Pidfile Class Reference

A wrapper around the pidfile\_\* family of commands implementing the RAII pattern.

```
#include <pidfile.hpp>
```

#### Public Member Functions

- [Pidfile](#) (char const \*const pfname, mode\_t const mode)  
*Attempts to open the pidfile.*
- [~Pidfile](#) ()  
*Removes the pidfile.*
- pid\_t [other](#) ()  
*Returns the PID of the other process holding the lock.*
- void [write](#) ()  
*Write PID to the file, should be called after [daemon\(\)](#).*

## Private Attributes

- `pid_t otherpid`  
*In case of failure to acquire the lock, the PID of the other process holding it is stored here.*
- `pidfh * pfh`  
*Pointer to the pidfile state data structure.*

## 11.12.1 Detailed Description

A wrapper around the `pidfile_*` family of commands implementing the RAII pattern.

## 11.12.2 Constructor &amp; Destructor Documentation

## 11.12.2.1 Pidfile()

```
sys::pid::Pidfile::Pidfile (
    char const *const pfname,
    mode_t const mode ) [inline]
```

Attempts to open the pidfile.

## Parameters

<code>pfname,mode</code>	Arguments to <a href="#">pidfile_open()</a>
--------------------------	---

## Exceptions

<code>pid_t</code>	Throws the PID of the other process already holding the requested pidfile
<code>sys::sc_error&lt;error&gt;</code>	Throws with the errno of <a href="#">pidfile_open()</a>

## 11.12.3 Member Function Documentation

## 11.12.3.1 write()

```
void sys::pid::Pidfile::write ( ) [inline]
```

Write PID to the file, should be called after [daemon\(\)](#).

## Exceptions

<code>sys::sc_error&lt;error&gt;</code>	Throws with the errno of <a href="#">pidfile_write()</a>
---	--

### 11.12.4 Member Data Documentation

#### 11.12.4.1 pfh

```
pidfh* sys::pid::Pidfile::pfh [private]
```

Pointer to the pidfile state data structure.

Thus is allocated by [pidfile\\_open\(\)](#) and assumedly freed by [pidfile\\_remove\(\)](#).

The documentation for this class was generated from the following file:

- [sys/pidfile.hpp](#)

### 11.13 sys::sc\_error< Domain > Struct Template Reference

Can be thrown by syscall function wrappers if the function returned with an error.

```
#include <error.hpp>
```

#### Public Member Functions

- [operator int \(\)](#) const  
*Cast to integer.*
- [char const \\* c\\_str \(\)](#) const  
*Return c style string.*

#### Public Attributes

- [int error](#)  
*The errno set by the native C function.*

#### 11.13.1 Detailed Description

```
template<class Domain>
struct sys::sc_error< Domain >
```

Can be thrown by syscall function wrappers if the function returned with an error.

This is its own type for easy catching, but implicitly casts to int for easy comparison.

#### Template Parameters

<i>Domain</i>	A type marking the domain the error comes from, e.g. <a href="#">sys::ctl::error</a>
---------------	--

### 11.13.2 Member Function Documentation

#### 11.13.2.1 c\_str()

```
template<class Domain >  
char const* sys::sc_error< Domain >::c_str ( ) const [inline]
```

Return c style string.

#### Returns

A string representation of the error

#### 11.13.2.2 operator int()

```
template<class Domain >  
sys::sc_error< Domain >::operator int ( ) const [inline]
```

Cast to integer.

#### Returns

The errno code

The documentation for this struct was generated from the following file:

- [sys/error.hpp](#)

## 11.14 sys::sig::Signal Class Reference

Sets up a given signal handler and restores the old handler when going out of scope.

```
#include <signal.hpp>
```

### Public Member Functions

- [Signal](#) (int const [sig](#), [sig\\_t](#) const [handler](#))  
*Sets up the given handler.*
- [~Signal](#) ()  
*Restore previous signal handler.*

### Private Attributes

- int const [sig](#)  
*The signal this handler is handling.*
- [sig\\_t](#) const [handler](#)  
*The previous signal handler.*

### 11.14.1 Detailed Description

Sets up a given signal handler and restores the old handler when going out of scope.

### 11.14.2 Constructor & Destructor Documentation

#### 11.14.2.1 Signal()

```
sys::sig::Signal::Signal (
    int const sig,
    sig_t const handler ) [inline]
```

Sets up the given handler.

#### Parameters

<i>sig</i>	The signal to set a handler for
<i>handler</i>	The signal handling function

#### Exceptions

<code>sys::sc_error&lt;error&gt;</code>	Throws with the errno of signal()
---	-----------------------------------

The documentation for this class was generated from the following file:

- [sys/signal.hpp](#)

## 11.15 sys::ctl::Sync< T, SysctlT > Class Template Reference

This is a wrapper around [Sysctl](#) that allows semantically transparent use of a sysctl.

```
#include <sysctl.hpp>
```

#### Public Member Functions

- constexpr [Sync](#) ()  
*The default constructor.*
- constexpr [Sync](#) (SysctlT const &[sysctl](#)) noexcept  
*The constructor copies the given [Sysctl](#) instance.*
- [Sync](#) & [operator=](#) (T const &value)  
*Transparently assigns values of type T to the represented [Sysctl](#) instance.*
- [operator T](#) () const  
*Implicitly cast to the represented type.*



## Private Attributes

- SysctlT [sysctl](#)  
*A sysctl to represent.*

## 11.15.1 Detailed Description

```
template<typename T, class SysctlT>
class sys::ctl::Sync< T, SysctlT >
```

This is a wrapper around [Sysctl](#) that allows semantically transparent use of a sysctl.

```
Sync<int, Sysctl<>> sndUnit{{"hw.snd.default_unit"}};
if (sndUnit != 3) {    // read from sysctl
    sndUnit = 3;      // assign to sysctl
}
```

Note that both assignment and read access (implemented through type casting to T) may throw an exception.

## Template Parameters

<i>T</i>	The type to represent the sysctl as
<i>SysctlT</i>	The <a href="#">Sysctl</a> type

## 11.15.2 Constructor &amp; Destructor Documentation

## 11.15.2.1 Sync() [1/2]

```
template<typename T, class SysctlT>
constexpr sys::ctl::Sync< T, SysctlT >::Sync ( ) [inline]
```

The default constructor.

This is available to defer initialisation to a later moment. This might be useful when initialising global or static instances by a character string represented name.

## 11.15.2.2 Sync() [2/2]

```
template<typename T, class SysctlT>
constexpr sys::ctl::Sync< T, SysctlT >::Sync (
    SysctlT const & sysctl ) [inline], [noexcept]
```

The constructor copies the given [Sysctl](#) instance.

## Parameters

<i>sysctl</i>	The <a href="#">Sysctl</a> instance to represent
---------------	--

### 11.15.3 Member Function Documentation

#### 11.15.3.1 operator T()

```
template<typename T, class SysctlT>
sys::ctl::Sync< T, SysctlT >::operator T ( ) const [inline]
```

Implicitly cast to the represented type.

#### Returns

Returns the value from the sysctl

#### 11.15.3.2 operator=()

```
template<typename T, class SysctlT>
Sync& sys::ctl::Sync< T, SysctlT >::operator= (
    T const & value ) [inline]
```

Transparently assigns values of type T to the represented [Sysctl](#) instance.

#### Parameters

<i>value</i>	The value to assign
--------------	---------------------

#### Returns

A self reference

The documentation for this class was generated from the following file:

- [sys/sysctl.hpp](#)

### 11.16 sys::ctl::Sysctl< MibDepth > Class Template Reference

Represents a sysctl MIB address.

```
#include <sysctl.hpp>
```

## Public Member Functions

- template<typename... Tail>  
constexpr [Sysctl](#) ([mib\\_t](#) const head, Tail const ... tail) noexcept  
*Initialise the MIB address directly.*
- void [get](#) (void \*const buf, size\_t const bufsize) const  
*Update the given buffer with a value retrieved from the sysctl.*
- template<typename T >  
void [get](#) (T &value) const  
*Update the given value with a value retrieved from the sysctl.*
- template<typename T >  
std::unique\_ptr< T[] > [get](#) () const  
*Retrieve an array from the sysctl address.*
- void [set](#) (void const \*const buf, size\_t const bufsize)  
*Update the the sysctl value with the given buffer.*
- template<typename T >  
void [set](#) (T const &value)  
*Update the the sysctl value with the given value.*

## Private Attributes

- [mib\\_t](#) [mib](#) [MibDepth]  
*Stores the MIB address.*

## 11.16.1 Detailed Description

```
template<size_t MibDepth = 0>
class sys::ctl::Sysctl< MibDepth >
```

Represents a sysctl MIB address.

It offers [set\(\)](#) and [get\(\)](#) methods to access these sysctls.

There are two ways of initialising a [Sysctl](#) instance, by symbolic name or by directly using the MIB address. The latter one only makes sense for sysctls with a fixed address, known at compile time, e.g. [Sysctl](#)<2>{CTL\_HW, HW\_NCPU} for "hw.ncpu". Check /usr/include/sys/sysctl.h for predefined MIBs.

For all other sysctls, symbolic names must be used. E.g. [Sysctl](#)<>{ "dev.cpu.0.freq" }. Creating a [Sysctl](#) from a symbolic name may throw.

Fixed address sysctls may be created using the [make\\_Sysctl\(\)](#) function, e.g. [make\\_Sysctl](#)(CTL\_HW, HW\_NCPU).

Instances created from symbolic names must use the [Sysctl](#)<0> specialisation, this can be done by omitting the template argument [Sysctl](#)<>.

## Template Parameters

<i>MibDepth</i>	The MIB level, e.g. "hw.ncpu" is two levels deep
-----------------	--

## 11.16.2 Constructor & Destructor Documentation

### 11.16.2.1 Sysctl()

```
template<size_t MibDepth = 0>
template<typename... Tail>
constexpr sys::ctl::Sysctl< MibDepth >::Sysctl (
    mib_t const head,
    Tail const ... tail ) [inline], [noexcept]
```

Initialise the MIB address directly.

Some important sysctl values have a fixed address that can be initialised at compile time with a noexcept guarantee.

Splitting the MIB address into head and tail makes sure that `Sysctl(char *)` does not match the template and is instead implicitly cast to invoke `Sysctl(char const *)`.

#### Template Parameters

<i>Tail</i>	The types of the trailing MIB address values (must be <code>mib_t</code> )
-------------	--

#### Parameters

<i>head, tail</i>	The mib
-------------------	---------

## 11.16.3 Member Function Documentation

### 11.16.3.1 get() [1/3]

```
template<size_t MibDepth = 0>
void sys::ctl::Sysctl< MibDepth >::get (
    void *const buf,
    size_t const bufsize ) const [inline]
```

Update the given buffer with a value retrieved from the sysctl.

#### Parameters

<i>buf, bufsize</i>	The target buffer and its size
---------------------	--------------------------------

#### Exceptions

<code>sys::sc_error&lt;error&gt;</code>	Throws if value retrieval fails or is incomplete, e.g. because the value does not fit into the target buffer
---	--

**11.16.3.2 get()** [2/3]

```
template<size_t MibDepth = 0>
template<typename T >
void sys::ctl::Sysctl< MibDepth >::get (
    T & value ) const [inline]
```

Update the given value with a value retrieved from the sysctl.

**Template Parameters**

<i>T</i>	The type store the sysctl value in
----------	------------------------------------

**Parameters**

<i>value</i>	A reference to the target value
--------------	---------------------------------

**Exceptions**

<i>sys::sc_error&lt;error&gt;</i>	Throws if value retrieval fails or is incomplete, e.g. because the value does not fit into the target type
-----------------------------------	--

**11.16.3.3 get()** [3/3]

```
template<size_t MibDepth = 0>
template<typename T >
std::unique_ptr<T[]> sys::ctl::Sysctl< MibDepth >::get ( ) const [inline]
```

Retrieve an array from the sysctl address.

This is useful to retrieve variable length sysctls, like character strings.

**Template Parameters**

<i>T</i>	The type stored in the array
----------	------------------------------

**Returns**

And array of T with the right length to store the whole sysctl value

**Exceptions**

<i>sys::sc_error&lt;error&gt;</i>	May throw if the size of the sysctl increases after the length was queried
-----------------------------------	--

**11.16.3.4 set()** [1/2]

```
template<size_t MibDepth = 0>
void sys::ctl::Sysctl< MibDepth >::set (
    void const *const buf,
    size_t const bufsize ) [inline]
```

Update the the sysctl value with the given buffer.

**Parameters**

<i>buf, bufsize</i>	The source buffer
---------------------	-------------------

**Exceptions**

<i>sys::sc_error&lt;error&gt;</i>	If the source buffer cannot be stored in the sysctl
-----------------------------------	---

**11.16.3.5 set()** [2/2]

```
template<size_t MibDepth = 0>
template<typename T >
void sys::ctl::Sysctl< MibDepth >::set (
    T const & value ) [inline]
```

Update the the sysctl value with the given value.

**Template Parameters**

<i>T</i>	The value type
----------	----------------

**Parameters**

<i>value</i>	The value to set the sysctl to
--------------	--------------------------------

The documentation for this class was generated from the following file:

- [sys/sysctl.hpp](#)

**11.17 sys::ctl::Sysctl< 0 > Class Template Reference**

This is a specialisation of [Sysctl](#) for sysctls using symbolic names.

```
#include <sysctl.hpp>
```

## Public Member Functions

- constexpr [Sysctl](#) ()  
*The default constructor.*
- [Sysctl](#) (char const \*const name)  
*Initialise the MIB address from a character string.*
- void [get](#) (void \*const buf, size\_t const bufsize) const  
*Update the given buffer with a value retrieved from the sysctl.*
- template<typename T >  
void [get](#) (T &value) const  
*Update the given value with a value retrieved from the sysctl.*
- template<typename T >  
std::unique\_ptr< T[] > [get](#) () const  
*Retrieve an array from the sysctl address.*
- void [set](#) (void const \*const buf, size\_t const bufsize)  
*Update the the sysctl value with the given buffer.*
- template<typename T >  
void [set](#) (T const &value)  
*Update the the sysctl value with the given value.*

## Private Attributes

- [mib\\_t mib](#) [CTL\_MAXNAME]  
*Stores the MIB address.*
- [size\\_t depth](#)  
*The MIB depth.*

## 11.17.1 Detailed Description

```
template<>
class sys::ctl::Sysctl< 0 >
```

This is a specialisation of [Sysctl](#) for sysctls using symbolic names.

A [Sysctl](#) instance created with the default constructor is uninitialised, initialisation can be deferred to a later moment by using copy assignment. This can be used to create globals but construct them inline where exceptions can be handled.

## 11.17.2 Constructor &amp; Destructor Documentation

## 11.17.2.1 Sysctl() [1/2]

```
constexpr sys::ctl::Sysctl< 0 >::Sysctl ( ) [inline]
```

The default constructor.

This is available to defer initialisation to a later moment.

## 11.17.2.2 Sysctl() [2/2]

```
sys::ctl::Sysctl< 0 >::Sysctl (
    char const *const name ) [inline]
```

Initialise the MIB address from a character string.

**Parameters**

<i>name</i>	The symbolic name of the sysctl
-------------	---------------------------------

**Exceptions**

<code>sys::sc_error&lt;error&gt;</code>	May throw an exception if the addressed sysctl does not exist or if the address is too long to store
---	--

**11.17.3 Member Function Documentation****11.17.3.1 get()** [1/3]

```
void sys::ctl::Sysctl< 0 >::get (
    void *const buf,
    size_t const bufsize ) const [inline]
```

Update the given buffer with a value retrieved from the sysctl.

**Parameters**

<i>buf, bufsize</i>	The target buffer and its size
---------------------	--------------------------------

**Exceptions**

<code>sys::sc_error&lt;error&gt;</code>	Throws if value retrieval fails or is incomplete, e.g. because the value does not fit into the target buffer
---	--

**11.17.3.2 get()** [2/3]

```
template<typename T >
void sys::ctl::Sysctl< 0 >::get (
    T & value ) const [inline]
```

Update the given value with a value retrieved from the sysctl.

**Template Parameters**

<i>T</i>	The type store the sysctl value in
----------	------------------------------------

**Parameters**

<i>value</i>	A reference to the target value
--------------	---------------------------------



## Exceptions

<code>sys::sc_error&lt;error&gt;</code>	Throws if value retrieval fails or is incomplete, e.g. because the value does not fit into the target type
---	--

## 11.17.3.3 get() [3/3]

```
template<typename T >
std::unique_ptr<T[]> sys::ctl::Sysctl< 0 >::get ( ) const [inline]
```

Retrieve an array from the sysctl address.

This is useful to retrieve variable length sysctls, like character strings.

## Template Parameters

<code>T</code>	The type stored in the array
----------------	------------------------------

## Returns

And array of T with the right length to store the whole sysctl value

## Exceptions

<code>sys::sc_error&lt;error&gt;</code>	May throw if the size of the sysctl increases after the length was queried
---	--

## 11.17.3.4 set() [1/2]

```
void sys::ctl::Sysctl< 0 >::set (
    void const *const buf,
    size_t const bufsize ) [inline]
```

Update the the sysctl value with the given buffer.

## Parameters

<code>buf, bufsize</code>	The source buffer
---------------------------	-------------------

## Exceptions

<code>sys::sc_error&lt;error&gt;</code>	If the source buffer cannot be stored in the sysctl
---	---

### 11.17.3.5 set() [2/2]

```
template<typename T >
void sys::ctl::Sysctl< 0 >::set (
    T const & value ) [inline]
```

Update the the sysctl value with the given value.

#### Template Parameters

<i>T</i>	The value type
----------	----------------

#### Parameters

<i>value</i>	The value to set the sysctl to
--------------	--------------------------------

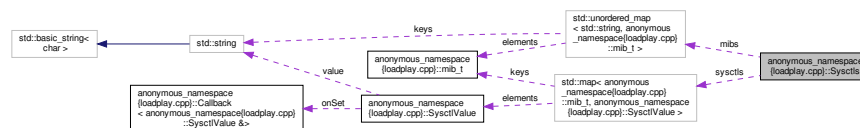
The documentation for this class was generated from the following file:

- [sys/sysctl.hpp](#)

## 11.18 anonymous\_namespace{loadplay.cpp}::Sysctls Class Reference

Singleton class representing the sysctl table for this library.

Collaboration diagram for anonymous\_namespace{loadplay.cpp}::Sysctls:



#### Public Member Functions

- void [addValue](#) ([mib\\_t](#) const &mib, std::string const &value)  
*Add a value to the sysctls map.*
- void [addValue](#) (std::string const &name, std::string const &value)  
*Add a value to the sysctls map.*
- [mib\\_t](#) const & [getMib](#) (std::string const &name) const  
*Returns a mib for a given symbolic name.*
- [SysctlValue](#) & [operator\[\]](#) ([mib\\_t](#) const &mib)  
*Returns a reference to a sysctl value container.*

#### Private Types

- typedef std::lock\_guard< decltype(mtx)> [lock\\_guard](#)  
*The appropriate lock guard type for mtx.*

## Private Attributes

- std::mutex [mtx](#)  
*A simple mutex.*
- std::unordered\_map< std::string, [mib\\_t](#) > [mibs](#)  
*Maps name → mib.*
- std::map< [mib\\_t](#), [SysctlValue](#) > [sysctls](#)  
*Maps mib → (type, value).*

## 11.18.1 Detailed Description

Singleton class representing the sysctl table for this library.

## 11.18.2 Member Function Documentation

## 11.18.2.1 addValue() [1/2]

```
void anonymous_namespace{loadplay.cpp}::Sysctls::addValue (
    mib\_t const & mib,
    std::string const & value ) [inline]
```

Add a value to the sysctls map.

## Parameters

<i>mib</i>	The mib to add the value for
<i>value</i>	The value to store

## 11.18.2.2 addValue() [2/2]

```
void anonymous_namespace{loadplay.cpp}::Sysctls::addValue (
    std::string const & name,
    std::string const & value ) [inline]
```

Add a value to the sysctls map.

## Parameters

<i>name</i>	The symbolic name of the mib to add the value for
<i>value</i>	The value to store

## 11.18.2.3 getMib()

```
mib\_t const& anonymous_namespace{loadplay.cpp}::Sysctls::getMib (
```

```
std::string const & name ) const [inline]
```

Returns a mib for a given symbolic name.

#### Parameters

<i>name</i>	The MIB name
-------------	--------------

#### Returns

The MIB

#### 11.18.2.4 operator[]()

```
SysctlValue& anonymous_namespace{loadplay.cpp}::Sysctls::operator[] (
    mib_t const & mib ) [inline]
```

Returns a reference to a sysctl value container.

#### Parameters

<i>mib</i>	The MIB to return the reference for
------------	-------------------------------------

#### Returns

A [SysctlValue](#) reference

### 11.18.3 Member Data Documentation

#### 11.18.3.1 mibs

```
std::unordered_map<std::string, mib_t> anonymous_namespace{loadplay.cpp}::Sysctls::mibs [private]
```

#### Initial value:

```
{
    {"hw.machine", {CTL_HW, HW_MACHINE}},
    {"hw.model", {CTL_HW, HW_MODEL}},
    {"hw.ncpu", {CTL_HW, HW_NCPU}},
    {ACLINE, {1000}},
    {FREQ, {1001}},
    {FREQ_LEVELS, {1002}},
    {CP_TIMES, {1003}}
}
```

Maps name → mib.

## 11.18.3.2 sysctls

```
std::map<mib_t, SysctlValue> anonymous_namespace{loadplay.cpp}::Sysctls::sysctls [private]
```

## Initial value:

```
{
    {{CTL_HW, HW_MACHINE}, {CTLTYPE_STRING, "hw.machine"}},
    {{CTL_HW, HW_MODEL}, {CTLTYPE_STRING, "hw.model"}},
    {{CTL_HW, HW_NCPU}, {CTLTYPE_INT, "0"}},
    {{1000}, {CTLTYPE_INT, "2"}},
    {{1001}, {CTLTYPE_INT, "0"}},
    {{1002}, {CTLTYPE_STRING, ""}},
    {{1003}, {CTLTYPE_LONG, ""}}
}
```

Maps mib  $\rightarrow$  (type, value).

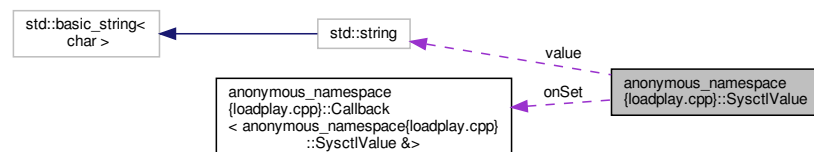
The documentation for this class was generated from the following file:

- [loadplay.cpp](#)

## 11.19 anonymous\_namespace{loadplay.cpp}::SysctlValue Class Reference

Instances of this class represents a specific sysctl value.

Collaboration diagram for anonymous\_namespace{loadplay.cpp}::SysctlValue:



## Public Member Functions

- [SysctlValue](#) ()  
*Default constructor.*
- [SysctlValue](#) ([SysctlValue](#) const &copy)  
*Copy constructor.*
- [SysctlValue](#) ([SysctlValue](#) &&move)  
*Move constructor.*
- [SysctlValue](#) (unsigned int [type](#), std::string const &[value](#), [callback\\_function](#) const callback=nullptr)  
*Construct from a type, value and optionally callback tuple.*
- [SysctlValue](#) & [operator=](#) ([SysctlValue](#) const &copy)  
*Copy assignment operator.*
- [SysctlValue](#) & [operator=](#) ([SysctlValue](#) &&move)  
*Move assignment operator.*
- [size\\_t](#) [size](#) () const

- Returns the required storage size according to the CTLTYPE.*

  - `template<typename T >`  
`int get (T *dst, size_t &size) const`  
*Copy a list of values into the given buffer.*
  - `int get (char *dst, size_t &size) const`  
*Copy a C string into the given buffer.*
  - `template<typename T >`  
`T get () const`  
*Returns a single value.*
  - `int get (void *dst, size_t &size) const`  
*Copy a list of values into the given buffer.*
  - `template<typename T >`  
`void set (T const *const newp, size_t newlen)`  
*Set this value to the values in the given buffer.*
  - `int set (void const *const newp, size_t newlen)`  
*Set this value to the values in the given buffer.*
  - `void set (std::string &&value)`  
*Move a string to the value.*
  - `void set (std::string const &value)`  
*Copy a string to the value.*
  - `template<typename T >`  
`void set (T const &value)`  
*Set the value.*
  - `void registerOnSet (callback_function &&callback)`  
*Register a callback function.*
  - `void registerOnSet (callback_function const &callback)`  
*Register a callback function.*

### Private Types

- `typedef std::lock_guard< decltype(mtx)> lock_guard`  
*Lock guard type, fitting the mutex.*

### Private Attributes

- `decltype(onSet) typedef ::function_t callback_function`  
*Callback function type.*
- `std::recursive_mutex mtx`  
*A stackable mutex.*
- `unsigned int type`  
*The sysctl type.*
- `std::string value`  
*The value of the sysctl.*
- `Callback< SysctlValue & > onSet`  
*Callback function handle.*

### 11.19.1 Detailed Description

Instances of this class represents a specific sysctl value.

There should only be one instance of this class per MIB.

Instances are thread safe.

## 11.19.2 Constructor & Destructor Documentation

### 11.19.2.1 SysctlValue() [1/3]

```
anonymous_namespace{loadplay.cpp}::SysctlValue::SysctlValue (
    SysctlValue const & copy ) [inline]
```

Copy constructor.

#### Parameters

<i>copy</i>	The instance to copy
-------------	----------------------

### 11.19.2.2 SysctlValue() [2/3]

```
anonymous_namespace{loadplay.cpp}::SysctlValue::SysctlValue (
    SysctlValue && move ) [inline]
```

Move constructor.

#### Parameters

<i>move</i>	The instance to move
-------------	----------------------

### 11.19.2.3 SysctlValue() [3/3]

```
anonymous_namespace{loadplay.cpp}::SysctlValue::SysctlValue (
    unsigned int type,
    std::string const & value,
    callback_function const callback = nullptr ) [inline]
```

Construct from a type, value and optionally callback tuple.

#### Parameters

<i>type</i>	The CTLTYPE
<i>value</i>	A string representation of the value
<i>callback</i>	A callback function that is called for each <a href="#">set()</a> call

## 11.19.3 Member Function Documentation

**11.19.3.1 get()** [1/4]

```
template<typename T >
int anonymous_namespace{loadplay.cpp}::SysctlValue::get (
    T * dst,
    size_t & size ) const [inline]
```

Copy a list of values into the given buffer.

**Template Parameters**

<i>T</i>	The type of the values to extract
----------	-----------------------------------

**Parameters**

<i>dst,size</i>	The destination buffer and size
-----------------	---------------------------------

**Return values**

<i>0</i>	On success
<i>-1</i>	On failure to fit all values into the taget buffer, also sets errno=ENOMEM

**11.19.3.2 get()** [2/4]

```
int anonymous_namespace{loadplay.cpp}::SysctlValue::get (
    char * dst,
    size_t & size ) const [inline]
```

Copy a C string into the given buffer.

**Parameters**

<i>dst,size</i>	The destination buffer and size
-----------------	---------------------------------

**Return values**

<i>0</i>	On success
<i>-1</i>	On failure to fit all values into the taget buffer, also sets errno=ENOMEM

**11.19.3.3 get()** [3/4]

```
template<typename T >
T anonymous_namespace{loadplay.cpp}::SysctlValue::get ( ) const [inline]
```

Returns a single value.



## Template Parameters

<i>T</i>	The type of the value
----------	-----------------------

## Returns

The value

11.19.3.4 `get()` [4/4]

```
int anonymous_namespace{loadplay.cpp}::SysctlValue::get (
    void * dst,
    size_t & size ) const [inline]
```

Copy a list of values into the given buffer.

## Parameters

<i>dst,size</i>	The destination buffer and size
-----------------	---------------------------------

## Return values

<i>0</i>	On success
<i>-1</i>	On failure to fit all values into the taget buffer, also sets errno=ENOMEM

11.19.3.5 `operator=()` [1/2]

```
SysctlValue& anonymous_namespace{loadplay.cpp}::SysctlValue::operator= (
    SysctlValue const & copy ) [inline]
```

Copy assignment operator.

## Parameters

<i>copy</i>	The instance to copy
-------------	----------------------

## Returns

A self reference

11.19.3.6 `operator=()` [2/2]

```
SysctlValue& anonymous_namespace{loadplay.cpp}::SysctlValue::operator= (
    SysctlValue && move ) [inline]
```

Move assignment operator.

**Parameters**

<i>move</i>	The instance to move
-------------	----------------------

**Returns**

A self reference

**11.19.3.7 registerOnSet()** [1/2]

```
void anonymous_namespace{loadplay.cpp}::SysctlValue::registerOnSet (
    callback_function && callback ) [inline]
```

Register a callback function.

**Parameters**

<i>callback</i>	The function to move to the callback handler
-----------------	--

**11.19.3.8 registerOnSet()** [2/2]

```
void anonymous_namespace{loadplay.cpp}::SysctlValue::registerOnSet (
    callback_function const & callback ) [inline]
```

Register a callback function.

**Parameters**

<i>callback</i>	The function to copy to the callback handler
-----------------	--

**11.19.3.9 set()** [1/5]

```
template<typename T >
void anonymous_namespace{loadplay.cpp}::SysctlValue::set (
    T const *const newp,
    size_t newlen ) [inline]
```

Set this value to the values in the given buffer.

**Template Parameters**

<i>T</i>	The type of the values
----------	------------------------

## Parameters

<i>newp, newlen</i>	The source buffer and size
---------------------	----------------------------

11.19.3.10 **set()** [2/5]

```
int anonymous_namespace{loadplay.cpp}::SysctlValue::set (
    void const *const newp,
    size_t newlen ) [inline]
```

Set this value to the values in the given buffer.

The buffer will be treated as an array of CTLTYPE values.

## Parameters

<i>newp, newlen</i>	The source buffer and size
---------------------	----------------------------

11.19.3.11 **set()** [3/5]

```
void anonymous_namespace{loadplay.cpp}::SysctlValue::set (
    std::string && value ) [inline]
```

Move a string to the value.

## Parameters

<i>value</i>	The new value
--------------	---------------

11.19.3.12 **set()** [4/5]

```
void anonymous_namespace{loadplay.cpp}::SysctlValue::set (
    std::string const & value ) [inline]
```

Copy a string to the value.

## Parameters

<i>value</i>	The new value
--------------	---------------

11.19.3.13 **set()** [5/5]

```
template<typename T >
```

```
void anonymous_namespace{loadplay.cpp}::SysctlValue::set (
    T const & value ) [inline]
```

Set the value.

#### Template Parameters

<i>T</i>	The value type
----------	----------------

#### Parameters

<i>value</i>	The value to set
--------------	------------------

#### 11.19.3.14 size()

```
size_t anonymous_namespace{loadplay.cpp}::SysctlValue::size ( ) const [inline]
```

Returns the required storage size according to the CTLTYPE.

#### Returns

The required buffer size to hold the values.

#### Exceptions

<i>int</i>	Throws -1 if the current CTLTYPE is not implemented.
------------	--

### 11.19.4 Member Data Documentation

#### 11.19.4.1 mtx

```
std::recursive_mutex anonymous_namespace{loadplay.cpp}::SysctlValue::mtx [mutable], [private]
```

A stackable mutex.

nice for exposing methods publicly and still let them allow accessing each other.

#### 11.19.4.2 value

```
std::string anonymous_namespace{loadplay.cpp}::SysctlValue::value [private]
```

The value of the sysctl.

This is stored as a string and converted to the appropriate type by the [set\(\)](#) and [get\(\)](#) methods.

The documentation for this class was generated from the following file:

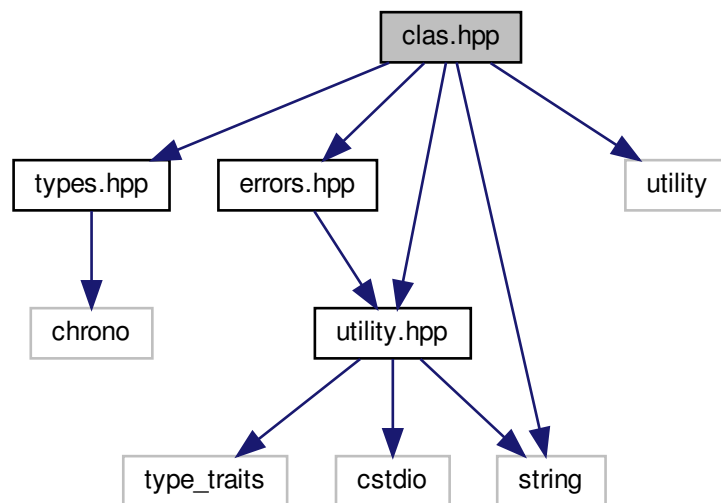
- [loadplay.cpp](#)

## 12 File Documentation

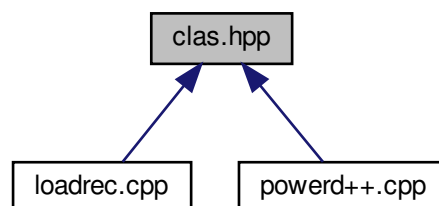
### 12.1 clas.hpp File Reference

Implements functions to process command line arguments.

```
#include "types.hpp"
#include "errors.hpp"
#include "utility.hpp"
#include <string>
#include <utility>
Include dependency graph for clas.hpp:
```



This graph shows which files directly or indirectly include this file:



## Namespaces

- [clas](#)  
*A collection of functions to process command line arguments.*

## Functions

- [types::cptime\\_t clas::load](#) (char const \*const str)  
*Convert string to load in the range [0, 1024].*
- [types::mhz\\_t clas::freq](#) (char const \*const str)  
*Convert string to frequency in MHz.*
- [types::ms clas::ival](#) (char const \*const str)  
*Convert string to time interval in milliseconds.*
- [size\\_t clas::samples](#) (char const \*const str)  
*A string encoded number of samples.*
- [types::decikelvin\\_t clas::temperature](#) (char const \*const str)  
*Convert string to temperature in dK.*
- [int clas::celsius](#) ([types::decikelvin\\_t](#) const val)  
*Converts dK into °C for display purposes.*
- [template<typename T> std::pair< T, T > clas::range](#) (char const \*const str, T(\*func)(char const \*const))  
*Takes a string encoded range of values and returns them.*

### 12.1.1 Detailed Description

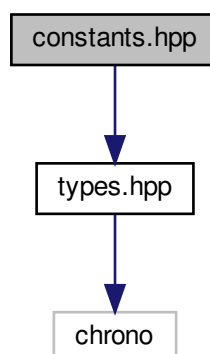
Implements functions to process command line arguments.

## 12.2 constants.hpp File Reference

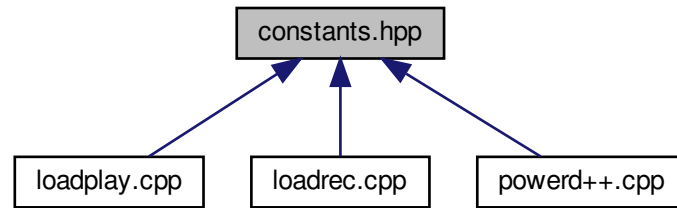
Defines a collection of constants.

```
#include "types.hpp"
```

Include dependency graph for constants.hpp:



This graph shows which files directly or indirectly include this file:



### Namespaces

- [constants](#)

*A collection of constants.*

### Variables

- `char const *const constants::CP\_TIMES = "kern.cp_times"`  
*The MIB name for per-CPU time statistics.*
- `char const *const constants::ACLIN = "hw.acpi.acline"`  
*The MIB name for the AC line state.*
- `char const *const constants::FREQ = "dev.cpu.%d.freq"`  
*The MIB name for CPU frequencies.*
- `char const *const constants::FREQ\_LEVELS = "dev.cpu.%d.freq_levels"`  
*The MIB name for CPU frequency levels.*
- `char const *const constants::TEMPERATURE = "dev.cpu.%d.temperature"`  
*The MIB name for CPU temperatures.*
- `char const *const constants::TJMAX\_SOURCES []`  
*An array of maximum temperature sources.*
- `types::mhz\_t const constants::FREQ\_DEFAULT\_MAX {1000000}`  
*Default maximum clock frequency value.*
- `types::mhz\_t const constants::FREQ\_DEFAULT\_MIN {0}`  
*Default minimum clock frequency value.*
- `types::mhz\_t const constants::FREQ\_UNSET {1000001}`  
*Clock frequency representing an uninitialised value.*
- `char const *const constants::POWERD\_PIDFILE = "/var/run/powerd.pid"`  
*The default pidfile name of powerd.*
- `types::cptime\_t const constants::ADP {512}`  
*The load target for adaptive mode, equals 50% load.*
- `types::cptime\_t const constants::HADP {384}`  
*The load target for hiadaptive mode, equals 37.5% load.*
- `types::decikelvin\_t const constants::HITEMP\_OFFSET {100}`  
*The default temperautre offset between high and critical temperature.*

### 12.2.1 Detailed Description

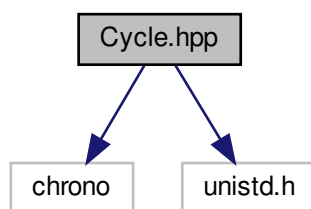
Defines a collection of constants.

## 12.3 Cycle.hpp File Reference

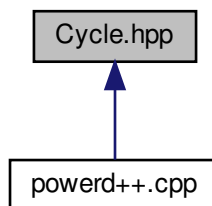
Implements [timing::Cycle](#), a cyclic sleep functor.

```
#include <chrono>
#include <unistd.h>
```

Include dependency graph for Cycle.hpp:



This graph shows which files directly or indirectly include this file:



### Classes

- class [timing::Cycle](#)  
*Implements an interruptible cyclic sleeping functor.*

### Namespaces

- [timing](#)  
*Namespace for time management related functionality.*



### 12.3.1 Detailed Description

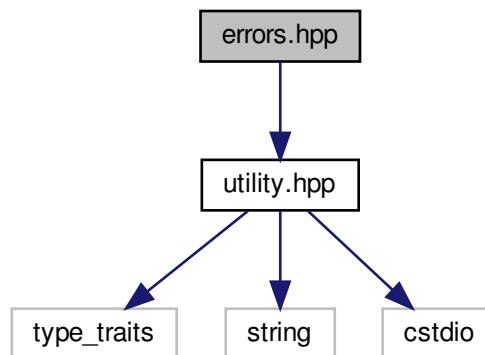
Implements [timing::Cycle](#), a cyclic sleep functor.

## 12.4 errors.hpp File Reference

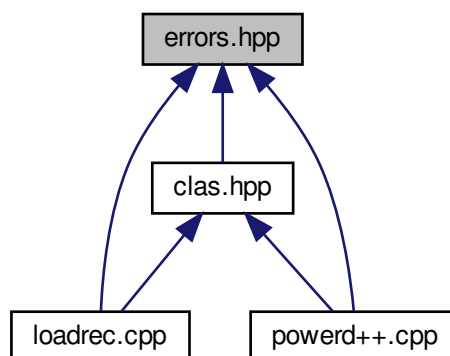
Common error handling code.

```
#include "utility.hpp"
```

Include dependency graph for errors.hpp:



This graph shows which files directly or indirectly include this file:



### Classes

- struct [errors::Exception](#)

*Exceptions bundle an exit code, errno value and message. [More...](#)*

## Namespaces

- [errors](#)

*Common error handling types and functions.*

## Enumerations

- enum [errors::Exit](#) : int {  
[errors::Exit::OK](#), [errors::Exit::ECLARG](#), [errors::Exit::EOUTOFRANGE](#), [errors::Exit::ELOAD](#),  
[errors::Exit::EFREQ](#), [errors::Exit::EMODE](#), [errors::Exit::EIVAL](#), [errors::Exit::ESAMPLES](#),  
[errors::Exit::ESYSCTL](#), [errors::Exit::ENOFREQ](#), [errors::Exit::ECONFLICT](#), [errors::Exit::EPID](#),  
[errors::Exit::EFORBIDDEN](#), [errors::Exit::EDAEMON](#), [errors::Exit::EWOPEN](#), [errors::Exit::ESIGNAL](#),  
[errors::Exit::ERANGEFMT](#), [errors::Exit::ETEMPERATURE](#), [errors::Exit::LENGTH](#) }

*Exit codes.*

## Functions

- void [errors::fail](#) (Exit const exitcode, int const err, std::string const &msg)  
*Throws an [Exception](#) instance with the given message.*

## Variables

- const char \*const [errors::ExitStr](#) []  
*Printable strings for exit codes.*

### 12.4.1 Detailed Description

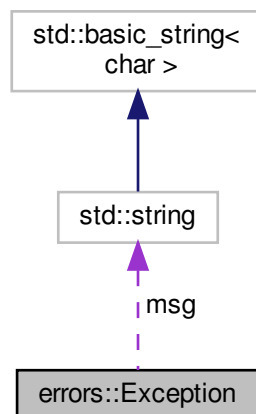
Common error handling code.

### 12.4.2 Class Documentation

#### 12.4.2.1 struct errors::Exception

Exceptions bundle an exit code, errno value and message.

Collaboration diagram for errors::Exception:



## Class Members

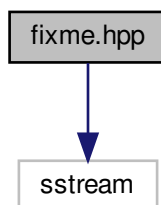
int	err	The errno value at the time of creation.
<a href="#">Exit</a>	exitcode	The code to exit with.
string	msg	An error message.

## 12.5 fixme.hpp File Reference

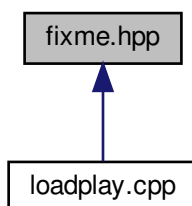
Implementations in the fixme namespace.

```
#include <sstream>
```

Include dependency graph for fixme.hpp:



This graph shows which files directly or indirectly include this file:



## Namespaces

- [fixme](#)

*Workarounds for compiler/library bugs.*

## Functions

- `template<typename T>`  
`std::string fixme::to_string (T const &op)`  
*G++ 5.3 does not believe in `std::to_string()`.*

### 12.5.1 Detailed Description

Implementations in the `fixme` namespace.

## 12.6 loadplay.cpp File Reference

Implements a library intended to be injected into a clock frequency daemon via `LD_PRELOAD`.

```
#include "utility.hpp"
#include "constants.hpp"
#include "fixme.hpp"
#include <iostream>
#include <iomanip>
#include <unordered_map>
#include <map>
#include <string>
#include <regex>
#include <sstream>
#include <memory>
#include <thread>
#include <exception>
#include <mutex>
#include <chrono>
#include <vector>
#include <cstring>
#include <cassert>
#include <csignal>
#include <sys/types.h>
#include <sys/sysctl.h>
#include <sys/resource.h>
#include <libutil.h>
#include <dlfcn.h>
#include <unistd.h>
```

Include dependency graph for `loadplay.cpp`:



## Classes

- struct `anonymous_namespace{loadplay.cpp}::mib_t`  
*Represents MIB, but wraps it to provide the necessary operators to use it as an `std::map` key.*
- class `anonymous_namespace{loadplay.cpp}::Callback< FunctionArgs >`

*Implements a recursion safe std::function wrapper.*

- class [anonymous\\_namespace{loadplay.cpp}::SysctlValue](#)  
*Instances of this class represents a specific sysctl value.*
- class [anonymous\\_namespace{loadplay.cpp}::Sysctls](#)  
*Singleton class representing the sysctl table for this library.*
- class [anonymous\\_namespace{loadplay.cpp}::Emulator](#)  
*Instances of this class represent an emulator session.*
- class [anonymous\\_namespace{loadplay.cpp}::Main](#)  
*Singleton class representing the main execution environment.*
- class [anonymous\\_namespace{loadplay.cpp}::Hold< T >](#)  
*Sets a referenced variable to a given value and restores it when going out of context.*

## Namespaces

- [anonymous\\_namespace{loadplay.cpp}](#)  
*File local scope.*

## Functions

- `template<size_t Size>`  
`int anonymous\_namespace{loadplay.cpp}::strcmp (char const *const s1, char const (&s2)[Size])`  
*Safe wrapper around strcmp, which automatically determines the buffer size of s2.*
- `std::regex anonymous\_namespace{loadplay.cpp}::operator"" \_r (char const *const str, size_t const len)`  
*User defined literal for regular expressions.*
- `template<>`  
`std::string anonymous\_namespace{loadplay.cpp}::SysctlValue::get< std::string > \(\) const`  
*Returns a copy of the value string.*
- `void anonymous\_namespace{loadplay.cpp}::warn (std::string const &msg)`  
*Print a warning.*
- `void anonymous\_namespace{loadplay.cpp}::fail (std::string const &msg)`  
*This prints an error message and sets sys\_results to make the hijacked process fail.*
- `int sysctl (const int *name, u_int namelen, void *oldp, size_t *oldlenp, const void *newp, size_t newlen)`  
*Functions to intercept.*
- `int sysctlnametomib (const char *name, int *mibp, size_t *sizep)`  
*Intercept calls to [sysctlnametomib\(\)](#).*
- `int sysctlbyname (const char *name, void *oldp, size_t *oldlenp, const void *newp, size_t newlen)`  
*Intercept calls to [sysctlbyname\(\)](#).*
- `int daemon (int, int)`  
*Intercept calls to [daemon\(\)](#).*
- `uid_t geteuid (void)`  
*Intercept calls to [geteuid\(\)](#).*
- `pid_t * pidfile\_open (const char *, mode_t, pid_t *)`  
*Intercept calls to [pidfile\\_open\(\)](#).*
- `int pidfile\_write (pid_t *)`  
*Intercept calls to [pidfile\\_write\(\)](#).*
- `int pidfile\_close (pid_t *)`  
*Intercept calls to [pidfile\\_close\(\)](#).*
- `int pidfile\_remove (pid_t *)`  
*Intercept calls to [pidfile\\_remove\(\)](#).*
- `int pidfile\_fileno (pid_t const *)`  
*Intercept calls to [pidfile\\_fileno\(\)](#).*

## Variables

- int `anonymous_namespace{loadplay.cpp}::sys_results` = 0  
*The success return value of intercepted functions.*
- class `anonymous_namespace{loadplay.cpp}::Sysctl` `anonymous_namespace{loadplay.cpp}::sysctl`  
*Sole instance of `Sysctl`.*
- class `anonymous_namespace{loadplay.cpp}::Main` `anonymous_namespace{loadplay.cpp}::main`  
*Sole instance of `Main`.*
- bool `anonymous_namespace{loadplay.cpp}::sysctl_fallback` = false  
*Set to activate fallback to the original sysctl functions.*

### 12.6.1 Detailed Description

Implements a library intended to be injected into a clock frequency daemon via LD\_PRELOAD.

This library reads instructions from `std::cin` and outputs statistics about the hijacked process on `std::cout`.

### 12.6.2 Function Documentation

#### 12.6.2.1 `daemon()`

```
int daemon (
    int ,
    int )
```

Intercept calls to `daemon()`.

Prevents process from separating from the controlling terminal.

#### Returns

The value of `sys_results`

#### 12.6.2.2 `geteuid()`

```
uid_t geteuid (
    void )
```

Intercept calls to `geteuid()`.

Tells the asking process that it is running as root.

#### Returns

Always returns 0

### 12.6.2.3 pidfile\_close()

```
int pidfile_close (
    pidfh * )
```

Intercept calls to [pidfile\\_close\(\)](#).

#### Returns

The value of sys\_results

### 12.6.2.4 pidfile\_fileno()

```
int pidfile_fileno (
    pidfh const * )
```

Intercept calls to [pidfile\\_fileno\(\)](#).

#### Returns

The value of sys\_results

### 12.6.2.5 pidfile\_open()

```
pidfh* pidfile_open (
    const char * ,
    mode_t ,
    pid_t * )
```

Intercept calls to [pidfile\\_open\(\)](#).

Prevents pidfile locking and creation by the hijacked process.

#### Returns

A dummy pointer

### 12.6.2.6 pidfile\_remove()

```
int pidfile_remove (
    pidfh * )
```

Intercept calls to [pidfile\\_remove\(\)](#).

#### Returns

The value of sys\_results

### 12.6.2.7 pidfile\_write()

```
int pidfile_write (
    pidfh * )
```

Intercept calls to [pidfile\\_write\(\)](#).

#### Returns

The value of `sys_results`

### 12.6.2.8 sysctl()

```
int sysctl (
    const int * name,
    u_int namelen,
    void * oldp,
    size_t * oldlenp,
    const void * newp,
    size_t newlen )
```

Functions to intercept.

Intercept calls to [sysctl\(\)](#).

Uses the local [anonymous\\_namespace{loadplay::cpp}::sysctls](#) store.

Falls back to the original if `kern.usrstack` is requested or `sysctl_fallback` is set.

The call may fail for 3 reasons:

1. The [fail\(\)](#) function was called and `sys_results` was assigned -1
2. A target buffer was too small (`errno == ENOMEM`)
3. The given `sysctl` is not in the `sysctls` store (`errno == ENOENT`)

#### Parameters

<i>name, namelen, oldp, oldlenp, newp, newlen</i>	Please refer to <code>sysctl(3)</code>
---	--

#### Return values

0	The call succeeded
-1	The call failed

### 12.6.2.9 sysctlbyname()

```
int sysctlbyname (
```



```

    const char * name,
    void * oldp,
    size_t * oldlenp,
    const void * newp,
    size_t newlen )

```

Intercept calls to [sysctlbyname\(\)](#).

Falls back on the original [sysctlbyname\(\)](#) for the following names:

- vm.overcommit
- kern.smp.cpus

May fail for the same reasons as [sysctl\(\)](#).

#### Parameters

<i>name, oldp, oldlenp, newp, newlen</i>	Please refer to <a href="#">sysctl(3)</a>
--	---

#### Return values

0	The call succeeded
-1	The call failed

#### 12.6.2.10 sysctlnametomib()

```

int sysctlnametomib (
    const char * name,
    int * mibp,
    size_t * sizep )

```

Intercept calls to [sysctlnametomib\(\)](#).

#### Parameters

<i>name, mibp, sizep</i>	Please refer to <a href="#">sysctl(3)</a>
--------------------------	---

#### Return values

0	The call succeeded
-1	The call failed

## 12.7 loadrec.cpp File Reference

Implements a load recorder, useful for simulating loads to test CPU clock daemons and settings.



## Variables

- - struct {
    - bool **verbose** {false}
      - Verbosity flag.*
    - ms **duration** {30000}
      - Recording duration in ms.*
    - ms **interval** {25}
      - Recording sample interval in ms.*
    - std::ofstream **outfile** {}
      - The output file stream to use if an outfilename is provided on the CLI.*
    - std::ostream \* **out** = &std::cout
      - A pointer to the stream to use for output, either std::cout or outfile.*
    - char const \* **outfilename** {nullptr}
      - The user provided output file name.*
    - char const \* **pidfilename** {POWERD\_PIDFILE}
      - The PID file location for clock frequency daemons.*
    - [sys::ctl::SysctlOnce](#)< coreid\_t, 2 > const **ncpu** {1U, {CTL\_HW, HW\_NCPU}}
    - The number of CPU cores/threads.*
  - } [anonymous\\_namespace{loadrec.cpp}::g](#)
    - The global state.*
  - char const \*const [anonymous\\_namespace{loadrec.cpp}::USAGE](#) = "[-hv] [-d ival] [-p ival] [-o file]"
    - The short usage string.*
  - Option< OE > const [anonymous\\_namespace{loadrec.cpp}::OPTIONS](#) []
    - Definitions of command line options.*

## 12.7.1 Detailed Description

Implements a load recorder, useful for simulating loads to test CPU clock daemons and settings.

## 12.7.2 Function Documentation

## 12.7.2.1 main()

```
int main (
    int argc,
    char * argv[] )
```

Main routine, setup and execute daemon, print errors.

## Parameters

<i>argc,argv</i>	The command line arguments
------------------	----------------------------

## Returns

An exit code

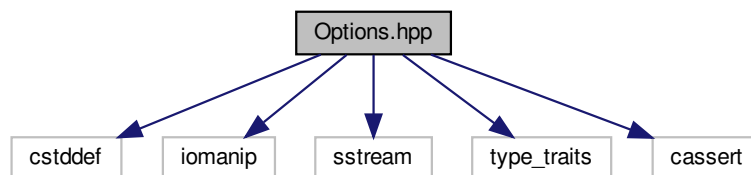
See also

Exit

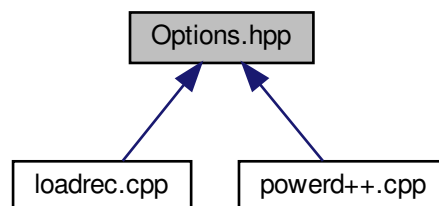
## 12.8 Options.hpp File Reference

This file provides `nih::Options<>`, a substitute for `getopt(3)`.

```
#include <cstdint>
#include <iomanip>
#include <sstream>
#include <type_traits>
#include <cassert>
Include dependency graph for Options.hpp:
```



This graph shows which files directly or indirectly include this file:



### Classes

- struct `nih::enum_has_members< Enum, class >`  
Tests whether the given enum provides all the required definitions.
- struct `nih::Option< Enum >`  
Container for an option definition. [More...](#)
- class `nih::Options< Enum, DefCount >`  
An instance of this class offers operators to retrieve command line options and arguments.

## Namespaces

- [nih](#)

*Not invented here namespace, for code that substitutes already commonly available functionality.*

## Typedefs

- `template<class... >`  
`using nih::void\_t = void`

*See `std::void_t` in C++17 <type\_traits>.*

## Functions

- `template<class Enum >`  
`size_t nih::argCount (Option< Enum > const &def)`  
*Retrieves the count of arguments in an option definition.*
- `template<class Enum , size_t DefCount>`  
`constexpr Options< Enum, DefCount > nih::make\_Options (int const argc, char const *const argv[], char const *const usage, Option< Enum > const (&defs)[DefCount])`

*Wrapper around the Options<> constructor, that uses function template matching to deduce template arguments.*

## 12.8.1 Detailed Description

This file provides [nih::Options<>](#), a substitute for `getopt(3)`.

The `getopt(3)` interface takes the command line arguments as `char * const` instead of `char const *`. I.e. it reserves the right to mutate the provided arguments, which it actually does.

The [nih::Options<>](#) functor is not a drop in substitute, but tries to be easily adoptable and does not change the data given to it.

To use the options an enum or enum class is required, e.g.:

```
enum class MyOptions {
    USAGE, FILE_IN, FILE_OUT, FLAG_VERBOSE,
    OPT_UNKNOWN, OPT_NOOPT, OPT_DASH, OPT_LDASH, OPT_DONE
};
```

The options prefixed with `OPT_` are obligatory. Their meaning is documented in `nih::enum_has_members<>`. Their presence is validated at compile time.

The enum values are returned whe selecting the next option, in order to do that a usage string and a list of definitions are needed:

```
static char const * const USAGE = "[-hv] [-i file] [-o file] [command ...]";

static nih::Option<MyOptions> const OPTIONS[]{
    {MyOptions::USAGE,      'h', "help", "", "Show this help"},
    {MyOptions::USAGE,      0, "usage", "", ""},
    {MyOptions::FILE_IN,     'i', "in", "file", "Input file"},
    {MyOptions::FILE_OUT,    'o', "out", "file", "Output file"},
    {MyOptions::FLAG_VERBOSE, 'v', "verbose", "", "Verbose output"}
};
```

Every array entry defines an option consisting of the enum value that represents it, a short and a long version (either of which are optional) and a comma separated list of arguments. The final string appears in the usage() output. The details are documented by nih::Option<>.

Aliases are created by adding a definition that returns the same enum value.

For the short version it does not matter whether -ifile or -i file is provided, the long version must be --in file. Short options without arguments may be directly followed by another short option, e.g. -vofile is equivalent to -v -o file.

The option definitions should be passed to nih::make\_Options() to create the functor:

```
#include <iostream>
...

int main(int argc, char * argv[]) {
    char const * infile = "-";
    char const * outfile = "-";
    bool verbose = false;

    auto getopt = nih::make_Options(argc, argv, USAGE, OPTIONS);
    while (true) switch (getopt()) { // get new option/argument
    case MyOptions::USAGE:
        std::cerr << getopt.usage(); // show usage
        return 0;
    case MyOptions::FILE_IN:
        infile = getopt[1]; // get first argument
        break;
    case MyOptions::FILE_OUT:
        outfile = getopt[1]; // get first argument
        break;
    case MyOptions::FLAG_VERBOSE:
        verbose = true;
        break;
    case MyOptions::OPT_UNKNOWN:
    case MyOptions::OPT_NOOPT:
    case MyOptions::OPT_DASH:
    case MyOptions::OPT_LDASH:
        std::cerr << "Unexpected command line argument: "
                    << getopt[0] << '\n'; // output option/argument
        return 1;
    case MyOptions::OPT_DONE:
        return do_something(infile, outfile, verbose);
    }
    return 0;
}
```

Every call of the functor moves on to the next option or argument. For non-option arguments it returns OPT\_NOOPT.

The getopt[1] calls return the first argument following the option. It is possible to retrieve more arguments than were defined in the options definition. The [] operator always returns a valid, terminated string (provided the command line arguments are valid, terminated strings). So it is always safe to dereference the pointer, even when reading beyond the end of command line arguments.

The getopt[0] calls return the command line argument that contains the selected option. So in the FILE\_IN case it could be any of -i, --in, -vi, -ifile or -vifile. This is useful for the OPT\_UNKNOWN and OPT\_NOOPT cases. The getopt[1] call on the other hand would always return file regardless of argument chaining.

## 12.8.2 Class Documentation

### 12.8.2.1 struct nih::Option

```
template<class Enum>
struct nih::Option< Enum >
```

Container for an option definition.

Aliases can be defined by creating definitions with the same enumval member.

The lopt, args and usage members have to be 0 terminated, using string literals is safe.

## Template Parameters

<i>Enum</i>	An enum or enum class representing the available options
-------------	--

## Class Members

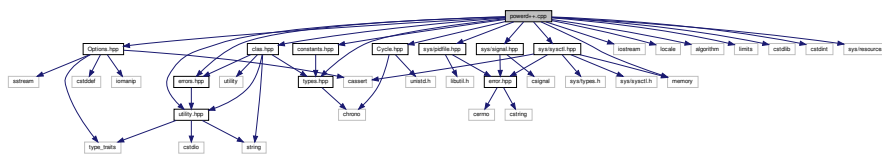
char const *	args	A comma separated list of arguments. Set to nullptr or "" if no argument is available.
Enum	enumval	The enum value to return for this option.
char const *	lopt	The long version of this option. Set to nullptr or "" if no long option is available.
char	sopt	The short version of this option. Set to 0 if no short option is available.
char const *	usage	A usage string.

## 12.9 power++.cpp File Reference

Implements power++ a drop in replacement for FreeBSD's powerd.

```
#include "Options.hpp"
#include "Cycle.hpp"
#include "types.hpp"
#include "constants.hpp"
#include "errors.hpp"
#include "clas.hpp"
#include "utility.hpp"
#include "sys/sysctl.hpp"
#include "sys/pidfile.hpp"
#include "sys/signal.hpp"
#include <iostream>
#include <locale>
#include <memory>
#include <algorithm>
#include <limits>
#include <cstdlib>
#include <cstdint>
#include <sys/resource.h>
```

Include dependency graph for power++.cpp:



## Classes

- struct [anonymous\\_namespace{power++.cpp}::Core](#)  
Contains the management information for a single CPU core. [More...](#)
- struct [anonymous\\_namespace{power++.cpp}::Global](#)  
A collection of all the gloabl, mutable states. [More...](#)
- struct [anonymous\\_namespace{power++.cpp}::Global::ACSet](#)  
Per AC line state settings. [More...](#)
- class [anonymous\\_namespace{power++.cpp}::FreqGuard](#)  
A core frequency guard.

## Namespaces

- `anonymous_namespace{power++.cpp}`

*File local scope.*

## Enumerations

- `enum anonymous_namespace{power++.cpp}::AcLineState : unsigned int { anonymous_namespace{power++.cpp}::AcLineState::BATTERY, anonymous_namespace{power++.cpp}::AcLineState::ONLINE, anonymous_namespace{power++.cpp}::AcLineState::UNKNOWN, anonymous_namespace{power++.cpp}::AcLineState::LENGTH }`

*The available AC line states.*

- `enum anonymous_namespace{power++.cpp}::OE { anonymous_namespace{power++.cpp}::OE::USAGE, anonymous_namespace{power++.cpp}::OE::MODE_BATT, anonymous_namespace{power++.cpp}::OE::FREQ_MIN, anonymous_namespace{power++.cpp}::OE::FREQ_MAX, anonymous_namespace{power++.cpp}::OE::FREQ_MIN_AC, anonymous_namespace{power++.cpp}::OE::FREQ_MAX_AC, anonymous_namespace{power++.cpp}::OE::FREQ_MIN_BATT, anonymous_namespace{power++.cpp}::OE::FREQ_MAX_BATT, anonymous_namespace{power++.cpp}::OE::FREQ_RANGE, anonymous_namespace{power++.cpp}::OE::FREQ_RANGE_AC, anonymous_namespace{power++.cpp}::OE::FREQ_RANGE_BATT, anonymous_namespace{power++.cpp}::OE::HITEMP_RANGE, anonymous_namespace{power++.cpp}::OE::MODE_UNKNOWN, anonymous_namespace{power++.cpp}::OE::IVAL_POLL, anonymous_namespace{power++.cpp}::OE::FILE_PID, anonymous_namespace{power++.cpp}::OE::FLAG_VERBOSE, anonymous_namespace{power++.cpp}::OE::FLAG_FOREGROUND, anonymous_namespace{power++.cpp}::OE::CNT_SAMPLES, anonymous_namespace{power++.cpp}::OE::IGNORE, anonymous_namespace{power++.cpp}::OE::OPT_UNKNOWN, anonymous_namespace{power++.cpp}::OE::OPT_NOOPT, anonymous_namespace{power++.cpp}::OE::OPT_DASH, anonymous_namespace{power++.cpp}::OE::OPT_LDASH, anonymous_namespace{power++.cpp}::OE::OPT_DONE }`

*An enum for command line parsing.*

## Functions

- `void anonymous_namespace{power++.cpp}::verbose (std::string const &msg)`  
*Outputs the given message on stderr if g.verbose is set.*
- `void anonymous_namespace{power++.cpp}::sysctl_fail (sys::sc_error< sys::ctl::error > const err)`  
*Treat sysctl errors.*
- `void anonymous_namespace{power++.cpp}::init ()`  
*Perform initial tasks.*
- `template<bool Load = 1, bool Temperature = 0>`  
`void anonymous_namespace{power++.cpp}::update_loads ()`  
*Updates the cp\_times ring buffer and computes the load average for each core.*
- `template<>`  
`void anonymous_namespace{power++.cpp}::update_loads< 0, 0 > ()`  
*Do nada if neither load nor temperature are to be updated.*
- `template<bool Foreground, bool Temperature, bool Fixed>`  
`void anonymous_namespace{power++.cpp}::update_freq (Global::ACSet const &acstate)`  
*Update the CPU clocks depending on the AC line state and targets.*
- `void anonymous_namespace{power++.cpp}::update_freq ()`  
*Dispatch update\_freq<>().*



- void `anonymous_namespace{powerd++.cpp}::init_loads ()`  
*Fill the loads buffers with n samples.*
- void `anonymous_namespace{powerd++.cpp}::set_mode (AcLineState const line, char const *const str)`  
*Sets a load target or fixed frequency for the given AC line state.*
- void `anonymous_namespace{powerd++.cpp}::read_args (int const argc, char const *const argv[])`  
*Parse command line arguments.*
- void `anonymous_namespace{powerd++.cpp}::show_settings ()`  
*Prints the configuration on stderr in verbose mode.*
- void `anonymous_namespace{powerd++.cpp}::signal_rcv (int signal)`  
*Sets g.signal, terminating the main loop.*
- void `anonymous_namespace{powerd++.cpp}::run_daemon ()`  
*Daemonise and run the main loop.*
- int `main (int argc, char *argv[])`  
*Main routine, setup and execute daemon, print errors.*

## Variables

- struct `anonymous_namespace{powerd++.cpp}::Global anonymous_namespace{powerd++.cpp}::g`  
*The global state.*
- char const \*const `anonymous_namespace{powerd++.cpp}::USAGE = "[-hvf] [-abn mode] [-mM freq] [-FAB freq:freq] [-H temp:temp] [-p ival] [-s cnt] [-P file]"`  
*The short usage string.*
- Option< OE > const `anonymous_namespace{powerd++.cpp}::OPTIONS []`  
*Definitions of command line options.*

### 12.9.1 Detailed Description

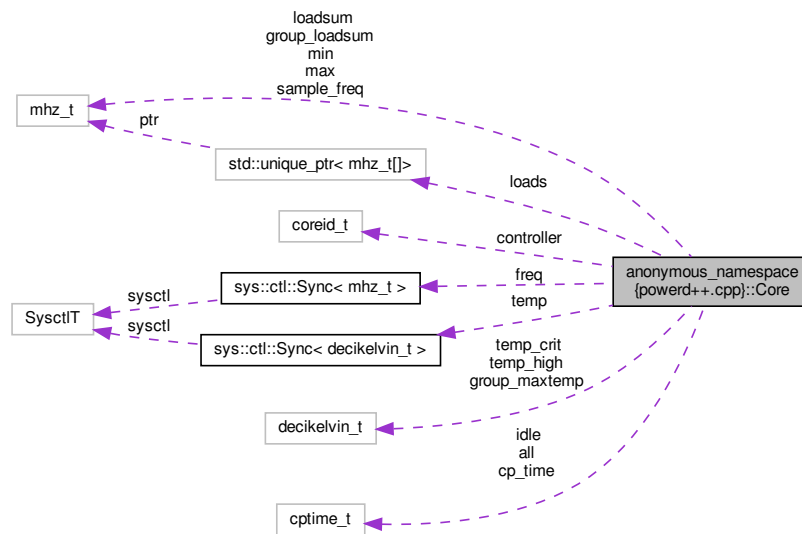
Implements powerd++ a drop in replacement for FreeBSD's powerd.

### 12.9.2 Class Documentation

#### 12.9.2.1 struct `anonymous_namespace{powerd++.cpp}::Core`

Contains the management information for a single CPU core.

Collaboration diagram for `anonymous_namespace{powerd++.cpp}::Core`:



#### Class Members

<code>cptime_t</code>	<code>all</code>	Count of all ticks.
<code>coreid_t</code>	<code>controller</code>	The core that controls the frequency for this core.
<code>cptime_t const *</code>	<code>cp_time</code>	A pointer to the <code>kern.cp_times</code> section for this core.
<code>SysctlSync&lt;mhz_t&gt;</code>	<code>freq</code>	The <code>sysctl kern.cpu.N.freq</code> , if present.
<code>mhz_t</code>	<code>group_loadsum</code>	For the controlling core this is set to the group loadsum. This is updated by <a href="#">update_loads()</a> .
<code>decikelvin_t</code>	<code>group_maxtemp</code>	For the controlling core this is set to the maximum temperature measurement taken in the group.
<code>cptime_t</code>	<code>idle</code>	The idle ticks count.
<code>unique_ptr&lt;mhz_t[]&gt;</code>	<code>loads</code>	A ring buffer of load samples for this core. Each load sample is weighted with the core frequency at which it was taken. This is updated by <a href="#">update_loads()</a> .
<code>mhz_t</code>	<code>loadsum</code>	The sum of all load samples. This is updated by <a href="#">update_loads()</a> .
<code>mhz_t</code>	<code>max</code>	The maximum core clock rate.
<code>mhz_t</code>	<code>min</code>	The minimum core clock rate.
<code>mhz_t</code>	<code>sample_freq</code>	The <code>kern.cpu.N.freq</code> value for the current load sample. This is updated by <a href="#">update_loads()</a> .
<code>SysctlSync&lt;decikelvin_t&gt;</code>	<code>temp</code>	The <code>dev.cpu. d.temperature sysctl</code> , if present.
<code>decikelvin_t</code>	<code>temp_crit</code>	Critical core temperature in dK.
<code>decikelvin_t</code>	<code>temp_high</code>	High core temperature in dK.

#### 12.9.2.2 struct `anonymous_namespace{powerd++.cpp}::Global`

A collection of all the global, mutable states.

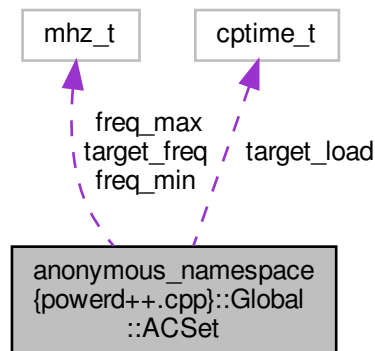
This is mostly for semantic clarity.



### 12.9.2.3 struct anonymous\_namespace{power++.cpp}::Global::ACSet

Per AC line state settings.

Collaboration diagram for anonymous\_namespace{power++.cpp}::Global::ACSet:



#### Class Members

mhz_t	freq_max	Highest frequency to set in MHz.
mhz_t	freq_min	Lowest frequency to set in MHz.
char const *const	name	The string representation of this state.
mhz_t	target_freq	Fixed clock frequencies to use if the target load is set to 0.
cptime_t	target_load	Target load times [0, 1024]. The value 0 indicates the corresponding fixed frequency setting from target_freqs should be used.

## 12.9.3 Function Documentation

### 12.9.3.1 main()

```

int main (
    int argc,
    char * argv[] )

```

Main routine, setup and execute daemon, print errors.

#### Parameters

<i>argc,argv</i>	The command line arguments
------------------	----------------------------

**Returns**

An exit code

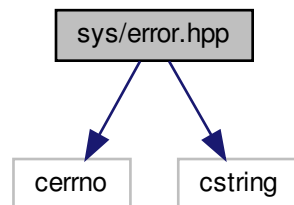
**See also**

Exit

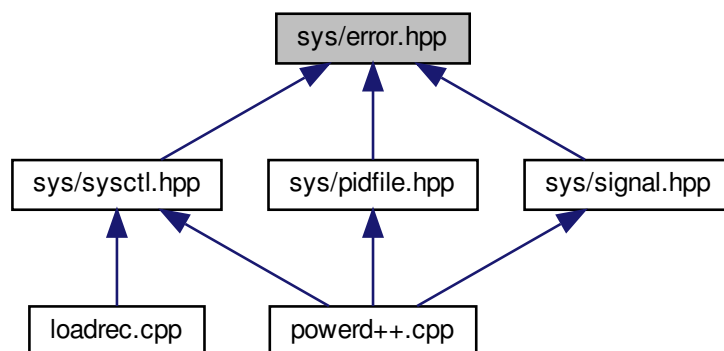
**12.10 sys/error.hpp File Reference**

Provides system call error handling.

```
#include <cerrno>
#include <cstring>
Include dependency graph for error.hpp:
```



This graph shows which files directly or indirectly include this file:

**Classes**

- struct `sys::sc_error< Domain >`

*Can be thrown by syscall function wrappers if the function returned with an error.*

## Namespaces

- [sys](#)

*Wrappers around native system interfaces.*

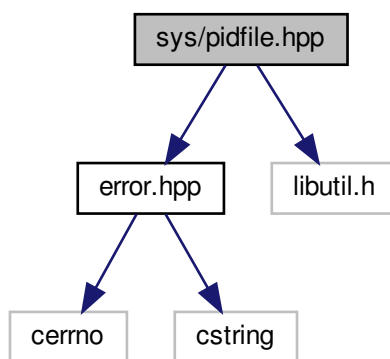
### 12.10.1 Detailed Description

Provides system call error handling.

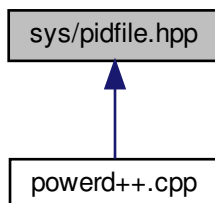
### 12.11 sys/pidfile.hpp File Reference

Implements safer c++ wrappers for the pidfile\_\*() interface.

```
#include "error.hpp"
#include <libutil.h>
Include dependency graph for pidfile.hpp:
```



This graph shows which files directly or indirectly include this file:



## Classes

- struct [sys::pid::error](#)  
*The domain error type. [More...](#)*
- class [sys::pid::Pidfile](#)  
*A wrapper around the `pidfile_*` family of commands implementing the RAII pattern.*

## Namespaces

- [sys](#)  
*Wrappers around native system interfaces.*
- [sys::pid](#)  
*This namespace contains safer c++ wrappers for the `pidfile_*`() interface.*

## 12.11.1 Detailed Description

Implements safer c++ wrappers for the `pidfile_*`() interface.

Requires linking with `-lutil`.

## 12.11.2 Class Documentation

## 12.11.2.1 struct sys::pid::error

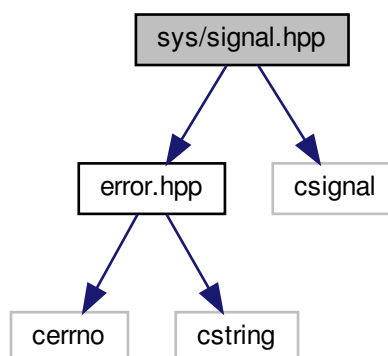
The domain error type.

## 12.12 sys/signal.hpp File Reference

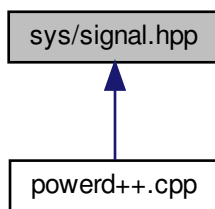
Implements a c++ wrapper for the `signal(3)` call.

```
#include "error.hpp"  
#include <csignal>
```

Include dependency graph for `signal.hpp`:



This graph shows which files directly or indirectly include this file:



## Classes

- struct [sys::sig::error](#)  
*The domain error type. [More...](#)*
- class [sys::sig::Signal](#)  
*Sets up a given signal handler and restores the old handler when going out of scope.*

## Namespaces

- [sys](#)  
*Wrappers around native system interfaces.*
- [sys::sig](#)  
*This namespace provides c++ wrappers for `signal(3)`.*

## Typedefs

- using [sys::sig::sig\\_t](#) = void(\*)(int)  
*Convenience type for signal handlers.*

### 12.12.1 Detailed Description

Implements a c++ wrapper for the `signal(3)` call.

### 12.12.2 Class Documentation

#### 12.12.2.1 struct `sys::sig::error`

The domain error type.

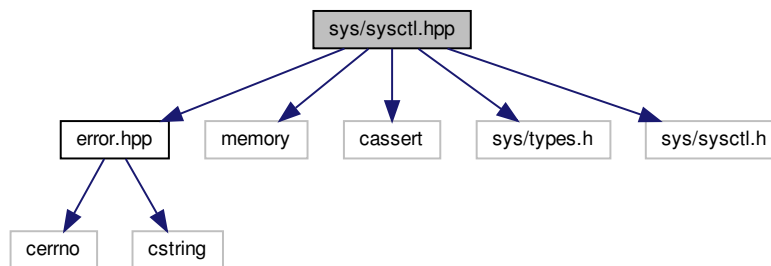


## 12.13 sys/sysctl.hpp File Reference

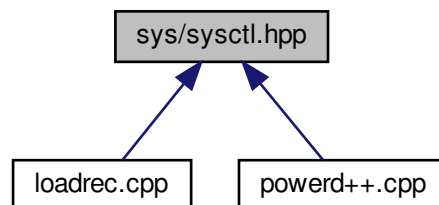
Implements safer c++ wrappers for the [sysctl\(\)](#) interface.

```
#include "error.hpp"
#include <memory>
#include <cassert>
#include <sys/types.h>
#include <sys/sysctl.h>
```

Include dependency graph for sysctl.hpp:



This graph shows which files directly or indirectly include this file:



### Classes

- struct `sys::ctl::error`  
The domain error type. [More...](#)
- class `sys::ctl::Sysctl< MibDepth >`  
Represents a sysctl MIB address.
- class `sys::ctl::Sysctl< 0 >`  
This is a specialisation of `Sysctl` for sysctls using symbolic names.
- class `sys::ctl::Sync< T, SysctlT >`  
This is a wrapper around `Sysctl` that allows semantically transparent use of a `sysctl`.
- class `sys::ctl::Once< T, SysctlT >`  
A read once representation of a `Sysctl`.

## Namespaces

- [sys](#)  
*Wrappers around native system interfaces.*
- [sys::ctl](#)  
*This namespace contains safer c++ wrappers for the [sysctl\(\)](#) interface.*

## Typedefs

- typedef int [sys::ctl::mib\\_t](#)  
*Management Information Base identifier type (see [sysctl\(3\)](#)).*
- template<typename T, size\_t MibDepth = 0>  
using [sys::ctl::SysctlSync](#) = Sync< T, Sysctl< MibDepth > >  
*A convenience alias around [Sync](#).*
- template<typename T, size\_t MibDepth>  
using [sys::ctl::SysctlOnce](#) = Once< T, Sysctl< MibDepth > >  
*A convenience alias around [Once](#).*

## Functions

- void [sys::ctl::sysctl\\_raw](#) (mib\_t const \*name, u\_int const namelen, void \*const oldp, size\_t \*const oldlenp, void const \*const newp, size\_t const newlen)  
*A wrapper around the [sysctl\(\)](#) function.*
- template<size\_t MibDepth>  
void [sys::ctl::sysctl\\_get](#) (mib\_t const (&mib)[MibDepth], void \*const oldp, size\_t &oldlen)  
*Returns a [sysctl\(\)](#) value to a buffer.*
- template<size\_t MibDepth>  
void [sys::ctl::sysctl\\_set](#) (mib\_t const (&mib)[MibDepth], void const \*const newp, size\_t const newlen)  
*Sets a [sysctl\(\)](#) value.*
- template<typename... Args>  
constexpr Sysctl< sizeof...(Args)> [sys::ctl::make\\_Sysctl](#) (Args const ... args)  
*Create a [Sysctl](#) instances.*
- template<typename T, class SysctlT >  
constexpr Once< T, SysctlT > [sys::ctl::make\\_Once](#) (T const &value, SysctlT const &[sysctl](#)) noexcept  
*This creates a [Once](#) instance.*

### 12.13.1 Detailed Description

Implements safer c++ wrappers for the [sysctl\(\)](#) interface.

### 12.13.2 Class Documentation

#### 12.13.2.1 struct sys::ctl::error

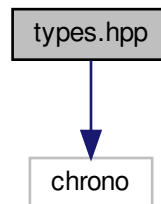
The domain error type.

## 12.14 types.hpp File Reference

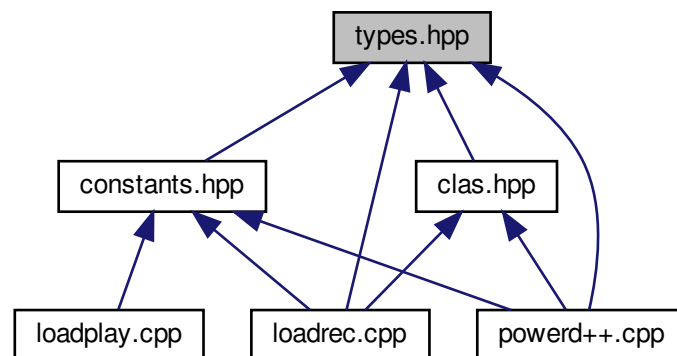
A collection of type aliases.

```
#include <chrono>
```

Include dependency graph for types.hpp:



This graph shows which files directly or indirectly include this file:



### Namespaces

- [types](#)  
*A collection of type aliases.*

### Typedefs

- `typedef std::chrono::milliseconds types::ms`  
*Millisecond type for polling intervals.*
- `typedef int types::coreid\_t`

*Type for CPU core indexing.*

- typedef unsigned long [types::cptime\\_t](#)

*Type for load counting.*

- typedef unsigned int [types::mhz\\_t](#)

*Type for CPU frequencies in MHz.*

- typedef int [types::decikelvin\\_t](#)

*Type for temperatures in dK.*

### 12.14.1 Detailed Description

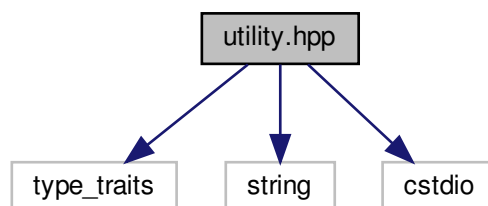
A collection of type aliases.

### 12.15 utility.hpp File Reference

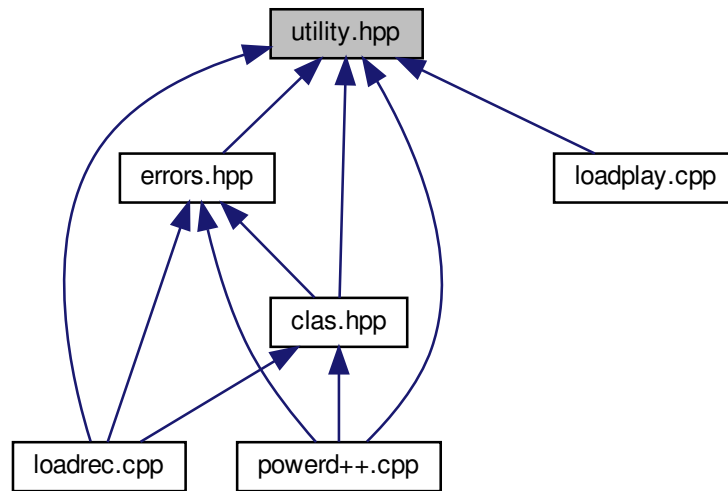
Implements generally useful functions.

```
#include <type_traits>
#include <string>
#include <cstdio>
```

Include dependency graph for utility.hpp:



This graph shows which files directly or indirectly include this file:



#### Classes

- class [utility::Formatter< BufSize >](#)  
A formatting wrapper around string literals.

#### Namespaces

- [utility](#)  
A collection of generally useful functions.
- [utility::literals](#)  
Contains literals.

#### Functions

- `template<typename T , size_t Count>`  
`constexpr size_t utility::countof (T(&)[Count])`  
Like `sizeof()`, but it returns the number of elements an array consists of instead of the number of bytes.
- `std::string utility::literals::operator"" \_s (char const *const op, size_t const size)`  
A string literal operator equivalent to the `operator "" s` literal provided by C++14 in `<string>`.
- `template<typename... Args>`  
`void utility::sprintf (Args...)`  
This is a safeguard against accidentally using `sprintf()`.
- `template<size_t Size, typename... Args>`  
`int utility::sprintf\_safe (char(&dst)[Size], char const *const format, Args const ... args)`  
A wrapper around `snprintf()` that automatically pulls in the destination buffer size.
- `template<class ET , typename VT = typename std::underlying_type<ET>::type>`  
`constexpr VT utility::to\_value (ET const op)`  
Casts an enum to its underlying value.
- `constexpr Formatter< 16384 > utility::literals::operator"" \_fmt (char const *const fmt, size_t const)`  
Literal to convert a string literal to a [Formatter](#) instance.

#### 12.15.1 Detailed Description

Implements generally useful functions.

## Index

### A

AcLineState  
    anonymous\_namespace{powerd++.cpp}, 29  
addValue  
    anonymous\_namespace{loadplay.cpp}::Sysctls, 93  
anonymous\_namespace{clas.cpp}, 18  
    Unit, 18  
    unit, 19  
    UnitStr, 19  
anonymous\_namespace{loadplay.cpp}, 19  
    fail, 20  
    main, 22  
    operator""\_r, 20  
    strcmp, 21  
    SysctlValue::get< std::string >, 21  
    sysctls, 22  
    warn, 21  
anonymous\_namespace{loadplay.cpp}::Callback  
    Callback, 55  
    operator(), 57  
anonymous\_namespace{loadplay.cpp}::Callback<  
    FunctionArgs >, 54  
anonymous\_namespace{loadplay.cpp}::Emulator, 59  
    Emulator, 60  
    freqs, 61  
    operator(), 61  
anonymous\_namespace{loadplay.cpp}::Hold  
    Hold, 66  
    ref, 67  
    restore, 67  
anonymous\_namespace{loadplay.cpp}::Hold< T >, 66  
anonymous\_namespace{loadplay.cpp}::Main, 67  
    Main, 68  
anonymous\_namespace{loadplay.cpp}::SysctlValue, 95  
    get, 97–99  
    mtx, 102  
    operator=, 99  
    registerOnSet, 100  
    set, 100, 101  
    size, 102  
    SysctlValue, 97  
    value, 102  
anonymous\_namespace{loadplay.cpp}::Sysctls, 92  
    addValue, 93  
    getMib, 93  
    mibs, 94  
    operator[], 94  
    sysctls, 94  
anonymous\_namespace{loadplay.cpp}::mib\_t, 68  
    mib\_t, 69  
    operator int \*, 69  
    operator int const \*, 70  
    operator<, 70  
    operator==, 70  
anonymous\_namespace{loadrec.cpp}, 22

    OPTIONS, 24  
    OE, 23  
    read\_args, 24  
    run, 24  
    verbose, 24  
anonymous\_namespace{powerd++.cpp}, 25  
    AcLineState, 29  
    g, 33  
    init, 30  
    init\_loads, 30  
    OPTIONS, 33  
    OE, 29  
    read\_args, 30  
    set\_mode, 30  
    signal\_recv, 31  
    sysctl\_fail, 31  
    update\_freq, 32  
    update\_loads, 32  
    verbose, 32  
anonymous\_namespace{powerd++.cpp}::Core, 26, 123  
anonymous\_namespace{powerd++.cpp}::FreqGuard,  
    64  
anonymous\_namespace{powerd++.cpp}::Global, 27,  
    124  
anonymous\_namespace{powerd++.cpp}::Global::AC↔  
    Set, 126  
argCount  
    nih, 43  
  
B  
bmatch  
    nih::Options, 75  
  
C  
c\_str  
    sys::sc\_error, 81  
Callback  
    anonymous\_namespace{loadplay.cpp}::Callback,  
    55  
celsius  
    clas, 34  
clas, 33  
    celsius, 34  
    freq, 34  
    ival, 34  
    load, 35  
    range, 35  
    samples, 36  
    temperature, 36  
clas.hpp, 103  
constants, 37  
    TJMAX\_SOURCES, 37  
constants.hpp, 104  
countof  
    utility, 51  
cptime\_t

- types, 50
- Cycle.hpp, 106
- D
- daemon
  - loadplay.cpp, 112
- E
- Emulator
  - anonymous\_namespace{loadplay.cpp}::Emulator, 60
- errors, 38
  - Exit, 39
  - ExitStr, 40
  - fail, 40
  - OK, 39
- errors.hpp, 107
- errors::Exception, 39, 108
- Exit
  - errors, 39
- ExitStr
  - errors, 40
- expose
  - nih::Options, 78
- F
- fail
  - anonymous\_namespace{loadplay.cpp}, 20
  - errors, 40
- fixme, 41
  - to\_string, 41
- fixme.hpp, 109
- freq
  - clas, 34
- freqs
  - anonymous\_namespace{loadplay.cpp}::Emulator, 61
- G
- g
  - anonymous\_namespace{powerd++.cpp}, 33
- get
  - anonymous\_namespace{loadplay.cpp}::Sysctl↔Value, 97–99
  - nih::Options, 75
  - sys::ctl::Sysctl, 86, 87
  - sys::ctl::Sysctl< 0 >, 90, 91
- getMib
  - anonymous\_namespace{loadplay.cpp}::Sysctls, 93
- geteuid
  - loadplay.cpp, 112
- H
- Hold
  - anonymous\_namespace{loadplay.cpp}::Hold, 66
- I
- init
  - anonymous\_namespace{powerd++.cpp}, 30
- init\_loads
  - anonymous\_namespace{powerd++.cpp}, 30
- ival
  - clas, 34
- L
- load
  - clas, 35
- loadplay.cpp, 110
  - daemon, 112
  - geteuid, 112
  - pidfile\_close, 112
  - pidfile\_fileno, 113
  - pidfile\_open, 113
  - pidfile\_remove, 113
  - pidfile\_write, 113
  - sysctl, 114
  - sysctlbyname, 114
  - sysctlnametomib, 115
- loadrec.cpp, 115
  - main, 117
- M
- Main
  - anonymous\_namespace{loadplay.cpp}::Main, 68
- main
  - anonymous\_namespace{loadplay.cpp}, 22
  - loadrec.cpp, 117
  - powerd++.cpp, 126
- make\_Once
  - sys::ctl, 46
- make\_Options
  - nih, 43
- make\_Sysctl
  - sys::ctl, 47
- match
  - nih::Options, 76
- mib\_t
  - anonymous\_namespace{loadplay.cpp}::mib\_t, 69
- mibs
  - anonymous\_namespace{loadplay.cpp}::Sysctls, 94
- mtx
  - anonymous\_namespace{loadplay.cpp}::Sysctl↔Value, 102
- N
- nih, 42
  - argCount, 43
  - make\_Options, 43
- nih::Option, 42, 120
- nih::Options
  - bmatch, 75
  - expose, 78
  - get, 75
  - match, 76
  - operator(), 76
  - operator[], 77
  - Options, 74
  - removePath, 77
  - usage, 77



nih::Options< Enum, DefCount >, 73  
 nih::enum\_has\_members< Enum, class >, 62  
 O  
 OPTIONS  
     anonymous\_namespace{loadrec.cpp}, 24  
     anonymous\_namespace{powerd++.cpp}, 33  
 OE  
     anonymous\_namespace{loadrec.cpp}, 23  
     anonymous\_namespace{powerd++.cpp}, 29  
 OK  
     errors, 39  
 Once  
     sys::ctl::Once, 72  
 operator int  
     sys::sc\_error, 81  
 operator int \*  
     anonymous\_namespace{loadplay.cpp}::mib\_t, 69  
 operator int const \*  
     anonymous\_namespace{loadplay.cpp}::mib\_t, 70  
 operator T  
     sys::ctl::Sync, 84  
 operator T const &  
     sys::ctl::Once, 72  
 operator<  
     anonymous\_namespace{loadplay.cpp}::mib\_t, 70  
 operator()  
     anonymous\_namespace{loadplay.cpp}::Callback, 57  
     anonymous\_namespace{loadplay.cpp}::Emulator, 61  
     nih::Options, 76  
     timing::Cycle, 58, 59  
     utility::Formatter, 64  
 operator=  
     anonymous\_namespace{loadplay.cpp}::Sysctl←  
         Value, 99  
     sys::ctl::Sync, 84  
 operator==  
     anonymous\_namespace{loadplay.cpp}::mib\_t, 70  
 operator"" \_fmt  
     utility::literals, 53  
 operator"" \_r  
     anonymous\_namespace{loadplay.cpp}, 20  
 operator"" \_s  
     utility::literals, 54  
 operator[]  
     anonymous\_namespace{loadplay.cpp}::Sysctls, 94  
     nih::Options, 77  
 Options  
     nih::Options, 74  
 Options.hpp, 118  
 P  
 pfh  
     sys::pid::Pidfile, 80  
 Pidfile  
     sys::pid::Pidfile, 79  
 pidfile\_close  
     loadplay.cpp, 112  
 pidfile\_fileno  
     loadplay.cpp, 113  
 pidfile\_open  
     loadplay.cpp, 113  
 pidfile\_remove  
     loadplay.cpp, 113  
 pidfile\_write  
     loadplay.cpp, 113  
 powerd++.cpp, 121  
     main, 126  
 R  
 range  
     clas, 35  
 read\_args  
     anonymous\_namespace{loadrec.cpp}, 24  
     anonymous\_namespace{powerd++.cpp}, 30  
 ref  
     anonymous\_namespace{loadplay.cpp}::Hold, 67  
 registerOnSet  
     anonymous\_namespace{loadplay.cpp}::Sysctl←  
         Value, 100  
 removePath  
     nih::Options, 77  
 restore  
     anonymous\_namespace{loadplay.cpp}::Hold, 67  
 run  
     anonymous\_namespace{loadrec.cpp}, 24  
 S  
 samples  
     clas, 36  
 set  
     anonymous\_namespace{loadplay.cpp}::Sysctl←  
         Value, 100, 101  
     sys::ctl::Sysctl, 87, 88  
     sys::ctl::Sysctl< 0 >, 91  
 set\_mode  
     anonymous\_namespace{powerd++.cpp}, 30  
 Signal  
     sys::sig::Signal, 82  
 signal\_recv  
     anonymous\_namespace{powerd++.cpp}, 31  
 size  
     anonymous\_namespace{loadplay.cpp}::Sysctl←  
         Value, 102  
 sprintf  
     utility, 52  
 sprintf\_safe  
     utility, 52  
 strcmp  
     anonymous\_namespace{loadplay.cpp}, 21  
 Sync  
     sys::ctl::Sync, 83  
 sys, 44  
 sys/error.hpp, 127  
 sys/pidfile.hpp, 128  
 sys/signal.hpp, 129

sys/sysctl.hpp, 131  
 sys::ctl, 44  
     make\_Once, 46  
     make\_Sysctl, 47  
     sysctl\_get, 47  
     sysctl\_raw, 48  
     sysctl\_set, 48  
     SysctlOnce, 45  
     SysctlSync, 46  
 sys::ctl::Once  
     Once, 72  
     operator T const &, 72  
 sys::ctl::Once< T, SysctlT >, 71  
 sys::ctl::Sync  
     operator T, 84  
     operator=, 84  
     Sync, 83  
 sys::ctl::Sync< T, SysctlT >, 82  
 sys::ctl::Sysctl  
     get, 86, 87  
     set, 87, 88  
     Sysctl, 86  
 sys::ctl::Sysctl< 0 >, 88  
     get, 90, 91  
     set, 91  
     Sysctl, 89  
 sys::ctl::Sysctl< MibDepth >, 84  
 sys::ctl::error, 45, 132  
 sys::pid, 49  
 sys::pid::Pidfile, 78  
     pfh, 80  
     Pidfile, 79  
     write, 79  
 sys::pid::error, 49, 129  
 sys::sc\_error  
     c\_str, 81  
     operator int, 81  
 sys::sc\_error< Domain >, 80  
 sys::sig, 49  
 sys::sig::Signal, 81  
     Signal, 82  
 sys::sig::error, 50, 130  
 Sysctl  
     sys::ctl::Sysctl, 86  
     sys::ctl::Sysctl< 0 >, 89  
 sysctl  
     loadplay.cpp, 114  
 sysctl\_fail  
     anonymous\_namespace{powerd++.cpp}, 31  
 sysctl\_get  
     sys::ctl, 47  
 sysctl\_raw  
     sys::ctl, 48  
 sysctl\_set  
     sys::ctl, 48  
 SysctlOnce  
     sys::ctl, 45  
 SysctlSync  
     sys::ctl, 46  
 SysctlValue  
     anonymous\_namespace{loadplay.cpp}::Sysctl↔  
         Value, 97  
 SysctlValue::get< std::string >  
     anonymous\_namespace{loadplay.cpp}, 21  
 sysctlbyname  
     loadplay.cpp, 114  
 sysctlnametomib  
     loadplay.cpp, 115  
 sysctls  
     anonymous\_namespace{loadplay.cpp}, 22  
     anonymous\_namespace{loadplay.cpp}::Sysctls, 94  
  
 T  
 TJMAX\_SOURCES  
     constants, 37  
 temperature  
     clas, 36  
 timing, 50  
 timing::Cycle, 57  
     operator(), 58, 59  
 to\_string  
     fixme, 41  
 to\_value  
     utility, 53  
 types, 50  
     cptime\_t, 50  
 types.hpp, 133  
  
 U  
 Unit  
     anonymous\_namespace{clas.cpp}, 18  
 unit  
     anonymous\_namespace{clas.cpp}, 19  
 UnitStr  
     anonymous\_namespace{clas.cpp}, 19  
 update\_freq  
     anonymous\_namespace{powerd++.cpp}, 32  
 update\_loads  
     anonymous\_namespace{powerd++.cpp}, 32  
 usage  
     nih::Options, 77  
 utility, 51  
     countof, 51  
     sprintf, 52  
     sprintf\_safe, 52  
     to\_value, 53  
 utility.hpp, 134  
 utility::Formatter  
     operator(), 64  
 utility::Formatter< BufSize >, 63  
 utility::literals, 53  
     operator""\_fmt, 53  
     operator""\_s, 54  
  
 V  
 value

- anonymous\_namespace{loadplay.cpp}::Sysctl↔  
Value, [102](#)
- verbose
  - anonymous\_namespace{loadrec.cpp}, [24](#)
  - anonymous\_namespace{powerd++.cpp}, [32](#)
- W
- warn
  - anonymous\_namespace{loadplay.cpp}, [21](#)
- write
  - sys::pid::Pidfile, [79](#)