powerd++

0.4.1

Generated by Doxygen 1.8.15

1 Main Page

```
                                      /\  __
   __ __ ___  ___  __ _____  ___  ___/ /_/ /_
  /_//_//   \/   \/ // / _ \/ __\/    /_   _/__
 __ __ / // / // /    / ___/ /  / // /  /_/_/ /_
/_//_// ___/\___/\_/_/\___/_/   \___/     /_  _/
     / /                                   /_/
     \/ multi-core CPU clock daemon for FreeBSD®
```

The powerd++ daemon is a drop-in replacement for FreeBSD's native powerd. Its purpose is to reduce the energy consumption of CPUs for the following benefits:

- Avoid unnecessary fan noise from portable devices
- Improve the battery runtime of portable devices
- Improve hardware lifetime by reducing thermal stress
- Energy conservation

Contents

1.1 Using powerd++

Powerd++ offers the following features:

- Load target based clock frequency control
- Tunable sampling with moving average filter
- Load recording and replay tooling for benchmarking, tuning and reporting issues
- Command line compatibility with `powerd(8)`
- Temperature based throttling
- Expressive command line arguments with units, ranges and argument chaining
- Helpful error messages
- Comprehensive manual pages

1.1.1 Packages

The `FreeBSD` port is `sysutils/powerdxx`, the package name `powerdxx`.

1.1.2 Running powerd++

It is not intended to run powerd++ simultaneously with powerd. To prevent this powerd++ uses the same default pidfile as powerd:

```
# service powerdxx onestart
Starting powerdxx.
powerd++: (ECONFLICT) a power daemon is already running under PID: 59866
/usr/local/etc/rc.d/powerdxx: WARNING: failed to start powerdxx
```

So if powerd is already setup, it first needs to be disabled:

```
# service powerd stop
Stopping powerd.
Waiting for PIDS: 50127.
# service powerd disable
powerd disabled in /etc/rc.conf
```

Afterwards powerd++ can be enabled:

```
# service powerdxx enable
powerdxx enabled in /etc/rc.conf
# service powerdxx start
Starting powerdxx.
```

1.1.3 Manuals

Comprehensive manual pages exist for powerd++ and its accompanying tools loadrec and loadplay:

```
> man powerd++ loadrec loadplay
```

The current version of the manual pages may be read directly from the repository:

```
> man man/*
```

The manual pages as of the last release can also be `read online`.

1.1.4 Tuning

Three parameters affect the responsiveness of powerd++:

- The load target (refer to `-a`, `-b` and `-n`)
- The polling interval (refer to `-p`)
- The sample count (refer to `-s`)

The key to tuning powerd++ is the `-f` flag, which keeps powerd++ in foreground and causes it to report its activity. This allows directly observing the effects of a parameter set.

Observing the defaults in action may be a good start:

```
# powerd++ -f
power:  online, load:  693 MHz,  42 C, cpu.0.freq: 2401 MHz, wanted: 1848 MHz
power:  online, load:  475 MHz,  43 C, cpu.0.freq: 1800 MHz, wanted: 1266 MHz
power:  online, load:  271 MHz,  43 C, cpu.0.freq: 1300 MHz, wanted:  722 MHz
power:  online, load:   64 MHz,  43 C, cpu.0.freq:  768 MHz, wanted:  170 MHz
power:  online, load:   55 MHz,  42 C, cpu.0.freq:  768 MHz, wanted:  146 MHz
power:  online, load:   57 MHz,  42 C, cpu.0.freq:  768 MHz, wanted:  152 MHz
power:  online, load:   60 MHz,  44 C, cpu.0.freq:  768 MHz, wanted:  160 MHz
power:  online, load:   67 MHz,  42 C, cpu.0.freq:  768 MHz, wanted:  178 MHz
...
```

Note, the immediate high load is due to the load buffer being filled under the assumption that the past load fits the current clock frequency when powerd++ starts.

1.1.5 Reporting Issues / Requesting Features

Please report issues and feature requests on `GitHub` or to `kamikaze@bsdforen.de`.

If powerd++ behaves in some unexpected or undesired manner, please mention all the command line flags (e.g. from `/etc/rc.conf` `powerdxx_flags`) and provide a load recording:

```
> loadrec -o myissue.load
```

The default recording duration is 30 s. Do not omit the `-o` parameter, printing the output on the terminal may create significant load and impact the recorded load significantly.

Before submitting the report, try to reproduce the behaviour using the recorded load:

```
> loadplay -i myissue.load -o /dev/null powerd++ -f
power:  online, load:  224 MHz, cpu.0.freq:  768 MHz, wanted:  597 MHz
power:  online, load:  155 MHz, cpu.0.freq:  768 MHz, wanted:  413 MHz
power:  online, load:   85 MHz, cpu.0.freq:  768 MHz, wanted:  226 MHz
power:  online, load:   29 MHz, cpu.0.freq:  768 MHz, wanted:   77 MHz
power:  online, load:   23 MHz, cpu.0.freq:  768 MHz, wanted:   61 MHz
...
```

1.2 Building/Installing

The `Makefile` offers a set of targets, it is written for FreeBSD's `make(1)`:

Target	Description
all	Build everything
debug	Build with `CXXFLAGS=-O0 -g -DEBUG`
paranoid	Turn on undefined behaviour canaries
install	Install tools and manuals
deinstall	Deinstall tools and manuals
clean	Clear build directory `obj/`
doc	Build HTML documentation
gh-pages	Build and publish HTML and PDF documentation

1.2.1 Building

The `all` target is the default target that is called implicitly if make is run without arguments:

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

The `debug` and `paranoid` flags perform the same build as the `all` target, but with different/additional `CXXFLAGS`. The `debug` and `paranoid` targets can be combined.

1.2.2 Installing

The installer installs the tools and manual pages according to a recipe in `pkg/files`. The following variables can be passed to `make install` or `make deinstall` to affect the install destination:

Variable	Default
`DESTDIR`	
`PREFIX`	`/usr/local`
`DOCSDIR`	`${PREFIX}/share/doc/powerdxx`

`DESTDIR` can be used to install powerd++ into a chroot or jail, e.g. to put it into the staging area when building a package using the FreeBSD ports. Unlike `PREFIX` and `DOCSDIR` it does not affect the installed files themselves.

1.2.3 Documentation

Building the documentation requires `doxygen` 1.8.15 or later, building the PDF version of the documentation requires `xelatex` as provided by the tex-xetex package.

The `doc` target populates `doc/html` and `doc/latex`, to create the PDF documentation `doc/latex/refman.pdf` must be built.

The `gh-pages` target builds the HTML and PDF documentation and drops it into the `gh-pages` submodule for publishing on `github.io`.

1.3 Development

The following table provides an overview of repository contents:

File/Folder	Contents
`doc/`	Output directory for doxygen documentation
`doxy/`	Doxygen configuration and filter scripts
`gh-pages/`	Submodule for publishing the documentation
`man/`	Manual pages written using mdoc(7) markup
`obj/`	Build output

File/Folder	Contents
`pkg/`	Installer scripts and instructions
`src/`	C++ source files
`src/sys/`	C++ wrappers for common C interfaces
`powerd++.rc`	Init script / service description
`LICENSE.md`	ISC license
`Makefile`	Build instructions
`README.md`	Project overview

1.3.1 Design

The life cycle of the powerd++ process goes through three stages:

1. Command line argument parsing
2. Initialisation and optionally printing the detected/configured parameters
3. Clock frequency control

The first stage is designed to maximise usability by providing both, the compact short option syntax (e.↩
g. `-vfbhadp`) as well as the more self-descriptive long option syntax (e.g. `--verbose --foreground --batt
hiadaptive`).

The second stage is designed to trigger all known error conditions in order to fail before calling daemon(3) at the start of the third stage. Both the first and second stage are meant to provide specific, helpful error messages.

The third stage tracks the CPU load and performs clock frequency control. It is designed to provide its functionality with as little runtime as possible. This is achieved by:

- Using integer arithmetic only
- Minimising branching

The latter is achieved by using function templates to roll out possible runtime state combinations as multiple functions. A single, central switch/case selects the correct function each cycle. This basically rolls out multiple code paths through a single function into multiple functions with a single code path.

The trade-off made is for runtime over code size. With every bit of state rolled out like this the number of functions that need to be generated doubles, thus this approach is limited to the few bits of state that control the most expensive functionality, e.g. the foreground mode.

1.3.2 License

This project is published under the ISC license.

2 LICENSE

3 Manual 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 injects libloadplay.so into command.  This library
     simulates the load from the input and outputs load statistics.


   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 0.001 second resolution.

`cpu.%d.rec.freq[MHz]`
The recorded clock frequency, sampled at the end of the frame.

`cpu.%d.rec.load[MHz]`
The recorded load in 0.1 MHz resolution.

`cpu.%d.run.freq[MHz]`
The simulated clock frequency set by the host process, sampled at the end of the frame.

`cpu.%d.run.load[MHz]`
The simulated load in 0.1 MHz resolution.

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.

IMPLEMENTATION NOTES

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 host 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 host process happens solely through the sysctl
     table. The simulation reads the recorded loads and the current core
     frequencies to update kern.cp_times.  The host 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.


ENVIRONMENT
     LOADPLAY_IN
             If set the file named is used for input instead of stdin.  This
             only affects the input of loadplay, the host process is not
             affected.


     LOADPLAY_OUT
             If set the file named is used for output instead of stdout.  This
             only affects the output of loadplay, the host process is not
             affected.


     LD_PRELOAD
             Used to inject libloadplay.so into the host process.


FILES
     /usr/local/lib/libloadplay.so
             A library injected into command via the LD_PRELOAD environment
             variable.


EXAMPLES
     Play a load recording with loadplay:


         > loadplay -i loads/freq_tracking.load powerd++
      time[s] cpu.0.rec.freq[MHz] cpu.0.rec.load[MHz] cpu.0.run.freq[MHz] cpu.0.run.load[MHz] cpu.1.rec.freq[MHz]
       0.025 1700 1700.0 1700 1700.0 1700 0.0 1700 0.0 1700 1700.0 1700 1700.0 1700 850.0 1700 850.0
        0.050 1700 1700.0 1700 1700.0 1700 1700.0 1700 1700.0 1700 0.0 1700 0.0 1700 0.0 1700 0.0
        0.075 1700 566.7 1700 566.6 1700 1700.0 1700 1700.0 1700 0.0 1700 0.0 1700 566.7 1700 566.6
         0.100 1700 0.0 1700 0.0 1700 0.0 1700 0.0 1700 0.0 1700 0.0 1700 0.0 1700 0.0
         0.125 1700 0.0 1700 0.0 1700 0.0 1700 0.0 1700 0.0 1700 0.0 1700 0.0 1700 0.0
         0.150 1700 0.0 1700 0.0 1700 0.0 1700 0.0 1700 0.0 1700 0.0 1700 0.0 1700 0.0
         0.175 1700 0.0 1700 0.0 1700 0.0 1700 0.0 1700 0.0 1700 0.0 1700 0.0 1700 0.0
         0.200 1700 0.0 1700 0.0 1700 0.0 1700 0.0 1700 0.0 1700 0.0 1700 0.0 1700 0.0
         0.225 1700 0.0 1700 0.0 1700 0.0 1700 0.0 1700 0.0 1700 0.0 1700 0.0 1700 0.0
         0.250 1700 0.0 1700 0.0 1700 0.0 1700 0.0 1700 0.0 1700 0.0 1700 0.0 1700 0.0
         0.275 1700 0.0 1700 0.0 1700 0.0 1700 0.0 1700 0.0 1700 0.0 1700 0.0 1700 0.0


     Capture load and loadplay output simultaneously into two different files:


         > loadplay -i loads/freq_tracking.load -o load.csv powerd++ -f > load.out
```

Capture and display loadplay output:

```
> loadplay -i loads/freq_tracking.load -o load.csv powerd++ -f $|$ tee load.out
power:  online, load:  527 MHz, cpu0.freq: 1700 MHz, wanted: 1405 MHz
power:  online, load:  459 MHz, cpu0.freq: 1400 MHz, wanted: 1224 MHz
power:  online, load:  502 MHz, cpu0.freq: 1200 MHz, wanted: 1338 MHz
power:  online, load:  548 MHz, cpu0.freq: 1300 MHz, wanted: 1461 MHz
power:  online, load:  704 MHz, cpu0.freq: 1500 MHz, wanted: 1877 MHz
power:  online, load:  750 MHz, cpu0.freq: 1900 MHz, wanted: 2000 MHz
power:  online, load:  805 MHz, cpu0.freq: 2000 MHz, wanted: 2146 MHz
power:  online, load:  772 MHz, cpu0.freq: 2200 MHz, wanted: 2058 MHz
power:  online, load:  574 MHz, cpu0.freq: 2000 MHz, wanted: 1530 MHz
power:  online, load:  515 MHz, cpu0.freq: 1500 MHz, wanted: 1373 MHz
```

SEE ALSO
    loadrec(1), powerd(8), powerd++(8), rtld(1), signal(3), tee(1)

AUTHORS
    Implementation and manual by Dominic Fandrey <kami@freebsd.org>

FreeBSD 12.0-STABLE   20 February, 2019   FreeBSD 12.0-STABLE

4 Manual 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 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 1700/968
```

```
     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
     steppings 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.
    # powerd++ -M1700
```

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

```
    > loadrec -d.25s
    usr.app.powerdxx.loadrec.features=1
    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=768
 dev.cpu.0.freq_levels=2401/15000 2400/15000 2300/14088 2200/13340 2000/11888 1900/11184 1800/10495 1700/9680
 0 768 768 768 768 728001 0 278439 54957 10215972 753315 0 245117 7838 10270972 767662 0 241991 37110 10230545 77
    25 768 768 768 768 0 0 0 0 3 2 0 0 0 1 0 0 0 0 3 0 0 0 0 4
    25 768 768 768 768 0 0 0 0 3 1 0 0 0 2 0 0 0 0 3 1 0 0 0 2
    25 768 768 768 768 0 0 1 0 2 1 0 2 0 0 0 0 0 0 3 0 0 1 0 2
    25 768 768 768 768 3 0 0 0 1 1 0 2 0 1 1 0 3 0 0 2 0 2 0 0
    25 768 768 768 768 0 0 0 0 3 0 0 1 0 2 0 0 0 0 3 3 0 0 0 0
    25 768 768 768 768 0 0 0 0 3 0 0 0 0 3 0 0 0 0 3 2 0 1 0 0
    25 768 768 768 768 0 0 0 0 3 0 0 0 0 3 0 0 0 0 3 2 0 1 0 0
    25 768 768 768 768 2 0 0 0 1 1 0 1 0 1 0 0 2 0 1 2 0 1 0 0
    25 768 768 768 768 0 0 0 0 4 1 0 2 0 1 0 0 0 0 4 0 0 1 0 2
    25 768 768 768 768 0 0 0 0 3 2 0 1 0 0 0 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>

FreeBSD 12.0-STABLE            4 February, 2019           FreeBSD 12.0-STABLE
```

5 Manual 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 hadp).


  -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 hadp).


  -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 runtime 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 at 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).

/usr/local/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++ -s1 -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>


FreeBSD 12.0-STABLE              9 May, 2017             FreeBSD 12.0-STABLE
```

6 Namespace Index

6.1 Namespace List

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

7 Hierarchical Index

7.1 Class Hierarchy

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

8 Class Index

8.1 Class List

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

9 File Index

9.1 File List

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

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
THZ	thz
CELSIUS	C.
KELVIN	K.
FAHRENHEIT	F.
RANKINE	R.
UNKNOWN	Unknown unit.

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

str	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{libloadplay.cpp} Namespace Reference

File local scope.

Classes

- class Callback

  Implements a recursion safe std::function wrapper.
- struct CoreFrameReport

  The report frame information for a single CPU pipeline. More...
- struct CoreReport

  The reported state of a single CPU pipeline. More...
- 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 Report

  Provides a mechanism to provide frame wise per core load information.
- 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 strncmp, 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<typename ... ArgTs>
  constexpr void dprintf (ArgTs &&... args)

  Calls fprintf(stderr, ...) if built with -DEBUG.
- template<>
  std::string SysctlValue::get< std::string > () const

  Returns a copy of the value string.
- void debug (std::string const &msg)

  Print a debugging message if built with -DEBUG.
- 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.
- std::ostream & operator<< (std::ostream &out, CoreReport const &core)

  Print a core clock frequency and load.
- std::ostream & operator<< (std::ostream &out, CoreFrameReport const &frame)

  Print recorded and running clock frequency and load for a frame.

Variables

- constexpr flag_t const FEATURES

  The set of supported features.
- int sys_results = 0

  The success return value of intercepted functions.
- class anonymous_namespace{libloadplay.cpp}::Sysctls sysctls

  Sole instance of Sysctls.
- class anonymous_namespace{libloadplay.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

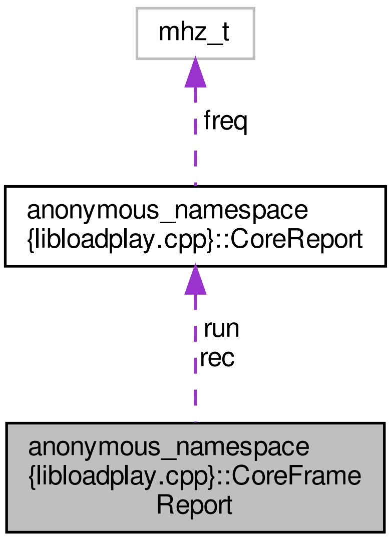
File local scope.

10.2.2 Class Documentation

10.2.2.1 struct anonymous_namespace{libloadplay.cpp}::CoreFrameReport

The report frame information for a single CPU pipeline.

Collaboration diagram for anonymous_namespace{libloadplay.cpp}::CoreFrameReport:


mhz_t
freq
anonymous_namespace {libloadplay.cpp}::CoreReport
run
rec
anonymous_namespace {libloadplay.cpp}::CoreFrame Report

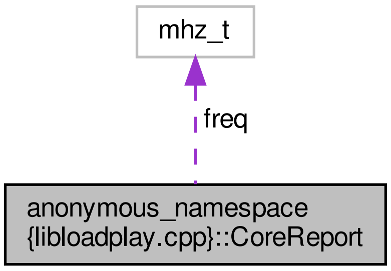
Class Members

CoreReport	rec	The recorded core state.
CoreReport	run	The running core state.

10.2.2.2 struct anonymous_namespace{libloadplay.cpp}::CoreReport

The reported state of a single CPU pipeline.

Collaboration diagram for anonymous_namespace{libloadplay.cpp}::CoreReport:



Class Members

mhz_t	freq	The core clock frequency in [MHz].
double	load	The core load as a fraction.

10.2.3 Function Documentation

10.2.3.1 debug()

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

Print a debugging message if built with -DEBUG.

Parameters

msg	The warning message

10.2.3.2 dprintf()

```
template<typename ... ArgTs>
constexpr void anonymous_namespace{libloadplay.cpp}::dprintf (
              ArgTs &&... args )
```

Calls fprintf(stderr, ...) if built with -DEBUG.

Template Parameters

Arg↩ Ts	The argument types to forward

Parameters

args	Arguments are forwarded to fprintf()

10.2.3.3 fail()

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

This prints an error message and sets sys_results to make the hijacked process fail.

Parameters

msg	The error message

10.2.3.4 operator""""_r()

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

User defined literal for regular expressions.

Parameters

str,len	The literal string and its length

Returns

A regular expression

10.2.3.5 operator<<() [1/2]

```
std::ostream& anonymous_namespace{libloadplay.cpp}::operator<< (
              std::ostream & out,
              CoreReport const & core )
```

Print a core clock frequency and load.

The clock frequency is printed at 1 MHz resolution, the load at 0.1 MHz.

Parameters

out	The stream to print to
core	The core information to print

Returns

A reference to the out stream

10.2.3.6 operator<<() [2/2]

```
std::ostream& anonymous_namespace{libloadplay.cpp}::operator<< (
              std::ostream & out,
              CoreFrameReport const & frame )
```

Print recorded and running clock frequency and load for a frame.

Parameters

out	The stream to print to
frame	The frame information to print

Returns

A reference to the out stream

10.2.3.7 strcmp()

```
template<size_t Size>
int anonymous_namespace{libloadplay.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

Size	The size of the buffer s2

Parameters

s1,s2	The strings to compare

Return values

0	Strings are equal
!0	Strings are not equal

10.2.3.8 SysctlValue::get< std::string >()

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

Returns a copy of the value string.

Returns

The value

10.2.3.9 warn()

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

Print a warning.

Parameters

msg	The warning message

10.2.4 Variable Documentation

10.2.4.1 FEATURES

```
constexpr flag_t const anonymous_namespace{libloadplay.cpp}::FEATURES
```

Initial value:

```
{
    1_FREQ_TRACKING
}
```

The set of supported features.

This value is used to ensure correct input data interpretation.

10.2.4.2 main

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

Sole instance of Main.

10.2.4.3 sysctls

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

Sole instance of Sysctls.

10.3 anonymous_namespace{loadplay.cpp} Namespace Reference

File local scope.

Enumerations

- enum OE {
  OE::USAGE, OE::FILE_IN, OE::FILE_OUT, OE::CMD,
  OE::OPT_NOOPT = CMD, OE::OPT_UNKNOWN, OE::OPT_DASH, OE::OPT_LDASH,
  OE::OPT_DONE }

  An enum for command line parsing.

Functions

- char const ∗ filename (char const ∗const path)

  Performs very rudimentary file name argument checks.
- void execute (char const ∗const file, char ∗const argv[ ])

  Executes the given command, substituting this process.

Variables

- char const ∗const USAGE = ”[-h] [-i file] [-o file] command [...]”

  The short usage string.
- Parameter< OE > const PARAMETERS [ ]

  Definitions of command line parameters.

10.3.1 Detailed Description

File local scope.

10.3.2 Enumeration Type Documentation

10.3.2.1 OE

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

An enum for command line parsing.

Enumerator

USAGE	Print help.
FILE_IN	Set input file instead of stdin.
FILE_OUT	Set output file instead of stdout.
CMD	The command to execute.
OPT_NOOPT	Obligatory.
OPT_UNKNOWN	Obligatory.
OPT_DASH	Obligatory.
OPT_LDASH	Obligatory.
OPT_DONE	Obligatory.

10.3.3 Function Documentation

10.3.3.1 execute()

```
void anonymous_namespace{loadplay.cpp}::execute (
            char const *const file,
            char *const argv[] )
```

Executes the given command, substituting this process.

This function is a wrapper around execvp(3) and does not return.

Parameters

file	The command to execute, looked up in PATH if no path is provided
argv	The command line arguments of the command

Exceptions

errors::Exception{Exit::EEXEC}	

10.3.3.2 filename()

```
char const* anonymous_namespace{loadplay.cpp}::filename (
            char const *const path )
```

Performs very rudimentary file name argument checks.

- Fail on empty path
- Return nullptr on '-'

Parameters

path	The file path to check

Returns

The given path or nullptr if the given path is '-'

10.3.4 Variable Documentation

10.3.4.1 PARAMETERS

```
Parameter<OE> const anonymous_namespace{loadplay.cpp}::PARAMETERS[]
```

Initial value:

```
{
    {OE::USAGE,    'h', "help",   "",              "Show usage and exit"},
    {OE::FILE_IN,  'i', "input",  "file",          "Input file (load recording)"},
    {OE::FILE_OUT, 'o', "output", "file",          "Output file (replay stats)"},
    {OE::CMD,       0 , "",       "command,[...]", "The command to execute"},
}
```

Definitions of command line parameters.

10.4 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

- constexpr flag_t const FEATURES

  The set of supported features.
- 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.
  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.
- Parameter< OE > const PARAMETERS [ ]

  Definitions of command line parameters.

10.4.1 Detailed Description

File local scope.

10.4.2 Enumeration Type Documentation

10.4.2.1 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.

10.4.3 Function Documentation

10.4.3.1 read_args()

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

Parse command line arguments.

Parameters

argc,argv	The command line arguments

10.4.3.2 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.

10.4.3.3 verbose()

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

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

Parameters

msg	The message to output

10.4.4 Variable Documentation

10.4.4.1 FEATURES

```
constexpr flag_t const anonymous_namespace{loadrec.cpp}::FEATURES
```

Initial value:
```
{
    1_FREQ_TRACKING
}
```

The set of supported features.

This value is stored in load recordings to allow loadplay to correctly interpret the data.

10.4.4.2 PARAMETERS

```
Parameter<OE> const anonymous_namespace{loadrec.cpp}::PARAMETERS[]
```

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", "Ignored"},
}
```

Definitions of command line parameters.

10.5 anonymous_namespace{powerd++.cpp} Namespace Reference

File local scope.

Classes

- struct Core

  Contains the management information for a single CPU core. More...
- struct CoreGroup

  Contains the management information for a group of cores with a common clock frequency. 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.
- Parameter< OE > const PARAMETERS [ ]

  Definitions of command line parameters.

10.5.1 Detailed Description

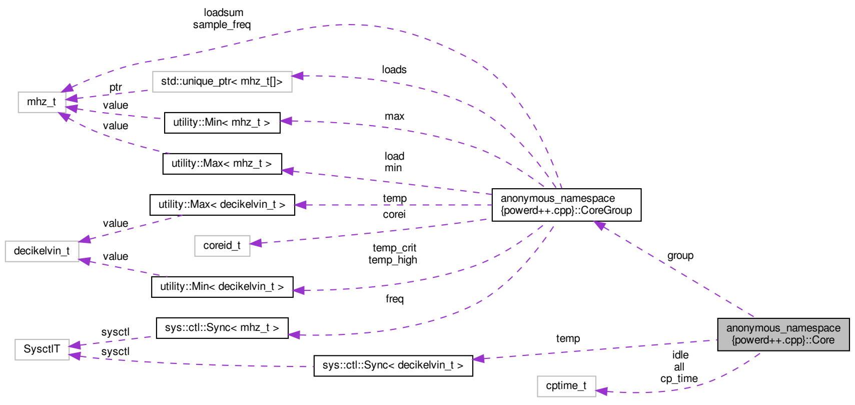
File local scope.

10.5.2 Class Documentation

10.5.2.1 struct anonymous_namespace{powerd++.cpp}::Core

Contains the management information for a single CPU core.

Collaboration diagram for anonymous_namespace{powerd++.cpp}::Core:



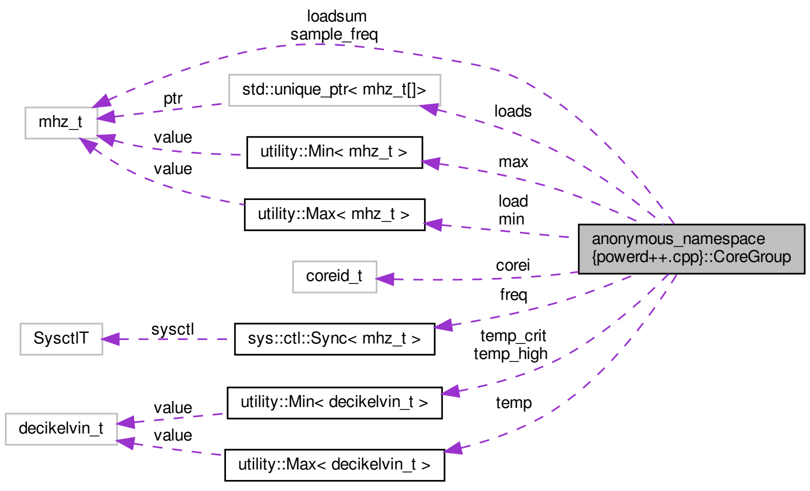
Class Members

cptime_t	all	Count of all ticks.
cptime_t const ∗	cp_time	A pointer to the kern.cp_times section for this core.
CoreGroup ∗	group	The core that controls the frequency for this core.
cptime_t	idle	The idle ticks count.
SysctlSync< decikelvin_t >	temp	The dev.cpu. d.temperature sysctl, if present.

10.5.2.2 struct anonymous_namespace{powerd++.cpp}::CoreGroup

Contains the management information for a group of cores with a common clock frequency.

Collaboration diagram for anonymous_namespace{powerd++.cpp}::CoreGroup:



Class Members

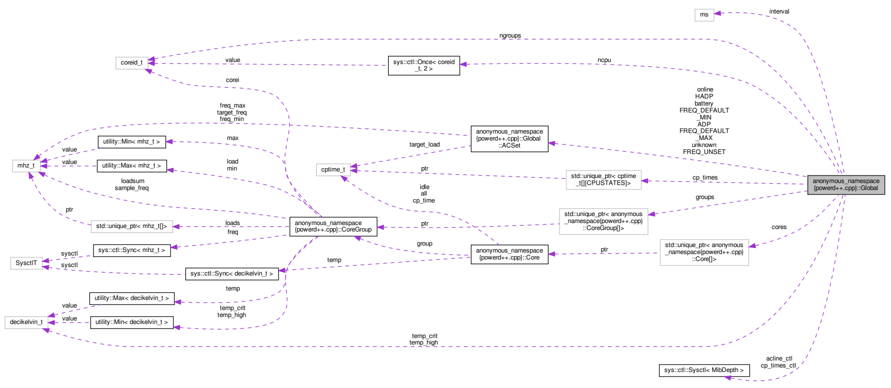
coreid_t	corei	The number of the core owning dev.cpu. d.freq.
SysctlSync< mhz_t >	freq	The sysctl dev.cpu. d.freq.
Max< mhz_t >	load	The maximum load reported by all cores in the group. This is updated by update_loads().
unique_ptr< mhz_t[ ]>	loads	A ring buffer of maximum load samples for this core group. Each maximum load sample is weighted with the core frequency at which it was taken. This is updated by update_loads().
mhz_t	loadsum	The maximum load sum of all controlled cores. This is updated by update_loads().
Min< mhz_t >	max	The maximum group clock rate. The least of all core maxima in the group.
Max< mhz_t >	min	The minimum group clock rate. The greatest of all core minima in the group.
mhz_t	sample_freq	The dev.cpu. d.freq value for the current load sample. This is updated by update_loads().
Max< decikelvin_t >	temp	The maximum temperature measurement taken in the group.
Min< decikelvin_t >	temp_crit	Critical core temperature in dK.
Min< decikelvin_t >	temp_high	High core temperature in dK.

10.5.2.3 struct anonymous_namespace{powerd++.cpp}::Global

A collection of all the gloabl, mutable states.

This is mostly for semantic clarity.

Collaboration diagram for anonymous_namespace{powerd++.cpp}::Global:


ms
interval
ngroups
coreid_t
value
sys::ctl::Once< coreid _t, 2 >
ncpu
corei
freq_max
target_freq
freq_min
online
HADP
battery
FREQ_DEFAULT _MIN
ADP
FREQ_DEFAULT _MAX
unknown
FREQ_UNSET
anonymous_namespace {powerd++.cpp}::Global ::ACSet
target_load
utility::Min< mhz_t >
max
value
mhz_t
value
utility::Max< mhz_t >
load
min
cptime_t
ptr
std::unique_ptr< cptime _t[][CPUSTATES]>
cp_times
anonymous_namespace {powerd++.cpp}::Global
loadsum
sample_freq
idle
all
cp_time
groups
ptr
std::unique_ptr< mhz_t[]>
loads
freq
anonymous_namespace {powerd++.cpp}::CoreGroup
ptr
std::unique_ptr< anonymous _namespace{powerd++.cpp} ::CoreGroup[]>
cores
sys::ctl::Sync< mhz_t >
sysctl
SysctlT
sysctl
group
anonymous_namespace {powerd++.cpp}::Core
ptr
std::unique_ptr< anonymous _namespace{powerd++.cpp} ::Core[]>
temp
sys::ctl::Sync< decikelvin_t >
temp
utility::Max< decikelvin_t >
value
decikelvin_t
value
utility::Min< decikelvin_t >
temp_crit
temp_high
temp_crit
temp_high
acline_ctl
cp_times_ctl
sys::ctl::Sysctl< MibDepth >

Class Members

Sysctl	acline_ctl	The hw.acpi.acline ctl.
struct anonymous_namespace{powerd++	ADP[3]	
struct anonymous_namespace{powerd++	battery[3]	
unique_ptr< Core[ ]>	cores	This buffer is to be allocated with ncpu instances of the Core struct to store the management information of every core.
unique_ptr< cptime_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 anonymous_namespace{powerd++	FREQ_DEFAULT_MAX[3]	
struct anonymous_namespace{powerd++	FREQ_DEFAULT_MIN[3]	
struct anonymous_namespace{powerd++	FREQ_UNSET[3]	
unique_ptr< CoreGroup[ ]>	groups	This buffer is to be allocated with the number of core groups. A core group is created by init() for each core that has a dev.cpu.d.freq handle.
struct anonymous_namespace{powerd++	HADP[3]	
ms	interval	The polling interval.
SysctlOnce< coreid_t, 2 > const	ncpu	The number of CPU cores or threads.
coreid_t	ngroups	The number of frequency controlling core groups.
struct anonymous_namespace{powerd++	online[3]	
char const *	pidfilename	Name of an alternative pidfile. If not given pidfile_open() uses a default name.

Class Members

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.
decikelvin_t	temp_crit	User set critical core temperature in dK.
decikelvin_t	temp_high	User set high core temperature in dK.
bool	temp_throttling	Temperature throttling mode.
struct anonymous_namespace{powerd++	unknown[3]	The power states.
bool	verbose	Verbose mode.

10.5.3 Enumeration Type Documentation

10.5.3.1 AcLineState

```
enum anonymous_namespace{powerd++.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.

10.5.3.2 OE

```
enum anonymous_namespace{powerd++.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.

Enumerator

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.

10.5.4 Function Documentation

10.5.4.1 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

10.5.4.2 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.

10.5.4.3 read_args()

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

Parse command line arguments.

Parameters

argc,argv	The command line arguments

10.5.4.4 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" | "hiadptive" | "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	
hiadptive	A target load of 37.5%
hadp	

Parameters

line	The power line state to set the mode for
str	A mode string

10.5.4.5 signal_recv()

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

Sets g.signal, terminating the main loop.

Parameters

signal	The signal number received

10.5.4.6 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

err	The errno value after calling sysctl

10.5.4.7 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

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

Parameters

acstate	The set of acline dependent variables

10.5.4.8 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

Load	Determines whether CoreGroup::loadsum is updated
Temperature	Determines whether CoreGroup::temp is updated

10.5.4.9 verbose()

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

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

Parameters

msg	The message to output

10.5.5 Variable Documentation

10.5.5.1 g

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

The gobal state.

10.5.5.2 PARAMETERS

```
Parameter<OE> const anonymous_namespace{powerd++.cpp}::PARAMETERS[]
```

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 parameters.

10.6 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 ℃ for display purposes.
- template< typename T >
  std::pair< T, T > range (T(&func)(char const ∗const), char const ∗const str)

  Takes a string encoded range of values and returns them.

10.6.1 Detailed Description

A collection of functions to process command line arguments.

10.6.2 Function Documentation

10.6.2.1 celsius()

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

Converts dK into ℃ for display purposes.

Parameters

val	A temperature in dK

Returns

The temperature in ℃

10.6.2.2 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

str	A string encoded frequency

Returns

The frequency given by str

10.6.2.3 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

str	A string encoded time interval

Returns

The interval in milliseconds

10.6.2.4 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

str	A string encoded load

Return values

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

10.6.2.5 range()

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

Takes a string encoded range of values and returns them.

A range has the format `from:to`.

Template Parameters

T	The return type of the conversion function

Parameters

func	The function that converts the values from the string
str	The string containing the range

Returns

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

10.6.2.6 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

str	The string containing the number of samples

Returns

The number of samples

10.6.2.7 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 ℃ is assumed.

Parameters

str	A string encoded temperature

Returns

The temperature given by str

10.7 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 ACLINE = "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.

10.7.1 Detailed Description

A collection of constants.

10.7.2 Variable Documentation

10.7.2.1 TJMAX_SOURCES

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

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

An array of maximum temperature sources.

10.8 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::EEXCEPT, Exit::EFILE,
  Exit::EEXEC, 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.

10.8.1 Detailed Description

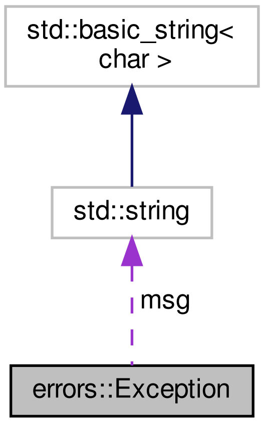
Common error handling types and functions.

10.8.2 Class Documentation

10.8.2.1 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.
Exit	exitcode	The code to exit with.
string	msg	An error message.

10.8.3 Enumeration Type Documentation

10.8.3.1 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.
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.
EEXCEPT	Untreated exception.
EFILE	Not a valid file name.
EEXEC	Command execution failed.
LENGTH	Enum length.

10.8.4 Function Documentation

10.8.4.1 fail()

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

Throws an Exception instance with the given message.

Parameters

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

10.8.5 Variable Documentation

10.8.5.1 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",
    "EEXCEPT", "EFILE", "EEXEC"
}
```

Printable strings for exit codes.

10.9 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().

10.9.1 Detailed Description

Workarounds for compiler/library bugs.

10.9.2 Function Documentation

10.9.2.1 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

T	The argument type to convert

Parameters

op	The argument to convert

Returns

A string of the given argument

10.10 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.
- class Options

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

  Container for an option definition. More...

Typedefs

- template<class... >
  using void_t = void

  See std::void_t in C++17 <type_traits>.

Functions

- template<class OptionT >
  size_t argCount (Parameter< OptionT > const &def)

  Retrieves the count of arguments in an option definition.
- template<class OptionT , size_t DefCount>
  constexpr Options< OptionT, DefCount > make_Options (int const argc, char const *const *const argv, char const *const usage, Parameter< OptionT > const (&defs)[DefCount])

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

10.10.1 Detailed Description

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

10.10.2 Class Documentation

10.10.2.1 struct nih::Parameter

template<class OptionT>
struct nih::Parameter< OptionT >

Container for an option definition.

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

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

Template Parameters

OptionT	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.
char const ∗	lparam	The long version of this parameter. Set to nullptr or "" if no long parameter is available.
OptionT	option	The enum value to return for this option.
char	sparam	The short version of this parameter. Set to 0 if no short parameter is available.
char const ∗	usage	A usage string.

10.10.3 Function Documentation

10.10.3.1 argCount()

```
template<class OptionT >
size_t nih::argCount (
            Parameter< OptionT > const & def )
```

Retrieves the count of arguments in an option definition.

Template Parameters

OptionT	An enum or enum class representing the available options

Parameters

def	The parameter definition

Returns

The number of arguments specified in the given definition

10.10.3.2 make_Options()

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

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

Template Parameters

OptionT	An enum for all the available options
DefCount	The number of option definitions

Parameters

argc,argv	The command line arguments
usage	A usage string that is used in the header of the usage output
defs	An array of parameter definitions

10.11 sys Namespace Reference

Wrappers around native system interfaces.

Namespaces

- ctl

  This namespace contains safer c++ wrappers for the sysctl() interface.
- env

  Provides wrappers around the getenv() family of functions.
- 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.

10.11.1 Detailed Description

Wrappers around native system interfaces.

10.12 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.

10.12.1 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.

10.12.2 Class Documentation

10.12.2.1 struct sys::ctl::error

The domain error type.

10.12.3 Typedef Documentation

10.12.3.1 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

T	The type to represent the sysctl as
MibDepth	The maximum allowed MIB depth

10.12.3.2 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

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

10.12.4 Function Documentation

10.12.4.1 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

T	The value type
SysctlT	The Sysctl type

Parameters

value	The default value to fall back to
sysctl	The sysctl to try and read from

10.12.4.2 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

Args	List of argument types, should all be pid_t

Parameters

args	List of initialising arguments

Returns

A Sysctl instance with the depth matching the number of arguments

10.12.4.3 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

MibDepth	The length of the MIB buffer

Parameters

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

Exceptions

sys::sc_error<error>	Throws if sysctl() fails for any reason

10.12.4.4 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

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

Exceptions

sys::sc_error<error>	Throws if sysctl() fails for any reason

10.12.4.5 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

MibDepth	The length of the MIB buffer

Parameters

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

Exceptions

sys::sc_error<error>	Throws if sysctl() fails for any reason

10.13 sys::env Namespace Reference

Provides wrappers around the getenv() family of functions.

Classes

- struct error

  The domain error type. More...
- class Var

  A reference type refering to an environment variable.
- struct Vars

  A singleton class providing access to environment variables.

Variables

- struct sys::env::Vars vars

  Singleton providing access to environment variables.

10.13.1 Detailed Description

Provides wrappers around the getenv() family of functions.

10.13.2 Class Documentation

10.13.2.1 struct sys::env::error

The domain error type.

10.13.3 Variable Documentation

10.13.3.1 vars

```
struct sys::env::Vars sys::env::vars
```

Singleton providing access to environment variables.

10.14 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.

10.14.1 Detailed Description

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

The class Pidfile implements the RAII pattern for holding a pidfile.

10.14.2 Class Documentation

10.14.2.1 struct sys::pid::error

The domain error type.

10.15 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.

10.15.1 Detailed Description

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

10.15.2 Class Documentation

10.15.2.1 struct sys::sig::error

The domain error type.

10.16 timing Namespace Reference

Namespace for time management related functionality.

Classes

- class Cycle

  Implements an interruptible cyclic sleeping functor.

10.16.1 Detailed Description

Namespace for time management related functionality.

10.17 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.

10.17.1 Detailed Description

A collection of type aliases.

10.17.2 Typedef Documentation

10.17.2.1 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.

10.18 utility Namespace Reference

A collection of generally useful functions.

Namespaces

- literals

    Contains literals.

Classes

- class Formatter

    A formatting wrapper around string literals.
- class Max

    A simple value container that provides the maximum of assigned values.
- class Min

    A simple value container that provides the minimum of assigned values.
- class Sum

    A simple value container only allowing += and copy assignment.

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.

10.18.1 Detailed Description

A collection of generally useful functions.

10.18.2 Function Documentation

10.18.2.1 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

T,Count	The type and number of array elements

Returns

The number of array entries

10.18.2.2 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

Args	Catch all arguments

10.18.2.3 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

Size	The destination buffer size
Args	The types of the arguments

Parameters

dst	A reference to the destination buffer
format	A printf style formatting string
args	The printf arguments

Returns

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

10.18.2.4 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

ET,VT	The enum and value type

Parameters

op	The operand to convert

Returns

The integer representation of the operand

10.19 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.

10.19.1 Detailed Description

Contains literals.

10.19.2 Function Documentation

10.19.2.1 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

fmt	A printf style format string

Returns

A Formatter instance

10.19.2.2 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

op	The raw string to turn into an std::string object
size	The size of the raw string

Returns

An std::string instance

10.20 version Namespace Reference

Version information constants and types.

Namespaces

- literals

  Literals to set flag bits.

Typedefs

- typedef uint64_t flag_t

  The data type to use for feature flags.

Enumerations

- enum LoadrecBits { LoadrecBits::FREQ_TRACKING }

  Feature flags for load recordings.

Variables

- char const ∗const LOADREC_FEATURES = ”usr.app.powerdxx.loadrec.features”

  The pseudo MIB name for the load recording feature flags.

10.20.1 Detailed Description

Version information constants and types.

10.20.2 Enumeration Type Documentation

10.20.2.1 LoadrecBits

```
enum version::LoadrecBits [strong]
```

Feature flags for load recordings.

Enumerator

FREQ_TRACKING	Record clock frequencies per frame.

10.21 version::literals Namespace Reference

Literals to set flag bits.

Functions

- constexpr flag_t operator"" _FREQ_TRACKING (unsigned long long int value)
  Set the FREQ_TRACKING bit.

10.21.1 Detailed Description

Literals to set flag bits.

10.21.2 Function Documentation

10.21.2.1 operator""""_FREQ_TRACKING()

```
constexpr flag_t version::literals::operator""_FREQ_TRACKING (
                unsigned long long int value )
```

Set the FREQ_TRACKING bit.

Parameters

value	The bit value

Returns

The flag at the correct bit position

11 Class Documentation

11.1 anonymous_namespace{libloadplay.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{libloadplay.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

Function↩ Args	The argument types of the callback function

11.1.2 Constructor & Destructor Documentation

11.1.2.1 Callback() [1/2]

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

Construct from function.

Parameters

callback	The callback function

11.1.2.2 Callback() [2/2]

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

Construct from temporary function.

Parameters

callback	The callback function

11.1.3 Member Function Documentation

11.1.3.1 operator()()

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

Forward call to callback functions.

Parameters

args	The arguments to the callback function

Exceptions

std::bad_function_call	In case this handler was default constructed or constructed from a nullptr

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

- src/libloadplay.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

true	Sleep completed uninterrupted
false	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

Dur↩ Traits	The traits of the duration type

Parameters

cycle↩ Time	The duration of the cycle to complete

Return values

true	Command completed uninterrupted
false	Command was interrupted

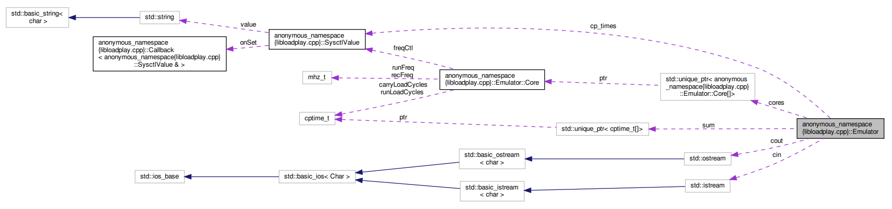
The documentation for this class was generated from the following file:

- src/Cycle.hpp

11.3 anonymous_namespace{libloadplay.cpp}::Emulator Class Reference

Instances of this class represent an emulator session.

Collaboration diagram for anonymous_namespace{libloadplay.cpp}::Emulator:



std::basic_string< char >
std::string
value
anonymous_namespace {libloadplay.cpp}::SysctlValue
cp_times
anonymous_namespace {libloadplay.cpp}::Callback < anonymous_namespace{libloadplay.cpp} ::SysctlValue & >
onSet
freqCtl
mhz_t
runFreq
recFreq
carryLoadCycles
runLoadCycles
anonymous_namespace {libloadplay.cpp}::Emulator::Core
ptr
std::unique_ptr< anonymous _namespace{libloadplay.cpp} ::Emulator::Core[]>
cores
cptime_t
ptr
std::unique_ptr< cptime_t[]>
sum
anonymous_namespace {libloadplay.cpp}::Emulator
cout
cin
std::basic_ostream < char >
std::ostream
std::ios_base
std::basic_ios< Char >
std::basic_istream < char >
std::istream

Classes

- struct Core

    Per core information. More...

Public Member Functions

- Emulator (std::istream &cin, std::ostream &cout, bool const &die)

    The constructor initialises all the members necessary for emulation.

- void operator() ()

    Performs load emulation and prints statistics on cout.

Private Attributes

- std::istream & cin

  The input data source.
- std::ostream & cout

  The output data sink.
- bool const & die

  A reference to a bool that tells the emulator to die.
- size_t const size = sysctls[CP_TIMES].size()

  The size of the kern.cp_times buffer.
- int const ncpu = this->size / sizeof(cptime_t[CPUSTATES])

  The number of CPUs in kern.cp_times, may be greater than the hw.ncpu value (e.g.
- std::unique_ptr< Core[ ]> cores {new Core[this->ncpu]{}}

  Simulation state information for each core.
- SysctlValue & cp_times = sysctls[CP_TIMES]

  The kern.cp_times sysctl handler.
- std::unique_ptr< cptime_t[ ]> sum {new cptime_t[CPUSTATES ∗ ncpu]{}}

  The current kern.cp_times values.

11.3.1 Detailed Description

Instances of this class represent an emulator session.

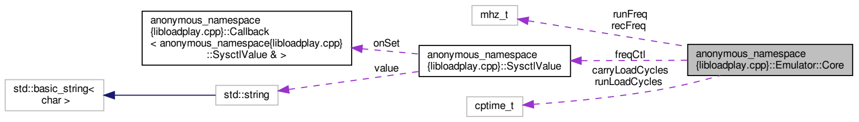
This should be run in its own thread and expects the sysctl table to be complete.

11.3.2 Class Documentation

11.3.2.1 struct anonymous_namespace{libloadplay.cpp}::Emulator::Core

Per core information.

Collaboration diagram for anonymous_namespace{libloadplay.cpp}::Emulator::Core:


anonymous_namespace {libloadplay.cpp}::Callback < anonymous_namespace{libloadplay.cpp} ::SysctlValue & >
onSet
value
std::basic_string< char >
std::string
mhz_t
runFreq
recFreq
anonymous_namespace {libloadplay.cpp}::SysctlValue
freqCtl
carryLoadCycles
runLoadCycles
anonymous_namespace {libloadplay.cpp}::Emulator::Core
cptime_t

Class Members

cptime_t	carryLoadCycles	The load cycles carried over to the next frame in [kcycles]. This is determined at the beginning of frame and used to calculated the simulation load at the beginning of the next frame.
SysctlValue ∗	freqCtl	The sysctl handler. The constructor ensures this points to a valid handler.
mhz_t	recFreq	The recorded clock frequency. If FREQ_TRACKING is enabled this is updated at during the preliminary stage and used at the beginning of frame stage.
mhz_t	runFreq	The clock frequency the simulation is running at. Updated at the end of frame and used in the next frame.
cptime_t	runLoadCycles	The load cycles simulated for this frame in [kcycles]. This is determined at the beginning of frame and used to calculate the reported load at the end of frame.

11.3.3 Constructor & Destructor Documentation

11.3.3.1 Emulator()

```
anonymous_namespace{libloadplay.cpp}::Emulator::Emulator (
              std::istream & cin,
              std::ostream & cout,
              bool const & die )  [inline]
```

The constructor initialises all the members necessary for emulation.

It also prints the column headers on stdout.

Exceptions

std::out_of_range	In case one of the required sysctls is missing

Parameters

cin,cout	The character input and output streams
die	If the referenced bool is true, emulation is terminated prematurely

11.3.4 Member Function Documentation

11.3.4.1 operator()()

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

Performs load emulation and prints statistics on cout.

Reads 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.

11.3.5 Member Data Documentation

11.3.5.1 ncpu

```
int const anonymous_namespace{libloadplay.cpp}::Emulator::ncpu = this->size / sizeof(cptime_t[CPUSTATES])
[private]
```

The number of CPUs in kern.cp_times, may be greater than the hw.ncpu value (e.g.

if hyperthreading was turned off).

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

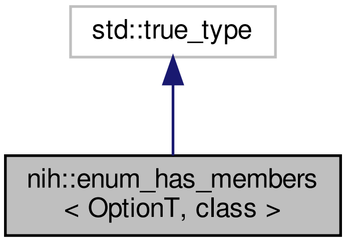
- src/libloadplay.cpp

11.4 nih::enum_has_members< OptionT, class > Struct Template Reference

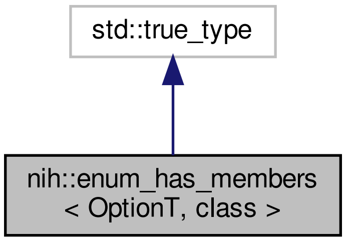
Tests whether the given enum provides all the required definitions.

`#include <Options.hpp>`

Inheritance diagram for nih::enum_has_members< OptionT, class >:



Collaboration diagram for nih::enum_has_members< OptionT, class >:



11.4.1 Detailed Description

template<class OptionT, class = void>
struct nih::enum_has_members< OptionT, 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

OptionT	An enum or enum class representing the available options

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

- src/Options.hpp

11.5 utility::Formatter< BufSize > 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

Buf↔ Size	The buffer size for formatting, resulting strings cannot grow beyond `BufSize - 1`

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

Arg↩ Ts	Variadic argument types

Parameters

args	Variadic arguments

Returns

An std::string formatted according to fmt

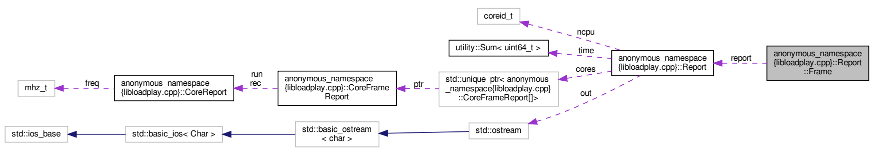
The documentation for this class was generated from the following file:

- src/utility.hpp

11.6 anonymous_namespace{libloadplay.cpp}::Report::Frame Class Reference

Represents a frame of the report.

Collaboration diagram for anonymous_namespace{libloadplay.cpp}::Report::Frame:


coreid_t
ncpu
utility::Sum< uint64_t >
time
anonymous_namespace {libloadplay.cpp}::Report
report
anonymous_namespace {libloadplay.cpp}::Report ::Frame
cores
mhz_t
freq
anonymous_namespace {libloadplay.cpp}::CoreReport
run
rec
anonymous_namespace {libloadplay.cpp}::CoreFrame Report
ptr
std::unique_ptr< anonymous _namespace{libloadplay.cpp} ::CoreFrameReport[]>
out
std::ios_base
std::basic_ios< Char >
std::basic_ostream < char >
std::ostream

Public Member Functions

- Frame (Report &report, uint64_t const duration)
  Construct a report frame.
- CoreFrameReport & operator [ ] (coreid_t const i)
  Subscript operator for per core frame report data.
- CoreFrameReport const & operator [ ] (coreid_t const i) const
  Subscript operator for per core frame report data.
- ∼Frame ()
  Finalises the frame by outputting it.

Private Attributes

- Report & report

  The report this frame belongs to.

11.6.1 Detailed Description

Represents a frame of the report.

It provides access to each CoreFrameReport via the subscript operator [].

The frame data is output when the frame goes out of scope.

11.6.2 Constructor & Destructor Documentation

11.6.2.1 Frame()

```
anonymous_namespace{libloadplay.cpp}::Report::Frame::Frame (
            Report & report,
            uint64_t const duration )   [inline]
```

Construct a report frame.

Parameters

report	The report this frame belongs to
duration	The frame duration

11.6.3 Member Function Documentation

11.6.3.1 operator []() [1/2]

```
CoreFrameReport& anonymous_namespace{libloadplay.cpp}::Report::Frame::operator [] (
            coreid_t const i )   [inline]
```

Subscript operator for per core frame report data.

Parameters

i	The core index

Returns

A reference to the core frame data

11.6.3.2 operator []() [2/2]

```
CoreFrameReport const& anonymous_namespace{libloadplay.cpp}::Report::Frame::operator [] (
             coreid_t const i ) const  [inline]
```

Subscript operator for per core frame report data.

Parameters

i	The core index

Returns

A const reference to the core frame data

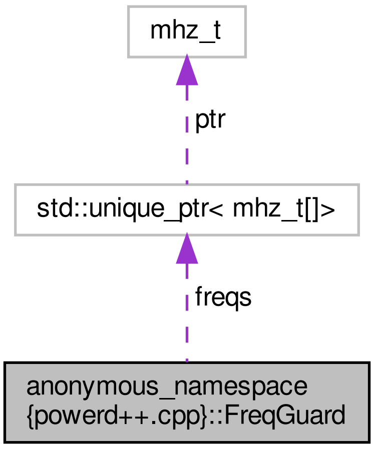
The documentation for this class was generated from the following file:

- src/libloadplay.cpp

11.7 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.

11.7.1 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:

- src/powerd++.cpp

11.8 anonymous_namespace{libloadplay.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 varibale to the given value.
- ~Hold ()
  Restores the original value.

Private Attributes

- T const restore
  The original value.
- T & ref
  Reference to the variable.

11.8.1 Detailed Description

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

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

Template Parameters

T	The type of the value to hold

11.8.2 Constructor & Destructor Documentation

11.8.2.1 Hold()

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

The constructor sets the referenced varibale to the given value.

Parameters

ref	The variable to hold and restore
value	The value to set the variable to

11.8.3 Member Data Documentation

11.8.3.1 ref

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

Reference to the variable.

11.8.3.2 restore

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

The original value.

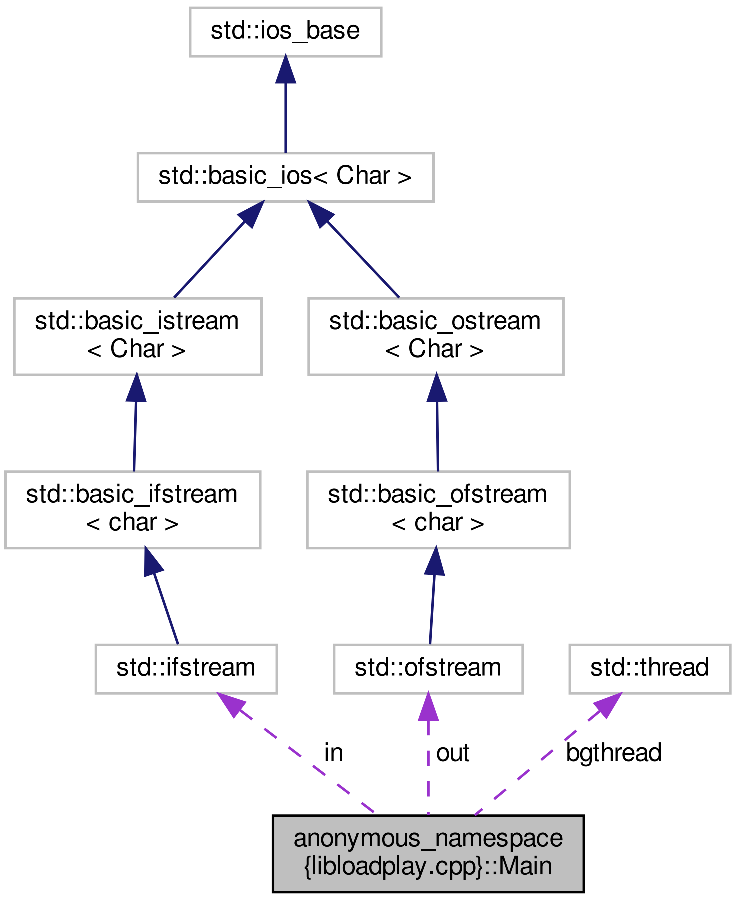
The documentation for this class was generated from the following file:

- src/libloadplay.cpp

11.9 anonymous_namespace{libloadplay.cpp}::Main Class Reference

Singleton class representing the main execution environment.

Collaboration diagram for anonymous_namespace{libloadplay.cpp}::Main:


std::ios_base
std::basic_ios< Char >
std::basic_istream < Char >
std::basic_ostream < Char >
std::basic_ifstream < char >
std::basic_ofstream < char >
std::ifstream
std::ofstream
std::thread
in
out
bgthread
anonymous_namespace {libloadplay.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.
- std::ifstream in {sys::env::vars["LOADPLAY_IN"]}

The optional input file.

- std::ofstream out {sys::env::vars["LOADPLAY_OUT"]}

  The optional output file.

- bool die {false}

  Used to request premature death from the emulation thread.

11.9.1 Detailed Description

Singleton class representing the main execution environment.

11.9.2 Constructor & Destructor Documentation

11.9.2.1 Main()

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

The constructor starts up the emulation.

- Read the headers from input 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:

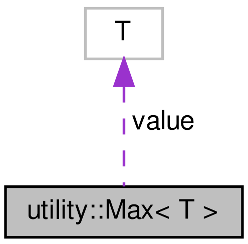
- src/libloadplay.cpp

11.10 utility::Max< T > Class Template Reference

A simple value container that provides the maximum of assigned values.

`#include <utility.hpp>`

Collaboration diagram for utility::Max< T >:



Public Member Functions

- constexpr Max (T const &value)

  Construct from an initial value.
- constexpr operator T const & () const

  Returns the current maximum.
- constexpr Max & operator= (T const &value)

  Assign a new value, if it is greater than the current value.

Private Attributes

- T value

  The maximum of the assigned values.

11.10.1 Detailed Description

template<typename T>
class utility::Max< T >

A simple value container that provides the maximum of assigned values.

Template Parameters

T	The value type

11.10.2 Constructor & Destructor Documentation

11.10.2.1 Max()

```
template<typename T>
constexpr utility::Max< T >::Max (
              T const & value )  [inline], [explicit]
```

Construct from an initial value.

Parameters

value	The initial value

11.10.3 Member Function Documentation

11.10.3.1 operator T const &()

```
template<typename T>
constexpr utility::Max< T >::operator T const & ( ) const  [inline]
```

Returns the current maximum.

Returns

The maximum by const reference

11.10.3.2 operator=()

```
template<typename T>
constexpr Max& utility::Max< T >::operator= (
            T const & value )  [inline]
```

Assign a new value, if it is greater than the current value.

Parameters

value	The value to assign

Returns

A self reference

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

- src/utility.hpp

11.11 anonymous_namespace{libloadplay.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.

11.11.1 Detailed Description

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

11.11.2 Constructor & Destructor Documentation

11.11.2.1 mib_t() [1/2]

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

Construct a mib with the given number of arguments.

Template Parameters

Ints	A list of integer types

Parameters

ints	A list of integers to create a mib from

11.11.2.2 mib_t() [2/2]

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

Initialise from a pointer to an int array.

Parameters

mibs,len	The array and its length

11.11.3 Member Function Documentation

11.11.3.1 operator int *()

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

Cast to `int *` for value access.

Returns

A pointer to mibs

11.11.3.2 operator int const *()

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

Cast to `int const *` for value access.

Returns

A pointer to mibs

11.11.3.3 operator<()

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

Less than operator required by std::map.

Parameters

op	Another mib_t instance

Returns

Whether this mib is less than the given one

11.11.3.4 operator==()

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

Equality operator required by std::map.

Parameters

op	Another mib_t instance

Returns

Whether all values in this and the given mib are equal

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

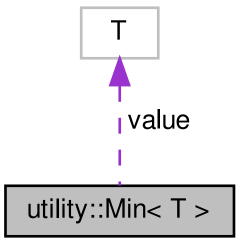
- src/libloadplay.cpp

11.12 utility::Min< T > Class Template Reference

A simple value container that provides the minimum of assigned values.

`#include <utility.hpp>`

Collaboration diagram for utility::Min< T >:



Public Member Functions

- constexpr Min (T const &value)

    Construct from an initial value.
- constexpr operator T const & () const

    Returns the current minimum.
- constexpr Min & operator= (T const &value)

    Assign a new value, if it is less than the current value.

Private Attributes

- T value

    The minimum of the assigned values.

11.12.1 Detailed Description

template<typename T>
class utility::Min< T >

A simple value container that provides the minimum of assigned values.

Template Parameters

T	The value type

11.12.2 Constructor & Destructor Documentation

11.12.2.1 Min()

```
template<typename T>
constexpr utility::Min< T >::Min (
              T const & value )  [inline], [explicit]
```

Construct from an initial value.

Parameters

value	The initial value

11.12.3 Member Function Documentation

11.12.3.1 operator T const &()

```
template<typename T>
constexpr utility::Min< T >::operator T const & ( ) const  [inline]
```

Returns the current minimum.

Returns

The minimum by const reference

11.12.3.2 operator=()

```
template<typename T>
constexpr Min& utility::Min< T >::operator= (
              T const & value )  [inline]
```

Assign a new value, if it is less than the current value.

Parameters

value	The value to assign

Returns

A self reference

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

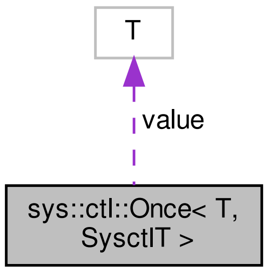
- src/utility.hpp

11.13 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.

11.13.1 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

T	The type to represent the sysctl as
SysctlT	The Sysctl type

11.13.2 Constructor & Destructor Documentation

11.13.2.1 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

value	The fallback value
sysctl	The sysctl to represent

11.13.3 Member Function Documentation

11.13.3.1 operator T const &()

```
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:

- src/sys/sysctl.hpp

11.14 nih::Options< OptionT, DefCount > 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 ∗const argv, char const ∗const usage, Parameter< OptionT > const (&defs)[DefCount])

  Construct an options functor.
- Options & operator() ()

  Updates the internal state by parsing the next option.
- operator OptionT () const

  Implicitly cast to the current option.
- 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.
- std::string show (int const i, int const n=1) const

  Provide a string containing the entire command line, with the indexed argument highlighted.
- int offset () const

  Returns the argument offset of the current parameter/argument.

Private Member Functions

- Parameter< OptionT > const & get (char const ch)

  Finds the short option matching the given character.
- Parameter< OptionT > const & 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 str, char const ∗const prefix)

  Returns true if the given string starts with the given prefix.

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.
- Parameter< OptionT > const (& defs)[DefCount]

  A reference to the option definitions.
- Parameter< OptionT > const opt_unknown

  The option definition to use for unknown options.
- Parameter< OptionT > const opt_noopt

  The option definition to use for non-options.
- Parameter< OptionT > const opt_dash

  The option definition to use for a single dash.
- Parameter< OptionT > const opt_ldash

  The option definition to use for a single double-dash.
- int argi

  The index of the command line argument containing the current option.
- char const ∗ argp

  Points to the current short option character.
- Parameter< OptionT > const ∗ current

  Points to the current option definition.

11.14.1 Detailed Description

template<class OptionT, size_t DefCount>
class nih::Options< OptionT, 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

OptionT	An enum or enum class matching the requirements set by enum_has_members
DefCount	The number of option definitions

11.14.2 Constructor & Destructor Documentation

11.14.2.1 Options()

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

Construct an options functor.

Parameters

argc,argv	The command line arguments
usage	A usage string following "usage: progname "
defs	An array of parameter definitions

11.14.3 Member Function Documentation

11.14.3.1 bmatch()

```
template<class OptionT , size_t DefCount>
static bool nih::Options< OptionT, DefCount >::bmatch (
              char const *const str,
              char const *const prefix )  [inline], [static], [private]
```

Returns true if the given string starts with the given prefix.

Parameters

str,prefix	Two 0 terminated strings

Return values

true	The string starts with the prefix
false	The string does not start with the prefix

11.14.3.2 get() [1/2]

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

Finds the short option matching the given character.

Parameters

ch	The short option to find

Returns

An option definition by reference

11.14.3.3 get() [2/2]

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

Finds the long option matching the given string.

Parameters

str	The long option to find

Returns

An option definition by reference

11.14.3.4 match()

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

Returns true if the given strings match.

Parameters

lstr,rstr	Two 0 terminated strings

Return values

true	The given strings match
false	The strings do not match

11.14.3.5 offset()

```
template<class OptionT , size_t DefCount>
int nih::Options< OptionT, DefCount >::offset ( ) const  [inline]
```

Returns the argument offset of the current parameter/argument.

Warning

This may return a value >= argc if the current state is OptionT::OPT_DONE

Returns

The current argument index

11.14.3.6 operator []()

```
template<class OptionT , size_t DefCount>
char const* nih::Options< OptionT, 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	The index of the argument to retrieve

Returns

The option or one of its arguments

11.14.3.7 operator OptionT()

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

Implicitly cast to the current option.

Returns

An OptionT member representing the current option

Return values

OPT_UNKNOWN	An option that was not in the list of option definitions was encountered
OPT_NOOPT	An argument that is not an option was encountered
OPT_DASH	A lone dash ”-” was encountered
OPT_LDASH	A lone long dash ”--” was encountered
OPT_DONE	All arguments have been processed, or argument processing has not yet started

11.14.3.8 operator()()

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

Updates the internal state by parsing the next option.

When reaching the end of the argument list, the internal state is reset, so a successive call will restart the argument parsing.

Returns

A self-reference

11.14.3.9 removePath()

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

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

Parameters

file	The string containing the path to the file

Returns

A pointer to the file name portion of the path

11.14.3.10 show()

```
template<class OptionT , size_t DefCount>
std::string nih::Options< OptionT, DefCount >::show (
                int const i,
                int const n = 1 ) const  [inline]
```

Provide a string containing the entire command line, with the indexed argument highlighted.

The current implementation highlights arguments by underlining them with ^~~~.

Parameters

i	The argument index, like operator []
n	The number of arguments to highlight, highlights all remaining arguments if n <= 0

Returns

A string formatted to highlight the given argument

11.14.3.11 usage()

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

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

Returns

A usage string for printing on the CLI

11.14.4 Member Data Documentation

11.14.4.1 opt_dash

```
template<class OptionT , size_t DefCount>
Parameter<OptionT> const nih::Options< OptionT, DefCount >::opt_dash  [private]
```

Initial value:

```
{
        OptionT::OPT_DASH, 0, nullptr, nullptr, nullptr
    }
```

The option definition to use for a single dash.

11.14.4.2 opt_ldash

```
template<class OptionT , size_t DefCount>
Parameter<OptionT> const nih::Options< OptionT, DefCount >::opt_ldash  [private]
```

Initial value:

```
{
        OptionT::OPT_LDASH, 0, nullptr, nullptr, nullptr
    }
```

The option definition to use for a single double-dash.

11.14.4.3 opt_noopt

```
template<class OptionT , size_t DefCount>
Parameter<OptionT> const nih::Options< OptionT, DefCount >::opt_noopt  [private]
```

Initial value:

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

The option definition to use for non-options.

11.14.4.4 opt_unknown

```
template<class OptionT , size_t DefCount>
Parameter<OptionT> const nih::Options< OptionT, DefCount >::opt_unknown  [private]
```

Initial value:

```
{
        OptionT::OPT_UNKNOWN, 0, nullptr, nullptr, nullptr
    }
```

The option definition to use for unknown options.

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

- src/Options.hpp

11.15 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.

11.15.1 Detailed Description

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

11.15.2 Constructor & Destructor Documentation

11.15.2.1 Pidfile()

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

Attempts to open the pidfile.

Parameters

pfname,mode	Arguments to pidfile_open()

Exceptions

pid_t	Throws the PID of the other process already holding the requested pidfile
sys::sc_error<error>	Throws with the errno of pidfile_open()

11.15.3 Member Function Documentation

11.15.3.1 write()

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

Write PID to the file, should be called after daemon().

Exceptions

sys::sc_error<error>	Throws with the errno of pidfile_write()

11.15.4 Member Data Documentation

11.15.4.1 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().

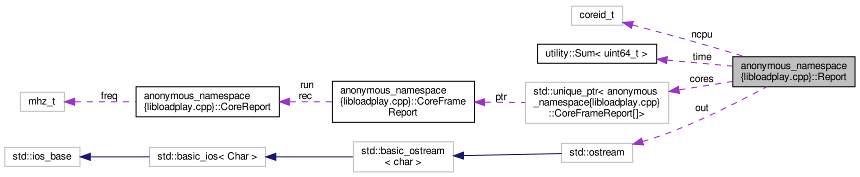
The documentation for this class was generated from the following file:

- src/sys/pidfile.hpp

11.16 anonymous_namespace{libloadplay.cpp}::Report Class Reference

Provides a mechanism to provide frame wise per core load information.

Collaboration diagram for anonymous_namespace{libloadplay.cpp}::Report:


coreid_t
ncpu
utility::Sum< uint64_t >
time
anonymous_namespace {libloadplay.cpp}::Report
cores
mhz_t
freq
anonymous_namespace {libloadplay.cpp}::CoreReport
run rec
anonymous_namespace {libloadplay.cpp}::CoreFrame Report
ptr
std::unique_ptr< anonymous _namespace{libloadplay.cpp} ::CoreFrameReport[]>
out
std::ios_base
std::basic_ios< Char >
std::basic_ostream < char >
std::ostream

Classes

- class Frame

    Represents a frame of the report.

Public Member Functions

- Report (std::ostream &out, coreid_t const ncpu)

    Construct a report.
- template<typename ... ArgTs>
  Frame frame (ArgTs &&... args)

    Constructs a frame for this report.

Private Attributes

- std::ostream & out

    The output stream to report to.
- coreid_t const ncpu

    The number of cpu cores to provide reports for.
- Sum< uint64_t > time

    The time passed in [ms].
- std::unique_ptr< CoreFrameReport[ ]> cores

    Per frame per core data.

11.16.1 Detailed Description

Provides a mechanism to provide frame wise per core load information.

11.16.2 Constructor & Destructor Documentation

11.16.2.1 Report()

```
anonymous_namespace{libloadplay.cpp}::Report::Report (
            std::ostream & out,
            coreid_t const ncpu )  [inline]
```

Construct a report.

Parameters

out	The stream to output to
ncpu	The number of CPU cores to report

11.16.3 Member Function Documentation

11.16.3.1 frame()

```
template<typename ... ArgTs>
Frame anonymous_namespace{libloadplay.cpp}::Report::frame (
            ArgTs &&... args )  [inline]
```

Constructs a frame for this report.

Template Parameters

Arg↩ Ts	The constructor argument types

Parameters

args	The constructor arguments

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

- src/libloadplay.cpp

11.17 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.

11.17.1 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

Domain	A type marking the domain the error comes from, e.g. sys::ctl::error

11.17.2 Member Function Documentation

11.17.2.1 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

11.17.2.2 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:

- src/sys/error.hpp

11.18 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.

11.18.1 Detailed Description

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

11.18.2 Constructor & Destructor Documentation

11.18.2.1 Signal()

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

Sets up the given handler.

Parameters

sig	The signal to set a handler for
handler	The signal handling function

Exceptions

sys::sc_error<error>	Throws with the errno of signal()

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

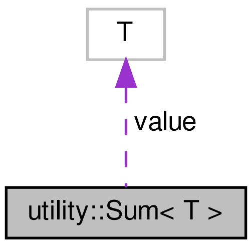
- src/sys/signal.hpp

11.19 utility::Sum< T > Class Template Reference

A simple value container only allowing += and copy assignment.

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

Collaboration diagram for utility::Sum< T >:



Public Member Functions

- constexpr Sum (T const &value)

  Construct from an initial value.
- constexpr Sum ()

  Default construct.
- constexpr operator T const & () const

  Returns the current sum of values.
- constexpr Sum & operator+= (T const &value)

  Add a value to the sum.

Private Attributes

- T value

    The sum of values accumulated.

11.19.1 Detailed Description

template<typename T>
class utility::Sum< T >

A simple value container only allowing += and copy assignment.

Template Parameters

T	The value type

11.19.2 Constructor & Destructor Documentation

11.19.2.1 Sum()

```
template<typename T>
constexpr utility::Sum< T >::Sum (
              T const & value )  [inline], [explicit]
```

Construct from an initial value.

Parameters

value	The initial value

11.19.3 Member Function Documentation

11.19.3.1 operator T const &()

```
template<typename T>
constexpr utility::Sum< T >::operator T const & ( ) const  [inline]
```

Returns the current sum of values.

Returns

The sum of values by const reference

11.19.3.2 operator+=()

```
template<typename T>
constexpr Sum& utility::Sum< T >::operator+= (
              T const & value )  [inline]
```

Add a value to the sum.

Parameters

value	The value to add to the current sum

Returns

A self reference

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

- src/utility.hpp

11.20 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 assiges 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.

11.20.1 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

T	The type to represent the sysctl as
SysctlT	The Sysctl type

11.20.2 Constructor & Destructor Documentation

11.20.2.1 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 repesented name.

11.20.2.2 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

sysctl	The Sysctl instance to represent

11.20.3 Member Function Documentation

11.20.3.1 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

11.20.3.2 operator=()

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

Transparently assiges values of type T to the represented Sysctl instance.

Parameters

value	The value to assign

Returns

A self reference

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

- src/sys/sysctl.hpp

11.21 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.
- size_t size () const

  The size of the sysctl.
- 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 retreived 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.

11.21.1 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

MibDepth	The MIB level, e.g. "hw.ncpu" is two levels deep

11.21.2 Constructor & Destructor Documentation

11.21.2.1 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.

Spliting 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

Tail	The types of the trailing MIB address values (must be mib_t)

Parameters

head,tail	The mib

11.21.3 Member Function Documentation

11.21.3.1 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

buf,bufsize	The target buffer and its size

Exceptions

sys::sc_error<error>	Throws if value retrieval fails or is incomplete, e.g. because the value does not fit into the target buffer

11.21.3.2 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 retreived from the sysctl.

Template Parameters

T	The type store the sysctl value in

Parameters

value	A reference to the target value

Exceptions

sys::sc_error<error>	Throws if value retrieval fails or is incomplete, e.g. because the value does not fit into the target type

11.21.3.3 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 characer strings.

Template Parameters

T	The type stored in the array

Returns

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

Exceptions

sys::sc_error<error>	May throw if the size of the sysctl increases after the length was queried

11.21.3.4 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

buf,bufsize	The source buffer

Exceptions

sys::sc_error<error>	If the source buffer cannot be stored in the sysctl

11.21.3.5 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

T	The value type

Parameters

value	The value to set the sysctl to

11.21.3.6 size()

```
template<size_t MibDepth = 0>
size_t sys::ctl::Sysctl< MibDepth >::size ( ) const  [inline]
```

The size of the sysctl.

Returns

The size in characters

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

- src/sys/sysctl.hpp

11.22 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.
- size_t size () const
  The size of the sysctl.
- 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 retreived 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.

11.22.1 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 unitialised, 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.

11.22.2 Constructor & Destructor Documentation

11.22.2.1 Sysctl() [1/2]

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

The default constructor.

This is available to defer initialisation to a later moment.

11.22.2.2 Sysctl() [2/2]

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

Initialise the MIB address from a character string.

Parameters

name	The symbolic name of the sysctl

Exceptions

sys::sc_error<error>	May throw an exception if the addressed sysct does not exist or if the address is too long to store

11.22.3 Member Function Documentation

11.22.3.1 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

buf,bufsize	The target buffer and its size

Exceptions

sys::sc_error<error>	Throws if value retrieval fails or is incomplete, e.g. because the value does not fit into the target buffer

11.22.3.2 get() [2/3]

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

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

Template Parameters

T	The type store the sysctl value in

Parameters

value	A reference to the target value

Exceptions

sys::sc_error<error>	Throws if value retrieval fails or is incomplete, e.g. because the value does not fit into the target type

11.22.3.3 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 characer strings.

Template Parameters

T	The type stored in the array

Returns

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

Exceptions

sys::sc_error<error>	May throw if the size of the sysctl increases after the length was queried

11.22.3.4 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

buf,bufsize	The source buffer

Exceptions

sys::sc_error<error>	If the source buffer cannot be stored in the sysctl

11.22.3.5 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

T	The value type

Parameters

value	The value to set the sysctl to

11.22.3.6 size()

```
size_t sys::ctl::Sysctl< 0 >::size ( ) const  [inline]
```

The size of the sysctl.

Returns

The size in characters

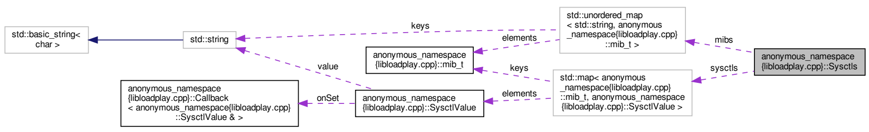
The documentation for this class was generated from the following file:

- src/sys/sysctl.hpp

11.23 anonymous_namespace{libloadplay.cpp}::Sysctls Class Reference

Singleton class representing the sysctl table for this library.

Collaboration diagram for anonymous_namespace{libloadplay.cpp}::Sysctls:


std::basic_string< char >
std::string
keys
std::unordered_map < std::string, anonymous _namespace{libloadplay.cpp} ::mib_t >
elements
anonymous_namespace {libloadplay.cpp}::mib_t
mibs
anonymous_namespace {libloadplay.cpp}::Sysctls
value
keys
sysctls
std::map< anonymous _namespace{libloadplay.cpp} ::mib_t, anonymous_namespace {libloadplay.cpp}::SysctlValue >
anonymous_namespace {libloadplay.cpp}::Callback < anonymous_namespace{libloadplay.cpp} ::SysctlValue & >
onSet
anonymous_namespace {libloadplay.cpp}::SysctlValue
elements

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 (char const ∗const name) const

  Returns a mib for a given symbolic name.
- SysctlValue & operator [ ] (char const ∗const name)

  Returns a reference to a sysctl value container.
- 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 $\longrightarrow$ mib.
- std::map< mib_t, SysctlValue > sysctls

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

11.23.1 Detailed Description

Singleton class representing the sysctl table for this library.

11.23.2 Member Function Documentation

11.23.2.1 addValue() [1/2]

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

Add a value to the sysctls map.

Parameters

mib	The mib to add the value for
value	The value to store

11.23.2.2 addValue() [2/2]

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

Add a value to the sysctls map.

Parameters

name	The symbolic name of the mib to add the value for
value	The value to store

11.23.2.3 getMib()

```
mib_t const& anonymous_namespace{libloadplay.cpp}::Sysctls::getMib (
              char const *const name ) const  [inline]
```

Returns a mib for a given symbolic name.

Parameters

name	The MIB name

Returns

The MIB

11.23.2.4 operator []() [1/2]

```
SysctlValue& anonymous_namespace{libloadplay.cpp}::Sysctls::operator [] (
              char const *const name )  [inline]
```

Returns a reference to a sysctl value container.

Parameters

name	The MIB name to return the reference for

Returns

A SysctlValue reference

11.23.2.5 operator []() [2/2]

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

Returns a reference to a sysctl value container.

Parameters

mib	The MIB to return the reference for

Returns

A SysctlValue reference

11.23.3 Member Data Documentation

11.23.3.1 mibs

```
std::unordered_map<std::string, mib_t> anonymous_namespace{libloadplay.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}},
        {LOADREC_FEATURES, {1004}}
    }
```

Maps name → mib.

11.23.3.2 sysctls

```
std::map<mib_t, SysctlValue> anonymous_namespace{libloadplay.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,   ""}},
        {{1004},               {CTLTYPE_U64,    "0"}}
    }
```

Maps mib → (type, value).

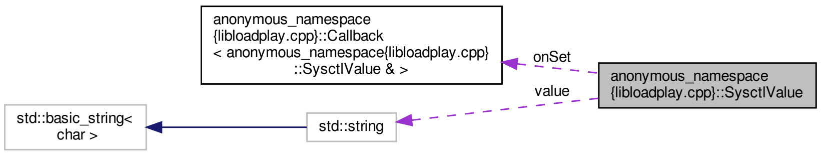
The documentation for this class was generated from the following file:

- src/libloadplay.cpp

11.24 anonymous_namespace{libloadplay.cpp}::SysctlValue Class Reference

Instances of this class represents a specific sysctl value.

Collaboration diagram for anonymous_namespace{libloadplay.cpp}::SysctlValue:


anonymous_namespace {libloadplay.cpp}::Callback < anonymous_namespace{libloadplay.cpp} ::SysctlValue & >
onSet
anonymous_namespace {libloadplay.cpp}::SysctlValue
value
std::basic_string< char >
std::string

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 Member Functions

- template<typename T >
  size_t size () const

  Provide the size of this value represented as a string of Ts.

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.

11.24.1 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.

11.24.2 Constructor & Destructor Documentation

11.24.2.1 SysctlValue() [1/3]

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

Copy constructor.

Parameters

copy	The instance to copy

11.24.2.2 SysctlValue() [2/3]

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

Move constructor.

Parameters

move	The instance to move

11.24.2.3 SysctlValue() [3/3]

```
anonymous_namespace{libloadplay.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

type	The CTLTYPE
value	A string representation of the value
callback	A callback function that is called for each set() call

11.24.3 Member Function Documentation

11.24.3.1 get() [1/4]

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

Copy a list of values into the given buffer.

Template Parameters

T	The type of the values to extract

Parameters

dst,size	The destination buffer and size

Return values

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

11.24.3.2 get() [2/4]

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

Copy a C string into the given buffer.

Parameters

dst,size	The destination buffer and size

Return values

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

11.24.3.3 get() [3/4]

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

Returns a single value.

Template Parameters

T	The type of the value

Returns

The value

11.24.3.4 get() [4/4]

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

Copy a list of values into the given buffer.

Parameters

dst,size	The destination buffer and size

Return values

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

11.24.3.5 operator=() [1/2]

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

Copy assignment operator.

Parameters

copy	The instance to copy

Returns

A self reference

11.24.3.6 operator=() [2/2]

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

Move assignment operator.

Parameters

move	The instance to move

Returns

A self reference

11.24.3.7 registerOnSet() [1/2]

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

Register a callback function.

Parameters

callback	The function to move to the callback handler

11.24.3.8 registerOnSet() [2/2]

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

Register a callback function.

Parameters

callback	The function to copy to the callback handler

11.24.3.9 set() [1/5]

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

Set this value to the values in the given buffer.

Template Parameters

T	The type of the values

Parameters

newp,newlen	The source buffer and size

11.24.3.10 set() [2/5]

```
int anonymous_namespace{libloadplay.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

newp,newlen	The source buffer and size

11.24.3.11 set() [3/5]

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

Move a string to the value.

Parameters

value	The new value

11.24.3.12 set() [4/5]

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

Copy a string to the value.

Parameters

value	The new value

11.24.3.13 set() [5/5]

```
template<typename T >
void anonymous_namespace{libloadplay.cpp}::SysctlValue::set (
              T const & value )  [inline]
```

Set the value.

Template Parameters

T	The value type

Parameters

value	The value to set

11.24.3.14 size() [1/2]

```
template<typename T >
size_t anonymous_namespace{libloadplay.cpp}::SysctlValue::size ( ) const  [inline], [private]
```

Provide the size of this value represented as a string of Ts.

Template Parameters

T	The type this value is supposed to be a array of

Returns

The size of the whole string of Ts

11.24.3.15 size() [2/2]

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

Returns the required storage size according to the CTLTYPE.

Returns

The required buffer size to hold the values.

Exceptions

int	Throws -1 if the current CTLTYPE is not implemented.

11.24.4 Member Data Documentation

11.24.4.1 mtx

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

A stackable mutex.

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

11.24.4.2 value

```
std::string anonymous_namespace{libloadplay.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:

- src/libloadplay.cpp

11.25 sys::env::Var Class Reference

A reference type refering to an environment variable.

`#include <env.hpp>`

Public Member Functions

- template<size_t Size>
  Var (char const (&name)[Size])

  Construct an environment variable reference.
- Var (Var const &)=delete

  Do not permit copy construction.
- Var & operator= (Var const &)=delete

  Do not permit copy assignment.
- operator char const ∗ () const

  Retrieve the value of the environment variable.
- Var & operator= (char const ∗const assign)

  Assign a new value to the environment variable.
- Var & erase ()

  Explicitly deletes the environment variable.
- char const ∗ c_str () const

  Explicitly retrieve the value as a character array.
- std::string str () const

  Explicitly retrieve the value as a std::string.

Private Attributes

- char const ∗const name

  A pointer to the variable name.

11.25.1 Detailed Description

A reference type refering to an environment variable.

To avoid issues with the lifetime of the name string this is not copy constructible or assignable.

11.25.2 Constructor & Destructor Documentation

11.25.2.1 Var()

```
template<size_t Size>
sys::env::Var::Var (
              char const (&) name[Size] )  [inline]
```

Construct an environment variable reference.

Template Parameters

Size	The size of the name buffer

Parameters

name	The name of the environment variable

11.25.3 Member Function Documentation

11.25.3.1 c_str()

```
char const* sys::env::Var::c_str ( ) const  [inline]
```

Explicitly retrieve the value as a character array.

Returns

A pointer to the character array with the variable value

Return values

nullptr	The variable does not exist

11.25.3.2 erase()

```
Var& sys::env::Var::erase ( )  [inline]
```

Explicitly deletes the environment variable.

Returns

A self-reference

Exceptions

sc_error<error>{EINVAL}	Invalid variable name
sc_error<error>{ENOMEM}	Failed to allocate memory when updating the environment

11.25.3.3 operator char const *()

```
sys::env::Var::operator char const * ( ) const  [inline]
```

Retrieve the value of the environment variable.

Returns

A pointer to the character array with the variable value

Return values

nullptr	The variable does not exist

11.25.3.4 operator=()

```
Var& sys::env::Var::operator= (
            char const *const assign )  [inline]
```

Assign a new value to the environment variable.

Deletes the variable if nullptr is assigned.

Parameters

assign	The new value

Returns

A self-reference

Exceptions

sc_error<error>{EINVAL}	Invalid variable name
sc_error<error>{ENOMEM}	Failed to allocate memory when updating the environment

11.25.3.5 str()

```
std::string sys::env::Var::str ( ) const  [inline]
```

Explicitly retrieve the value as a std::string.

Returns an empty string if the variable does not exist. Use c_str() to distinguish between an empty string and an inexistant variable.

Returns

A string containing the variable value

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

- src/sys/env.hpp

11.26 sys::env::Vars Struct Reference

A singleton class providing access to environment variables.

```
#include <env.hpp>
```

Public Member Functions

- template<typename T >
  Var const operator [ ] (T const &name) const

  Access environment variable by name.
- template<typename T >
  Var operator [ ] (T const &name)

  Access environment variable by name.

11.26.1 Detailed Description

A singleton class providing access to environment variables.

11.26.2 Member Function Documentation

11.26.2.1 operator []() [1/2]

```
template<typename T >
Var const sys::env::Vars::operator [] (
            T const & name ) const  [inline]
```

Access environment variable by name.

Template Parameters

T	The name argument type

Parameters

name	The name of the variable by reference

11.26.2.2 operator []() [2/2]

```
template<typename T >
Var sys::env::Vars::operator [] (
              T const & name )  [inline]
```

Access environment variable by name.

Template Parameters

T	The name argument type

Parameters

name	The name of the variable by reference

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

- src/sys/env.hpp

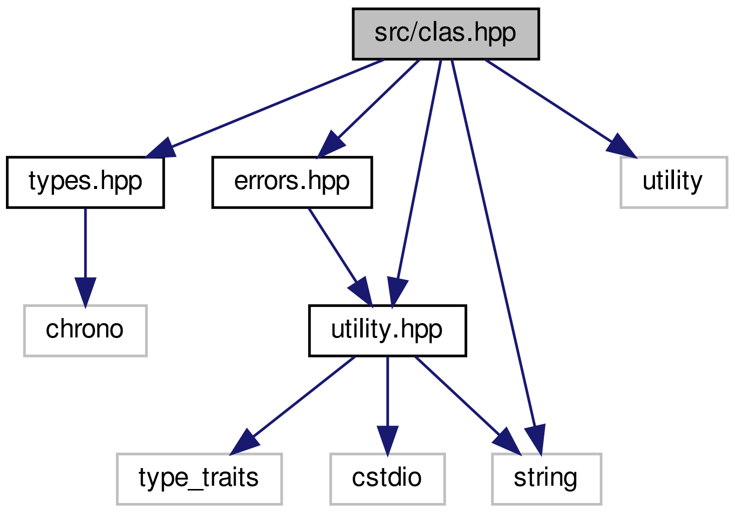
12 File Documentation

12.1 src/clas.hpp File Reference

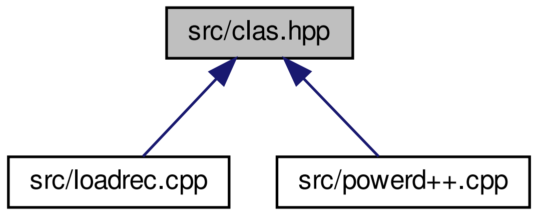
Implements functions to process command line arguments.

```
#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 ℃ for display purposes.

- template< typename T >
  std::pair< T, T > clas::range (T(&func)(char const ∗const), char const ∗const str)

  Takes a string encoded range of values and returns them.

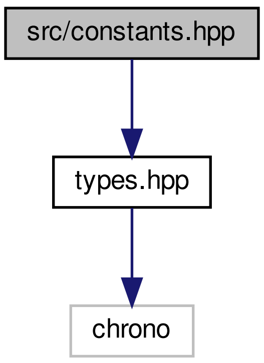
12.1.1 Detailed Description

Implements functions to process command line arguments.

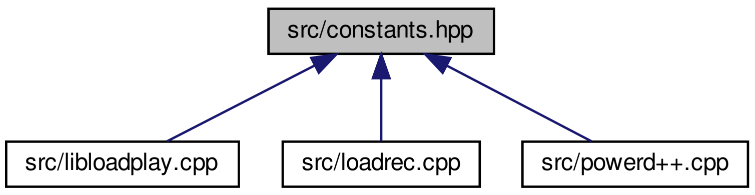
12.2 src/constants.hpp File Reference

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::ACLINE = ”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

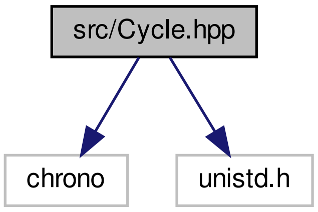
Defines a collection of constants.

12.3 src/Cycle.hpp File Reference

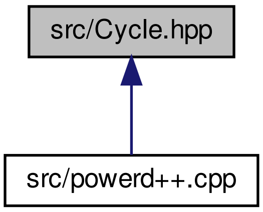
Implements timing::Cycle, a cyclic sleep functor.

```
#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.

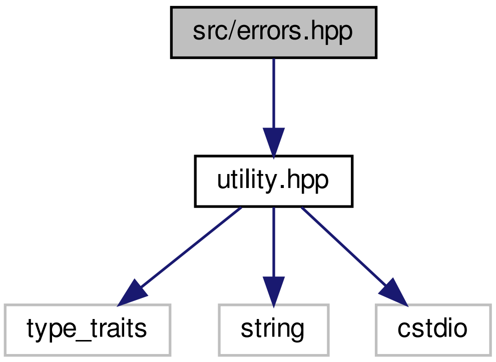
12.3.1 Detailed Description

Implements timing::Cycle, a cyclic sleep functor.

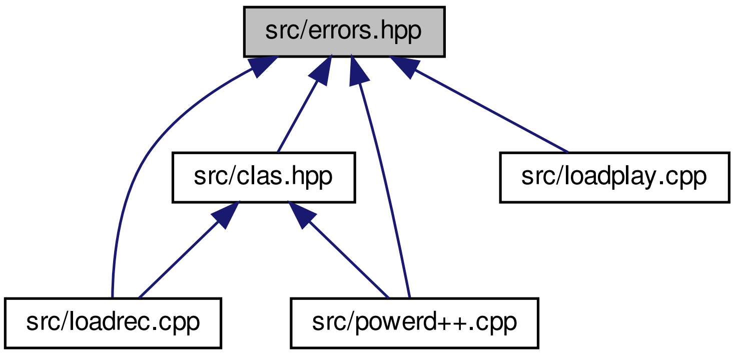
12.4 src/errors.hpp File Reference

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::EEXCEPT, errors::Exit::EFILE,
errors::Exit::EEXEC, 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

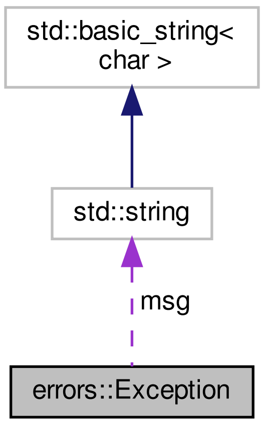
Common error handling code.

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:



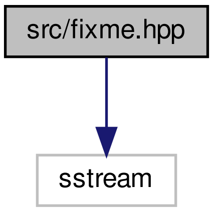
Class Members

int	err	The errno value at the time of creation.
Exit	exitcode	The code to exit with.
string	msg	An error message.

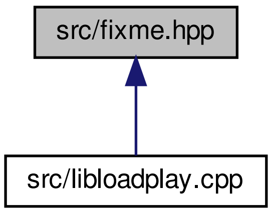
12.5 src/fixme.hpp File Reference

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().

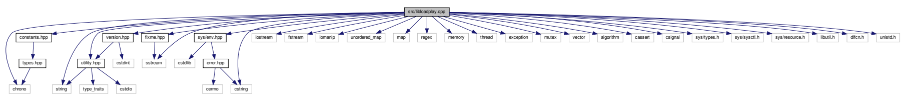
12.5.1 Detailed Description

Implementations in the fixme namespace.

12.6 src/libloadplay.cpp File Reference

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

```
#include "utility.hpp"
#include "constants.hpp"
#include "fixme.hpp"
#include "version.hpp"
#include "sys/env.hpp"
#include <iostream>
#include <fstream>
#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 <algorithm>
#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 libloadplay.cpp:



Classes

- struct anonymous_namespace{libloadplay.cpp}::mib_t

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

  Implements a recursion safe std::function wrapper.
- class anonymous_namespace{libloadplay.cpp}::SysctlValue

  Instances of this class represents a specific sysctl value.
- class anonymous_namespace{libloadplay.cpp}::Sysctls

  Singleton class representing the sysctl table for this library.
- struct anonymous_namespace{libloadplay.cpp}::CoreReport

  The reported state of a single CPU pipeline. More...
- struct anonymous_namespace{libloadplay.cpp}::CoreFrameReport

  The report frame information for a single CPU pipeline. More...
- class anonymous_namespace{libloadplay.cpp}::Report

  Provides a mechanism to provide frame wise per core load information.
- class anonymous_namespace{libloadplay.cpp}::Report::Frame

  Represents a frame of the report.
- class anonymous_namespace{libloadplay.cpp}::Emulator

  Instances of this class represent an emulator session.
- struct anonymous_namespace{libloadplay.cpp}::Emulator::Core

  Per core information. More...
- class anonymous_namespace{libloadplay.cpp}::Main

  Singleton class representing the main execution environment.
- class anonymous_namespace{libloadplay.cpp}::Hold< T >

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

Namespaces

- anonymous_namespace{libloadplay.cpp}

  File local scope.

Functions

- template<size_t Size>
  int anonymous_namespace{libloadplay.cpp}::strcmp (char const ∗const s1, char const (&s2)[Size])

  Safe wrapper around strncmp, which automatically determines the buffer size of s2.
- std::regex anonymous_namespace{libloadplay.cpp}::operator""_r (char const ∗const str, size_t const len)

  User defined literal for regular expressions.
- template<typename ... ArgTs>
  constexpr void anonymous_namespace{libloadplay.cpp}::dprintf (ArgTs &&... args)

  Calls fprintf(stderr, ...) if built with -DEBUG.
- template<>
  std::string anonymous_namespace{libloadplay.cpp}::SysctlValue::get< std::string > () const

  Returns a copy of the value string.
- void anonymous_namespace{libloadplay.cpp}::debug (std::string const &msg)

  Print a debugging message if built with -DEBUG.
- void anonymous_namespace{libloadplay.cpp}::warn (std::string const &msg)

  Print a warning.
- void anonymous_namespace{libloadplay.cpp}::fail (std::string const &msg)

This prints an error message and sets sys_results to make the hijacked process fail.

- std::ostream & anonymous_namespace{libloadplay.cpp}::operator<< (std::ostream &out, CoreReport const &core)

  Print a core clock frequency and load.
- std::ostream & anonymous_namespace{libloadplay.cpp}::operator<< (std::ostream &out, CoreFrame↩ Report const &frame)

  Print recorded and running clock frequency and load for a frame.
- 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().
- pidfh ∗ pidfile_open (const char ∗, mode_t, pid_t ∗)

  Intercept calls to pidfile_open().
- int pidfile_write (pidfh ∗)

  Intercept calls to pidfile_write().
- int pidfile_close (pidfh ∗)

  Intercept calls to pidfile_close().
- int pidfile_remove (pidfh ∗)

  Intercept calls to pidfile_remove().
- int pidfile_fileno (pidfh const ∗)

  Intercept calls to pidfile_fileno().

Variables

- constexpr flag_t const anonymous_namespace{libloadplay.cpp}::FEATURES

  The set of supported features.
- int anonymous_namespace{libloadplay.cpp}::sys_results = 0

  The success return value of intercepted functions.
- class anonymous_namespace{libloadplay.cpp}::Sysctls anonymous_namespace{libloadplay.cpp}::sysctls

  Sole instance of Sysctls.
- class anonymous_namespace{libloadplay.cpp}::Main anonymous_namespace{libloadplay.cpp}::main

  Sole instance of Main.
- bool anonymous_namespace{libloadplay.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 deamon via LD_PRELOAD.

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

The following environment variables affect the operation of loadplay:

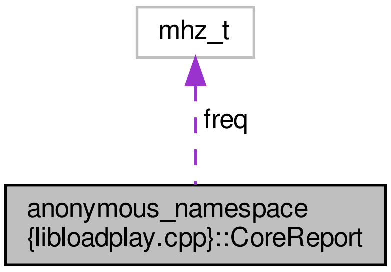
Variable	Description
LOADPLAY_IN	Alternative input file
LOADPLAY_O↵ UT	Alternative output file

12.6.2 Class Documentation

12.6.2.1 struct anonymous_namespace{libloadplay.cpp}::CoreReport

The reported state of a single CPU pipeline.

Collaboration diagram for anonymous_namespace{libloadplay.cpp}::CoreReport:



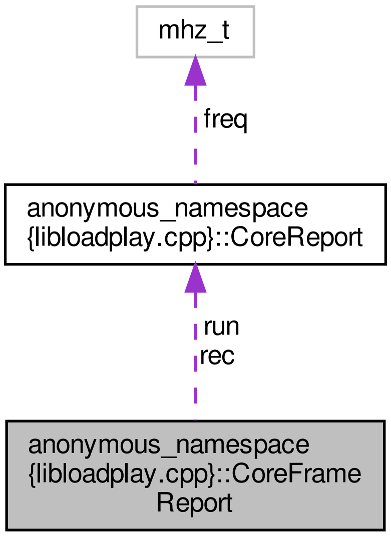
Class Members

mhz_t	freq	The core clock frequency in [MHz].
double	load	The core load as a fraction.

12.6.2.2 struct anonymous_namespace{libloadplay.cpp}::CoreFrameReport

The report frame information for a single CPU pipeline.

Collaboration diagram for anonymous_namespace{libloadplay.cpp}::CoreFrameReport:


mhz_t
freq
anonymous_namespace {libloadplay.cpp}::CoreReport
run
rec
anonymous_namespace {libloadplay.cpp}::CoreFrame Report

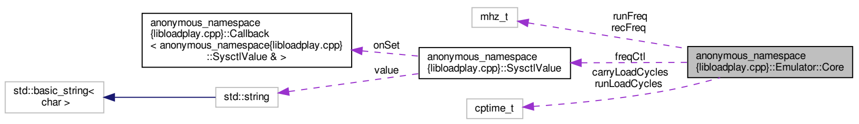
Class Members

CoreReport	rec	The recorded core state.
CoreReport	run	The running core state.

12.6.2.3 struct anonymous_namespace{libloadplay.cpp}::Emulator::Core

Per core information.

Collaboration diagram for anonymous_namespace{libloadplay.cpp}::Emulator::Core:


anonymous_namespace {libloadplay.cpp}::Callback < anonymous_namespace{libloadplay.cpp} ::SysctlValue & >
onSet
value
std::basic_string< char >
std::string
anonymous_namespace {libloadplay.cpp}::SysctlValue
mhz_t
runFreq
recFreq
freqCtl
carryLoadCycles
runLoadCycles
cptime_t
anonymous_namespace {libloadplay.cpp}::Emulator::Core

Class Members

cptime_t	carryLoadCycles	The load cycles carried over to the next frame in [kcycles]. This is determined at the beginning of frame and used to calculated the simulation load at the beginning of the next frame.
SysctlValue ∗	freqCtl	The sysctl handler. The constructor ensures this points to a valid handler.

Class Members

mhz_t	recFreq	The recorded clock frequency. If FREQ_TRACKING is enabled this is updated at during the preliminary stage and used at the beginning of frame stage.
mhz_t	runFreq	The clock frequency the simulation is running at. Updated at the end of frame and used in the next frame.
cptime_t	runLoadCycles	The load cycles simulated for this frame in [kcycles]. This is determined at the beginning of frame and used to calculate the reported load at the end of frame.

12.6.3 Function Documentation

12.6.3.1 daemon()

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

Intercept calls to daemon().

Prevents process from separating from the controlling terminal.

Returns

The value of sys_results

12.6.3.2 geteuid()

```
uid_t geteuid (
              void  )
```

Intercept calls to geteuid().

Tells the asking process that it is running as root.

Returns

Always returns 0

12.6.3.3 pidfile_close()

```
int pidfile_close (
              pidfh * )
```

Intercept calls to pidfile_close().

Returns

The value of sys_results

12.6.3.4 pidfile_fileno()

```
int pidfile_fileno (
              pidfh const * )
```

Intercept calls to pidfile_fileno().

Returns

The value of sys_results

12.6.3.5 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

12.6.3.6 pidfile_remove()

```
int pidfile_remove (
              pidfh * )
```

Intercept calls to pidfile_remove().

Returns

The value of sys_results

12.6.3.7 pidfile_write()

```
int pidfile_write (
              pidfh * )
```

Intercept calls to pidfile_write().

Returns

The value of sys_results

12.6.3.8 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{libloadplay::cpp}::sysctls store.

Falls back to the original under the following conditions:

- sysctl_fallback is set
- kern.usrstack is requested
- vm.∗ is requested

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

name,namelen,oldp,oldlenp,newp,newlen	Please refer to sysctl(3)

Return values

0	The call succeeded

Return values

-1	The call failed

12.6.3.9 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:

- kern.smp.cpus

May fail for the same reasons as sysctl().

Parameters

name,oldp,oldlenp,newp,newlen	Please refer to sysctl(3)

Return values

0	The call succeeded
-1	The call failed

12.6.3.10 sysctlnametomib()

```
int sysctlnametomib (
              const char * name,
              int * mibp,
              size_t * sizep )
```

Intercept calls to sysctlnametomib().

Parameters

name,mibp,sizep	Please refer to sysctl(3)

Return values

0	The call succeeded

Return values

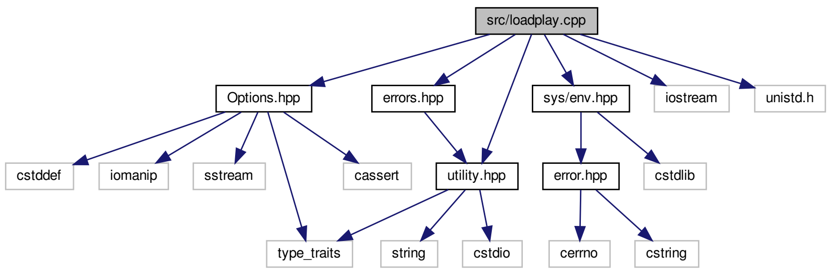
-1	The call failed

12.7 src/loadplay.cpp File Reference

Implements the loadplay a bootstrapping tool for libloadplay.

```
#include "Options.hpp"
#include "errors.hpp"
#include "utility.hpp"
#include "sys/env.hpp"
#include <iostream>
#include <unistd.h>
```

Include dependency graph for loadplay.cpp:



Namespaces

- anonymous_namespace{loadplay.cpp}

  File local scope.

Enumerations

- enum anonymous_namespace{loadplay.cpp}::OE {
  anonymous_namespace{loadplay.cpp}::OE::USAGE, anonymous_namespace{loadplay.cpp}::OE::FILE_IN,
  anonymous_namespace{loadplay.cpp}::OE::FILE_OUT, anonymous_namespace{loadplay.cpp}::OE::CMD,
  anonymous_namespace{loadplay.cpp}::OE::OPT_NOOPT = CMD, anonymous_namespace{loadplay.cpp}::OE::OPT_UNKNO
  anonymous_namespace{loadplay.cpp}::OE::OPT_DASH, anonymous_namespace{loadplay.cpp}::OE::OPT_LDASH,
  anonymous_namespace{loadplay.cpp}::OE::OPT_DONE }

  An enum for command line parsing.

Functions

- char const ∗ anonymous_namespace{loadplay.cpp}::filename (char const ∗const path)

  Performs very rudimentary file name argument checks.
- void anonymous_namespace{loadplay.cpp}::execute (char const ∗const file, char ∗const argv[ ])

  Executes the given command, substituting this process.
- int main (int argc, char ∗argv[ ])

  Parse command line arguments and execute the given command.

Variables

- char const ∗const anonymous_namespace{loadplay.cpp}::USAGE = ”[-h] [-i file] [-o file] command [...]”

  The short usage string.
- Parameter< OE > const anonymous_namespace{loadplay.cpp}::PARAMETERS [ ]

  Definitions of command line parameters.

12.7.1 Detailed Description

Implements the loadplay a bootstrapping tool for libloadplay.

12.7.2 Function Documentation

12.7.2.1 main()

```
int main (
            int argc,
            char ∗ argv[] )
```

Parse command line arguments and execute the given command.

Parameters

argc,argv	The command line arguments

Returns

An exit code

See also

Exit

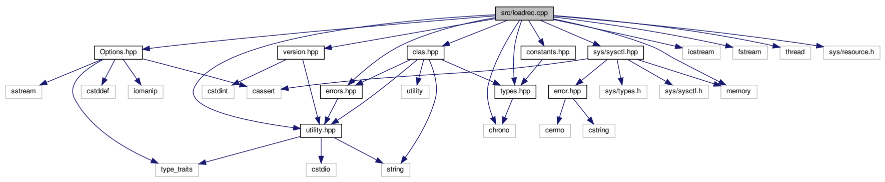
12.8 src/loadrec.cpp File Reference

Implements a load recorder, useful for simulating loads to test CPU clock daemons and settings.

```
#include "Options.hpp"
#include "types.hpp"
#include "constants.hpp"
#include "errors.hpp"
#include "utility.hpp"
#include "clas.hpp"
#include "version.hpp"
#include "sys/sysctl.hpp"
#include <iostream>
#include <fstream>
```

```
#include <chrono>
#include <thread>
#include <memory>
#include <sys/resource.h>
```

Include dependency graph for loadrec.cpp:



Namespaces

- anonymous_namespace{loadrec.cpp}

  File local scope.

Enumerations

- enum anonymous_namespace{loadrec.cpp}::OE {
  anonymous_namespace{loadrec.cpp}::OE::USAGE, anonymous_namespace{loadrec.cpp}::OE::IVAL_DURATION,
  anonymous_namespace{loadrec.cpp}::OE::IVAL_POLL, anonymous_namespace{loadrec.cpp}::OE::FILE_OUTPUT,
  anonymous_namespace{loadrec.cpp}::OE::FILE_PID, anonymous_namespace{loadrec.cpp}::OE::FLAG_VERBOSE,
  anonymous_namespace{loadrec.cpp}::OE::OPT_UNKNOWN, anonymous_namespace{loadrec.cpp}::OE::OPT_NOOPT,
  anonymous_namespace{loadrec.cpp}::OE::OPT_DASH, anonymous_namespace{loadrec.cpp}::OE::OPT_LDASH,
  anonymous_namespace{loadrec.cpp}::OE::OPT_DONE }

  An enum for command line parsing.

Functions

- void anonymous_namespace{loadrec.cpp}::verbose (std::string const &msg)

  Outputs the given message on stderr if g.verbose is set.
- void anonymous_namespace{loadrec.cpp}::init ()

  Set up output to the given file.
- void anonymous_namespace{loadrec.cpp}::read_args (int const argc, char const ∗const argv[ ])

  Parse command line arguments.
- void anonymous_namespace{loadrec.cpp}::print_sysctls ()

  Print the sysctls.
- void anonymous_namespace{loadrec.cpp}::run ()

  Report the load frames.
- int main (int argc, char ∗argv[ ])

  Main routine, setup and execute daemon, print errors.

Variables

- constexpr flag_t const anonymous_namespace{loadrec.cpp}::FEATURES

  The set of supported features.
-

  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.
  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.
- Parameter< OE > const anonymous_namespace{loadrec.cpp}::PARAMETERS [ ]

  Definitions of command line parameters.

12.8.1 Detailed Description

Implements a load recorder, useful for simulating loads to test CPU clock daemons and settings.

12.8.2 Function Documentation

12.8.2.1 main()

```
int main (
              int argc,
              char ∗ argv[] )
```

Main routine, setup and execute daemon, print errors.

Parameters

argc,argv	The command line arguments

Returns

An exit code

See also

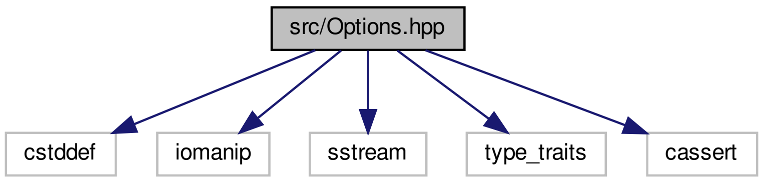
Exit

12.9 src/Options.hpp File Reference

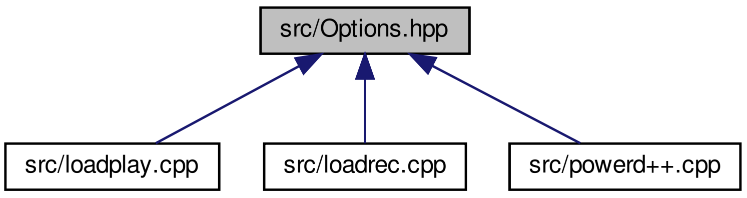
This file provides nih::Options<>, a substitute for `getopt(3)`.

```
#include <cstddef>
#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< OptionT, class >

  Tests whether the given enum provides all the required definitions.
- struct nih::Parameter< OptionT >

  Container for an option definition. More...
- class nih::Options< OptionT, 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 OptionT >
  size_t nih::argCount (Parameter< OptionT > const &def)

  Retrieves the count of arguments in an option definition.
- template<class OptionT , size_t DefCount>
  constexpr Options< OptionT, DefCount > nih::make_Options (int const argc, char const *const *const argv, char const *const usage, Parameter< OptionT > const (&defs)[DefCount])

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

12.9.1 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 when matching the next argument to a parameter. In order to do that a usage string and a list of parameter definitions are required:

```
static char const * const USAGE = "[-hv] [-i file] [-o file] [command ...]";
static nih::Parameter<MyOptions> const PARAMETERS[]{
    {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"}
};
```

Each entry in the array defines a parameter consisting of the following:

Field	Meaning
`option`	The option symbol (enum value)
`sparam`	An optional parameter character (short parameter)
`lparam`	An optional long parameter string
`args`	A comma separated list of parameter arguments
`usage`	A descriptive string

Multiple parameters may be mapped to a single option (e.g. `--help` and `--usage`). Parameters without arguments are called flags. It is possible to map parameters with different numbers of arguments to a single option, but this is arguably semantically confusing and should not be done.

Multiple flags' parameter characters can be concatenated in an argument. A parameter with arguments' character can appear at the end of a character chain. The first argument to the parameter may be concatenated as well. E.g. `-v -i file`, `-vi file` and `-vifile` are all equivalent. Parameters' string representations always stand alone, they can neither be combined with each other nor with parameter characters. E.g. `--verbose --in file` is the equivalent parameter string representation.

The usage string and the parameter usage strings are used to assemble the string provided by the nih::Options<>::usage() method.

The parameter 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, PARAMETERS);
    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 parameter 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 `[]` opterator 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 return `file` regardless of argument chaining.

12.9.2 Class Documentation

12.9.2.1 struct nih::Parameter

template<class OptionT>
struct nih::Parameter< OptionT >

Container for an option definition.

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

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

Template Parameters

OptionT	An enum or enum class representing the available options

Class Members

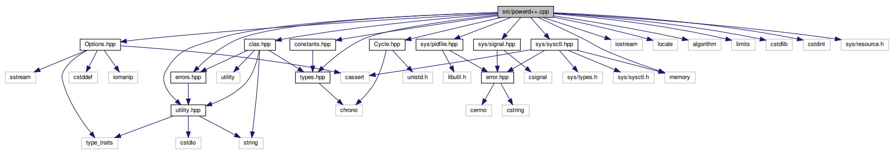
char const *	args	A comma separated list of arguments. Set to nullptr or "" if no argument is available.
char const *	lparam	The long version of this parameter. Set to nullptr or "" if no long parameter is available.
OptionT	option	The enum value to return for this option.
char	sparam	The short version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

12.10 src/powerd++.cpp File Reference

Implements powerd++ 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 powerd++.cpp:



Classes

- struct anonymous_namespace{powerd++.cpp}::CoreGroup

  Contains the management information for a group of cores with a common clock frequency. More...
- struct anonymous_namespace{powerd++.cpp}::Core

Contains the management information for a single CPU core. More...

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

  A collection of all the gloabl, mutable states. More...
- struct anonymous_namespace{powerd++.cpp}::Global::ACSet

  Per AC line state settings. More...
- class anonymous_namespace{powerd++.cpp}::FreqGuard

  A core frequency guard.

Namespaces

- anonymous_namespace{powerd++.cpp}

  File local scope.

Enumerations

- enum anonymous_namespace{powerd++.cpp}::AcLineState : unsigned int { anonymous_namespace{powerd++.cpp}::AcLineS
anonymous_namespace{powerd++.cpp}::AcLineState::ONLINE, anonymous_namespace{powerd++.cpp}::AcLineState::UNK
anonymous_namespace{powerd++.cpp}::AcLineState::LENGTH }

  The available AC line states.
- enum anonymous_namespace{powerd++.cpp}::OE {
anonymous_namespace{powerd++.cpp}::OE::USAGE, anonymous_namespace{powerd++.cpp}::OE::MODE_AC,
anonymous_namespace{powerd++.cpp}::OE::MODE_BATT, anonymous_namespace{powerd++.cpp}::OE::FREQ_MIN,
anonymous_namespace{powerd++.cpp}::OE::FREQ_MAX, anonymous_namespace{powerd++.cpp}::OE::FREQ_MIN_AC,
anonymous_namespace{powerd++.cpp}::OE::FREQ_MAX_AC, anonymous_namespace{powerd++.cpp}::OE::FREQ_MIN_BA
anonymous_namespace{powerd++.cpp}::OE::FREQ_MAX_BATT, anonymous_namespace{powerd++.cpp}::OE::FREQ_RANG
anonymous_namespace{powerd++.cpp}::OE::FREQ_RANGE_AC, anonymous_namespace{powerd++.cpp}::OE::FREQ_RANG
anonymous_namespace{powerd++.cpp}::OE::HITEMP_RANGE, anonymous_namespace{powerd++.cpp}::OE::MODE_UNKN
anonymous_namespace{powerd++.cpp}::OE::IVAL_POLL, anonymous_namespace{powerd++.cpp}::OE::FILE_PID,
anonymous_namespace{powerd++.cpp}::OE::FLAG_VERBOSE, anonymous_namespace{powerd++.cpp}::OE::FLAG_FOREG
anonymous_namespace{powerd++.cpp}::OE::CNT_SAMPLES, anonymous_namespace{powerd++.cpp}::OE::IGNORE,
anonymous_namespace{powerd++.cpp}::OE::OPT_UNKNOWN, anonymous_namespace{powerd++.cpp}::OE::OPT_NOOPT
anonymous_namespace{powerd++.cpp}::OE::OPT_DASH, anonymous_namespace{powerd++.cpp}::OE::OPT_LDASH,
anonymous_namespace{powerd++.cpp}::OE::OPT_DONE }

  An enum for command line parsing.

Functions

- void anonymous_namespace{powerd++.cpp}::verbose (std::string const &msg)

  Outputs the given message on stderr if g.verbose is set.
- void anonymous_namespace{powerd++.cpp}::sysctl_fail (sys::sc_error< sys::ctl::error > const err)

  Treat sysctl errors.
- void anonymous_namespace{powerd++.cpp}::init ()

  Perform initial tasks.
- 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<>
void anonymous_namespace{powerd++.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{powerd++.cpp}::update_freq (Global::ACSet const &acstate)

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

- void anonymous_namespace{powerd++.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_recv (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 gobal 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.
- Parameter< OE > const anonymous_namespace{powerd++.cpp}::PARAMETERS [ ]

  Definitions of command line parameters.

12.10.1 Detailed Description

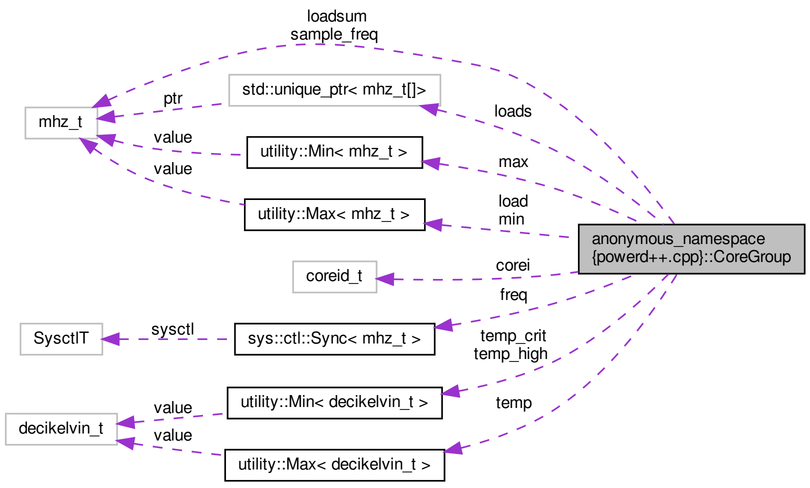
Implements powerd++ a drop in replacement for FreeBSD's powerd.

12.10.2 Class Documentation

12.10.2.1 struct anonymous_namespace{powerd++.cpp}::CoreGroup

Contains the management information for a group of cores with a common clock frequency.

Collaboration diagram for anonymous_namespace{powerd++.cpp}::CoreGroup:



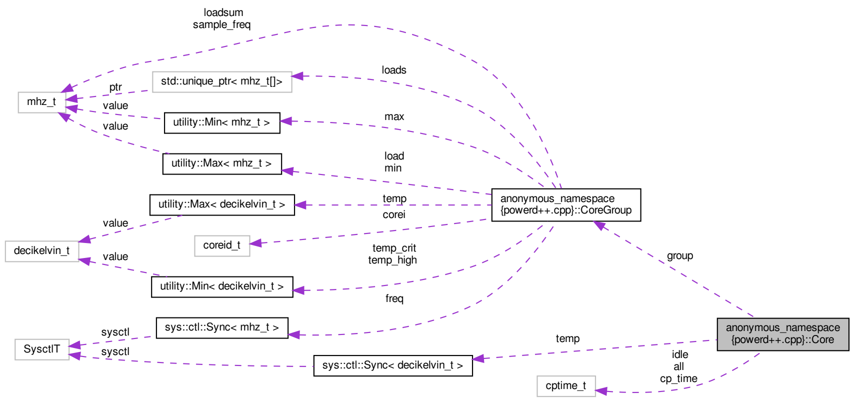
Class Members

coreid_t	corei	The number of the core owning dev.cpu. d.freq.
SysctlSync< mhz_t >	freq	The sysctl dev.cpu. d.freq.
Max< mhz_t >	load	The maximum load reported by all cores in the group. This is updated by update_loads().
unique_ptr< mhz_t[ ]>	loads	A ring buffer of maximum load samples for this core group. Each maximum load sample is weighted with the core frequency at which it was taken. This is updated by update_loads().
mhz_t	loadsum	The maximum load sum of all controlled cores. This is updated by update_loads().
Min< mhz_t >	max	The maximum group clock rate. The least of all core maxima in the group.
Max< mhz_t >	min	The minimum group clock rate. The greatest of all core minima in the group.
mhz_t	sample_freq	The dev.cpu. d.freq value for the current load sample. This is updated by update_loads().
Max< decikelvin_t >	temp	The maximum temperature measurement taken in the group.
Min< decikelvin_t >	temp_crit	Critical core temperature in dK.
Min< decikelvin_t >	temp_high	High core temperature in dK.

12.10.2.2 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

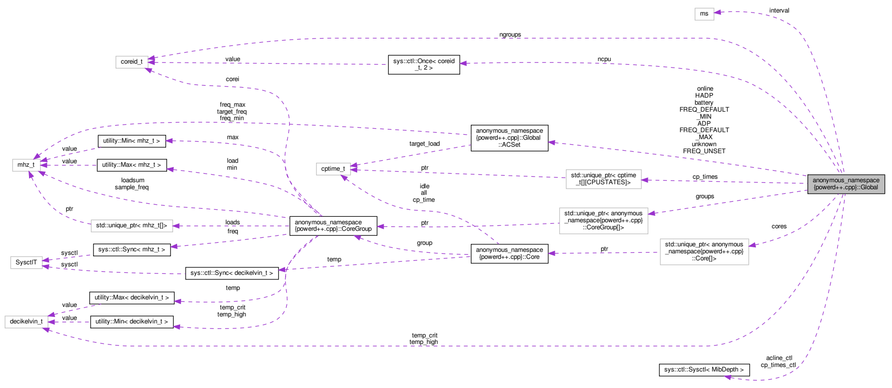
cptime_t	all	Count of all ticks.
cptime_t const ∗	cp_time	A pointer to the kern.cp_times section for this core.
CoreGroup ∗	group	The core that controls the frequency for this core.
cptime_t	idle	The idle ticks count.
SysctlSync< decikelvin_t >	temp	The dev.cpu. d.temperature sysctl, if present.

12.10.2.3 struct anonymous_namespace{powerd++.cpp}::Global

A collection of all the gloabl, mutable states.

This is mostly for semantic clarity.

Collaboration diagram for anonymous_namespace{powerd++.cpp}::Global:



Class Members

Sysctl	acline_ctl	The hw.acpi.acline ctl.

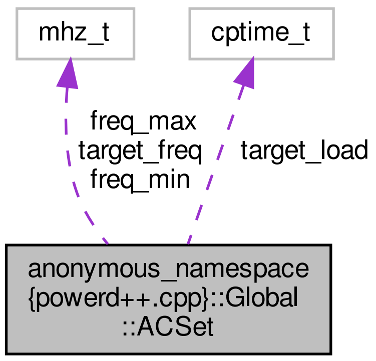
Class Members

struct anonymous_namespace{powerd++	ADP[3]	
struct anonymous_namespace{powerd++	battery[3]	
unique_ptr< Core[ ]>	cores	This buffer is to be allocated with ncpu instances of the Core struct to store the management information of every core.
unique_ptr< cptime_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 anonymous_namespace{powerd++	FREQ_DEFAULT_MAX[3]	
struct anonymous_namespace{powerd++	FREQ_DEFAULT_MIN[3]	
struct anonymous_namespace{powerd++	FREQ_UNSET[3]	
unique_ptr< CoreGroup[ ]>	groups	This buffer is to be allocated with the number of core groups. A core group is created by init() for each core that has a dev.cpu.d.freq handle.
struct anonymous_namespace{powerd++	HADP[3]	
ms	interval	The polling interval.
SysctlOnce< coreid_t, 2 > const	ncpu	The number of CPU cores or threads.
coreid_t	ngroups	The number of frequency controlling core groups.
struct anonymous_namespace{powerd++	online[3]	
char const ∗	pidfilename	Name of an alternative pidfile. If not given pidfile_open() 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.
decikelvin_t	temp_crit	User set critical core temperature in dK.
decikelvin_t	temp_high	User set high core temperature in dK.
bool	temp_throttling	Temperature throttling mode.
struct anonymous_namespace{powerd++	unknown[3]	The power states.
bool	verbose	Verbose mode.

12.10.2.4 struct anonymous_namespace{powerd++.cpp}::Global::ACSet

Per AC line state settings.

Collaboration diagram for anonymous_namespace{powerd++.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.

12.10.3 Function Documentation

12.10.3.1 main()

```
int main (
            int argc,
            char * argv[] )
```

Main routine, setup and execute daemon, print errors.

Parameters

argc,argv	The command line arguments

Returns

An exit code

See also

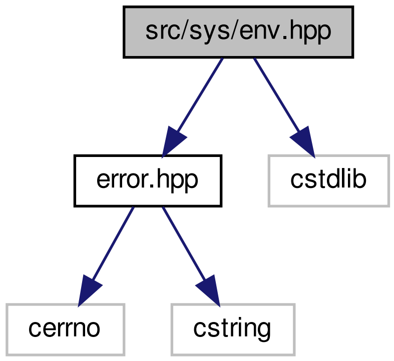
Exit

12.11 src/sys/env.hpp File Reference

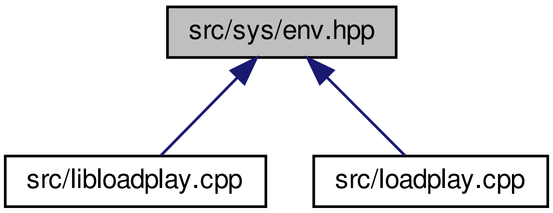
Implements zero-cost abstractions for the getenv(3) facilities.

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

Include dependency graph for env.hpp:



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



Classes

- struct sys::env::error

  The domain error type. More...
- class sys::env::Var

  A reference type refering to an environment variable.
- struct sys::env::Vars

  A singleton class providing access to environment variables.

Namespaces

- sys

  Wrappers around native system interfaces.
- sys::env

  Provides wrappers around the getenv() family of functions.

Variables

- struct sys::env::Vars sys::env::vars

  Singleton providing access to environment variables.

12.11.1 Detailed Description

Implements zero-cost abstractions for the getenv(3) facilities.

12.11.2 Class Documentation

12.11.2.1 struct sys::env::error

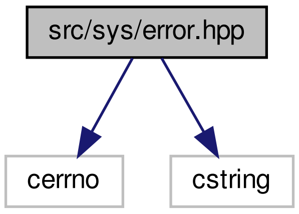
The domain error type.

12.12 src/sys/error.hpp File Reference

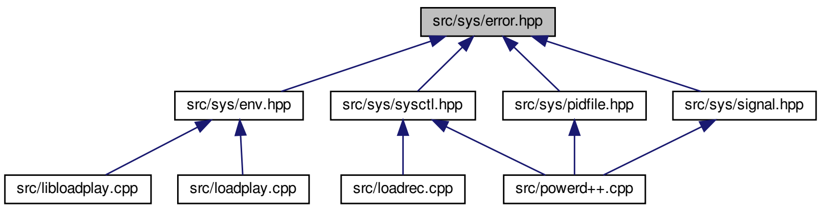
Provides system call error handling.

```
#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.

12.12.1 Detailed Description

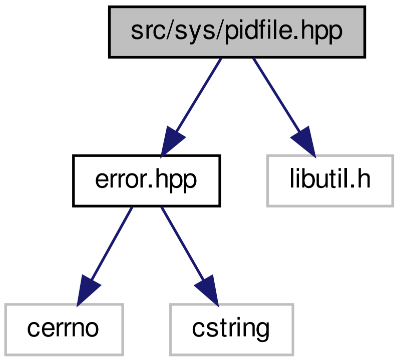
Provides system call error handling.

12.13 src/sys/pidfile.hpp File Reference

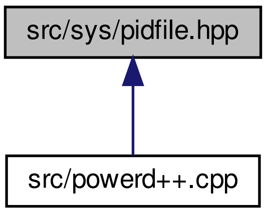
Implements safer c++ wrappers for the pidfile_∗() interface.

```
#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.

12.13.1 Detailed Description

Implements safer c++ wrappers for the pidfile_∗() interface.

Requires linking with -lutil.

12.13.2 Class Documentation

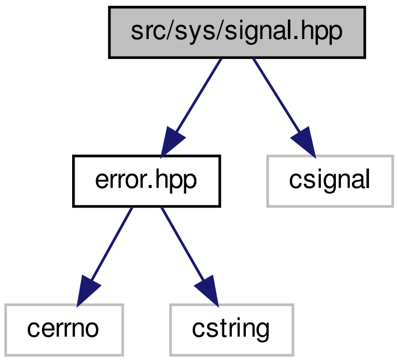
12.13.2.1 struct sys::pid::error

The domain error type.

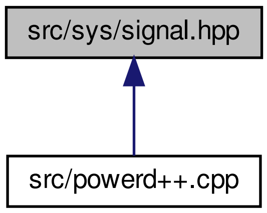
12.14 src/sys/signal.hpp File Reference

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.

12.14.1 Detailed Description

Implements a c++ wrapper for the signal(3) call.

12.14.2 Class Documentation

12.14.2.1 struct sys::sig::error

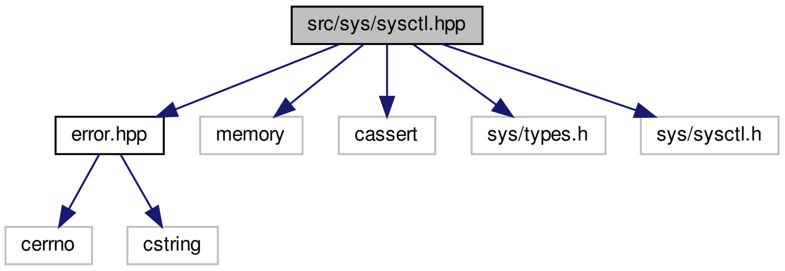
The domain error type.

12.15 src/sys/sysctl.hpp File Reference

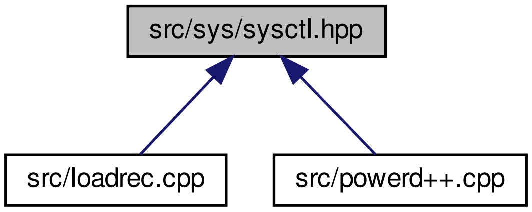
Implements safer c++ wrappers for the sysctl() interface.

```
#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.

12.15.1 Detailed Description

Implements safer c++ wrappers for the sysctl() interface.

12.15.2 Class Documentation

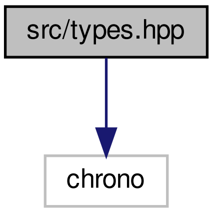
12.15.2.1 struct sys::ctl::error

The domain error type.

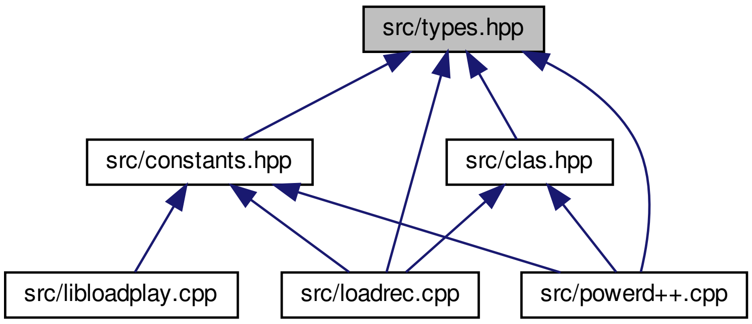
12.16 src/types.hpp File Reference

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.

12.16.1 Detailed Description

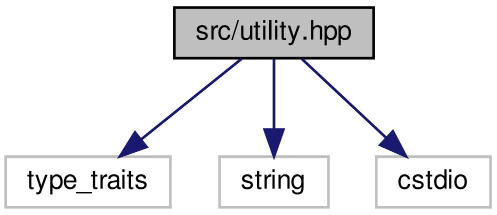
A collection of type aliases.

12.17 src/utility.hpp File Reference

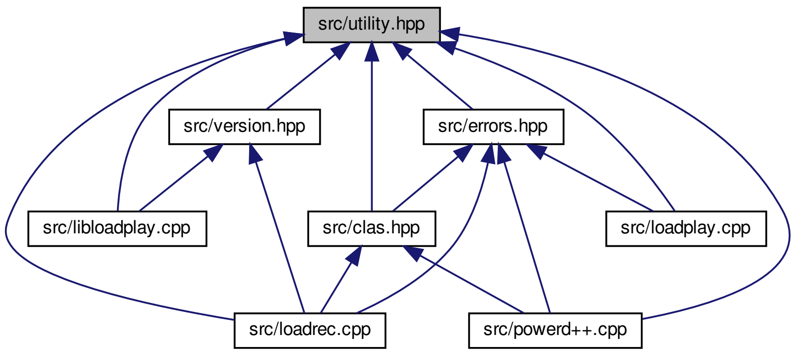
Implements generally useful functions.

```
#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.
- class utility::Sum< T >

  A simple value container only allowing += and copy assignment.
- class utility::Min< T >

  A simple value container that provides the minimum of assigned values.
- class utility::Max< T >

  A simple value container that provides the maximum of assigned values.

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.

12.17.1 Detailed Description

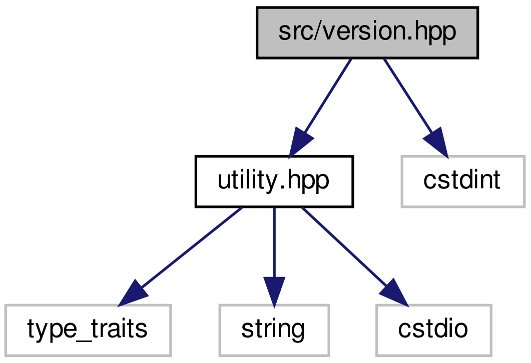
Implements generally useful functions.

12.18 src/version.hpp File Reference

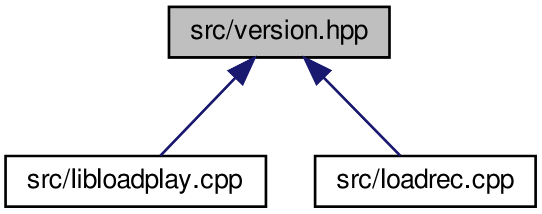
Defines types and constants used for version management.

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

Include dependency graph for version.hpp:



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



Namespaces

- version

  Version information constants and types.
- version::literals

  Literals to set flag bits.

Typedefs

- typedef uint64_t version::flag_t

  The data type to use for feature flags.

Enumerations

- enum version::LoadrecBits { version::LoadrecBits::FREQ_TRACKING }

  Feature flags for load recordings.

Functions

- constexpr flag_t version::literals::operator""_FREQ_TRACKING (unsigned long long int value)

  Set the FREQ_TRACKING bit.

Variables

- char const ∗const version::LOADREC_FEATURES = "usr.app.powerdxx.loadrec.features"

  The pseudo MIB name for the load recording feature flags.

12.18.1 Detailed Description

Defines types and constants used for version management.

Index