

powerd++

0.4.2+c9

Generated by Doxygen 1.8.16

1 Main Page	1
1.1 Using powerd++	2
1.1.1 Packages	2
1.1.2 Running powerd++	2
1.1.3 Manuals	2
1.1.4 Tuning	3
1.1.5 Reporting Issues / Requesting Features	3
1.2 Building/Installing	3
1.2.1 Building	4
1.2.2 Installing	4
1.2.3 Documentation	4
1.3 Development	4
1.3.1 Design	5
1.3.2 License	5
2 LICENSE	6
3 Manual loadplay(1)	6
4 Manual loadrec(1)	9
5 Manual powerd++(8)	11
6 Module Index	16
6.1 Modules	16
7 Namespace Index	16
7.1 Namespace List	16
8 Hierarchical Index	17
8.1 Class Hierarchy	17
9 Class Index	20
9.1 Class List	20
10 File Index	22
10.1 File List	22
11 Module Documentation	22
11.1 Standard I/O File Access	22
11.1.1 Detailed Description	23
12 Namespace Documentation	23
12.1 anonymous_namespace{clas.cpp} Namespace Reference	23
12.1.1 Detailed Description	23
12.1.2 Enumeration Type Documentation	24
12.1.3 Function Documentation	24

12.1.4 Variable Documentation	24
12.2 anonymous_namespace{libloadplay.cpp} Namespace Reference	25
12.2.1 Detailed Description	26
12.2.2 Class Documentation	27
12.2.3 Typedef Documentation	28
12.2.4 Function Documentation	29
12.2.5 Variable Documentation	32
12.3 anonymous_namespace{loadplay.cpp} Namespace Reference	32
12.3.1 Detailed Description	32
12.3.2 Enumeration Type Documentation	32
12.3.3 Function Documentation	33
12.3.4 Variable Documentation	34
12.4 anonymous_namespace{loadrec.cpp} Namespace Reference	34
12.4.1 Detailed Description	35
12.4.2 Typedef Documentation	35
12.4.3 Enumeration Type Documentation	35
12.4.4 Function Documentation	37
12.4.5 Variable Documentation	38
12.5 anonymous_namespace{powerd++.cpp} Namespace Reference	38
12.5.1 Detailed Description	40
12.5.2 Class Documentation	40
12.5.3 Enumeration Type Documentation	43
12.5.4 Function Documentation	44
12.5.5 Variable Documentation	47
12.6 clas Namespace Reference	47
12.6.1 Detailed Description	48
12.6.2 Function Documentation	48
12.7 constants Namespace Reference	52
12.7.1 Detailed Description	53
12.7.2 Variable Documentation	53
12.8 errors Namespace Reference	53
12.8.1 Detailed Description	54
12.8.2 Class Documentation	54
12.8.3 Enumeration Type Documentation	54
12.8.4 Function Documentation	55
12.8.5 Variable Documentation	55
12.9 nih Namespace Reference	56
12.9.1 Detailed Description	56
12.9.2 Class Documentation	56
12.9.3 Function Documentation	57
12.10 sys Namespace Reference	58
12.10.1 Detailed Description	58

12.11 sys::ctl Namespace Reference	58
12.11.1 Detailed Description	59
12.11.2 Class Documentation	60
12.11.3 Typedef Documentation	60
12.11.4 Function Documentation	60
12.12 sys::env Namespace Reference	63
12.12.1 Detailed Description	63
12.12.2 Class Documentation	63
12.13 sys::io Namespace Reference	63
12.13.1 Detailed Description	65
12.13.2 Class Documentation	65
12.13.3 Typedef Documentation	67
12.13.4 Enumeration Type Documentation	67
12.13.5 Variable Documentation	68
12.14 sys::pid Namespace Reference	69
12.14.1 Detailed Description	69
12.14.2 Class Documentation	69
12.15 sys::sig Namespace Reference	69
12.15.1 Detailed Description	70
12.15.2 Class Documentation	70
12.16 timing Namespace Reference	70
12.16.1 Detailed Description	70
12.17 types Namespace Reference	70
12.17.1 Detailed Description	71
12.17.2 Typedef Documentation	71
12.18 utility Namespace Reference	71
12.18.1 Detailed Description	72
12.18.2 Function Documentation	72
12.19 utility::literals Namespace Reference	73
12.19.1 Detailed Description	73
12.19.2 Function Documentation	74
12.20 version Namespace Reference	74
12.20.1 Detailed Description	74
12.20.2 Enumeration Type Documentation	74
12.21 version::literals Namespace Reference	75
12.21.1 Detailed Description	75
12.21.2 Function Documentation	75
13 Class Documentation	75
13.1 anonymous_namespace{libloadplay.cpp}::Callback< FunctionArgs > Class Template Reference . .	75
13.1.1 Detailed Description	76
13.1.2 Constructor & Destructor Documentation	76
13.1.3 Member Function Documentation	77

13.2 sys::io::contains< set< Set ... >, Value > Struct Template Reference	77
13.2.1 Detailed Description	78
13.3 sys::io::query::contains_ftor Struct Reference	79
13.3.1 Detailed Description	79
13.3.2 Member Function Documentation	79
13.4 timing::Cycle Class Reference	81
13.4.1 Detailed Description	81
13.4.2 Member Function Documentation	82
13.4.3 Member Data Documentation	83
13.5 anonymous_namespace{libloadplay.cpp}::Emulator Class Reference	83
13.5.1 Detailed Description	84
13.5.2 Class Documentation	84
13.5.3 Constructor & Destructor Documentation	85
13.5.4 Member Function Documentation	85
13.5.5 Member Data Documentation	85
13.6 sys::io::enable_if< bool, T > Struct Template Reference	86
13.6.1 Detailed Description	86
13.7 sys::io::enable_if< false, T > Struct Template Reference	86
13.7.1 Detailed Description	86
13.8 nih::enum_has_members< OptionT, class > Struct Template Reference	87
13.8.1 Detailed Description	87
13.9 sys::io::file< link, Features ... > Class Template Reference	88
13.9.1 Detailed Description	89
13.9.2 Constructor & Destructor Documentation	90
13.9.3 Member Function Documentation	91
13.10 sys::io::file< own, Features ... > Class Template Reference	92
13.10.1 Detailed Description	93
13.10.2 Constructor & Destructor Documentation	94
13.10.3 Member Function Documentation	95
13.11 sys::io::file_feature< FileT,... > Class Template Reference	97
13.11.1 Detailed Description	97
13.11.2 Constructor & Destructor Documentation	98
13.11.3 Member Function Documentation	98
13.12 sys::io::file_feature< FileT, read, Tail ... > Class Template Reference	99
13.12.1 Detailed Description	100
13.12.2 Member Function Documentation	101
13.13 sys::io::file_feature< FileT, seek, Tail ... > Class Template Reference	103
13.13.1 Detailed Description	104
13.13.2 Member Function Documentation	105
13.14 sys::io::file_feature< FileT, write, Tail ... > Class Template Reference	106
13.14.1 Detailed Description	107
13.14.2 Member Function Documentation	107

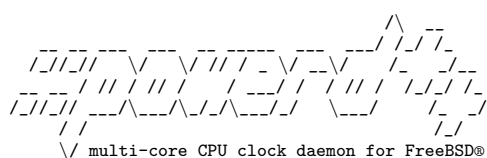
13.15 <code>utility::Formatter< BufSize ></code> Class Template Reference	111
13.15.1 Detailed Description	111
13.15.2 Member Function Documentation	112
13.16 <code>anonymous_namespace{libloadplay.cpp}::Report::Frame</code> Class Reference	112
13.16.1 Detailed Description	113
13.16.2 Constructor & Destructor Documentation	113
13.16.3 Member Function Documentation	113
13.17 <code>anonymous_namespace{powerd++.cpp}::FreqGuard</code> Class Reference	115
13.17.1 Detailed Description	116
13.18 <code>anonymous_namespace{libloadplay.cpp}::Hold< T ></code> Class Template Reference	116
13.18.1 Detailed Description	116
13.18.2 Constructor & Destructor Documentation	117
13.19 <code>sys::io::is_superset_of< LSetT, set< RSet ... > ></code> Struct Template Reference	117
13.19.1 Detailed Description	118
13.20 <code>anonymous_namespace{libloadplay.cpp}::Main</code> Class Reference	118
13.20.1 Detailed Description	119
13.20.2 Constructor & Destructor Documentation	119
13.21 <code>utility::Max< T ></code> Class Template Reference	120
13.21.1 Detailed Description	120
13.21.2 Constructor & Destructor Documentation	121
13.21.3 Member Function Documentation	121
13.22 <code>anonymous_namespace{libloadplay.cpp}::mib_t</code> Struct Reference	122
13.22.1 Detailed Description	122
13.22.2 Constructor & Destructor Documentation	122
13.22.3 Member Function Documentation	123
13.23 <code>utility::Min< T ></code> Class Template Reference	124
13.23.1 Detailed Description	125
13.23.2 Constructor & Destructor Documentation	125
13.23.3 Member Function Documentation	125
13.24 <code>sys::ctl::Once< T, SysctlT ></code> Class Template Reference	126
13.24.1 Detailed Description	127
13.24.2 Constructor & Destructor Documentation	127
13.24.3 Member Function Documentation	128
13.25 <code>nih::Options< OptionT, DefCount ></code> Class Template Reference	128
13.25.1 Detailed Description	129
13.25.2 Constructor & Destructor Documentation	130
13.25.3 Member Function Documentation	130
13.25.4 Member Data Documentation	134
13.26 <code>sys::pid::Pidfile</code> Class Reference	135
13.26.1 Detailed Description	135
13.26.2 Constructor & Destructor Documentation	135
13.26.3 Member Function Documentation	136

13.26.4 Member Data Documentation	136
13.27 anonymous_namespace{libloadplay.cpp}::Report Class Reference	136
13.27.1 Detailed Description	137
13.27.2 Constructor & Destructor Documentation	137
13.27.3 Member Function Documentation	137
13.28 sys::sc_error< Domain > Struct Template Reference	138
13.28.1 Detailed Description	138
13.28.2 Member Function Documentation	139
13.29 sys::sig::Signal Class Reference	139
13.29.1 Detailed Description	140
13.29.2 Constructor & Destructor Documentation	140
13.30 utility::Sum< T > Class Template Reference	140
13.30.1 Detailed Description	141
13.30.2 Constructor & Destructor Documentation	141
13.30.3 Member Function Documentation	142
13.31 sys::ctl::Sync< T, SysctlT > Class Template Reference	142
13.31.1 Detailed Description	143
13.31.2 Constructor & Destructor Documentation	143
13.31.3 Member Function Documentation	144
13.32 sys::ctl::Sysctl< MibDepth > Class Template Reference	144
13.32.1 Detailed Description	145
13.32.2 Constructor & Destructor Documentation	146
13.32.3 Member Function Documentation	146
13.33 sys::ctl::Sysctl< 0 > Class Template Reference	149
13.33.1 Detailed Description	149
13.33.2 Constructor & Destructor Documentation	149
13.33.3 Member Function Documentation	150
13.34 anonymous_namespace{libloadplay.cpp}::Sysctls Class Reference	152
13.34.1 Detailed Description	153
13.34.2 Member Function Documentation	153
13.34.3 Member Data Documentation	155
13.35 anonymous_namespace{libloadplay.cpp}::SysctlValue Class Reference	156
13.35.1 Detailed Description	157
13.35.2 Constructor & Destructor Documentation	157
13.35.3 Member Function Documentation	158
13.35.4 Member Data Documentation	163
13.36 sys::env::Var Class Reference	163
13.36.1 Detailed Description	164
13.36.2 Constructor & Destructor Documentation	164
13.36.3 Member Function Documentation	165
13.37 sys::env::Vars Struct Reference	166
13.37.1 Detailed Description	167

13.37.2 Member Function Documentation	167
14 File Documentation	167
14.1 src/clas.hpp File Reference	167
14.1.1 Detailed Description	169
14.2 src/constants.hpp File Reference	169
14.2.1 Detailed Description	171
14.3 src/Cycle.hpp File Reference	171
14.3.1 Detailed Description	172
14.4 src/errors.hpp File Reference	172
14.4.1 Detailed Description	173
14.4.2 Class Documentation	173
14.5 src/libloadplay.cpp File Reference	174
14.5.1 Detailed Description	176
14.5.2 Class Documentation	177
14.5.3 Function Documentation	179
14.6 src/loadplay.cpp File Reference	183
14.6.1 Detailed Description	185
14.6.2 Function Documentation	185
14.7 src/loadrec.cpp File Reference	185
14.7.1 Detailed Description	187
14.7.2 Function Documentation	187
14.8 src/Options.hpp File Reference	187
14.8.1 Detailed Description	189
14.8.2 Class Documentation	190
14.9 src/powerd++.cpp File Reference	191
14.9.1 Detailed Description	193
14.9.2 Class Documentation	193
14.9.3 Function Documentation	197
14.10 src/sys/env.hpp File Reference	197
14.10.1 Detailed Description	198
14.10.2 Class Documentation	198
14.11 src/sys/error.hpp File Reference	199
14.11.1 Detailed Description	199
14.12 src/sys/io.hpp File Reference	200
14.12.1 Detailed Description	202
14.12.2 Class Documentation	202
14.13 src/sys/pidfile.hpp File Reference	203
14.13.1 Detailed Description	205
14.13.2 Class Documentation	205
14.14 src/sys/signal.hpp File Reference	205
14.14.1 Detailed Description	206
14.14.2 Class Documentation	206

14.15 src/sys/sysctl.hpp File Reference	206
14.15.1 Detailed Description	208
14.15.2 Class Documentation	208
14.16 src/types.hpp File Reference	208
14.16.1 Detailed Description	209
14.17 src/utility.hpp File Reference	209
14.17.1 Detailed Description	211
14.18 src/version.hpp File Reference	211
14.18.1 Detailed Description	212
Index	213

1 Main Page



The power++ 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. [Using powerd++](#)
 - (a) [Packages](#)
 - (b) [Running powerd++](#)
 - (c) [Manuals](#)
 - (d) [Tuning](#)
 - (e) [Reporting Issues / Requesting Features](#)
2. [Building/Installing](#)
 - (a) [Building](#)
 - (b) [Installing](#)
 - (c) [Documentation](#)
3. [Development](#)
 - (a) [Design](#)
 - (b) [License](#)

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 <code>CXXFLAGS=-O0 -g -DEBUG</code>
paranoid	Turn on undefined behaviour canaries
install	Install tools and manuals
deinstall	Deinstall tools and manuals
clean	Clear build directory <code>obj/</code>
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++17 -Wall -Werror -pedantic -c src/powerd++.cpp -o powerd++.o
c++ -O2 -pipe -march=haswell -std=c++17 -Wall -Werror -pedantic -c src/clas.cpp -o clas.o
c++ -O2 -pipe -march=haswell -std=c++17 -Wall -Werror -pedantic powerd++.o clas.o -lutil -o powerd++
c++ -O2 -pipe -march=haswell -std=c++17 -Wall -Werror -pedantic -c src/loadrec.cpp -o loadrec.o
c++ -O2 -pipe -march=haswell -std=c++17 -Wall -Werror -pedantic loadrec.o clas.o -o loadrec
c++ -O2 -pipe -march=haswell -std=c++17 -Wall -Werror -pedantic -c src/loadplay.cpp -o loadplay.o
c++ -O2 -pipe -march=haswell -std=c++17 -Wall -Werror -pedantic loadplay.o clas.o -o loadplay
c++ -O2 -pipe -march=haswell -std=c++17 -Wall -Werror -pedantic -fPIC -c src/libloadplay.cpp -o libloadplay.o
c++ -O2 -pipe -march=haswell -std=c++17 -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](https://github.com).

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 <code>mdoc(7)</code> 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

Copyright © 2016 - 2019 Dominic Fandrey kami@freebsd.org

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

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

3 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(3), 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

3 September, 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 1700/9680

```

Supposedly the first mode, which is off by 1 MHz, invokes the turbo mode. However all modes down to 1800 MHz actually invoke the turbo mode for this model. The only way to determine this is by benchmarking the 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 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:

<u>mode</u>	The mode is either a <u>load</u> target or a fixed <u>freq</u> . 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.

<u>load</u>	A load is either a fraction in the range [0.0, 1.0] or a percentage in the range [0%, 100%].
<u>freq</u>	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).
<u>temp</u>	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.
<u>ival</u>	A time interval can be given in seconds or milliseconds. s, ms An interval without a unit is treated as milliseconds.
<u>cnt</u>	A positive integer.
<u>file</u>	A file name.

OPTIONS

The following options are supported:

<u>-h, --help</u>	Show usage and exit
<u>-v, --verbose</u>	Be verbose and produce initial diagnostics on <u>stderr</u> .
<u>-f, --foreground</u>	Stay in foreground, produce an event log on <u>stdout</u> .
<u>-a, --ac mode</u>	Mode to use while the AC power line is connected (default hadp).
<u>-b, --batt mode</u>	Mode to use while battery powered (default adp).
<u>-n, --unknown mode</u>	Mode to use while the power line state is unknown (default hadp).
<u>-m, --min freq</u>	The lowest CPU clock frequency to use (default 0Hz).
<u>-M, --max freq</u>	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 Module Index

6.1 Modules

Here is a list of all modules:

Standard I/O File Access	22
---------------------------------	-----------

7 Namespace Index

7.1 Namespace List

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

anonymous_namespace{clas.cpp}	
File local scope	23
anonymous_namespace{libloadplay.cpp}	
File local scope	25
anonymous_namespace{loadplay.cpp}	
File local scope	32
anonymous_namespace{loadrec.cpp}	
File local scope	34
anonymous_namespace{powerd++.cpp}	
File local scope	38
clas	
A collection of functions to process command line arguments	47
constants	
A collection of constants	52

errors	
Common error handling types and functions	53
nih	
Not invented here namespace, for code that substitutes already commonly available functionality	56
sys	58
sys::ctl	
This namespace contains safer c++ wrappers for the <code>sysctl()</code> interface	58
sys::env	
Provides wrappers around the <code>getenv()</code> family of functions	63
sys::io	
This namespace contains c++ wrappers for <code><cstdio></code> functionality	63
sys::pid	
This namespace contains safer c++ wrappers for the <code>pidfile_*</code> () interface	69
sys::sig	
This namespace provides c++ wrappers for <code>signal(3)</code>	69
timing	
Namespace for time management related functionality	70
types	
A collection of type aliases	70
utility	
A collection of generally useful functions	71
utility::literals	
Contains literal operators	73
version	
Version information constants and types	74
version::literals	
Literals to set flag bits	75

8 Hierarchical Index

8.1 Class Hierarchy

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

<code>anonymous_namespace{powerd++.cpp}::Global::ACSet</code>	38
<code>anonymous_namespace{libloadplay.cpp}::Callback< FunctionArgs ></code>	75
<code>anonymous_namespace{libloadplay.cpp}::Callback< anonymous_namespace{libloadplay.cpp}::SysctlValue & ></code>	75
<code>sys::io::contains< SetT, Value ></code>	63
<code>sys::io::query::contains_ftor</code>	79

<code>anonymous_namespace{libloadplay.cpp}::Emulator::Core</code>	83
<code>anonymous_namespace{powerd++.cpp}::Core</code>	38
<code>anonymous_namespace{libloadplay.cpp}::CoreFrameReport</code>	25
<code>anonymous_namespace{powerd++.cpp}::CoreGroup</code>	38
<code>anonymous_namespace{libloadplay.cpp}::CoreReport</code>	25
<code>timing::Cycle</code>	81
<code>anonymous_namespace{libloadplay.cpp}::Emulator</code>	83
<code>sys::io::enable_if< bool, T ></code>	86
<code>sys::io::enable_if< false, T ></code>	86
<code>sys::io::enable_if<((Set==Value) ...)></code>	86
<code>sys::io::contains< set< Set ... >, Value ></code>	77
<code>sys::io::enable_if<(contains_v< LSetT, RSet > &&...)></code>	86
<code>sys::io::is_superset_of< LSetT, set< RSet ... > ></code>	117
<code>sys::env::error</code>	63
<code>sys::pid::error</code>	69
<code>sys::sig::error</code>	69
<code>sys::ctl::error</code>	58
<code>errors::Exception</code>	53
<code>sys::io::file< Ownership, Features ></code>	63
<code>sys::io::file< io::link ></code>	63
<code>sys::io::file< io::own ></code>	63
<code>sys::io::file_feature< FileT,... ></code>	97
<code>sys::io::file_feature< file< link, Features ... >, Features ... ></code>	97
<code>sys::io::file< link, Features ... ></code>	88
<code>sys::io::file_feature< file< own, Features ... >, Features ... ></code>	97
<code>sys::io::file< own, Features ... ></code>	92
<code>sys::io::file_feature< FileT, Tail ... ></code>	97
<code>sys::io::file_feature< FileT, read, Tail ... ></code>	99
<code>sys::io::file_feature< FileT, seek, Tail ... ></code>	103
<code>sys::io::file_feature< FileT, write, Tail ... ></code>	106
<code>utility::Formatter< BufSize ></code>	111
<code>anonymous_namespace{libloadplay.cpp}::Report::Frame</code>	112

<code>anonymous_namespace{powerd++.cpp}::FreqGuard</code>	115
<code>anonymous_namespace{powerd++.cpp}::Global</code>	38
<code>anonymous_namespace{libloadplay.cpp}::Hold< T ></code>	116
<code>sys::io::is_superset_of< LSetT, RSetT ></code>	63
<code>anonymous_namespace{libloadplay.cpp}::Main</code>	118
<code>utility::Max< T ></code>	120
<code>utility::Max< decikelvin_t ></code>	120
<code>utility::Max< mhz_t ></code>	120
<code>anonymous_namespace{libloadplay.cpp}::mib_t</code>	122
<code>utility::Min< T ></code>	124
<code>utility::Min< decikelvin_t ></code>	124
<code>utility::Min< mhz_t ></code>	124
<code>sys::ctl::Once< T, SysctlT ></code>	126
<code>sys::ctl::Once< coreid_t, 2 ></code>	126
<code>nih::Options< OptionT, DefCount ></code>	128
<code>nih::Parameter< OptionT ></code>	56
<code>sys::pid::Pidfile</code>	135
<code>sys::io::query</code>	63
<code>anonymous_namespace{libloadplay.cpp}::Report</code>	136
<code>sys::sc_error< Domain ></code>	138
<code>sys::io::set< Set ></code>	63
<code>sys::sig::Signal</code>	139
<code>utility::Sum< T ></code>	140
<code>utility::Sum< uint64_t ></code>	140
<code>sys::ctl::Sync< T, SysctlT ></code>	142
<code>sys::ctl::Sync< decikelvin_t ></code>	142
<code>sys::ctl::Sync< mhz_t ></code>	142
<code>sys::ctl::Sysctl< MibDepth ></code>	144
<code>sys::ctl::Sysctl< 0 ></code>	149
<code>anonymous_namespace{libloadplay.cpp}::Sysctls</code>	152
<code>anonymous_namespace{libloadplay.cpp}::SysctlValue true_type</code>	156

<code>nih::enum_has_members< OptionT, class ></code>	87
<code>sys::env::Var</code>	163
<code>sys::env::Vars</code>	166

9 Class Index

9.1 Class List

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

<code>anonymous_namespace{libloadplay.cpp}::Callback< FunctionArgs ></code> Implements a recursion safe <code>std::function</code> wrapper	75
<code>sys::io::contains< set< Set ... >, Value ></code> Specialise <code>io::contains</code> to unpack <code>io::set</code>	77
<code>sys::io::query::contains_ftor</code> Test a string whether it contains a set of characters	79
<code>timing::Cycle</code> Implements an interruptible cyclic sleeping functor	81
<code>anonymous_namespace{libloadplay.cpp}::Emulator</code> Instances of this class represent an emulator session	83
<code>sys::io::enable_if< bool, T ></code> Similar to <code>std::enable_if</code> , but it also has the value of the expression	86
<code>sys::io::enable_if< false, T ></code> Specialise <code>enable_if</code> for a false expression	86
<code>nih::enum_has_members< OptionT, class ></code> Tests whether the given enum provides all the required definitions	87
<code>sys::io::file< link, Features ... ></code> Specialise for FILE object linking file instances	88
<code>sys::io::file< own, Features ... ></code> Specialise for FILE object owning file instances	92
<code>sys::io::file_feature< FileT,... ></code> Implements the base functionality of all file access types	97
<code>sys::io::file_feature< FileT, read, Tail ... ></code> Implement read support for file types	99
<code>sys::io::file_feature< FileT, seek, Tail ... ></code> Implement seek support for file types	103
<code>sys::io::file_feature< FileT, write, Tail ... ></code> Implement write support for file types	106
<code>utility::Formatter< BufSize ></code> A formatting wrapper around string literals	111

<code>anonymous_namespace{libloadplay.cpp}::Report::Frame</code>	
Represents a frame of the report	112
<code>anonymous_namespace{powerd++.cpp}::FreqGuard</code>	
A core frequency guard	115
<code>anonymous_namespace{libloadplay.cpp}::Hold< T ></code>	
Sets a referenced variable to a given value and restores it when going out of context	116
<code>sys::io::is_superset_of< LSetT, set< RSet ... > ></code>	
Specialise <code>is_superset_of</code> to unpack the right hand <code>io::set</code>	117
<code>anonymous_namespace{libloadplay.cpp}::Main</code>	
Singleton class representing the main execution environment	118
<code>utility::Max< T ></code>	
A simple value container that provides the maximum of assigned values	120
<code>anonymous_namespace{libloadplay.cpp}::mib_t</code>	
Represents MIB, but wraps it to provide the necessary operators to use it as an <code>std::map</code> key	122
<code>utility::Min< T ></code>	
A simple value container that provides the minimum of assigned values	124
<code>sys::ctl::Once< T, SysctlT ></code>	
A read once representation of a <code>Sysctl</code>	126
<code>nih::Options< OptionT, DefCount ></code>	
An instance of this class offers operators to retrieve command line options and arguments	128
<code>sys::pid::Pidfile</code>	
A wrapper around the <code>pidfile_*</code> family of commands implementing the RAIL pattern	135
<code>anonymous_namespace{libloadplay.cpp}::Report</code>	
Provides a mechanism to provide frame wise per core load information	136
<code>sys::sc_error< Domain ></code>	
Can be thrown by syscall function wrappers if the function returned with an error	138
<code>sys::sig::Signal</code>	
Sets up a given signal handler and restores the old handler when going out of scope	139
<code>utility::Sum< T ></code>	
A simple value container only allowing += and copy assignment	140
<code>sys::ctl::Sync< T, SysctlT ></code>	
This is a wrapper around <code>Sysctl</code> that allows semantically transparent use of a <code>sysctl</code>	142
<code>sys::ctl::Sysctl< MibDepth ></code>	
Represents a <code>sysctl</code> MIB address	144
<code>sys::ctl::Sysctl< 0 ></code>	
This is a specialisation of <code>Sysctl</code> for <code>sysctls</code> using symbolic names	149
<code>anonymous_namespace{libloadplay.cpp}::Sysctls</code>	
Singleton class representing the <code>sysctl</code> table for this library	152
<code>anonymous_namespace{libloadplay.cpp}::SysctlValue</code>	
Instances of this class represents a specific <code>sysctl</code> value	156

sys::env::Var	163
A reference type refering to an environment variable	
sys::env::Vars	166
A singleton class providing access to environment variables	

10 File Index

10.1 File List

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

src/clas.hpp	167
src/constants.hpp	169
src/Cycle.hpp	171
src/errors.hpp	172
src/libloadplay.cpp	174
src/loadplay.cpp	183
src/loadrec.cpp	185
src/Options.hpp	187
src/powerd++.cpp	191
src/types.hpp	208
src/utility.hpp	209
src/version.hpp	211
src/sys/env.hpp	197
src/sys/error.hpp	199
src/sys/io.hpp	200
src/sys/pidfile.hpp	203
src/sys/signal.hpp	205
src/sys/sysctl.hpp	206

11 Module Documentation

11.1 Standard I/O File Access

A set of file instances providing access to stderr, stdout and stdin.

Variables

- `file< link, write > sys::io::ferr {stderr}`
File access instances for stderr.
- `file< link, write > sys::io::fout {stdout}`
File access instances for stdout.
- `file< link, read > sys::io::fin {stdin}`
File access instances for stdin.

11.1.1 Detailed Description

A set of file instances providing access to stderr, stdout and stdin.

In theory these should be functions returning a reference to a local static file object, to avoid global object initialisation order issues.

This would be annoying to access, though. In practice it works the way it is and it would be hard to notice if it did not.

12 Namespace Documentation

12.1 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

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

12.1.1 Detailed Description

File local scope.

12.1.2 Enumeration Type Documentation

12.1.2.1 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.

12.1.3 Function Documentation

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

Determine the unit of a string encoded value.

Parameters

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

Returns

A unit

12.1.4 Variable Documentation

12.1.4.1 UnitStr `const char* 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.

12.2 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.

Typedefs

- `template<auto Ownership>`
`using ofile = io::file< Ownership, io::write >`
Output file type alias.
- `template<auto Ownership>`
`using ifile = io::file< Ownership, io::read >`
Input file type alias.

Functions

- `template<size_t Size>`
`int strcmp (char const *const s1, char const (&s2)[Size])`
Safe wrapper around strcmp, which automatically determines the buffer size of s2.
- `std::regex operator""_r (char const *const str, size_t const len)`
User defined literal for regular expressions.
- `template<typename ... ArgTs>`
`constexpr void dprintf (ArgTs &&... args)`
Calls io::ferr.printf(...) if built with -DEBUG.
- `template<>`
`std::string SysctlValue::get< std::string > () const`
Returns a copy of the value string.
- `template<typename... MsgTs>`
`ofile< io::link > debug (MsgTs &&... msg)`
Print a debugging message if built with -DEBUG.
- `template<typename... MsgTs>`
`ofile< io::link > warn (MsgTs &&... msg)`
Print a warning.
- `template<typename... MsgTs>`
`ofile< io::link > fail (MsgTs &&... msg)`
This prints an error message and sets sys_results to make the hijacked process fail.
- `ofile< io::link > operator<< (ofile< io::link > fout, CoreFrameReport const &frame)`
Print recorded and running clock frequency and load for a frame.

Variables

- `constexpr const flag_t 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.

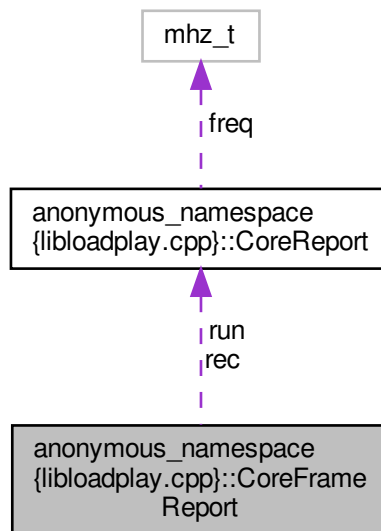
12.2.1 Detailed Description

File local scope.

12.2.2 Class Documentation

12.2.2.1 struct anonymous_namespace{libloadplay.cpp}::CoreFrameReport The report frame information for a single CPU pipeline.

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

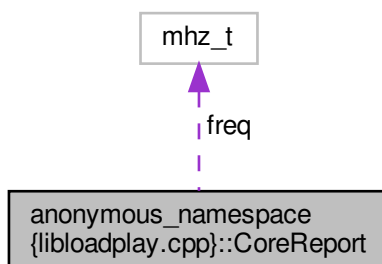


Class Members

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

12.2.2.2 struct anonymous_namespace{libloadplay.cpp}::CoreReport The reported state of a single CPU pipeline.

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



Class Members

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

12.2.3 Typedef Documentation

12.2.3.1 `ifile` `template<auto Ownership>`

```
using anonymous_namespace{libloadplay.cpp}::ifile = typedef io::file<Ownership, io::read>
```

Input file type alias.

Template Parameters

<i>Ownership</i>	The io::ownership type of the file
------------------	------------------------------------

12.2.3.2 `ofile` `template<auto Ownership>`

```
using anonymous_namespace{libloadplay.cpp}::ofile = typedef io::file<Ownership, io::write>
```

Output file type alias.

Template Parameters

<i>Ownership</i>	The io::ownership type of the file
------------------	------------------------------------

12.2.4 Function Documentation

12.2.4.1 debug() `template<typename... MsgTs>`
[ofile](#)<io::link> anonymous_namespace{libloadplay.cpp}::debug (
 MsgTs &&... *msg*) [inline]

Print a debugging message if built with -DEBUG.

Template Parameters

$MsgTs$	The message argument types
---------	----------------------------

Parameters

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

Returns

An output file handle for extending the message

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

Calls io::ferr.printf(...) if built with -DEBUG.

Template Parameters

$ArgTs$	The argument types to forward
---------	-------------------------------

Parameters

<i>args</i>	Arguments are forwarded to fprintf()
-------------	--------------------------------------

12.2.4.3 fail() `template<typename... MsgTs>`
[ofile](#)<io::link> anonymous_namespace{libloadplay.cpp}::fail (
 MsgTs &&... *msg*) [inline]

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

Template Parameters

<i>Msg</i> ↔ <i>Ts</i>	The message argument types
---------------------------	----------------------------

Parameters

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

Returns

An output file handle for extending the message

12.2.4.4 `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

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

Returns

A regular expression

12.2.4.5 `operator<<()` `ofile<io::link> anonymous_namespace{libloadplay.cpp}::operator<< (`
 `ofile< io::link > fout,`
 `CoreFrameReport const & frame)`

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

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

Parameters

<i>fout</i>	The stream to print to
<i>frame</i>	The frame information to print

Returns

A reference to the out stream

12.2.4.6 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

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

Parameters

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

Return values

<i>0</i>	Strings are equal
<i>!0</i>	Strings are not equal

12.2.4.7 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

12.2.4.8 warn() `template<typename... MsgTs>`
`ofile<io::link> anonymous_namespace{libloadplay.cpp}::warn (`
`MsgTs &&... msg) [inline]`

Print a warning.

Template Parameters

<i>MsgTs</i>	The message argument types
--------------	----------------------------

Parameters

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

Returns

An output file handle for extending the message

12.2.5 Variable Documentation

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

Initial value:

```
{  
    1_FREQ_TRACKING  
}
```

The set of supported features.

This value is used to ensure correct input data interpretation.

12.3 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

- const char * `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

- const char *const `USAGE` = "[-h] [-i file] [-o file] command [...]"
The short usage string.
- const `Parameter` < `OE` > `PARAMETERS` []
Definitions of command line parameters.

12.3.1 Detailed Description

File local scope.

12.3.2 Enumeration Type Documentation

12.3.2.1 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.

12.3.3 Function Documentation

12.3.3.1 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

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

Exceptions

<code>errors::Exception{Exit::EEXEC}</code>	
---	--

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

Performs very rudimentary file name argument checks.

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

Parameters

<i>path</i>	The file path to check
-------------	------------------------

Returns

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

12.3.4 Variable Documentation

12.3.4.1 PARAMETERS `const Parameter<OE> 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.

12.4 anonymous_namespace{loadrec.cpp} Namespace Reference

File local scope.

Typedefs

- `template<auto Ownership>`
`using ofile = io::file< Ownership, io::write >`
Output file type alias.

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

- `template<typename... MsgTs>`
`void verbose (MsgTs &&... msg)`
Outputs the given printf style 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 const flag_t **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.
 ofile< io::link > fout = io::fout
 The output stream either io::fout (stdout) or a file.
 const char * outfilename {nullptr}
 The user provided output file name.
 const sys::ctl::SysctlOnce< coreid_t, 2 > ncpu {1U, {CTL_HW, HW_NCPU}}
 The number of CPU cores/threads.
} g

```

*The global state.*
- const char \*const **USAGE** = "[ -hv] [ -d ival] [ -p ival] [ -o file]"  
*The short usage string.*
- const **Parameter**< OE > **PARAMETERS** []  
*Definitions of command line parameters.*

### 12.4.1 Detailed Description

File local scope.

### 12.4.2 Typedef Documentation

**12.4.2.1 ofile** template<auto Ownership>  
using anonymous\_namespace{loadrec.cpp}::ofile = typedef io::file<Ownership, io::write>

Output file type alias.

Template Parameters

|                  |                                    |
|------------------|------------------------------------|
| <i>Ownership</i> | The io::ownership type of the file |
|------------------|------------------------------------|

### 12.4.3 Enumeration Type Documentation

**12.4.3.1** **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.                        |

#### 12.4.4 Function Documentation

**12.4.4.1 print\_sysctls()** void anonymous\_namespace{loadrec.cpp}::print\_sysctls ( )

Print the sysctls.

**12.4.4.2 read\_args()** void anonymous\_namespace{loadrec.cpp}::read\_args (   
int const *argc*,  
char const \*const *argv*[] )

Parse command line arguments.

Parameters

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

**12.4.4.3 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.

**12.4.4.4 verbose()** template<typename... MsgTs>  
void anonymous\_namespace{loadrec.cpp}::verbose (   
MsgTs &&... *msg* ) [inline]

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

## Template Parameters

|                     |                            |
|---------------------|----------------------------|
| $Msg \leftarrow Ts$ | The message argument types |
|---------------------|----------------------------|

## Parameters

|                  |                       |
|------------------|-----------------------|
| <code>msg</code> | The message to output |
|------------------|-----------------------|

## 12.4.5 Variable Documentation

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

## 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.

12.4.5.2 PARAMETERS `const Parameter<OE> 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.

## 12.5 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

- template<typename... MsgTs>  
void `verbose` (MsgTs &&... msg)  
*Outputs the given printf style 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_rcv` (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.
- const char \*const [USAGE](#) = "[ -hvf] [-abn mode] [-mM freq] [-FAB freq:freq] [-H temp:temp] [-p ival] [-s cnt] [-P file]"  
The short usage string.
- const [Parameter](#)< [OE](#) > [PARAMETERS](#) []  
Definitions of command line parameters.

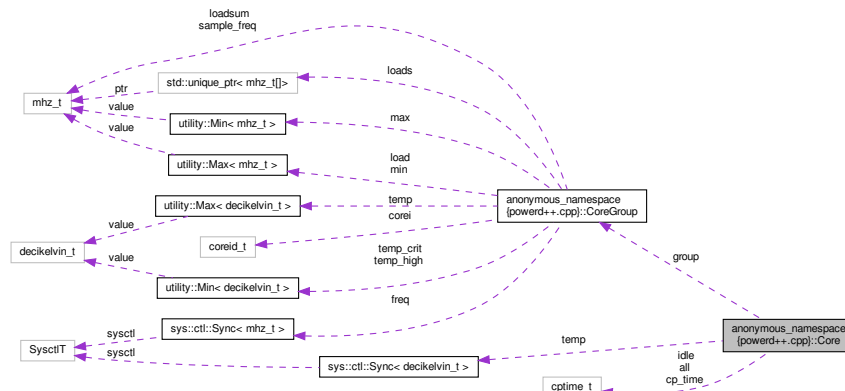
### 12.5.1 Detailed Description

File local scope.

### 12.5.2 Class Documentation

**12.5.2.1 struct anonymous\_namespace{powerd++.cpp}::Core** Contains the management information for a single CPU core.

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



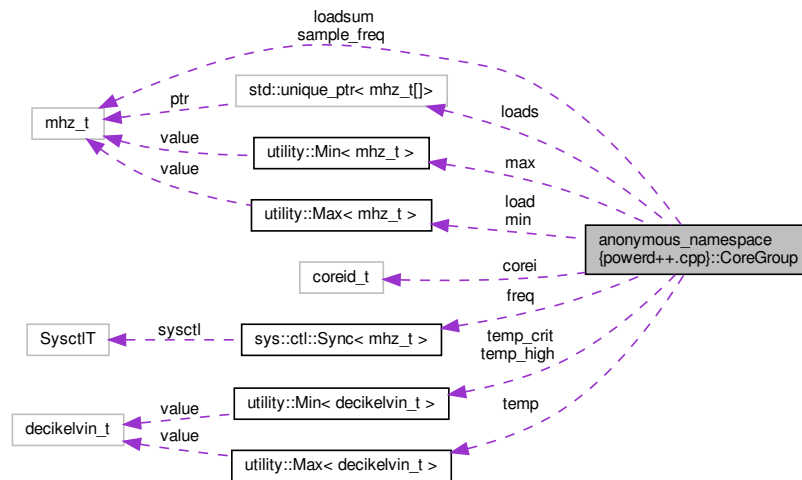
### Class Members

|                             |         |                                                       |
|-----------------------------|---------|-------------------------------------------------------|
| cptime_t                    | all     | Count of all ticks.                                   |
| const cptime_t *            | cp_time | A pointer to the kern.cp_times section for this core. |
| <a href="#">CoreGroup</a> * | 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.        |

**12.5.2.2 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:

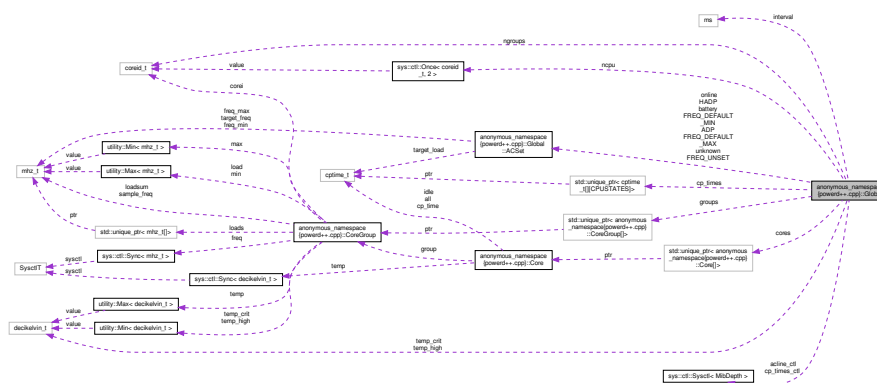


#### Class Members

|                                           |             |                                                                                                                                                                                                    |
|-------------------------------------------|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| coreid_t                                  | corei       | The number of the core owning dev.cpu. d.freq.                                                                                                                                                     |
| SysctlSync< mhz_t >                       | freq        | The sysctl dev.cpu. d.freq.                                                                                                                                                                        |
| <a href="#">Max&lt; mhz_t &gt;</a>        | load        | The maximum load reported by all cores in the group. This is updated by <a href="#">update_loads()</a> .                                                                                           |
| 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 <a href="#">update_loads()</a> . |
| mhz_t                                     | loadsum     | The maximum load sum of all controlled cores. This is updated by <a href="#">update_loads()</a> .                                                                                                  |
| <a href="#">Min&lt; mhz_t &gt;</a>        | max         | The maximum group clock rate. The least of all core maxima in the group.                                                                                                                           |
| <a href="#">Max&lt; mhz_t &gt;</a>        | 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 <a href="#">update_loads()</a> .                                                                                         |
| <a href="#">Max&lt; decikelvin_t &gt;</a> | temp        | The maximum temperature measurement taken in the group.                                                                                                                                            |
| <a href="#">Min&lt; decikelvin_t &gt;</a> | temp_crit   | Critical core temperature in dK.                                                                                                                                                                   |
| <a href="#">Min&lt; decikelvin_t &gt;</a> | temp_high   | High core temperature in dK.                                                                                                                                                                       |

**12.5.2.3 struct anonymous\_namespace{powerd++.cpp}::Global** A collection of all the gloabl, mutable states.

This is mostly for semantic clarity.



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

## 12.5.3 Enumeration Type Documentation

**12.5.3.1 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.             |

**12.5.3.2 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.      |
| FREQ_MIN_BATT | Set minimum clock frequency on battery power. |
| FREQ_MAX_BATT | Set maximum clock frequency on battery power. |

## Enumerator

|                 |                                             |
|-----------------|---------------------------------------------|
| 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.                                 |

## 12.5.4 Function Documentation

**12.5.4.1 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

**12.5.4.2 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.

**12.5.4.3 read\_args()** void anonymous\_namespace{powerd++.cpp}::read\_args (   
int const argc,   
char const \*const argv[] )

Parse command line arguments.

## Parameters

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

**12.5.4.4 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" | "hiadaptive" | "hadp";
mode = mode_predefined | load | freq;
```

Scalar values are treated as loads.

The predefined values have the following meaning:

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

## Parameters

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

**12.5.4.5 signal\_recv()** void anonymous\_namespace{powerd++.cpp}::signal\_recv (   
     int *signal* )

Sets g.signal, terminating the main loop.

## Parameters

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

**12.5.4.6 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

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

**12.5.4.7 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

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

Parameters

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

**12.5.4.8 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

|                    |                                                                  |
|--------------------|------------------------------------------------------------------|
| <i>Load</i>        | Determines whether <a href="#">CoreGroup::loadsum</a> is updated |
| <i>Temperature</i> | Determines whether <a href="#">CoreGroup::temp</a> is updated    |

**12.5.4.9 verbose()** `template<typename... MsgTs>`

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

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

Template Parameters

|              |                            |
|--------------|----------------------------|
| <i>MsgTs</i> | The message argument types |
|--------------|----------------------------|

Parameters

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

## 12.5.5 Variable Documentation

**12.5.5.1 PARAMETERS** `const Parameter<OE> 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.

## 12.6 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 °C 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.*

### 12.6.1 Detailed Description

A collection of functions to process command line arguments.

### 12.6.2 Function Documentation

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

Converts dK into °C for display purposes.

Parameters

|                  |                     |
|------------------|---------------------|
| <code>val</code> | A temperature in dK |
|------------------|---------------------|

Returns

The temperature in °C

**12.6.2.2 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

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

## Returns

The frequency given by *str*

**12.6.2.3 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

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

## Returns

The interval in milliseconds

**12.6.2.4 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

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

Return values

|                 |                                     |
|-----------------|-------------------------------------|
| <i>[0,1024]</i> | The load given by str               |
| >               | 1024 The given string is not a load |

#### 12.6.2.5 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

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

Parameters

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

Returns

A pair with the from and to values

#### 12.6.2.6 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

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

Returns

The number of samples

**12.6.2.7 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 °C is assumed.

Parameters

|                  |                              |
|------------------|------------------------------|
| <code>str</code> | A string encoded temperature |
|------------------|------------------------------|

Returns

The temperature given by `str`

## 12.7 constants Namespace Reference

A collection of constants.

### Variables

- `const char *const CP_TIMES = "kern.cp_times"`  
*The MIB name for per-CPU time statistics.*
- `const char *const ACLINE = "hw.acpi.acline"`  
*The MIB name for the AC line state.*
- `const char *const FREQ = "dev.cpu.%d.freq"`  
*The MIB name for CPU frequencies.*
- `const char *const FREQ_LEVELS = "dev.cpu.%d.freq_levels"`  
*The MIB name for CPU frequency levels.*
- `const char *const TEMPERATURE = "dev.cpu.%d.temperature"`  
*The MIB name for CPU temperatures.*
- `const char *const TJMAX_SOURCES [ ]`  
*An array of maximum temperature sources.*
- `const types::mhz_t FREQ_DEFAULT_MAX {1000000}`  
*Default maximum clock frequency value.*
- `const types::mhz_t FREQ_DEFAULT_MIN {0}`  
*Default minimum clock frequency value.*
- `const types::mhz_t FREQ_UNSET {1000001}`  
*Clock frequency representing an uninitialised value.*
- `const char *const POWERD_PIDFILE = "/var/run/powerd.pid"`  
*The default pidfile name of powerd.*
- `const types::cptime_t ADP {512}`  
*The load target for adaptive mode, equals 50% load.*
- `const types::cptime_t HADP {384}`  
*The load target for hiadaptive mode, equals 37.5% load.*
- `const types::decikelvin_t HITEMP_OFFSET {100}`  
*The default temperautre offset between high and critical temperature.*

### 12.7.1 Detailed Description

A collection of constants.

### 12.7.2 Variable Documentation

#### 12.7.2.1 TJMAX\_SOURCES `const char* const constants::TJMAX_SOURCES[]`

**Initial value:**

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

An array of maximum temperature sources.

## 12.8 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.*

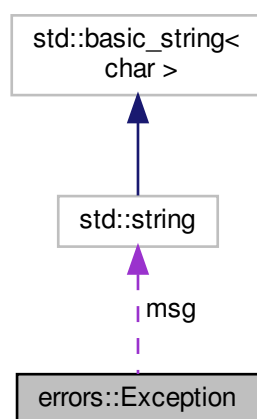
### 12.8.1 Detailed Description

Common error handling types and functions.

### 12.8.2 Class Documentation

**12.8.2.1 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. |
| <a href="#">Exit</a> | exitcode | The code to exit with.                   |
| string               | msg      | An error message.                        |

### 12.8.3 Enumeration Type Documentation

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

Exit codes.

Enumerator

|        |                                   |
|--------|-----------------------------------|
| OK     | Regular termination.              |
| ECLARG | Unexpected command line argument. |

## Enumerator

|              |                                                    |
|--------------|----------------------------------------------------|
| 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.                                       |

## 12.8.4 Function Documentation

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

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

## Parameters

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

## 12.8.5 Variable Documentation

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

#### Initial value:

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

Printable strings for exit codes.

## 12.9 nih Namespace Reference

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

### Classes

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

### 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.*

### 12.9.1 Detailed Description

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

### 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

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

## Class Members

|              |        |                                                                                             |
|--------------|--------|---------------------------------------------------------------------------------------------|
| const char * | args   | A comma separated list of arguments. Set to nullptr or "" if no argument is available.      |
| const char * | 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.           |
| const char * | usage  | A usage string.                                                                             |

## 12.9.3 Function Documentation

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

Retrieves the count of arguments in an option definition.

## Template Parameters

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

## Parameters

|            |                          |
|------------|--------------------------|
| <i>def</i> | The parameter definition |
|------------|--------------------------|

## Returns

The number of arguments specified in the given definition

**12.9.3.2 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] ) [constexpr]`

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

## Template Parameters

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

## Parameters

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

## 12.10 sys Namespace Reference

### Namespaces

- [ctl](#)

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

- [env](#)

*Provides wrappers around the [getenv\(\)](#) family of functions.*

- [io](#)

*This namespace contains c++ wrappers for [<stdio>](#) functionality.*

- [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.*

#### 12.10.1 Detailed Description

Wrappers around native system interfaces.

## 12.11 sys::ctl Namespace Reference

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

## Classes

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

## Typedefs

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

## Functions

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

### 12.11.1 Detailed Description

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

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

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

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

## 12.11.2 Class Documentation

**12.11.2.1 struct sys::ctl::error** The domain error type.

## 12.11.3 Typedef Documentation

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

A convenience alias around `Once`.

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

Template Parameters

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

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

A convenience alias around `Sync`.

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

Template Parameters

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

## 12.11.4 Function Documentation

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

This creates a `Once` instance.

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

## Template Parameters

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

## Parameters

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

**12.11.4.2 make\_Sysctl()** `template<typename... Args>  
constexpr Sysctl<sizeof...(Args)> sys::ctl::make_Sysctl (  
    Args const ... args ) [constexpr]`

Create a [Sysctl](#) instances.

This is only compatible with creating sysctls from predefined MIBs.

## Template Parameters

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

## Parameters

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

## Returns

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

**12.11.4.3 sysctl\_get()** `template<size_t MibDepth>  
void sys::ctl::sysctl_get (  
    mib\_t const (&) mib[MibDepth],  
    void *const oldp,  
    size_t & oldlen )`

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

## Template Parameters

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

## Parameters

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

## Exceptions

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

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

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

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

## Parameters

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

## Exceptions

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

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

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

## Template Parameters

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

## Parameters

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

## Exceptions

|                                         |                                                      |
|-----------------------------------------|------------------------------------------------------|
| <code>sys::sc_error&lt;error&gt;</code> | Throws if <code>sysctl()</code> fails for any reason |
|-----------------------------------------|------------------------------------------------------|

## 12.12 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 referring 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.*

#### 12.12.1 Detailed Description

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

#### 12.12.2 Class Documentation

**12.12.2.1 struct `sys::env::error`** The domain error type.

## 12.13 sys::io Namespace Reference

This namespace contains c++ wrappers for `<stdio>` functionality.

## Classes

- struct `contains`  
*Check whether a set type contains a value. [More...](#)*
- struct `contains< set< Set ... >, Value >`  
*Specialise `io::contains` to unpack `io::set`.*
- struct `enable_if`  
*Similar to `std::enable_if`, but it also has the value of the expression.*
- struct `enable_if< false, T >`  
*Specialise `enable_if` for a false expression.*
- class `file`  
*Produces file access types around the C file handling facilities. [More...](#)*
- class `file< link, Features ... >`  
*Specialise for FILE object linking file instances.*
- class `file< own, Features ... >`  
*Specialise for FILE object owning file instances.*
- class `file_feature`  
*Implements the base functionality of all file access types.*
- class `file_feature< FileT, read, Tail ... >`  
*Implement read support for file types.*
- class `file_feature< FileT, seek, Tail ... >`  
*Implement seek support for file types.*
- class `file_feature< FileT, write, Tail ... >`  
*Implement write support for file types.*
- struct `is_superset_of`  
*Check whether the left hand set is a superest of the right hand set. [More...](#)*
- struct `is_superset_of< LSetT, set< RSet ... > >`  
*Specialise `is_superset_of` to unpack the right hand `io::set`.*
- struct `query`  
*Ask questions about the contents of a string. [More...](#)*
- struct `set`  
*Pack a set of integral values in a type. [More...](#)*

## Typedefs

- template<class SetT, auto Value>  
using `contains_t` = typename `contains< SetT, Value >::type`  
*Check whether a set type contains a value.*
- template<class LSetT, class RSetT >  
using `is_superset_of_t` = typename `is_superset_of< LSetT, RSetT >::type`  
*Check whether the left hand set is a superest of the right hand set.*

## Enumerations

- enum `feature` { `feature::read`, `feature::write`, `feature::seek` }  
*Feature flags for file type composition.*
- enum `ownership` { `ownership::own`, `ownership::link` }  
*Ownership relation to the underlying FILE object.*



## Variables

- `template<class SetT, auto Value>`  
`constexpr const auto contains_v = contains<SetT, Value>::value`  
*Check whether a set type contains a value.*
- `template<class LSetT, class RSetT >`  
`constexpr const auto is_superset_of_v = is_superset_of<LSetT, RSetT>::value`  
*Check whether the left hand set is a superest of the right hand set.*
- `file< link, write > ferr {stderr}`  
*File access instances for stderr.*
- `file< link, write > fout {stdout}`  
*File access instances for stdout.*
- `file< link, read > fin {stdin}`  
*File access instances for stdin.*

### 12.13.1 Detailed Description

This namespace contains c++ wrappers for `<cstdio>` functionality.

### 12.13.2 Class Documentation

#### 12.13.2.1 struct sys::io::contains

```
template<class SetT, auto Value>
struct sys::io::contains< SetT, Value >
```

Check whether a set type contains a value.

Template Parameters

|              |                                                            |
|--------------|------------------------------------------------------------|
| <i>SetT</i>  | A set of integral values packed in <a href="#">io::set</a> |
| <i>Value</i> | The value to look up                                       |

#### 12.13.2.2 class sys::io::file

```
template<ownership Ownership, feature ... Features>
class sys::io::file< Ownership, Features >
```

Produces file access types around the C file handling facilities.

Template Parameters

|                  |                                                                    |
|------------------|--------------------------------------------------------------------|
| <i>Ownership</i> | Determine the ownership relationship to the underlying FILE object |
| <i>Features</i>  | A list of features the file type supports                          |

See also

[ownership](#)  
[feature](#)  
[file<own, Features ...>](#)  
[file<link, Features ...>](#)  
[file\\_feature](#)

### 12.13.2.3 struct sys::io::is\_superset\_of

```
template<class LSetT, class RSetT>
struct sys::io::is_superset_of< LSetT, RSetT >
```

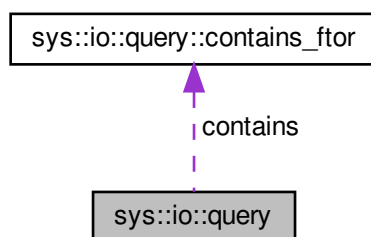
Check whether the left hand set is a superest of the right hand set.

Template Parameters

|                     |                                       |
|---------------------|---------------------------------------|
| <i>LSetT, RSetT</i> | Two <a href="#">io::set</a> instances |
|---------------------|---------------------------------------|

### 12.13.2.4 struct sys::io::query Ask questions about the contents of a string.

Collaboration diagram for sys::io::query:



Class Members

|                                      |          |                                  |
|--------------------------------------|----------|----------------------------------|
| struct <a href="#">contains_ftor</a> | contains | Query the string for characters. |
|--------------------------------------|----------|----------------------------------|

### 12.13.2.5 struct sys::io::set

```
template<auto ... Set>
struct sys::io::set< Set >
```

Pack a set of integral values in a type.

Template Parameters

|            |                          |
|------------|--------------------------|
| <i>Set</i> | A set of integral values |
|------------|--------------------------|

### 12.13.3 Typedef Documentation

**12.13.3.1 contains\_t** `template<class SetT , auto Value>`  
using `sys::io::contains_t` = typedef typename `contains<SetT, Value>::type`

Check whether a set type contains a value.

Template Parameters

|              |                                                         |
|--------------|---------------------------------------------------------|
| <i>SetT</i>  | A set of integral values packed in <code>io::set</code> |
| <i>Value</i> | The value to look up                                    |

**12.13.3.2 is\_superset\_of\_t** `template<class LSetT , class RSetT >`  
using `sys::io::is_superset_of_t` = typedef typename `is_superset_of<LSetT, RSetT>::type`

Check whether the left hand set is a superest of the right hand set.

Template Parameters

|                     |                                    |
|---------------------|------------------------------------|
| <i>LSetT, RSetT</i> | Two <code>io::set</code> instances |
|---------------------|------------------------------------|

### 12.13.4 Enumeration Type Documentation

**12.13.4.1 feature** `enum sys::io::feature` [strong]

Feature flags for file type composition.

See also

[file\\_feature](#)

## Enumerator

|       |                                                                                                                    |
|-------|--------------------------------------------------------------------------------------------------------------------|
| read  | The file type supports read operations.<br>See also<br><a href="#">file_feature&lt;FileT, read, Tail ...&gt;</a>   |
| write | The file type supports write operations.<br>See also<br><a href="#">file_feature&lt;FileT, write, Tail ...&gt;</a> |
| seek  | The file type supports seek operations.<br>See also<br><a href="#">file_feature&lt;FileT, seek, Tail ...&gt;</a>   |

**12.13.4.2 ownership** enum `sys::io::ownership` [strong]

Ownership relation to the underlying FILE object.

## Enumerator

|      |                                                                                                                                 |
|------|---------------------------------------------------------------------------------------------------------------------------------|
| own  | The file instance owns the FILE object.<br>See also<br><a href="#">file&lt;own, Features ...&gt;</a>                            |
| link | The file instance refers to a FILE object managed somewhere else.<br>See also<br><a href="#">file&lt;link, Features ...&gt;</a> |

**12.13.5 Variable Documentation****12.13.5.1 contains\_v** template<class SetT , auto Value>

```
constexpr const auto sys::io::contains_v = contains<SetT, Value>::value [constexpr]
```

Check whether a set type contains a value.

## Template Parameters

|              |                                                            |
|--------------|------------------------------------------------------------|
| <i>SetT</i>  | A set of integral values packed in <a href="#">io::set</a> |
| <i>Value</i> | The value to look up                                       |

**12.13.5.2 is\_superset\_of\_v** `template<class LSetT , class RSetT >`  
`constexpr const auto sys::io::is_superset_of_v = is_superset_of<LSetT, RSetT>::value` `[constexpr]`

Check whether the left hand set is a superest of the right hand set.

Template Parameters

|                     |                                       |
|---------------------|---------------------------------------|
| <i>LSetT, RSetT</i> | Two <a href="#">io::set</a> instances |
|---------------------|---------------------------------------|

## 12.14 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 RAI pattern.*

### 12.14.1 Detailed Description

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

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

### 12.14.2 Class Documentation

**12.14.2.1 struct sys::pid::error** The domain error type.

## 12.15 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.*

### 12.15.1 Detailed Description

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

### 12.15.2 Class Documentation

**12.15.2.1 struct sys::sig::error** The domain error type.

## 12.16 timing Namespace Reference

Namespace for time management related functionality.

## Classes

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

### 12.16.1 Detailed Description

Namespace for time management related functionality.

## 12.17 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.*

### 12.17.1 Detailed Description

A collection of type aliases.

### 12.17.2 Typedef Documentation

#### 12.17.2.1 `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.

## 12.18 utility Namespace Reference

A collection of generally useful functions.

### Namespaces

- [literals](#)

*Contains literal operators.*

### 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.*

### 12.18.1 Detailed Description

A collection of generally useful functions.

### 12.18.2 Function Documentation

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

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

Template Parameters

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

Returns

The number of array entries

**12.18.2.2 `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

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

**12.18.2.3 `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

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

## Parameters

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

## Returns

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

**12.18.2.4 to\_value()** `template<class ET , typename VT = typename std::underlying_type<ET>::type>  
constexpr VT utility::to_value (   
                    ET const op ) [constexpr]`

Casts an enum to its underlying value.

## Template Parameters

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

## Parameters

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

## Returns

The integer representation of the operand

## 12.19 utility::literals Namespace Reference

Contains literal operators.

### Functions

- constexpr [Formatter](#)< 16384 > [operator](#)""\_fmt (char const \*const fmt, size\_t const)  
*Literal to convert a string literal to a [Formatter](#) instance.*

### 12.19.1 Detailed Description

Contains literal operators.

## 12.19.2 Function Documentation

**12.19.2.1 operator""\_fmt()** constexpr [Formatter](#)<16384> utility::literals::operator""\_fmt (   
char const \*const *fmt*,   
size\_t const ) [constexpr]

Literal to convert a string literal to a [Formatter](#) instance.

Parameters

|            |                              |
|------------|------------------------------|
| <i>fmt</i> | A printf style format string |
|------------|------------------------------|

Returns

A [Formatter](#) instance

## 12.20 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

- const char \*const [LOADREC\\_FEATURES](#) = "usr.app.powerdxx.loadrec.features"

*The pseudo MIB name for the load recording feature flags.*

### 12.20.1 Detailed Description

Version information constants and types.

### 12.20.2 Enumeration Type Documentation

**12.20.2.1 LoadrecBits** enum [version::LoadrecBits](#) [strong]

Feature flags for load recordings.

Enumerator

|               |                                     |
|---------------|-------------------------------------|
| FREQ_TRACKING | Record clock frequencies per frame. |
|---------------|-------------------------------------|

## 12.21 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.

#### 12.21.1 Detailed Description

Literals to set flag bits.

#### 12.21.2 Function Documentation

**12.21.2.1 [operator""\\_FREQ\\_TRACKING\(\)](#)** constexpr [flag\\_t](#) version::literals::operator""\_FREQ\_TRACKING ( unsigned long long int *value* ) [constexpr]

Set the FREQ\_TRACKING bit.

Parameters

|              |               |
|--------------|---------------|
| <i>value</i> | The bit value |
|--------------|---------------|

Returns

The flag at the correct bit position

## 13 Class Documentation

### 13.1 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.*

### 13.1.1 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

|                                  |                                             |
|----------------------------------|---------------------------------------------|
| <i>Function</i> ↔<br><i>Args</i> | The argument types of the callback function |
|----------------------------------|---------------------------------------------|

### 13.1.2 Constructor & Destructor Documentation

**13.1.2.1 Callback()** [1/2] `template<typename... FunctionArgs>`  
`anonymous_namespace{libloadplay.cpp}::Callback< FunctionArgs >::Callback (`  
`function_t const & callback ) [inline]`

Construct from function.

Parameters

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

**13.1.2.2 Callback()** [2/2] `template<typename... FunctionArgs>`  
`anonymous_namespace{libloadplay.cpp}::Callback< FunctionArgs >::Callback (`  
`function_t && callback ) [inline]`

Construct from temporary function.

Parameters

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

### 13.1.3 Member Function Documentation

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

Forward call to callback functions.

Parameters

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

Exceptions

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

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

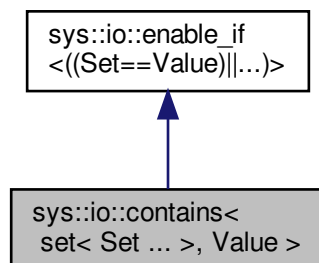
- [src/libloadplay.cpp](#)

## 13.2 sys::io::contains< set< Set ... >, Value > Struct Template Reference

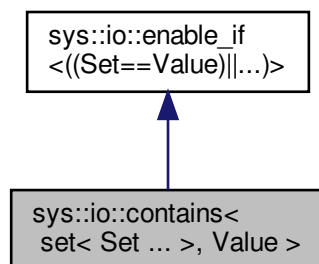
Specialise [io::contains](#) to unpack [io::set](#).

```
#include <io.hpp>
```

Inheritance diagram for `sys::io::contains< set< Set ... >, Value >`:



Collaboration diagram for `sys::io::contains< set< Set ... >, Value >`:



## Additional Inherited Members

### 13.2.1 Detailed Description

```
template<auto ... Set, auto Value>
struct sys::io::contains< set< Set ... >, Value >
```

Specialise `io::contains` to unpack `io::set`.

Template Parameters

|              |                                      |
|--------------|--------------------------------------|
| <i>Set</i>   | The set of integral values to search |
| <i>Value</i> | The value to find in Set             |

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

- [src/sys/io.hpp](#)

### 13.3 sys::io::query::contains\_ftor Struct Reference

Test a string whether it contains a set of characters.

```
#include <io.hpp>
```

#### Public Member Functions

- constexpr bool [operator\(\)](#) (char const ch) const  
*Check for a single character.*
- template<typename ... CharTs>  
constexpr bool [any](#) (CharTs const ... chars) const  
*Check for a set of characters if at least one is part of the string.*
- template<typename ... CharTs>  
constexpr bool [all](#) (CharTs const ... chars) const  
*Check for a set of characters if all of them are part of the string.*

#### Public Attributes

- const char \*const [str](#)  
*The string to ask about.*

#### 13.3.1 Detailed Description

Test a string whether it contains a set of characters.

#### 13.3.2 Member Function Documentation

**13.3.2.1 all()** template<typename ... CharTs>  
constexpr bool sys::io::query::contains\_ftor::all (  
CharTs const ... chars ) const [inline], [constexpr]

Check for a set of characters if all of them are part of the string.

Template Parameters

|               |                     |
|---------------|---------------------|
| <i>CharTs</i> | The character types |
|---------------|---------------------|

## Parameters

|              |                       |
|--------------|-----------------------|
| <i>chars</i> | The set of characters |
|--------------|-----------------------|

## Returns

Whether all of the given characters are in the string

```
13.3.2.2 any() template<typename ... CharTs>
constexpr bool sys::io::query::contains_ftor::any (
 CharTs const ... chars) const [inline], [constexpr]
```

Check for a set of characters if at least one is part of the string.

## Template Parameters

|               |                     |
|---------------|---------------------|
| <i>CharTs</i> | The character types |
|---------------|---------------------|

## Parameters

|              |                       |
|--------------|-----------------------|
| <i>chars</i> | The set of characters |
|--------------|-----------------------|

## Returns

Whether at least one of the given characters is in the string

```
13.3.2.3 operator()() constexpr bool sys::io::query::contains_ftor::operator() (
 char const ch) const [inline], [constexpr]
```

Check for a single character.

## Parameters

|           |                            |
|-----------|----------------------------|
| <i>ch</i> | The character to check for |
|-----------|----------------------------|

## Returns

Whether the given character is part of the string

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

- [src/sys/io.hpp](#)



## 13.4 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.*

#### 13.4.1 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.

### 13.4.2 Member Function Documentation

#### 13.4.2.1 `operator>()` [1/2] `bool timing::Cycle::operator() ( ) const` [inline]

Completes an interrupted sleep cycle.

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

Return values

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

#### 13.4.2.2 `operator>()` [2/2] `template<class... DurTraits>`

```

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

```

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

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

Template Parameters

|                        |                                 |
|------------------------|---------------------------------|
| <i>Dur↔<br/>Traits</i> | The traits of the duration type |
|------------------------|---------------------------------|

Parameters

|                        |                                       |
|------------------------|---------------------------------------|
| <i>cycle↔<br/>Time</i> | The duration of the cycle to complete |
|------------------------|---------------------------------------|



## Private Attributes

- `ifile` < io::link > `fin`  
The input data source.
- `ofile` < io::link > `fout`  
The output data sink.
- `const bool` & `die`  
A reference to a `bool` that tells the emulator to die.
- `const size_t size` = `sysctls[CP_TIMES].size()`  
The size of the `kern.cp_times` buffer.
- `const int 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.`

### 13.5.1 Detailed Description

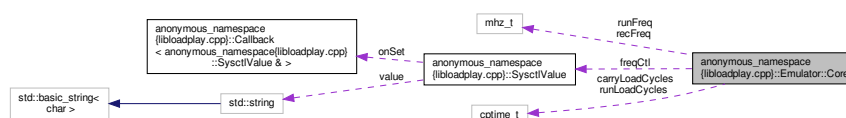
Instances of this class represent an emulator session.

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

### 13.5.2 Class Documentation

#### 13.5.2.1 struct anonymous\_namespace{libloadplay.cpp}::Emulator::Core Per core information.

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



### Class Members

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

### 13.5.3 Constructor & Destructor Documentation

**13.5.3.1 Emulator()** anonymous\_namespace{libloadplay.cpp}::Emulator::Emulator (   
     ifile< io::link > fin,   
     ofile< io::link > fout,   
     bool const & die ) [inline]

The constructor initialises all the members necessary for emulation.

It also prints the column headers on stdout.

Exceptions

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

Parameters

|                       |                                                                     |
|-----------------------|---------------------------------------------------------------------|
| <code>fin,fout</code> | The character input and output streams                              |
| <code>die</code>      | If the referenced bool is true, emulation is terminated prematurely |

### 13.5.4 Member Function Documentation

**13.5.4.1 operator>()** void anonymous\_namespace{libloadplay.cpp}::Emulator::operator() ( ) [inline]

Performs load emulation and prints statistics on io::fout.

Reads fin 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.

### 13.5.5 Member Data Documentation

**13.5.5.1 ncpu** const int 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](#)

## 13.6 sys::io::enable\_if< bool, T > Struct Template Reference

Similar to std::enable\_if, but it also has the value of the expression.

```
#include <io.hpp>
```

### Public Types

- using [type](#) = T  
*Provide the requested type.*

### Static Public Attributes

- static constexpr const bool [value](#) {true}  
*The given expression is true.*

#### 13.6.1 Detailed Description

```
template<bool, class T = void>
struct sys::io::enable_if< bool, T >
```

Similar to std::enable\_if, but it also has the value of the expression.

Template Parameters

|                   |                                           |
|-------------------|-------------------------------------------|
| <a href="#">T</a> | The return type if the expression is true |
|-------------------|-------------------------------------------|

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

- [src/sys/io.hpp](#)

## 13.7 sys::io::enable\_if< false, T > Struct Template Reference

Specialise [enable\\_if](#) for a false expression.

```
#include <io.hpp>
```

### Static Public Attributes

- static constexpr const bool [value](#) {false}  
*The given expression is false.*

#### 13.7.1 Detailed Description

```
template<class T>
struct sys::io::enable_if< false, T >
```

Specialise [enable\\_if](#) for a false expression.

Template Parameters

|          |                                            |
|----------|--------------------------------------------|
| <i>T</i> | The return type if the expression was true |
|----------|--------------------------------------------|

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

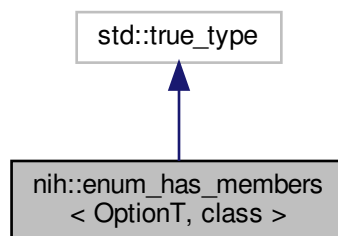
- [src/sys/io.hpp](#)

## 13.8 nih::enum\_has\_members< OptionT, class > Struct Template Reference

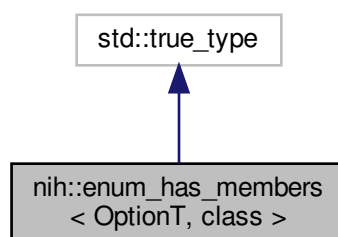
Tests whether the given enum provides all the required definitions.

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

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



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



### 13.8.1 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

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

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

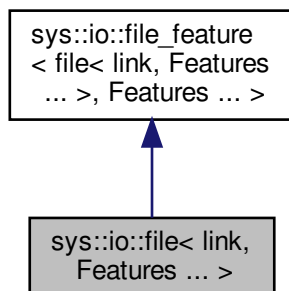
- [src/Options.hpp](#)

## 13.9 sys::io::file< link, Features ... > Class Template Reference

Specialise for FILE object linking file instances.

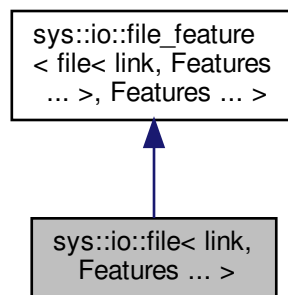
```
#include <io.hpp>
```

Inheritance diagram for sys::io::file< link, Features ... >:





Collaboration diagram for sys::io::file< link, Features ... >:



### Public Member Functions

- `file` (FILE \*const `handle`)  
*Use the given FILE object.*
- `template<ownership Ownership, feature ... Superset, class = is_superset_of_t<set<Superset ...>, set<Features ...>>>`  
`file` (`file`< Ownership, Superset ... > const &copy)  
*Copy construct from another file type instance.*
- `template<feature ... Superset, class = is_superset_of_t<set<Superset ...>, set<Features ...>>>`  
`file` (`file`< `own`, Superset ... > &&)=delete  
*Must not move construct from files with ownership of their handle.*
- `template<ownership Ownership, feature ... Superset, class = is_superset_of_t<set<Superset ...>, set<Features ...>>>`  
`file` & `operator=` (`file`< Ownership, Superset ... > const &copy)  
*Copy assign from another file type instance.*
- `template<feature ... Superset, class = is_superset_of_t<set<Superset ...>, set<Features ...>>>`  
`file` & `operator=` (`file`< `own`, Superset ... > &&)=delete  
*Must not move assign from files with ownership of their handle.*
- `FILE * get () const`  
*Provide the internal FILE object pointer.*

### Additional Inherited Members

#### 13.9.1 Detailed Description

```
template<feature ... Features>
class sys::io::file< link, Features ... >
```

Specialise for FILE object linking file instances.

Lack of ownership implies some semantics:

- Cannot be used to open files
- Can be copy constructed/assigned from other owning and non-owning file instances
- Cannot be move constructed/assigned from owning file instances
- Cannot close()

## Template Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>Features</i> | The set of file access features to support |
|-----------------|--------------------------------------------|

**13.9.2 Constructor & Destructor Documentation**

**13.9.2.1 file()** [1/3] `template<feature ... Features>`  
`sys::io::file< link, Features ... >::file (`  
`FILE *const handle ) [inline], [explicit]`

Use the given FILE object.

This can be used to refer to FILE objects managed by legacy C code.

## Parameters

|               |                            |
|---------------|----------------------------|
| <i>handle</i> | A pointer to a FILE object |
|---------------|----------------------------|

**13.9.2.2 file()** [2/3] `template<feature ... Features>`  
`template<ownership Ownership, feature ... Superset, class = is_superset_of_t<set<Superset ...>, set<←`  
`Features ...>>>`  
`sys::io::file< link, Features ... >::file (`  
`file< Ownership, Superset ... > const & copy ) [inline]`

Copy construct from another file type instance.

The origin file type instance must support all features supported by this file type.

## Template Parameters

|                  |                                                    |
|------------------|----------------------------------------------------|
| <i>Ownership</i> | The ownership status of the other file type        |
| <i>Superset</i>  | The feature set of another file type               |
| <i>Cond</i>      | Whether Superset is an actual superset of Features |

## Parameters

|             |                                                 |
|-------------|-------------------------------------------------|
| <i>copy</i> | The lvalue file to acquire the FILE object from |
|-------------|-------------------------------------------------|

**13.9.2.3 file()** [3/3] `template<feature ... Features>`  
`template<feature ... Superset, class = is_superset_of_t<set<Superset ...>, set<Features ...>>>`

```
sys::io::file< link, Features ... >::file (
 file< own, Superset ... > &&) [delete]
```

Must not move construct from files with ownership of their handle.

Template Parameters

|                 |                                                         |
|-----------------|---------------------------------------------------------|
| <i>Superset</i> | The feature set of another FILE object owning file type |
|-----------------|---------------------------------------------------------|

### 13.9.3 Member Function Documentation

#### 13.9.3.1 get() template<feature ... Features>

```
FILE* sys::io::file< link, Features ... >::get () const [inline]
```

Provide the internal FILE object pointer.

Can be used to pass the file to legacy C interfaces.

Returns

A pointer to the managed FILE object

#### 13.9.3.2 operator=() [1/2] template<feature ... Features>

```
template<feature ... Superset, class = is_superset_of_t<set<Superset ...>, set<Features ...>>>
file& sys::io::file< link, Features ... >::operator= (
 file< own, Superset ... > &&) [delete]
```

Must not move assign from files with ownership of their handle.

Template Parameters

|                 |                                                         |
|-----------------|---------------------------------------------------------|
| <i>Superset</i> | The feature set of another FILE object owning file type |
|-----------------|---------------------------------------------------------|

Returns

A self reference

#### 13.9.3.3 operator=() [2/2] template<feature ... Features>

```
template<ownership Ownership, feature ... Superset, class = is_superset_of_t<set<Superset ...>, set<←
Features ...>>>
file& sys::io::file< link, Features ... >::operator= (
 file< Ownership, Superset ... > const & copy) [inline]
```

Copy assign from another file type instance.

The origin file type instance must support all features supported by this file type.

Template Parameters

|                  |                                                    |
|------------------|----------------------------------------------------|
| <i>Ownership</i> | The ownership status of the other file type        |
| <i>Superset</i>  | The feature set of another file type               |
| <i>Cond</i>      | Whether Superset is an actual superset of Features |

Parameters

|             |                                                 |
|-------------|-------------------------------------------------|
| <i>copy</i> | The lvalue file to acquire the FILE object from |
|-------------|-------------------------------------------------|

Returns

A self reference

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

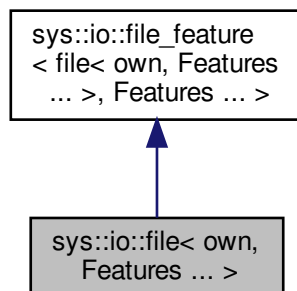
- [src/sys/io.hpp](#)

## 13.10 sys::io::file< own, Features ... > Class Template Reference

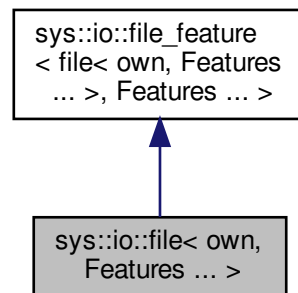
Specialise for FILE object owning file instances.

```
#include <io.hpp>
```

Inheritance diagram for sys::io::file< own, Features ... >:



Collaboration diagram for sys::io::file< own, Features ... >:



## Public Member Functions

- **file** (**file** const &)=delete  
*Must not copy construct for risk of multiple `close()` on the same file.*
- **file** (**file** &&move)  
*Move construct from a temporary.*
- **file** (FILE \*const **handle**)  
*Take ownership of the given FILE object.*
- template<feature ... Superset, class = is\_superset\_of\_t<set<Superset ...>, set<Features ...>>>  
**file** (**file**< own, Superset ... > &&move)  
*Move construct from another owning file type instance.*
- **file** (char const \*const filename, char const \*const mode)  
*Open a file by name.*
- **~file** ()  
*Free all resources.*
- template<feature ... Superset, class = is\_superset\_of\_t<set<Superset ...>, set<Features ...>>>  
**file** & **operator=** (**file**< own, Superset ... > &&move)  
*Move assign from another owning file type instance.*
- FILE \* **get** () const  
*Provide the internal FILE object pointer.*
- FILE \* **release** ()  
*Surrender ownership of the internal FILE object pointer.*
- **file** & **close** ()  
*Close the file.*

## Additional Inherited Members

### 13.10.1 Detailed Description

```
template<feature ... Features>
class sys::io::file< own, Features ... >
```

Specialise for FILE object owning file instances.

Ownership implies some semantics:

- Offers a constructor that opens a file
- Cannot be copy constructed/assigned
- Can be move constructed/assigned from other owning file instances
- Can `close()`
- Implicit `close()` when going out of scope

Template Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>Features</i> | The set of file access features to support |
|-----------------|--------------------------------------------|

## 13.10.2 Constructor & Destructor Documentation

**13.10.2.1 `file()`** [1/4] `template<feature ... Features>`  
`sys::io::file< own, Features ... >::file (`  
`file< own, Features ... > && move )` [inline]

Move construct from a temporary.

Parameters

|             |                                                 |
|-------------|-------------------------------------------------|
| <i>move</i> | The rvalue file to acquire the FILE object from |
|-------------|-------------------------------------------------|

**13.10.2.2 `file()`** [2/4] `template<feature ... Features>`  
`sys::io::file< own, Features ... >::file (`  
`FILE *const handle )` [inline], [explicit]

Take ownership of the given FILE object.

This can be used to take ownership of FILE objects provided by a legacy C interface.

Parameters

|               |                            |
|---------------|----------------------------|
| <i>handle</i> | A pointer to a FILE object |
|---------------|----------------------------|

**13.10.2.3 `file()`** [3/4] `template<feature ... Features>`  
`template<feature ... Superset, class = is_superset_of_t<set<Superset ...>, set<Features ...>>>`  
`sys::io::file< own, Features ... >::file (`  
`file< own, Superset ... > && move )` [inline]

Move construct from another owning file type instance.

The origin file type instance must support all features supported by this file type.

Template Parameters

|                 |                                                   |
|-----------------|---------------------------------------------------|
| <i>Superset</i> | The feature set of the original FILE object owner |
|-----------------|---------------------------------------------------|

Parameters

|             |                                                 |
|-------------|-------------------------------------------------|
| <i>move</i> | The rvalue file to acquire the FILE object from |
|-------------|-------------------------------------------------|

#### 13.10.2.4 `file()` [4/4] `template<feature ... Features>`

```
sys::io::file< own, Features ... >::file (
 char const *const filename,
 char const *const mode) [inline]
```

Open a file by name.

Failure to open a file occurs silently, but can be detected by boolean checking the file instance.

The arguments of this constructor are forwarded to `fopen()`, provided the mode argument does not contradict the feature set of this file type.

It is recommended to always add the 'b' (binary) character to the mode string, because text mode behaves quirkily.

The `feature::seek` feature is not supported with 'a' (append), it is available with 'a+', but it behaves quirkily. Which means two different files of the same type may have different seek behaviour, depending on how the file was opened. Refer to the `fopen()` spec for the unsettling details.

See also

`fopen()`

Parameters

|                 |                                                                             |
|-----------------|-----------------------------------------------------------------------------|
| <i>filename</i> | The name of the file                                                        |
| <i>mode</i>     | The file access mode, must not contradict the feature set of this file type |

### 13.10.3 Member Function Documentation

#### 13.10.3.1 `close()` `template<feature ... Features>`

```
file& sys::io::file< own, Features ... >::close () [inline]
```

Close the file.

Returns

A self reference

**13.10.3.2 get()** `template<feature ... Features>`  
`FILE* sys::io::file< own, Features ... >::get ( ) const [inline]`

Provide the internal FILE object pointer.

Can be used to pass the file to legacy C interfaces.

Returns

A pointer to the managed FILE object

**13.10.3.3 operator=()** `template<feature ... Features>`  
`template<feature ... Superset, class = is_superset_of_t<set<Superset ...>, set<Features ...>>>`  
`file& sys::io::file< own, Features ... >::operator= (`  
`file< own, Superset ... > && move ) [inline]`

Move assign from another owning file type instance.

The origin file type instance must support all features supported by this file type.

Template Parameters

|                 |                                                   |
|-----------------|---------------------------------------------------|
| <i>Superset</i> | The feature set of the original FILE object owner |
|-----------------|---------------------------------------------------|

Parameters

|             |                                                 |
|-------------|-------------------------------------------------|
| <i>move</i> | The rvalue file to acquire the FILE object from |
|-------------|-------------------------------------------------|

Returns

A self reference

**13.10.3.4 release()** `template<feature ... Features>`  
`FILE* sys::io::file< own, Features ... >::release ( ) [inline]`

Surrender ownership of the internal FILE object pointer.

Can be used to pass the file to legacy C interfaces.



Returns

A pointer to the managed FILE object

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

- [src/sys/io.hpp](#)

## 13.11 sys::io::file\_feature< FileT,... > Class Template Reference

Implements the base functionality of all file access types.

```
#include <io.hpp>
```

### Public Member Functions

- [operator bool](#) () const  
*Cast to boolean.*
- [bool eof](#) () const  
*Return whether the file instance is in EOF state.*
- [bool error](#) () const  
*Return whether the file instance is in an error state.*

### Protected Member Functions

- [operator FileT &](#) ()  
*Implicit cast up to inheriting file access type.*
- [file\\_feature](#) (FILE \*const [handle](#))  
*Construct from a FILE object pointer.*

### Protected Attributes

- FILE \* [handle](#)  
*A pointer to the underlying FILE object.*

#### 13.11.1 Detailed Description

```
template<class FileT, feature ...>
class sys::io::file_feature< FileT,... >
```

Implements the base functionality of all file access types.

Template Parameters

|              |                                             |
|--------------|---------------------------------------------|
| <i>FileT</i> | The file access type inheriting the feature |
|--------------|---------------------------------------------|

See also

[file\\_feature<FileT, read, Tail ...>](#)

[file\\_feature<FileT, write, Tail ...>](#)

[file\\_feature<FileT, seek, Tail ...>](#)

## 13.11.2 Constructor & Destructor Documentation

**13.11.2.1 file\_feature()** `template<class FileT, feature ... >`  
`sys::io::file_feature< FileT,... >::file_feature (`  
`FILE *const handle ) [inline], [protected]`

Construct from a FILE object pointer.

Parameters

|               |                                                       |
|---------------|-------------------------------------------------------|
| <i>handle</i> | A pointer to the object keeping file descriptor state |
|---------------|-------------------------------------------------------|

## 13.11.3 Member Function Documentation

**13.11.3.1 eof()** `template<class FileT, feature ... >`  
`bool sys::io::file_feature< FileT,... >::eof ( ) const [inline]`

Return whether the file instance is in EOF state.

See also

`feof()`

Returns

Whether the file instance points to a FILE object and is in EOF state

**13.11.3.2 error()** `template<class FileT, feature ... >`  
`bool sys::io::file_feature< FileT,... >::error ( ) const [inline]`

Return whether the file instance is in an error state.

See also

`ferror()`

Returns

Whether the file instance points to a FILE object and is in an error state

**13.11.3.3 operator bool()** `template<class FileT, feature ... >`  
`sys::io::file_feature< FileT,... >::operator bool ( ) const` `[inline], [explicit]`

Cast to boolean.

See also

`feof()`  
`ferror()`

Return values

|              |                                                                                                |
|--------------|------------------------------------------------------------------------------------------------|
| <i>true</i>  | The file instance point to a FILE object, which is not in EOF or error state                   |
| <i>false</i> | The file instance does not point to a FILE object, or the FILE object is in EOF or error state |

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

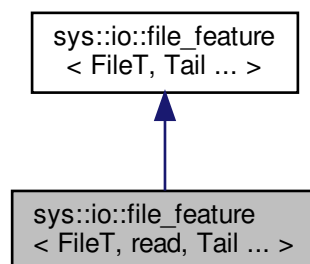
- [src/sys/io.hpp](#)

## 13.12 sys::io::file\_feature< FileT, read, Tail ... > Class Template Reference

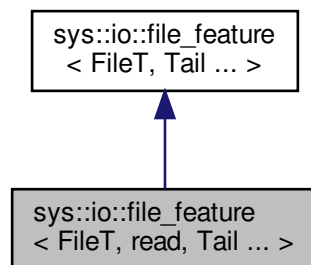
Implement read support for file types.

```
#include <io.hpp>
```

Inheritance diagram for sys::io::file\_feature< FileT, read, Tail ... >:



Collaboration diagram for `sys::io::file_feature< FileT, read, Tail ... >`:



## Public Member Functions

- `int getc ()`  
*Read a single character from the file.*
- `template<typename T >`  
`std::size_t read (T &dst)`  
*Read the given object from the file.*
- `template<typename T , auto Count>`  
`std::size_t read (T(&dst)[Count], std::size_t const count)`  
*Read the requested number of objects from the file.*
- `template<auto CountFmt, typename ... RefTs>`  
`int scanf (char const (&fmt)[CountFmt], RefTs &... refs)`  
*Read formatted input.*
- `template<auto Count>`  
`bool gets (char(&dst)[Count])`  
*Read a line from the file.*

## Additional Inherited Members

### 13.12.1 Detailed Description

```
template<class FileT, feature ... Tail>
class sys::io::file_feature< FileT, read, Tail ... >
```

Implement read support for file types.

Template Parameters

|              |                                             |
|--------------|---------------------------------------------|
| <i>FileT</i> | The file access type inheriting the feature |
| <i>Tail</i>  | The remaining features                      |

### 13.12.2 Member Function Documentation

**13.12.2.1 getc()** `template<class FileT , feature ... Tail>`  
`int sys::io::file_feature< FileT, read, Tail ... >::getc ( ) [inline]`

Read a single character from the file.

See also

`fgetc()`

Returns

The character or EOF

**13.12.2.2 gets()** `template<class FileT , feature ... Tail>`  
`template<auto Count>`  
`bool sys::io::file_feature< FileT, read, Tail ... >::gets (`  
`char(&) dst[Count] ) [inline]`

Read a line from the file.

Reads the file up to and including the first newline or terminating zero, as long as it fits into the destination buffer. Always zero terminated.

See also

`fgets()`

Template Parameters

|              |                                          |
|--------------|------------------------------------------|
| <i>Count</i> | The maximum number of characters to read |
|--------------|------------------------------------------|

Parameters

|            |                                       |
|------------|---------------------------------------|
| <i>dst</i> | A reference to the destination buffer |
|------------|---------------------------------------|

Return values

|              |                              |
|--------------|------------------------------|
| <i>true</i>  | Characters have been read    |
| <i>false</i> | Characters could not be read |

```
13.12.2.3 read() [1/2] template<class FileT , feature ... Tail>
template<typename T >
std::size_t sys::io::file_feature< FileT, read, Tail ... >::read (
 T & dst) [inline]
```

Read the given object from the file.

See also

`fread()`

Template Parameters

|          |                                       |
|----------|---------------------------------------|
| <i>T</i> | The object type, should be a POD type |
|----------|---------------------------------------|

Parameters

|            |                                        |
|------------|----------------------------------------|
| <i>dst</i> | A reference to the object to overwrite |
|------------|----------------------------------------|

Returns

The number of characters read

```
13.12.2.4 read() [2/2] template<class FileT , feature ... Tail>
template<typename T , auto Count>
std::size_t sys::io::file_feature< FileT, read, Tail ... >::read (
 T(&) dst[Count],
 std::size_t const count) [inline]
```

Read the requested number of objects from the file.

See also

`fread()`

Template Parameters

|              |                                                 |
|--------------|-------------------------------------------------|
| <i>T</i>     | The object type, should be a POD type           |
| <i>Count</i> | The number of objects in the destination buffer |

Parameters

|              |                                    |
|--------------|------------------------------------|
| <i>dst</i>   | A reference to an array of objects |
| <i>count</i> | The number of objects to read      |

Returns

The number of characters read

```
13.12.2.5 scanf() template<class FileT , feature ... Tail>
template<auto CountFmt, typename ... RefTs>
int sys::io::file_feature< FileT, read, Tail ... >::scanf (
 char const (&) fmt[CountFmt],
 RefTs &... refs) [inline]
```

Read formatted input.

See also

fscanf()

Template Parameters

|                 |                                               |
|-----------------|-----------------------------------------------|
| <i>CountFmt</i> | The number of characters in the format string |
| <i>RefTs</i>    | The argument types to read                    |

Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>fmt</i>  | The input format                |
| <i>refs</i> | A set of references to write to |

Returns

The number of inputs successfully parsed

Return values

|                                 |                                              |
|---------------------------------|----------------------------------------------|
| $E\leftrightarrow$<br><i>OF</i> | No inputs could be parsed due to end of file |
|---------------------------------|----------------------------------------------|

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

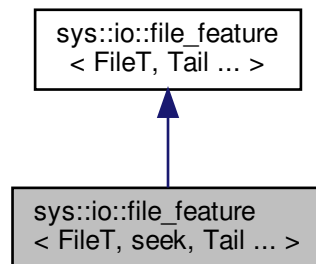
- src/sys/[io.hpp](#)

## 13.13 sys::io::file\_feature< FileT, seek, Tail ... > Class Template Reference

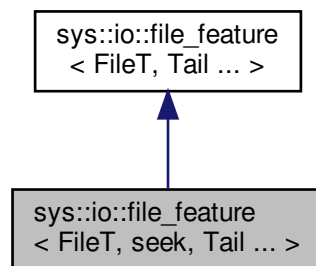
Implement seek support for file types.

```
#include <io.hpp>
```

Inheritance diagram for `sys::io::file_feature< FileT, seek, Tail ... >`:



Collaboration diagram for `sys::io::file_feature< FileT, seek, Tail ... >`:



## Public Member Functions

- `FileT` & `seek` (long int const offset, int const origin)  
*Seek file position.*
- `FileT` & `rewind` ()  
*Reset file position to the beginning of the file.*
- long int `tell` ()  
*Retrieve the current file position.*

## Additional Inherited Members

### 13.13.1 Detailed Description

```

template<class FileT, feature ... Tail>
class sys::io::file_feature< FileT, seek, Tail ... >

```

Implement seek support for file types.



## Template Parameters

|              |                                             |
|--------------|---------------------------------------------|
| <i>FileT</i> | The file access type inheriting the feature |
| <i>Tail</i>  | The remaining features                      |

## 13.13.2 Member Function Documentation

**13.13.2.1 rewind()** `template<class FileT , feature ... Tail>`  
`FileT& sys::io::file_feature< FileT, seek, Tail ... >::rewind ( ) [inline]`

Reset file position to the beginning of the file.

See also

`frewind()`

Returns

A self reference

**13.13.2.2 seek()** `template<class FileT , feature ... Tail>`  
`FileT& sys::io::file_feature< FileT, seek, Tail ... >::seek (`  
    `long int const offset,`  
    `int const origin ) [inline]`

Seek file position.

See also

`fseek()`

Parameters

|               |                                                                                                                           |
|---------------|---------------------------------------------------------------------------------------------------------------------------|
| <i>offset</i> | The origin relative file position for binary files or an absolute position returned by <code>tell()</code> for text files |
| <i>origin</i> | One of SEEK_SET, SEEK_CUR, SEEK_END                                                                                       |

Returns

A self reference

**13.13.2.3 tell()** `template<class FileT , feature ... Tail>`  
`long int sys::io::file_feature< FileT, seek, Tail ... >::tell ( ) [inline]`

Retrieve the current file position.

See also

`ftell()`

Returns

The current file offset

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

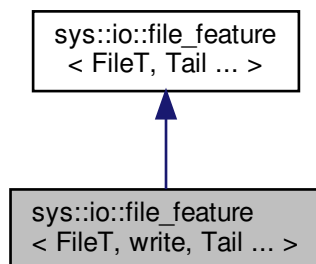
- [src/sys/io.hpp](#)

## 13.14 sys::io::file\_feature< FileT, write, Tail ... > Class Template Reference

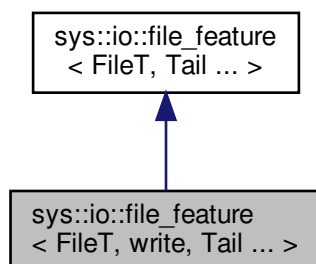
Implement write support for file types.

`#include <io.hpp>`

Inheritance diagram for `sys::io::file_feature< FileT, write, Tail ... >`:



Collaboration diagram for `sys::io::file_feature< FileT, write, Tail ... >`:



## Public Member Functions

- template<auto CountFmt, typename ... ArgTs>  
FileT & **printf** (char const (&fmt)[CountFmt], ArgTs const &... args)  
*Output with printf style formatting.*
- template<auto CountFmt>  
FileT & **printf** (char const (&fmt)[CountFmt])  
*Output a printf style formatted string.*
- template<auto Count>  
FileT & **print** (char const (&msg)[Count])  
*Print an unformatted string, excluding the last character.*
- FileT & **putc** (int const character)  
*Write a single character to the string.*
- template<typename T >  
FileT & **write** (T const &src)  
*Write an object to file.*
- template<typename T , auto Count>  
FileT & **write** (T const (&src)[Count], std::size\_t const count)  
*Write an objects to file.*
- FileT & **flush** ()  
*Flush file buffers.*

## Additional Inherited Members

### 13.14.1 Detailed Description

```
template<class FileT, feature ... Tail>
class sys::io::file_feature< FileT, write, Tail ... >
```

Implement write support for file types.

Template Parameters

|              |                                             |
|--------------|---------------------------------------------|
| <i>FileT</i> | The file access type inheriting the feature |
| <i>Tail</i>  | The remaining features                      |

### 13.14.2 Member Function Documentation

**13.14.2.1 flush()** template<class FileT , feature ... Tail>  
FileT& **sys::io::file\_feature**< FileT, **write**, Tail ... >::flush ( ) [inline]

Flush file buffers.

See also

fflush()

Returns

A self reference

```
13.14.2.2 print() template<class FileT , feature ... Tail>
template<auto Count>
FileT& sys::io::file_feature< FileT, write, Tail ... >::print (
 char const (&) msg[Count]) [inline]
```

Print an unformatted string, excluding the last character.

This method is built around the assumption that the argument is a string literal and the last character is a terminating zero.

See also

fwrite()

Template Parameters

|              |                                        |
|--------------|----------------------------------------|
| <i>Count</i> | The number of characters in the string |
|--------------|----------------------------------------|

Parameters

|            |                     |
|------------|---------------------|
| <i>msg</i> | The string to print |
|------------|---------------------|

Returns

A self reference

```
13.14.2.3 printf() [1/2] template<class FileT , feature ... Tail>
template<auto CountFmt>
FileT& sys::io::file_feature< FileT, write, Tail ... >::printf (
 char const (&) fmt[CountFmt]) [inline]
```

Output a printf style formatted string.

This overload exists as a workaround for a bug in clang++-8's -Wformat-security that does not recognise the format as a literal string if no arguments follow.

See also

fprintf()

## Template Parameters

|                 |                                                   |
|-----------------|---------------------------------------------------|
| <i>CountFmt</i> | The number of characters in the formatting string |
|-----------------|---------------------------------------------------|

## Parameters

|            |                   |
|------------|-------------------|
| <i>fmt</i> | The format string |
|------------|-------------------|

## Returns

A self reference

```
13.14.2.4 printf() [2/2] template<class FileT , feature ... Tail>
template<auto CountFmt, typename ... ArgTs>
FileT& sys::io::file_feature< FileT, write, Tail ... >::printf (
 char const (&) fmt[CountFmt],
 ArgTs const &... args) [inline]
```

Output with printf style formatting.

See also

fprintf()

## Template Parameters

|                 |                                                   |
|-----------------|---------------------------------------------------|
| <i>CountFmt</i> | The number of characters in the formatting string |
| <i>ArgTs</i>    | The argument types of the data to print           |

## Parameters

|             |                          |
|-------------|--------------------------|
| <i>fmt</i>  | The format string        |
| <i>args</i> | The set of data to print |

## Returns

A self reference

```
13.14.2.5 putc() template<class FileT , feature ... Tail>
FileT& sys::io::file_feature< FileT, write, Tail ... >::putc (
 int const character) [inline]
```

Write a single character to the string.

See also

`fputc()`

Parameters

|                  |                        |
|------------------|------------------------|
| <i>character</i> | The character to write |
|------------------|------------------------|

Returns

A self reference

```
13.14.2.6 write() [1/2] template<class FileT , feature ... Tail>
template<typename T >
FileT& sys::io::file_feature< FileT, write, Tail ... >::write (
 T const & src) [inline]
```

Write an object to file.

See also

`fwrite()`

Template Parameters

|          |                                       |
|----------|---------------------------------------|
| <i>T</i> | The object type, should be a POD type |
|----------|---------------------------------------|

Parameters

|            |                                     |
|------------|-------------------------------------|
| <i>src</i> | The object to write out to the file |
|------------|-------------------------------------|

Returns

A self reference

```
13.14.2.7 write() [2/2] template<class FileT , feature ... Tail>
template<typename T , auto Count>
FileT& sys::io::file_feature< FileT, write, Tail ... >::write (
 T const (&) src[Count],
 std::size_t const count) [inline]
```

Write an objects to file.

See also

`fwrite()`

## Template Parameters

|              |                                            |
|--------------|--------------------------------------------|
| <i>T</i>     | The object type, should be a POD type      |
| <i>Count</i> | The number of objects in the source buffer |

## Parameters

|              |                                     |
|--------------|-------------------------------------|
| <i>src</i>   | The object to write out to the file |
| <i>count</i> | The number of objects to write      |

## Returns

A self reference

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

- [src/sys/io.hpp](#)

## 13.15 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

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

#### 13.15.1 Detailed Description

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

A formatting wrapper around string literals.

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

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

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

## Template Parameters

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

## 13.15.2 Member Function Documentation

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

Returns a formatted string.

## Template Parameters

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

## Parameters

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

## Returns

An std::string formatted according to fmt

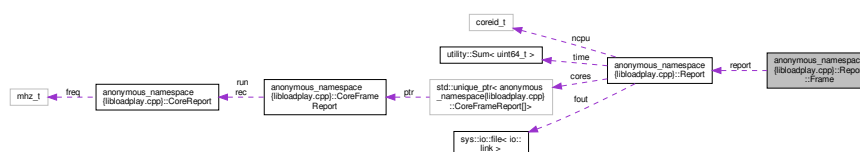
The documentation for this class was generated from the following file:

- [src/utility.hpp](#)

## 13.16 anonymous\_namespace{libloadplay.cpp}::Report::Frame Class Reference

Represents a frame of the report.

Collaboration diagram for anonymous\_namespace{libloadplay.cpp}::Report::Frame:





## 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.*
- const [CoreFrameReport](#) & [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.*

### 13.16.1 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.

### 13.16.2 Constructor & Destructor Documentation

**13.16.2.1 Frame()** anonymous\_namespace{libloadplay.cpp}::Report::Frame::Frame (  
    [Report](#) & *report*,  
    uint64\_t const *duration* ) [inline]

Construct a report frame.

Parameters

|                 |                                  |
|-----------------|----------------------------------|
| <i>report</i>   | The report this frame belongs to |
| <i>duration</i> | The frame duration               |

### 13.16.3 Member Function Documentation

**13.16.3.1 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>i</i> | The core index |
|----------|----------------|

## Returns

A reference to the core frame data

**13.16.3.2 operator[]()** [2/2] const [CoreFrameReport](#)& anonymous\_namespace{libloadplay.cpp}::Report::Frame←  
::operator[] (   
 coreid\_t const *i* ) const [inline]

Subscript operator for per core frame report data.

## Parameters

|          |                |
|----------|----------------|
| <i>i</i> | The core index |
|----------|----------------|

## Returns

A const reference to the core frame data

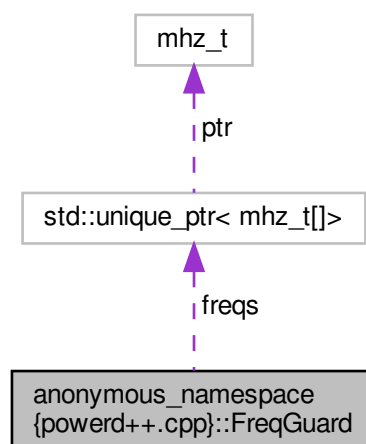
The documentation for this class was generated from the following file:

- [src/libloadplay.cpp](#)

## 13.17 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.*

#### 13.17.1 Detailed Description

A core frequency guard.

This uses the RAI 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](#)

## 13.18 `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

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

#### 13.18.1 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

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

### 13.18.2 Constructor & Destructor Documentation

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

The constructor sets the referenced variable to the given value.

Parameters

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

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

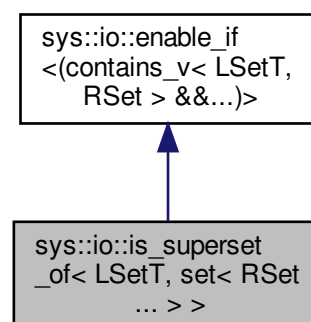
- [src/libloadplay.cpp](#)

## 13.19 sys::io::is\_superset\_of< LSetT, set< RSet ... > > Struct Template Reference

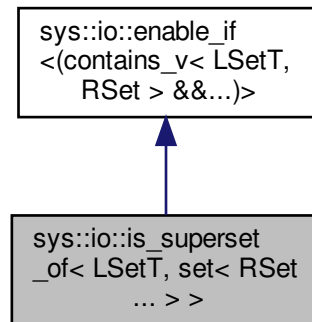
Specialise [is\\_superset\\_of](#) to unpack the right hand [io::set](#).

```
#include <io.hpp>
```

Inheritance diagram for `sys::io::is_superset_of< LSetT, set< RSet ... > >`:



Collaboration diagram for `sys::io::is_superset_of< LSetT, set< RSet ... > >`:



## Additional Inherited Members

### 13.19.1 Detailed Description

```
template<class LSetT, auto ... RSet>
struct sys::io::is_superset_of< LSetT, set< RSet ... > >
```

Specialise `is_superset_of` to unpack the right hand `io::set`.

Template Parameters

|              |                                            |
|--------------|--------------------------------------------|
| <i>LSetT</i> | The left hand <code>io::set</code>         |
| <i>RSet</i>  | The right hand <code>io::set</code> values |

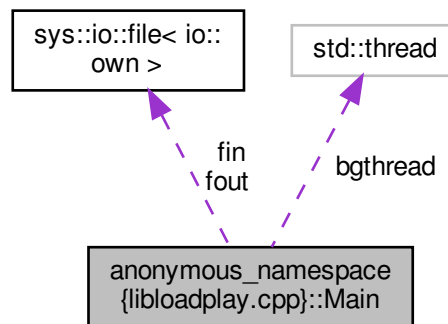
The documentation for this struct was generated from the following file:

- `src/sys/io.hpp`

## 13.20 anonymous\_namespace{libloadplay.cpp}::Main Class Reference

Singleton class representing the main execution environment.

Collaboration diagram for 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.*
- `ifile< io::own >` [fin](#) {`sys::env::vars`["LOADPLAY\_IN"], "rb"}  
*The optional input file.*
- `ofile< io::own >` [fout](#) {`sys::env::vars`["LOADPLAY\_OUT"], "wb"}  
*The optional output file.*
- `bool` [die](#) {false}  
*Used to request premature death from the emulation thread.*

### 13.20.1 Detailed Description

Singleton class representing the main execution environment.

### 13.20.2 Constructor & Destructor Documentation

### 13.20.2.1 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:

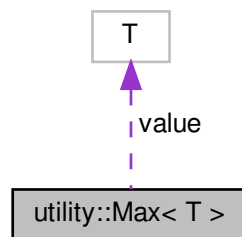
- [src/libloadplay.cpp](#)

## 13.21 `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.*

### 13.21.1 Detailed Description

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

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



Template Parameters

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

### 13.21.2 Constructor & Destructor Documentation

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

Construct from an initial value.

Parameters

|              |                   |
|--------------|-------------------|
| <i>value</i> | The initial value |
|--------------|-------------------|

### 13.21.3 Member Function Documentation

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

Returns the current maximum.

Returns

The maximum by const reference

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

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

Parameters

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

Returns

A self reference

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

- [src/utility.hpp](#)

## 13.22 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.*

#### 13.22.1 Detailed Description

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

#### 13.22.2 Constructor & Destructor Documentation

**13.22.2.1 mib\_t()** [1/2] `template<typename... Ints>`  
`constexpr anonymous_namespace{libloadplay.cpp}::mib_t::mib_t (`  
`Ints const ... ints ) [inline], [constexpr]`

Construct a mib with the given number of arguments.

## Template Parameters

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

## Parameters

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

**13.22.2.2 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

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

**13.22.3 Member Function Documentation**

**13.22.3.1 operator int \*()** anonymous\_namespace{libloadplay.cpp}::mib\_t::operator int \* ( ) [inline]

Cast to int \* for value access.

## Returns

A pointer to mibs

**13.22.3.2 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

**13.22.3.3 operator<()** bool anonymous\_namespace{libloadplay.cpp}::mib\_t::operator< (  
     mib\_t const & *op* ) const [inline]

Less than operator required by std::map.

## Parameters

|           |                                        |
|-----------|----------------------------------------|
| <i>op</i> | Another <a href="#">mib_t</a> instance |
|-----------|----------------------------------------|

## Returns

Whether this mib is less than the given one

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

Equality operator required by std::map.

## Parameters

|           |                                        |
|-----------|----------------------------------------|
| <i>op</i> | Another <a href="#">mib_t</a> instance |
|-----------|----------------------------------------|

## Returns

Whether all values in this and the given mib are equal

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

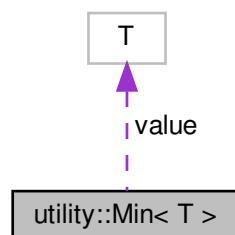
- [src/libloadplay.cpp](#)

## 13.23 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.*

### 13.23.1 Detailed Description

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

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

Template Parameters

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

### 13.23.2 Constructor & Destructor Documentation

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

Construct from an initial value.

Parameters

|              |                   |
|--------------|-------------------|
| <i>value</i> | The initial value |
|--------------|-------------------|

### 13.23.3 Member Function Documentation

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

Returns the current minimum.

Returns

The minimum by const reference

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

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

Parameters

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

Returns

A self reference

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

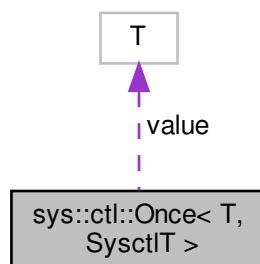
- [src/utility.hpp](#)

## 13.24 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.*

### 13.24.1 Detailed Description

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

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

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

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

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

Template Parameters

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

### 13.24.2 Constructor & Destructor Documentation

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

The constructor tries to read and store the requested sysctl.

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

Parameters

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

### 13.24.3 Member Function Documentation

**13.24.3.1 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](#)

## 13.25 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.*
- const char \* [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

- const [Parameter](#)< OptionT > & [get](#) (char const ch)  
*Finds the short option matching the given character.*
- const [Parameter](#)< OptionT > & [get](#) (char const \*const str)  
*Finds the long option matching the given string.*

### Static Private Member Functions

- static const char \* [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

- const int [argc](#)  
*The number of command line arguments.*
- const char \*const \*const [argv](#)  
*The command line arguments.*
- const char \*const [usageStr](#)  
*A string literal for the [usage\(\)](#) output.*
- const [Parameter](#)< OptionT >(& [defs](#))[DefCount]  
*A reference to the option definitions.*
- const [Parameter](#)< OptionT > [opt\\_unknown](#)  
*The option definition to use for unknown options.*
- const [Parameter](#)< OptionT > [opt\\_noopt](#)  
*The option definition to use for non-options.*
- const [Parameter](#)< OptionT > [opt\\_dash](#)  
*The option definition to use for a single dash.*
- const [Parameter](#)< OptionT > [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.*
- const char \* [argp](#)  
*Points to the current short option character.*
- const [Parameter](#)< OptionT > \* [current](#)  
*Points to the current option definition.*

#### 13.25.1 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

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

## 13.25.2 Constructor &amp; Destructor Documentation

**13.25.2.1 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

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

## 13.25.3 Member Function Documentation

**13.25.3.1 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

|                   |                          |
|-------------------|--------------------------|
| <i>str,prefix</i> | Two 0 terminated strings |
|-------------------|--------------------------|

## Return values

|              |                                           |
|--------------|-------------------------------------------|
| <i>true</i>  | The string starts with the prefix         |
| <i>false</i> | The string does not start with the prefix |

**13.25.3.2 get()** [1/2] `template<class OptionT , size_t DefCount>`  
`const Parameter<OptionT>& nih::Options< OptionT, DefCount >::get (`  
`char const *const str ) [inline], [private]`

Finds the long option matching the given string.

Parameters

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

Returns

An option definition by reference

**13.25.3.3 get()** [2/2] `template<class OptionT , size_t DefCount>`  
`const Parameter<OptionT>& nih::Options< OptionT, DefCount >::get (`  
`char const ch ) [inline], [private]`

Finds the short option matching the given character.

Parameters

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

Returns

An option definition by reference

**13.25.3.4 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

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

Return values

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

**13.25.3.5 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  $\geq$  argc if the current state is OptionT::OPT\_DONE

Returns

The current argument index

**13.25.3.6 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

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

**13.25.3.7 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

**13.25.3.8 operator[]()** `template<class OptionT , size_t DefCount>`  
`const char* 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>i</i> | The index of the argument to retrieve |
|----------|---------------------------------------|

Returns

The option or one of its arguments

**13.25.3.9 removePath()** `template<class OptionT , size_t DefCount>`  
`static const char* 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

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

Returns

A pointer to the file name portion of the path

**13.25.3.10 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>i</i> | The argument index, like operator []                                                   |
| <i>n</i> | The number of arguments to highlight, highlights all remaining arguments if $n \leq 0$ |

Returns

A string formatted to highlight the given argument

```
13.25.3.11 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

## 13.25.4 Member Data Documentation

```
13.25.4.1 opt_dash template<class OptionT , size_t DefCount>
const Parameter<OptionT> 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.

```
13.25.4.2 opt_ldash template<class OptionT , size_t DefCount>
const Parameter<OptionT> 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.

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

**Initial value:**

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

The option definition to use for non-options.

**13.25.4.4 opt\_unknown** `template<class OptionT , size_t DefCount>`  
`const Parameter<OptionT> 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](#)

## 13.26 sys::pid::Pidfile Class Reference

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

### 13.26.1 Detailed Description

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

### 13.26.2 Constructor & Destructor Documentation

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

Attempts to open the pidfile.

## Parameters

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

## Exceptions

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

## 13.26.3 Member Function Documentation

13.26.3.1 **write()** void sys::pid::Pidfile::write ( ) [inline]

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

## Exceptions

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

## 13.26.4 Member Data Documentation

13.26.4.1 **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\(\)](#).

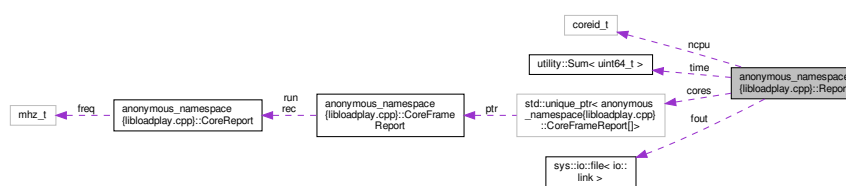
The documentation for this class was generated from the following file:

- [src/sys/pidfile.hpp](#)

## 13.27 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:





## Classes

- class [Frame](#)  
*Represents a frame of the report.*

## Public Member Functions

- [Report](#) ([ofile](#)< io::link > [fout](#), coreid\_t const [ncpu](#))  
*Construct a report.*
- template<typename ... ArgTs>  
[Frame frame](#) (ArgTs &&... args)  
*Constructs a frame for this report.*

## Private Attributes

- [ofile](#)< io::link > [fout](#)  
*The output stream to report to.*
- const coreid\_t [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.*

### 13.27.1 Detailed Description

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

### 13.27.2 Constructor & Destructor Documentation

**13.27.2.1 Report()** anonymous\_namespace{libloadplay.cpp}::Report::Report (  
    [ofile](#)< io::link > [fout](#),  
    coreid\_t const [ncpu](#) ) [inline]

Construct a report.

Parameters

|                      |                                   |
|----------------------|-----------------------------------|
| <a href="#">fout</a> | The stream to output to           |
| <a href="#">ncpu</a> | The number of CPU cores to report |

### 13.27.3 Member Function Documentation

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

Constructs a frame for this report.

Template Parameters

|         |                                |
|---------|--------------------------------|
| $ArgTs$ | The constructor argument types |
|---------|--------------------------------|

Parameters

|                   |                           |
|-------------------|---------------------------|
| <code>args</code> | The constructor arguments |
|-------------------|---------------------------|

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

- [src/libloadplay.cpp](#)

## 13.28 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.*
- `const char * c_str` () const  
*Return c style string.*

### Public Attributes

- `int error`  
*The errno set by the native C function.*

#### 13.28.1 Detailed Description

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

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

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

Template Parameters

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

## 13.28.2 Member Function Documentation

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

Return c style string.

Returns

A string representation of the error

**13.28.2.2 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](#)

## 13.29 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

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

### 13.29.1 Detailed Description

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

### 13.29.2 Constructor & Destructor Documentation

**13.29.2.1 Signal()** `sys::sig::Signal::Signal (`  
    `int const sig,`  
    `sig\_t const handler ) [inline]`

Sets up the given handler.

Parameters

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

Exceptions

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

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

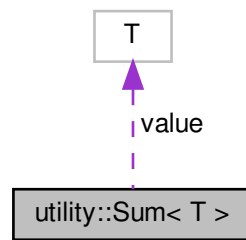
- src/sys/[signal.hpp](#)

## 13.30 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.*

#### 13.30.1 Detailed Description

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

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

Template Parameters

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

#### 13.30.2 Constructor & Destructor Documentation

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

Construct from an initial value.

Parameters

|              |                   |
|--------------|-------------------|
| <i>value</i> | The initial value |
|--------------|-------------------|

### 13.30.3 Member Function Documentation

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

Returns the current sum of values.

Returns

The sum of values by const reference

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

Add a value to the sum.

Parameters

|              |                                     |
|--------------|-------------------------------------|
| <i>value</i> | 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](#)

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

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

`#include <sysctl.hpp>`

## Public Member Functions

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

## Private Attributes

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

### 13.31.1 Detailed Description

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

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

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

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

Template Parameters

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

### 13.31.2 Constructor & Destructor Documentation

**13.31.2.1 Sync()** [1/2] `template<typename T, class SysctlT>`  
constexpr [sys::ctl::Sync](#)< T, SysctlT >::[Sync](#) ( ) [inline], [constexpr]

The default constructor.

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

**13.31.2.2 Sync()** [2/2] `template<typename T, class SysctlT>`  
constexpr [sys::ctl::Sync](#)< T, SysctlT >::[Sync](#) ( SysctlT const & *sysctl* ) [inline], [constexpr], [noexcept]

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

Parameters

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

### 13.31.3 Member Function Documentation

**13.31.3.1 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`

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

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

Parameters

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

Returns

A self reference

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

- `src/sys/sysctl.hpp`

## 13.32 `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 retrieved from the sysctl.*
- template<typename T >  
std::unique\_ptr< T[] > [get](#) () const  
*Retrieve an array from the sysctl address.*
- void [set](#) (void const \*const buf, size\_t const bufsize)  
*Update the the sysctl value with the given buffer.*
- template<typename T >  
void [set](#) (T const &value)  
*Update the the sysctl value with the given value.*

## Private Attributes

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

### 13.32.1 Detailed Description

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

Represents a sysctl MIB address.

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

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

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

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

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

Template Parameters

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

### 13.32.2 Constructor & Destructor Documentation

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

Initialise the MIB address directly.

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

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

Template Parameters

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

Parameters

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

### 13.32.3 Member Function Documentation

**13.32.3.1 get()** [1/3] `template<size_t MibDepth = 0>`  
`template<typename T >`  
`std::unique_ptr<T[]> sys::ctl::Sysctl< MibDepth >::get ( ) const [inline]`

Retrieve an array from the sysctl address.

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

Template Parameters

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

Returns

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

## Exceptions

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

**13.32.3.2 get()** [2/3] `template<size_t MibDepth = 0>`  
`template<typename T >`  
`void sys::ctl::Sysctl< MibDepth >::get (`  
`T & value ) const [inline]`

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

## Template Parameters

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

## Parameters

|                    |                                 |
|--------------------|---------------------------------|
| <code>value</code> | A reference to the target value |
|--------------------|---------------------------------|

## Exceptions

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

**13.32.3.3 get()** [3/3] `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

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

## Exceptions

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

**13.32.3.4 set()** [1/2] `template<size_t MibDepth = 0>`

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

Update the the sysctl value with the given value.

Template Parameters

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

Parameters

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

```
13.32.3.5 set() [2/2] template<size_t MibDepth = 0>
void sys::ctl::Sysctl< MibDepth >::set (
 void const *const buf,
 size_t const bufsize) [inline]
```

Update the the sysctl value with the given buffer.

Parameters

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

Exceptions

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

```
13.32.3.6 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](#)

## 13.33 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 retrieved from the sysctl.*
- template<typename T >  
std::unique\_ptr< T[ ]> [get](#) () const  
*Retrieve an array from the sysctl address.*
- void [set](#) (void const \*const buf, size\_t const bufsize)  
*Update the the sysctl value with the given buffer.*
- template<typename T >  
void [set](#) (T const &value)  
*Update the the sysctl value with the given value.*

### Private Attributes

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

#### 13.33.1 Detailed Description

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

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

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

#### 13.33.2 Constructor & Destructor Documentation

**13.33.2.1 Sysctl()** [1/2] constexpr `sys::ctl::Sysctl< 0 >::Sysctl ( )` [inline], [constexpr]

The default constructor.

This is available to defer initialisation to a later moment.

**13.33.2.2 Sysctl()** [2/2] `sys::ctl::Sysctl< 0 >::Sysctl ( char const *const name )` [inline]

Initialise the MIB address from a character string.

Parameters

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

Exceptions

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

### 13.33.3 Member Function Documentation

**13.33.3.1 get()** [1/3] template<typename T >  
`std::unique_ptr<T[]> sys::ctl::Sysctl< 0 >::get ( ) const` [inline]

Retrieve an array from the sysctl address.

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

Template Parameters

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

Returns

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

Exceptions

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

**13.33.3.2 get()** [2/3] template<typename T >

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

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

Template Parameters

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

Parameters

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

Exceptions

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

**13.33.3.3 get()** [3/3] void `sys::ctl::Sysctl< 0 >::get (`  
     void \*const *buf*,  
     size\_t const *bufsize* ) const [inline]

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

Parameters

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

Exceptions

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

**13.33.3.4 set()** [1/2] template<typename T >  
 void `sys::ctl::Sysctl< 0 >::set (`  
     T const & *value* ) [inline]

Update the the sysctl value with the given value.

Template Parameters

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

```
13.33.3.5 set() [2/2] void sys::ctl::Sysctl< 0 >::set (
 void const *const buf,
 size_t const bufsize) [inline]
```

## Parameters

## Exceptions

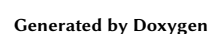
### 13.33.3.6 size() `size_t sys::ctl::Sysctl< 0 >::size ( ) const [inline]`

## Returns

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

- ### 13.34 anonymous\_namespace{libloadplay.cpp}::Sysctls Class Reference

Collaboration diagram for anonymous\_namespace{libloadplay.cpp}::Sysctl:





## 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.*
- const `mib_t` & `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 → mib.*
- `std::map< mib_t, SysctlValue >` `sysctls`  
*Maps mib → (type, value).*

### 13.34.1 Detailed Description

Singleton class representing the sysctl table for this library.

### 13.34.2 Member Function Documentation

**13.34.2.1 `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

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

**13.34.2.2 addValue()** [2/2] `void anonymous_namespace{libloadplay.cpp}::Sysctl::addValue (`  
    `std::string const & name,`  
    `std::string const & value ) [inline]`

Add a value to the sysctl map.

Parameters

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

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

Returns a mib for a given symbolic name.

Parameters

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

Returns

The MIB

**13.34.2.4 operator[]()** [1/2] `SysctlValue& anonymous_namespace{libloadplay.cpp}::Sysctl::operator[] (`  
    `char const *const name ) [inline]`

Returns a reference to a sysctl value container.

Parameters

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

Returns

A [SysctlValue](#) reference

**13.34.2.5 operator[]()** [2/2] `SysctlValue& anonymous_namespace{libloadplay.cpp}::Sysctl::operator[] (`  
    `mib_t const & mib ) [inline]`

Returns a reference to a sysctl value container.

## Parameters

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

## Returns

A [SysctlValue](#) reference

## 13.34.3 Member Data Documentation

**13.34.3.1 mibs** `std::unordered_map<std::string, mib\_t> anonymous_namespace{libloadplay.cpp}::Sysctl's↔`  
`::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.

**13.34.3.2 sysctl's** `std::map<mib\_t, SysctlValue> anonymous_namespace{libloadplay.cpp}::Sysctl's::sysctl's`  
`[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).

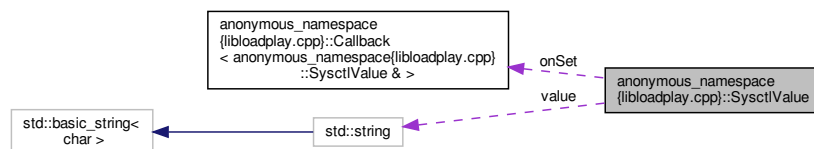
The documentation for this class was generated from the following file:

- [src/libloadplay.cpp](#)

### 13.35 anonymous\_namespace{libloadplay.cpp}::SysctlValue Class Reference

Instances of this class represents a specific sysctl value.

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



#### Public Member Functions

- [SysctlValue](#) ()  
*Default constructor.*
- [SysctlValue](#) ([SysctlValue](#) const &copy)  
*Copy constructor.*
- [SysctlValue](#) ([SysctlValue](#) &&move)  
*Move constructor.*
- [SysctlValue](#) (unsigned int [type](#), std::string const &[value](#), [callback\\_function](#) const callback=nullptr)  
*Construct from a type, value and optionally callback tuple.*
- [SysctlValue](#) & [operator=](#) ([SysctlValue](#) const &copy)  
*Copy assignment operator.*
- [SysctlValue](#) & [operator=](#) ([SysctlValue](#) &&move)  
*Move assignment operator.*
- size\_t [size](#) () const  
*Returns the required storage size according to the CTLTYPE.*
- template<typename T >  
int [get](#) (T \*dst, size\_t &[size](#)) const  
*Copy a list of values into the given buffer.*
- int [get](#) (char \*dst, size\_t &[size](#)) const  
*Copy a C string into the given buffer.*
- template<typename T >  
T [get](#) () const  
*Returns a single value.*
- int [get](#) (void \*dst, size\_t &[size](#)) const  
*Copy a list of values into the given buffer.*
- template<typename T >  
void [set](#) (T const \*const newp, size\_t newlen)  
*Set this value to the values in the given buffer.*
- int [set](#) (void const \*const newp, size\_t newlen)  
*Set this value to the values in the given buffer.*
- void [set](#) (std::string &&[value](#))  
*Move a string to the value.*
- void [set](#) (std::string const &[value](#))  
*Copy a string to the value.*

- template<typename T >  
void [set](#) (T const &[value](#))  
*Set the value.*
- void [registerOnSet](#) ([callback\\_function](#) &&callback)  
*Register a callback function.*
- void [registerOnSet](#) ([callback\\_function](#) const &callback)  
*Register a callback function.*

### Private Types

- typedef std::lock\_guard< decltype([mtx](#))> [lock\\_guard](#)  
*Lock guard type, fitting the mutex.*

### Private 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.*

## 13.35.1 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.

## 13.35.2 Constructor & Destructor Documentation

### 13.35.2.1 SysctlValue() [1/3] anonymous\_namespace{libloadplay.cpp}::SysctlValue::SysctlValue ( [SysctlValue](#) const & *copy* ) [inline]

Copy constructor.

## Parameters

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

**13.35.2.2 SysctlValue()** [2/3] `anonymous_namespace{libloadplay.cpp}::SysctlValue::SysctlValue (   
 SysctlValue && move ) [inline]`

Move constructor.

## Parameters

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

**13.35.2.3 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

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

### 13.35.3 Member Function Documentation

**13.35.3.1 get()** [1/4] `template<typename T >   
 T anonymous_namespace{libloadplay.cpp}::SysctlValue::get ( ) const [inline]`

Returns a single value.

## Template Parameters

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

## Returns

The value

**13.35.3.2 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

|                  |                                 |
|------------------|---------------------------------|
| <i>dst, size</i> | 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 |

**13.35.3.3 get()** [3/4] `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

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

Parameters

|                  |                                 |
|------------------|---------------------------------|
| <i>dst, size</i> | 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 |

**13.35.3.4 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

|                  |                                 |
|------------------|---------------------------------|
| <i>dst, size</i> | 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 |

**13.35.3.5 operator=()** [1/2] `SysctlValue& anonymous_namespace{libloadplay.cpp}::SysctlValue::operator= ( SysctlValue && move )` [inline]

Move assignment operator.

## Parameters

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

## Returns

A self reference

**13.35.3.6 operator=()** [2/2] `SysctlValue& anonymous_namespace{libloadplay.cpp}::SysctlValue::operator= ( SysctlValue const & copy )` [inline]

Copy assignment operator.

## Parameters

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

## Returns

A self reference

**13.35.3.7 registerOnSet()** [1/2] `void anonymous_namespace{libloadplay.cpp}::SysctlValue::registerOnSet ( callback_function && callback )` [inline]

Register a callback function.



Parameters

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

**13.35.3.8 registerOnSet()** [2/2] `void anonymous_namespace{libloadplay.cpp}::SysctlValue::registerOnSet (   
 callback_function const & callback ) [inline]`

Register a callback function.

Parameters

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

**13.35.3.9 set()** [1/5] `void anonymous_namespace{libloadplay.cpp}::SysctlValue::set (   
 std::string && value ) [inline]`

Move a string to the value.

Parameters

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

**13.35.3.10 set()** [2/5] `void anonymous_namespace{libloadplay.cpp}::SysctlValue::set (   
 std::string const & value ) [inline]`

Copy a string to the value.

Parameters

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

**13.35.3.11 set()** [3/5] `template<typename T >   
 void anonymous_namespace{libloadplay.cpp}::SysctlValue::set (   
 T const & value ) [inline]`

Set the value.

Template Parameters

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

## Parameters

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

```
13.35.3.12 set() [4/5] 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

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

## Parameters

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

```
13.35.3.13 set() [5/5] 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

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

```
13.35.3.14 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

|          |                                                  |
|----------|--------------------------------------------------|
| <i>T</i> | The type this value is supposed to be a array of |
|----------|--------------------------------------------------|

Returns

The size of the whole string of Ts

**13.35.3.15 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

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

#### 13.35.4 Member Data Documentation

**13.35.4.1 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.

**13.35.4.2 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](#)

## 13.36 sys::env::Var Class Reference

A reference type referring 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.*
- `const char * 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

- `const char *const name`  
*A pointer to the variable name.*

### 13.36.1 Detailed Description

A reference type referring to an environment variable.

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

### 13.36.2 Constructor & Destructor Documentation

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

Construct an environment variable reference.

Template Parameters

|             |                             |
|-------------|-----------------------------|
| <i>Size</i> | The size of the name buffer |
|-------------|-----------------------------|

Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>name</i> | The name of the environment variable |
|-------------|--------------------------------------|

### 13.36.3 Member Function Documentation

#### 13.36.3.1 c\_str() const char\* 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

|                |                             |
|----------------|-----------------------------|
| <i>nullptr</i> | The variable does not exist |
|----------------|-----------------------------|

#### 13.36.3.2 erase() Var& sys::env::Var::erase ( ) [inline]

Explicitly deletes the environment variable.

Returns

A self-reference

Exceptions

|                                      |                                                         |
|--------------------------------------|---------------------------------------------------------|
| <i>sc_error&lt;error&gt;{EINVAL}</i> | Invalid variable name                                   |
| <i>sc_error&lt;error&gt;{ENOMEM}</i> | Failed to allocate memory when updating the environment |

#### 13.36.3.3 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

|                |                             |
|----------------|-----------------------------|
| <i>nullptr</i> | The variable does not exist |
|----------------|-----------------------------|

**13.36.3.4 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

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

Returns

A self-reference

Exceptions

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

**13.36.3.5 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](#)

## 13.37 sys::env::Vars Struct Reference

A singleton class providing access to environment variables.

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

### Public Member Functions

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

### 13.37.1 Detailed Description

A singleton class providing access to environment variables.

### 13.37.2 Member Function Documentation

**13.37.2.1 operator[]()** [1/2] `template<typename T >`  
`Var sys::env::Vars::operator[] (`  
`T const & name ) [inline]`

Access environment variable by name.

Template Parameters

|          |                        |
|----------|------------------------|
| <i>T</i> | The name argument type |
|----------|------------------------|

Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>name</i> | The name of the variable by reference |
|-------------|---------------------------------------|

**13.37.2.2 operator[]()** [2/2] `template<typename T >`  
`const Var sys::env::Vars::operator[] (`  
`T const & name ) const [inline]`

Access environment variable by name.

Template Parameters

|          |                        |
|----------|------------------------|
| <i>T</i> | The name argument type |
|----------|------------------------|

Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>name</i> | The name of the variable by reference |
|-------------|---------------------------------------|

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

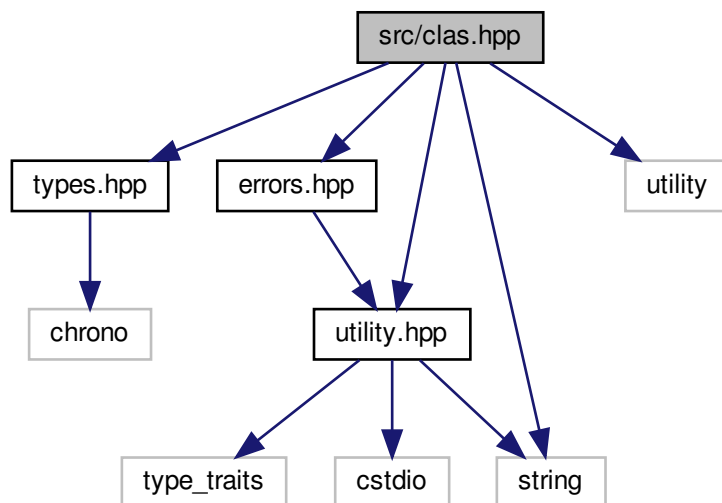
- `src/sys/env.hpp`

## 14 File Documentation

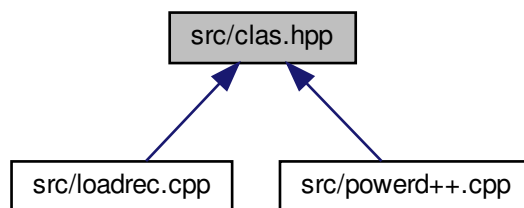
### 14.1 src/clas.hpp File Reference

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

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



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



## Namespaces

- [clas](#)

*A collection of functions to process command line arguments.*

## Functions

- [types::cptime\\_t clas::load](#) (char const \*const str)



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

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

*Convert string to frequency in MHz.*
- `types::ms clas::ival` (char const \*const str)

*Convert string to time interval in milliseconds.*
- `size_t clas::samples` (char const \*const str)

*A string encoded number of samples.*
- `types::decikelvin_t clas::temperature` (char const \*const str)

*Convert string to temperature in dK.*
- `int clas::celsius` (`types::decikelvin_t` const val)

*Converts dK into °C for display purposes.*
- `template<typename T > std::pair< T, T > clas::range` (T(&func)(char const \*const), char const \*const str)

*Takes a string encoded range of values and returns them.*

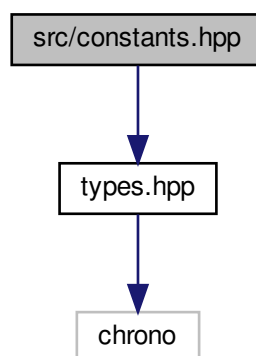
### 14.1.1 Detailed Description

Implements functions to process command line arguments.

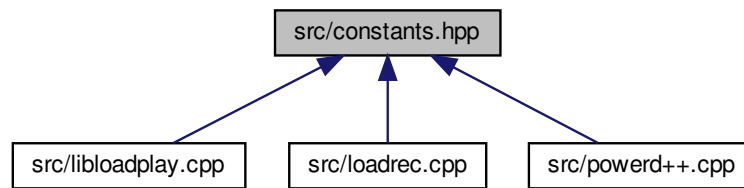
## 14.2 src/constants.hpp File Reference

```
#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

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

### 14.2.1 Detailed Description

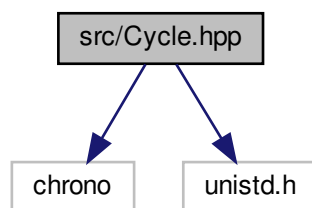
Defines a collection of constants.

## 14.3 src/Cycle.hpp File Reference

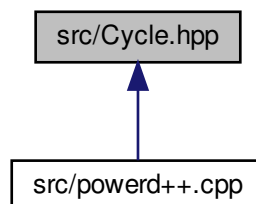
```
#include <chrono>
```

```
#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.*

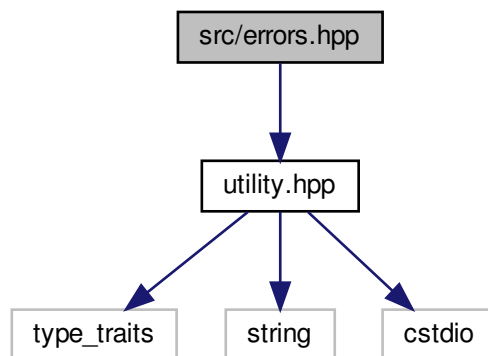
### 14.3.1 Detailed Description

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

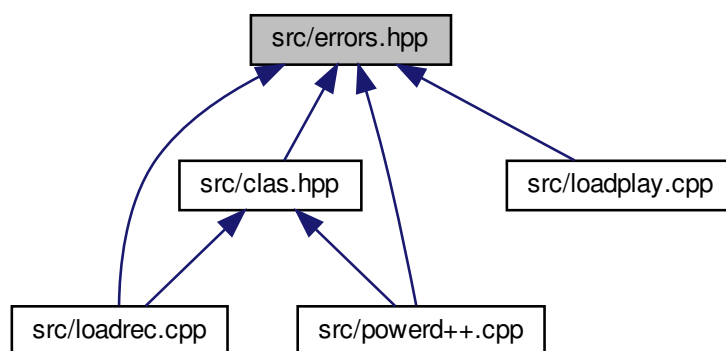
## 14.4 src/errors.hpp File Reference

```
#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.*

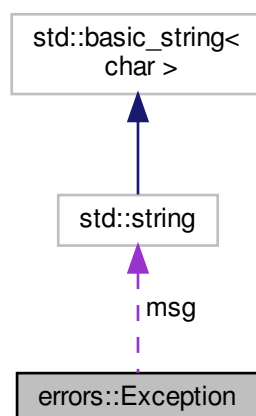
### 14.4.1 Detailed Description

Common error handling code.

### 14.4.2 Class Documentation

**14.4.2.1 struct errors::Exception** Exceptions bundle an exit code, errno value and message.

Collaboration diagram for errors::Exception:



## Class Members

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

## 14.5 src/libloadplay.cpp File Reference

```

#include "utility.hpp"
#include "constants.hpp"
#include "version.hpp"
#include "sys/env.hpp"
#include "sys/io.hpp"
#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.*

## Typedefs

- template<auto Ownership>  
using [anonymous\\_namespace{libloadplay.cpp}::ofile](#) = io::file< Ownership, io::write >  
*Output file type alias.*
- template<auto Ownership>  
using [anonymous\\_namespace{libloadplay.cpp}::ifile](#) = io::file< Ownership, io::read >  
*Input file type alias.*

## Functions

- template<size\_t Size>  
int [anonymous\\_namespace{libloadplay.cpp}::strcmp](#) (char const \*const s1, char const (&s2)[Size])  
*Safe wrapper around strcmp, which automatically determines the buffer size of s2.*
- std::regex [anonymous\\_namespace{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 io::ferr.printf(...) if built with -DEBUG.*
- template<>  
std::string [anonymous\\_namespace{libloadplay.cpp}::SysctlValue::get< std::string > \(\)](#) const  
*Returns a copy of the value string.*
- template<typename... MsgTs>  
ofile< io::link > [anonymous\\_namespace{libloadplay.cpp}::debug](#) (MsgTs &&... msg)  
*Print a debugging message if built with -DEBUG.*
- template<typename... MsgTs>  
ofile< io::link > [anonymous\\_namespace{libloadplay.cpp}::warn](#) (MsgTs &&... msg)  
*Print a warning.*

- `template<typename... MsgTs>`  
`ofile< io::link > anonymous_namespace{libloadplay.cpp}::fail (MsgTs &&... msg)`  
*This prints an error message and sets sys\_results to make the hijacked process fail.*
- `ofile< io::link > anonymous_namespace{libloadplay.cpp}::operator<< (ofile< io::link > fout, Core↵  
FrameReport 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 const flag_t 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.*

### 14.5.1 Detailed Description

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

This library reads instructions from io::fin (stdin) and outputs statistics about the hijacked process on io::fout (stdout).

The following environment variables affect the operation of loadplay:

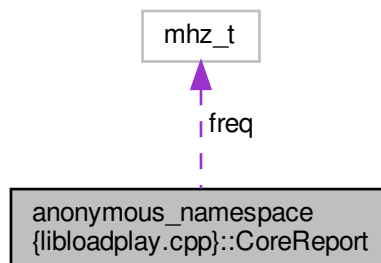


| Variable        | Description             |
|-----------------|-------------------------|
| LOADPLAY_IN     | Alternative input file  |
| LOADPLAY_OUTPUT | Alternative output file |

## 14.5.2 Class Documentation

**14.5.2.1 struct anonymous\_namespace{libloadplay.cpp}::CoreReport** The reported state of a single CPU pipeline.

Collaboration diagram for anonymous\_namespace{libloadplay.cpp}::CoreReport:

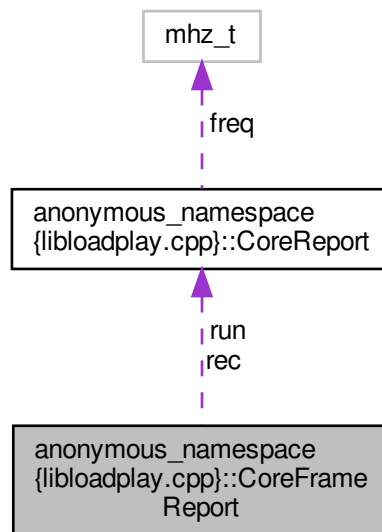


### Class Members

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

**14.5.2.2 struct anonymous\_namespace{libloadplay.cpp}::CoreFrameReport** The report frame information for a single CPU pipeline.

Collaboration diagram for anonymous\_namespace{libloadplay.cpp}::CoreFrameReport:

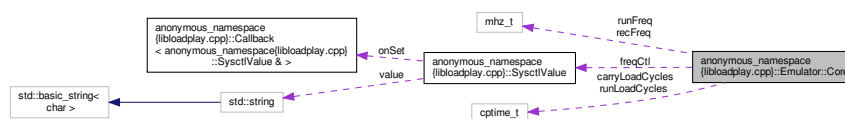


Class Members

|                            |     |                          |
|----------------------------|-----|--------------------------|
| <a href="#">CoreReport</a> | rec | The recorded core state. |
| <a href="#">CoreReport</a> | run | The running core state.  |

**14.5.2.3 struct anonymous\_namespace{libloadplay.cpp}::Emulator::Core** Per core information.

Collaboration diagram for 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. |
| <a href="#">SysctlValue</a> * | freqCtl         | The sysctl handler. The constructor ensures this points to a valid handler.                                                                                                              |
| mhz_t                         | recFreq         | The recorded clock frequency. If <code>FREQ_TRACKING</code> is enabled this is updated at during the preliminary stage and used at the beginning of frame stage.                         |

## Class Members

|          |               |                                                                                                                                                                  |
|----------|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| mhzt     | 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. |

## 14.5.3 Function Documentation

**14.5.3.1 daemon()** `int daemon (`  
`int ,`  
`int )`

Intercept calls to [daemon\(\)](#).

Prevents process from separating from the controlling terminal.

Returns

The value of `sys_results`

**14.5.3.2 geteuid()** `uid_t geteuid (`  
`void )`

Intercept calls to [geteuid\(\)](#).

Tells the asking process that it is running as root.

Returns

Always returns 0

**14.5.3.3 pidfile\_close()** `int pidfile_close (`  
`pidfh * )`

Intercept calls to [pidfile\\_close\(\)](#).

Returns

The value of `sys_results`

**14.5.3.4 pidfile\_fileno()** int pidfile\_fileno (  
pidfh const \* )

Intercept calls to [pidfile\\_fileno\(\)](#).

Returns

The value of sys\_results

**14.5.3.5 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

**14.5.3.6 pidfile\_remove()** int pidfile\_remove (  
pidfh \* )

Intercept calls to [pidfile\\_remove\(\)](#).

Returns

The value of sys\_results

**14.5.3.7 pidfile\_write()** int pidfile\_write (  
pidfh \* )

Intercept calls to [pidfile\\_write\(\)](#).

Returns

The value of sys\_results

```
14.5.3.8 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

|                                                   |                                           |
|---------------------------------------------------|-------------------------------------------|
| <i>name, namelen, oldp, oldlenp, newp, newlen</i> | Please refer to <a href="#">sysctl(3)</a> |
|---------------------------------------------------|-------------------------------------------|

Return values

|    |                    |
|----|--------------------|
| 0  | The call succeeded |
| -1 | The call failed    |

```
14.5.3.9 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

|                                      |                           |
|--------------------------------------|---------------------------|
| <i>name,oldp,oldlenp,newp,newlen</i> | Please refer to sysctl(3) |
|--------------------------------------|---------------------------|

Return values

|    |                    |
|----|--------------------|
| 0  | The call succeeded |
| -1 | The call failed    |

**14.5.3.10 sysctlnametomib()** int sysctlnametomib (  
    const char \* *name*,  
    int \* *mibp*,  
    size\_t \* *sizep* )

Intercept calls to [sysctlnametomib\(\)](#).

Parameters

|                        |                           |
|------------------------|---------------------------|
| <i>name,mibp,sizep</i> | Please refer to sysctl(3) |
|------------------------|---------------------------|

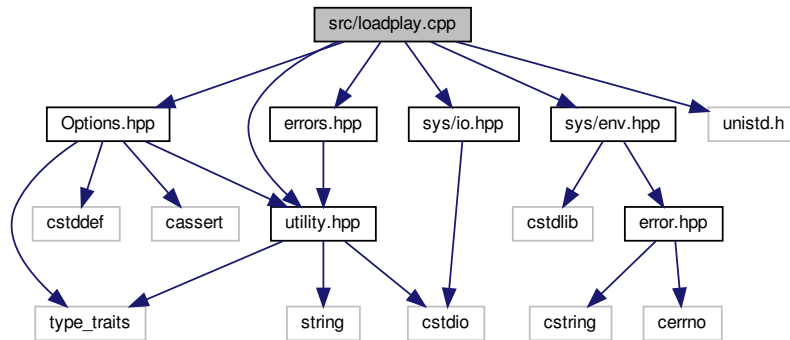
Return values

|    |                    |
|----|--------------------|
| 0  | The call succeeded |
| -1 | The call failed    |

## 14.6 src/loadplay.cpp File Reference

```
#include "Options.hpp"
#include "errors.hpp"
#include "utility.hpp"
#include "sys/env.hpp"
#include "sys/io.hpp"
#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_UNKN, 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

- `const char * 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

- `const char *const anonymous_namespace{loadplay.cpp}::USAGE = "[ -h ] [-i file] [-o file] command [...]"`  
*The short usage string.*
- `const Parameter< OE > anonymous_namespace{loadplay.cpp}::PARAMETERS [ ]`  
*Definitions of command line parameters.*





## Namespaces

- `anonymous_namespace{loadrec.cpp}`

*File local scope.*

## Typedefs

- `template<auto Ownership>`  
`using anonymous_namespace{loadrec.cpp}::ofile = io::file< Ownership, io::write >`

*Output file type alias.*

## 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

- `template<typename... MsgTs>`  
`void anonymous_namespace{loadrec.cpp}::verbose (MsgTs &&... msg)`  
*Outputs the given printf style 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 const flag_t 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.
 ofile< io::link > fout = io::fout
 The output stream either io::fout (stdout) or a file.
 const char * outfilename {nullptr}
 The user provided output file name.
 const sys::ctl::SysctlOnce< coreid_t, 2 > ncpu {1U, {CTL_HW, HW_NCPU}}
 The number of CPU cores/threads.
} anonymous_namespace{loadrec.cpp}:g

The global state.
• const char *const anonymous_namespace{loadrec.cpp}::USAGE = "[-hv] [-d ival] [-p ival] [-o file]"
 The short usage string.
• const Parameter< OE > anonymous_namespace{loadrec.cpp}::PARAMETERS []
 Definitions of command line parameters.

```

### 14.7.1 Detailed Description

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

### 14.7.2 Function Documentation

**14.7.2.1 main()** int main (
 int argc,
 char \* argv[] )

Main routine, setup and execute daemon, print errors.

Parameters

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

Returns

An exit code

See also

Exit

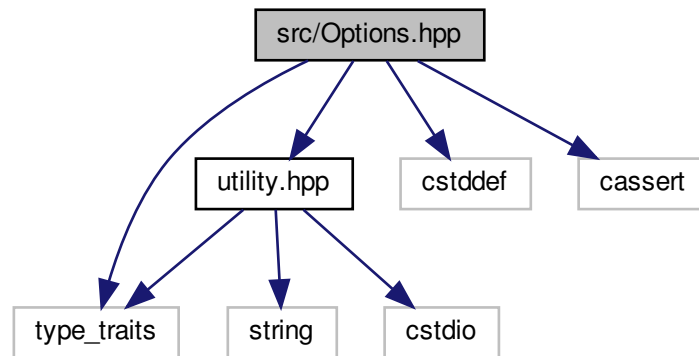
## 14.8 src/Options.hpp File Reference

```

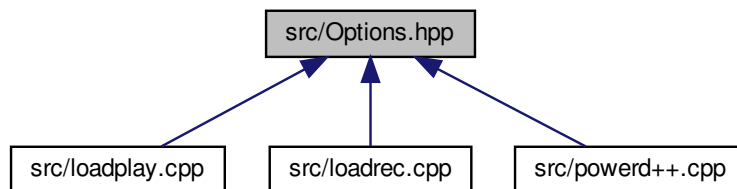
#include "utility.hpp"
#include <cstdint>

```

```
#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.

## 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.*

### 14.8.1 Detailed Description

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

The `getopt(3)` interface takes the command line arguments as `char * const` instead of `char const *`. I.e. it reserves the right to mutate the provided arguments, which it actually does.

The `nih::Options<>` functor is not a drop in substitute, but tries to be easily adoptable and does not change the data given to it.

To use the options an enum or enum class is required, e.g.:

```
enum class MyOptions {
 USAGE, FILE_IN, FILE_OUT, FLAG_VERBOSE,
 OPT_UNKNOWN, OPT_NOOPT, OPT_DASH, OPT_LDASH, OPT_DONE
};
```

The options prefixed with `OPT_` are obligatory. Their meaning is documented in `nih::enum_has_members<>`. Their presence is validated at compile time.

The enum values are returned 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 `[]` operator always returns a valid, terminated string (provided the command line arguments are valid, terminated strings). So it is always safe to dereference the pointer, even when reading beyond the end of command line arguments.

The `getopt[0]` calls return the command line argument that contains the selected option. So in the `FILE_IN` case it could be any of `-i`, `--in`, `-vi`, `-ifile` or `-vifile`. This is useful for the `OPT_UNKNOWN` and `OPT_NOOPT` cases. The `getopt[1]` call on the other hand would return file regardless of argument chaining.

## 14.8.2 Class Documentation

### 14.8.2.1 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

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

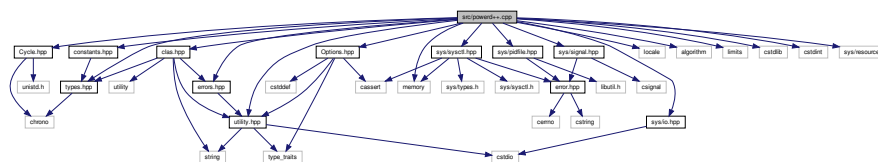
## Class Members

|              |        |                                                                                             |
|--------------|--------|---------------------------------------------------------------------------------------------|
| const char * | args   | A comma separated list of arguments. Set to nullptr or "" if no argument is available.      |
| const char * | 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.           |
| const char * | usage  | A usage string.                                                                             |

## 14.9 src/powerd++.cpp File Reference

```
#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 "sys/io.hpp"
#include <locale>
#include <memory>
#include <algorithm>
#include <limits>
#include <cstdlib>
#include <cstdint>
#include <sys/resource.h>
```

Include dependency graph for power++.cpp:



## Classes

- struct `anonymous_namespace{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}::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

- template<typename... MsgTs>  
void `anonymous_namespace{powerd++.cpp}::verbose` (MsgTs &&... msg)  
*Outputs the given printf style 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 global state.
- const char \*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.
- const Parameter< OE > `anonymous_namespace{powerd++.cpp}::PARAMETERS [ ]`  
Definitions of command line parameters.

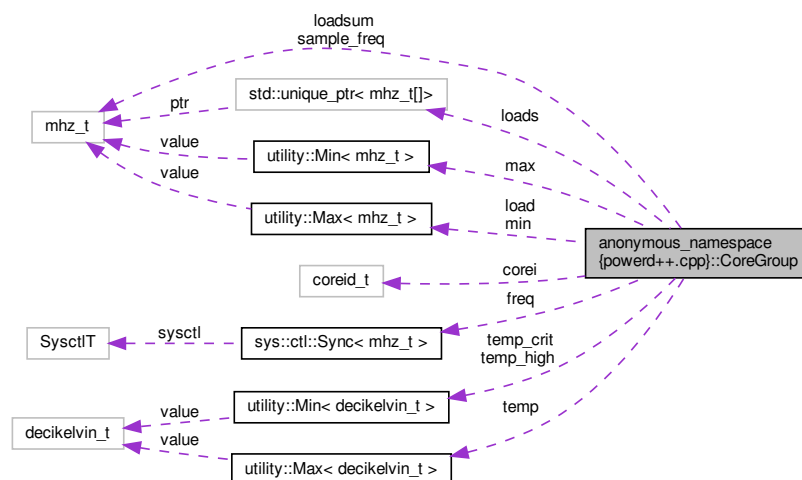
### 14.9.1 Detailed Description

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

### 14.9.2 Class Documentation

**14.9.2.1 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`:

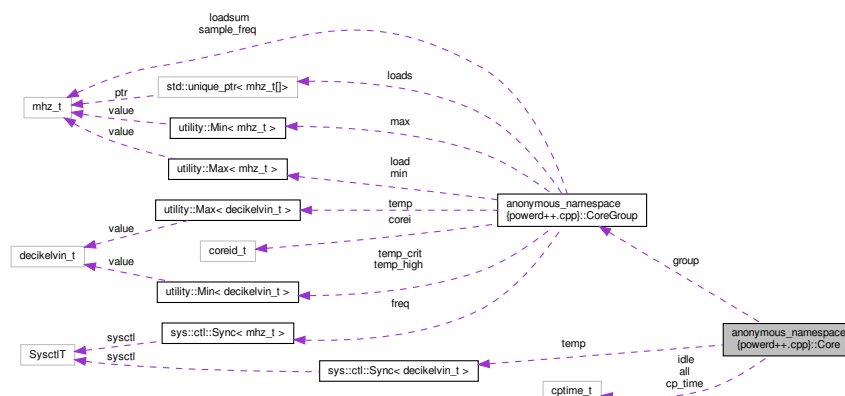


## Class Members

|                        |             |                                                                                                                                                                                                       |
|------------------------|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 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 <a href="#">update_loads()</a> .                                                                                              |
| 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.<br>This is updated by <a href="#">update_loads()</a> . |
| mhz_t                  | loadsum     | The maximum load sum of all controlled cores. This is updated by <a href="#">update_loads()</a> .                                                                                                     |
| 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 <a href="#">update_loads()</a> .                                                                                            |
| 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.                                                                                                                                                                          |

**14.9.2.2 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.                                   |
| const cptime_t *           | 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.        |

This is mostly for semantic clarity.

[illegible]

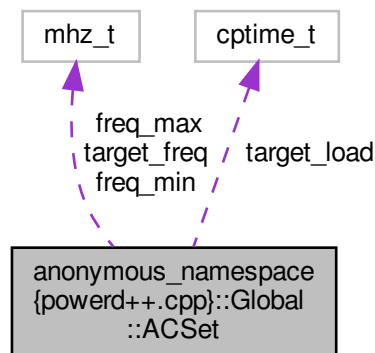
|                                           |                     |                                                                                                                                                                  |
|-------------------------------------------|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Sysctl                                    | acline_ctl          | The hw.acpi.acline ctl.                                                                                                                                          |
| struct<br>anonymous_namespace{powerd++    | ADP[3]              |                                                                                                                                                                  |
| struct<br>anonymous_namespace{powerd++    | battery[3]          |                                                                                                                                                                  |
| unique_ptr< <a href="#">Core</a> []>      | cores               | This buffer is to be allocated with ncpu instances of the <a href="#">Core</a> struct to store the management information of every core.                         |
| unique_ptr<<br>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<br>anonymous_namespace{powerd++    | FREQ_DEFAULT_MAX[3] |                                                                                                                                                                  |
| struct<br>anonymous_namespace{powerd++    | FREQ_DEFAULT_MIN[3] |                                                                                                                                                                  |
| struct<br>anonymous_namespace{powerd++    | FREQ_UNSET[3]       |                                                                                                                                                                  |
| unique_ptr< <a href="#">CoreGroup</a> []> | groups              | This buffer is to be allocated with the number of core groups. A core group is created by <a href="#">init()</a> for each core that has a dev.cpu.d.freq handle. |
| struct<br>anonymous_namespace{powerd++    | HADP[3]             |                                                                                                                                                                  |
| ms                                        | interval            | The polling interval.                                                                                                                                            |
| const SysctlOnce< coreid_t, 2 >           | ncpu                | The number of CPU cores or threads.                                                                                                                              |
| coreid_t                                  | ngroups             | The number of frequency controlling core groups.                                                                                                                 |

## Class Members

|                                        |                 |                                                                                                  |
|----------------------------------------|-----------------|--------------------------------------------------------------------------------------------------|
| struct<br>anonymous_namespace{powerd++ | online[3]       |                                                                                                  |
| const char *                           | pidfilename     | Name of an alternative pidfile. If not given <a href="#">pidfile_open()</a> uses a default name. |
| size_t                                 | sample          | The current sample.                                                                              |
| size_t                                 | samples         | The number of load samples to take.                                                              |
| volatile sig_atomic_t                  | signal          | The last signal received, used for terminating.                                                  |
| 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<br>anonymous_namespace{powerd++ | unknown[3]      | The power states.                                                                                |
| bool                                   | verbose         | Verbose mode.                                                                                    |

**14.9.2.4 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.                                                                                                |
| const char *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. |

### 14.9.3 Function Documentation

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

Main routine, setup and execute daemon, print errors.

Parameters

|                        |                            |
|------------------------|----------------------------|
| <code>argc,argv</code> | The command line arguments |
|------------------------|----------------------------|

Returns

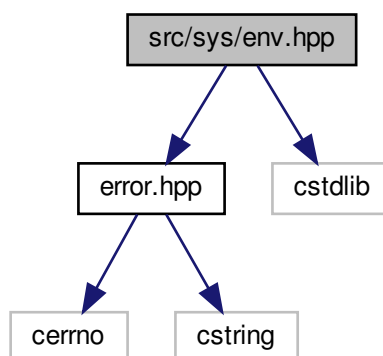
An exit code

See also

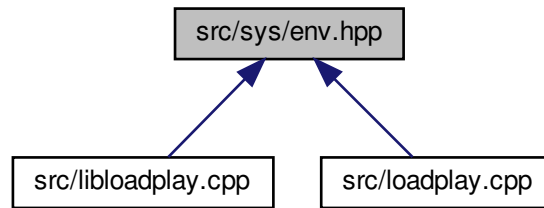
Exit

## 14.10 src/sys/env.hpp File Reference

```
#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 referring to an environment variable.*
- struct `sys::env::Vars`  
*A singleton class providing access to environment variables.*

## Namespaces

- `sys`
- `sys::env`  
*Provides wrappers around the `getenv()` family of functions.*

## Variables

- struct `sys::env::Vars sys::env::vars`  
*Singleton providing access to environment variables.*

### 14.10.1 Detailed Description

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

### 14.10.2 Class Documentation

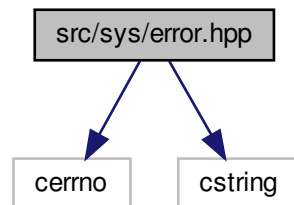
#### 14.10.2.1 struct `sys::env::error` The domain error type.

## 14.11 src/sys/error.hpp File Reference

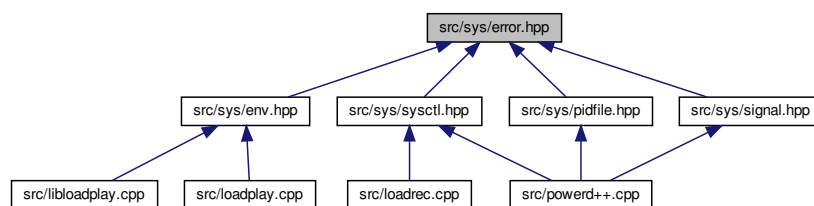
```
#include <cerrno>
```

```
#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](#)

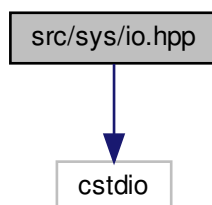
#### 14.11.1 Detailed Description

Provides system call error handling.

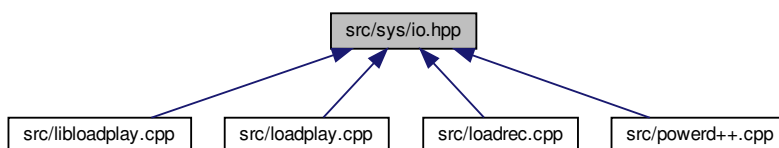
## 14.12 src/sys/io.hpp File Reference

```
#include <cstdio>
```

Include dependency graph for io.hpp:



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



### Classes

- class [sys::io::file< Ownership, Features >](#)  
Produces file access types around the C file handling facilities. [More...](#)
- struct [sys::io::enable\\_if< bool, T >](#)  
Similar to `std::enable_if`, but it also has the value of the expression.
- struct [sys::io::enable\\_if< false, T >](#)  
Specialise `enable_if` for a false expression.
- struct [sys::io::set< Set >](#)  
Pack a set of integral values in a type. [More...](#)
- struct [sys::io::contains< SetT, Value >](#)  
Check whether a set type contains a value. [More...](#)
- struct [sys::io::contains< set< Set ... >, Value >](#)  
Specialise `io::contains` to unpack `io::set`.
- struct [sys::io::is\\_superset\\_of< LSetT, RSetT >](#)  
Check whether the left hand set is a superest of the right hand set. [More...](#)
- struct [sys::io::is\\_superset\\_of< LSetT, set< RSet ... > >](#)  
Specialise `is_superset_of` to unpack the right hand `io::set`.
- struct [sys::io::query](#)  
Ask questions about the contents of a string. [More...](#)



- struct `sys::io::query::contains_ftor`  
*Test a string whether it contains a set of characters.*
- class `sys::io::file_feature< FileT,... >`  
*Implements the base functionality of all file access types.*
- class `sys::io::file_feature< FileT, read, Tail ... >`  
*Implement read support for file types.*
- class `sys::io::file_feature< FileT, write, Tail ... >`  
*Implement write support for file types.*
- class `sys::io::file_feature< FileT, seek, Tail ... >`  
*Implement seek support for file types.*
- class `sys::io::file< own, Features ... >`  
*Specialise for FILE object owning file instances.*
- class `sys::io::file< link, Features ... >`  
*Specialise for FILE object linking file instances.*

## Namespaces

- `sys`
- `sys::io`  
*This namespace contains c++ wrappers for <stdio> functionality.*

## Typedefs

- `template<class SetT , auto Value>`  
`using sys::io::contains_t = typename contains< SetT, Value >::type`  
*Check whether a set type contains a value.*
- `template<class LSetT , class RSetT >`  
`using sys::io::is_superset_of_t = typename is_superset_of< LSetT, RSetT >::type`  
*Check whether the left hand set is a superest of the right hand set.*

## Enumerations

- enum `sys::io::feature` { `sys::io::feature::read`, `sys::io::feature::write`, `sys::io::feature::seek` }  
*Feature flags for file type composition.*
- enum `sys::io::ownership` { `sys::io::ownership::own`, `sys::io::ownership::link` }  
*Ownership relation to the underlying FILE object.*

## Variables

- `template<class SetT , auto Value>`  
`constexpr const auto sys::io::contains_v = contains<SetT, Value>::value`  
*Check whether a set type contains a value.*
- `template<class LSetT , class RSetT >`  
`constexpr const auto sys::io::is_superset_of_v = is_superset_of<LSetT, RSetT>::value`  
*Check whether the left hand set is a superest of the right hand set.*
- `file< link, write > sys::io::ferr {stderr}`  
*File access instances for stderr.*
- `file< link, write > sys::io::fout {stdout}`  
*File access instances for stdout.*
- `file< link, read > sys::io::fin {stdin}`  
*File access instances for stdin.*

### 14.12.1 Detailed Description

Implements c++ wrappers for <stdio> I/O functionality.

### 14.12.2 Class Documentation

#### 14.12.2.1 class `sys::io::file`

```
template<ownership Ownership, feature ... Features>
class sys::io::file< Ownership, Features >
```

Produces file access types around the C file handling facilities.

Template Parameters

|                  |                                                                    |
|------------------|--------------------------------------------------------------------|
| <i>Ownership</i> | Determine the ownership relationship to the underlying FILE object |
| <i>Features</i>  | A list of features the file type supports                          |

See also

[ownership](#)

[feature](#)

[file<own, Features ...>](#)

[file<link, Features ...>](#)

[file\\_feature](#)

#### 14.12.2.2 struct `sys::io::set`

```
template<auto ... Set>
struct sys::io::set< Set >
```

Pack a set of integral values in a type.

Template Parameters

|            |                          |
|------------|--------------------------|
| <i>Set</i> | A set of integral values |
|------------|--------------------------|

#### 14.12.2.3 struct `sys::io::contains`

```
template<class SetT, auto Value>
struct sys::io::contains< SetT, Value >
```

Check whether a set type contains a value.

Template Parameters

|              |                                                            |
|--------------|------------------------------------------------------------|
| <i>SetT</i>  | A set of integral values packed in <a href="#">io::set</a> |
| <i>Value</i> | The value to look up                                       |

#### 14.12.2.4 struct sys::io::is\_superset\_of

```
template<class LSetT, class RSetT>
struct sys::io::is_superset_of< LSetT, RSetT >
```

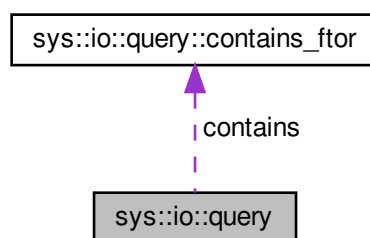
Check whether the left hand set is a superest of the right hand set.

Template Parameters

|                     |                                       |
|---------------------|---------------------------------------|
| <i>LSetT, RSetT</i> | Two <a href="#">io::set</a> instances |
|---------------------|---------------------------------------|

#### 14.12.2.5 struct sys::io::query Ask questions about the contents of a string.

Collaboration diagram for sys::io::query:



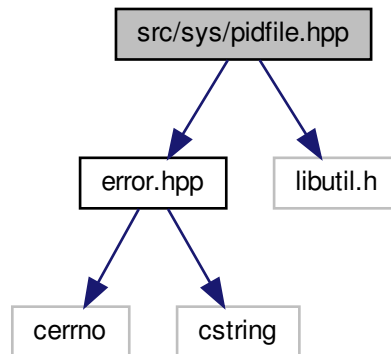
Class Members

|                                      |          |                                  |
|--------------------------------------|----------|----------------------------------|
| struct <a href="#">contains_ftor</a> | contains | Query the string for characters. |
|--------------------------------------|----------|----------------------------------|

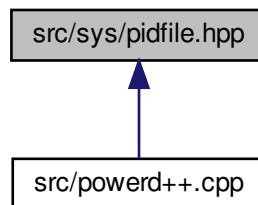
## 14.13 src/sys/pidfile.hpp File Reference

```
#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 RAI pattern.*

## Namespaces

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

### 14.13.1 Detailed Description

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

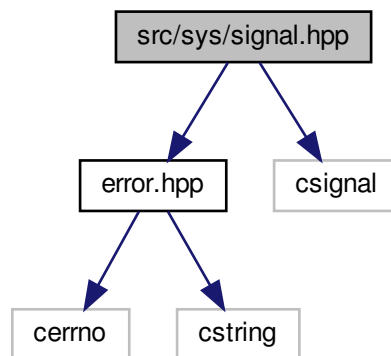
Requires linking with `-lutil`.

### 14.13.2 Class Documentation

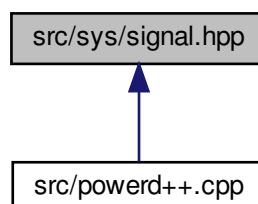
**14.13.2.1 struct sys::pid::error** The domain error type.

## 14.14 src/sys/signal.hpp File Reference

```
#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`
- `sys::sig`  
*This namespace provides c++ wrappers for `signal(3)`.*

## Typedefs

- using `sys::sig::sig_t` = `void(*)(int)`  
*Convenience type for signal handlers.*

### 14.14.1 Detailed Description

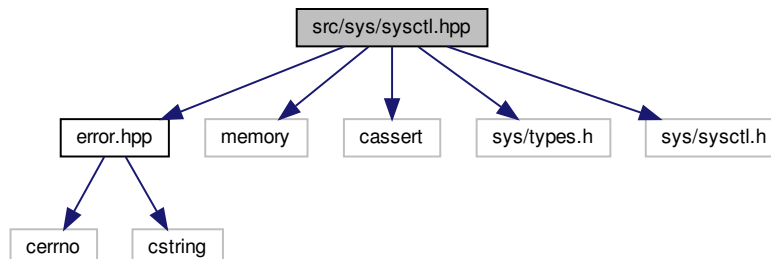
Implements a c++ wrapper for the `signal(3)` call.

### 14.14.2 Class Documentation

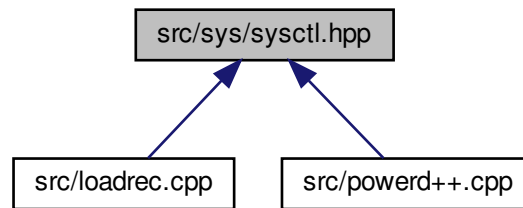
**14.14.2.1 struct `sys::sig::error`** The domain error type.

## 14.15 `src/sys/sysctl.hpp` File Reference

```
#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`
- `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.

### 14.15.1 Detailed Description

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

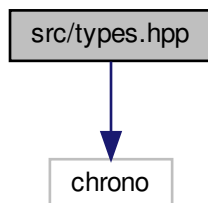
### 14.15.2 Class Documentation

**14.15.2.1 struct `sys::ctl::error`** The domain error type.

## 14.16 src/types.hpp File Reference

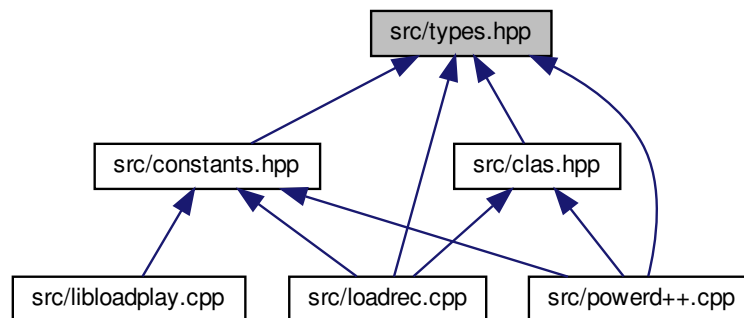
```
#include <chrono>
```

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.*

### 14.16.1 Detailed Description

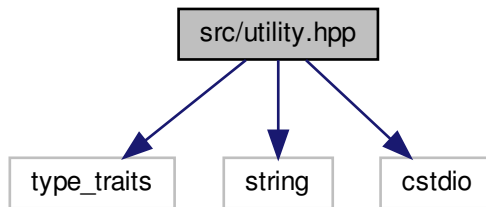
A collection of type aliases.

## 14.17 src/utility.hpp File Reference

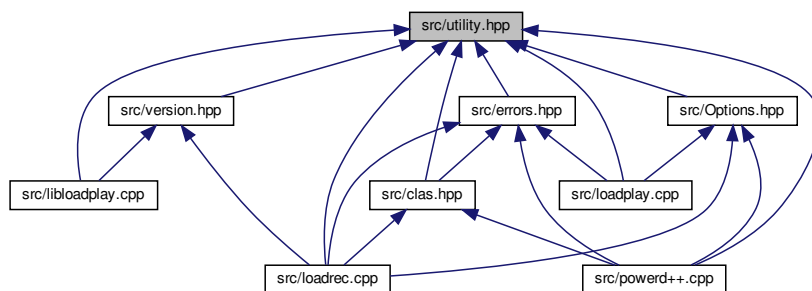
```
#include <type_traits>
#include <string>
```

```
#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 literal operators.

## Functions

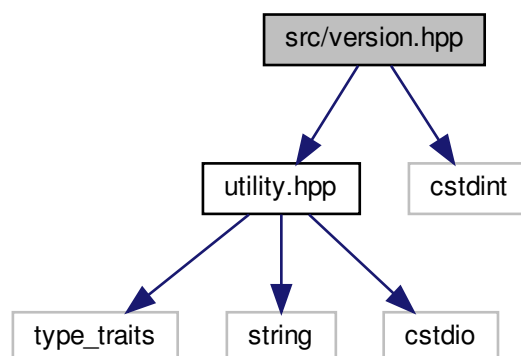
- `template<typename T, size_t Count>`  
`constexpr size_t utility::countof (T(&)[Count])`  
*Like `sizeof()`, but it returns the number of elements an array consists of instead of the number of bytes.*
- `template<typename... Args>`  
`void utility::sprintf (Args...)`  
*This is a safeguard against 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.*

### 14.17.1 Detailed Description

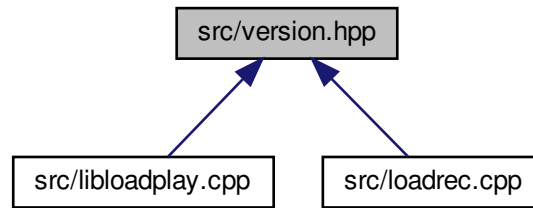
Implements generally useful functions.

## 14.18 src/version.hpp File Reference

```
#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

- const char \*const [version::LOADREC\\_FEATURES](#) = "usr.app.powerdxx.loadrec.features"  
*The pseudo MIB name for the load recording feature flags.*

### 14.18.1 Detailed Description

Defines types and constants used for version management.

# Index

- AcLineState
  - anonymous\_namespace{powerd++.cpp}, 43
- addValue
  - anonymous\_namespace{libloadplay.cpp}::SysctlS, 153
- all
  - sys::io::query::contains\_ftor, 79
- anonymous\_namespace{clas.cpp}, 23
  - Unit, 24
  - unit, 24
  - UnitStr, 24
- anonymous\_namespace{libloadplay.cpp}, 25
  - debug, 29
  - dprintf, 29
  - fail, 29
  - FEATURES, 32
  - ifile, 28
  - ofile, 28
  - operator<<, 30
  - operator""\_r, 30
  - strcmp, 30
  - SysctlValue::get< std::string >, 31
  - warn, 31
- anonymous\_namespace{libloadplay.cpp}::Callback< FunctionArgs >, 75
  - Callback, 76, 77
  - operator(), 77
- anonymous\_namespace{libloadplay.cpp}::CoreFrameReport, 27, 177
- anonymous\_namespace{libloadplay.cpp}::CoreReport, 27, 177
- anonymous\_namespace{libloadplay.cpp}::Emulator, 83
  - Emulator, 85
  - ncpu, 85
  - operator(), 85
- anonymous\_namespace{libloadplay.cpp}::Emulator::Core, 84, 178
- anonymous\_namespace{libloadplay.cpp}::Hold< T >, 116
  - Hold, 117
- anonymous\_namespace{libloadplay.cpp}::Main, 118
  - Main, 119
- anonymous\_namespace{libloadplay.cpp}::mib\_t, 122
  - mib\_t, 122, 123
  - operator int \*, 123
  - operator int const \*, 123
  - operator<, 123
  - operator==, 124
- anonymous\_namespace{libloadplay.cpp}::Report, 136
  - frame, 137
  - Report, 137
- anonymous\_namespace{libloadplay.cpp}::Report::Frame, 112
  - Frame, 113
  - operator[], 113, 115
- anonymous\_namespace{libloadplay.cpp}::SysctlS, 152
  - addValue, 153
  - getMib, 154
  - mibs, 155
  - operator[], 154
  - sysctlS, 155
- anonymous\_namespace{libloadplay.cpp}::SysctlValue, 156
  - get, 158, 159
  - mtx, 163
  - operator=, 160
  - registerOnSet, 160, 161
  - set, 161, 162
  - size, 162, 163
  - SysctlValue, 157, 158
  - value, 163
- anonymous\_namespace{loadplay.cpp}, 32
  - execute, 33
  - filename, 33
  - OE, 32
  - PARAMETERS, 34
- anonymous\_namespace{loadrec.cpp}, 34
  - FEATURES, 38
  - OE, 35
  - ofile, 35
  - PARAMETERS, 38
  - print\_sysctlS, 37
  - read\_args, 37
  - run, 37
  - verbose, 37
- anonymous\_namespace{powerd++.cpp}, 38
  - AcLineState, 43
  - init, 44
  - init\_loads, 44
  - OE, 43
  - PARAMETERS, 47
  - read\_args, 44
  - set\_mode, 45
  - signal\_recv, 45
  - sysctl\_fail, 46
  - update\_freq, 46
  - update\_loads, 46
  - verbose, 46
- anonymous\_namespace{powerd++.cpp}::Core, 40, 194
- anonymous\_namespace{powerd++.cpp}::CoreGroup, 40, 193
- anonymous\_namespace{powerd++.cpp}::FreqGuard, 115
- anonymous\_namespace{powerd++.cpp}::Global, 41, 195
- anonymous\_namespace{powerd++.cpp}::Global::ACSet, 196
- anonymous\_namespace{clas.cpp}
  - CELSIUS, 24
  - FAHRENHEIT, 24

- GHZ, [24](#)
- HZ, [24](#)
- KELVIN, [24](#)
- KHZ, [24](#)
- MHZ, [24](#)
- MILLISECOND, [24](#)
- PERCENT, [24](#)
- RANKINE, [24](#)
- SCALAR, [24](#)
- SECOND, [24](#)
- THZ, [24](#)
- UNKNOWN, [24](#)
- anonymous\_namespace{loadplay.cpp}
  - CMD, [33](#)
  - FILE\_IN, [33](#)
  - FILE\_OUT, [33](#)
  - OPT\_DASH, [33](#)
  - OPT\_DONE, [33](#)
  - OPT\_LDASH, [33](#)
  - OPT\_NOOPT, [33](#)
  - OPT\_UNKNOWN, [33](#)
  - USAGE, [33](#)
- anonymous\_namespace{loadrec.cpp}
  - FILE\_OUTPUT, [37](#)
  - FILE\_PID, [37](#)
  - FLAG\_VERBOSE, [37](#)
  - IVAL\_DURATION, [37](#)
  - IVAL\_POLL, [37](#)
  - OPT\_DASH, [37](#)
  - OPT\_DONE, [37](#)
  - OPT\_LDASH, [37](#)
  - OPT\_NOOPT, [37](#)
  - OPT\_UNKNOWN, [37](#)
  - USAGE, [37](#)
- anonymous\_namespace{powerd++.cpp}
  - BATTERY, [43](#)
  - CNT\_SAMPLES, [44](#)
  - FILE\_PID, [44](#)
  - FLAG\_FOREGROUND, [44](#)
  - FLAG\_VERBOSE, [44](#)
  - FREQ\_MAX, [43](#)
  - FREQ\_MAX\_AC, [43](#)
  - FREQ\_MAX\_BATT, [43](#)
  - FREQ\_MIN, [43](#)
  - FREQ\_MIN\_AC, [43](#)
  - FREQ\_MIN\_BATT, [43](#)
  - FREQ\_RANGE, [44](#)
  - FREQ\_RANGE\_AC, [44](#)
  - FREQ\_RANGE\_BATT, [44](#)
  - HITEMP\_RANGE, [44](#)
  - IGNORE, [44](#)
  - IVAL\_POLL, [44](#)
  - LENGTH, [43](#)
  - MODE\_AC, [43](#)
  - MODE\_BATT, [43](#)
  - MODE\_UNKNOWN, [44](#)
  - ONLINE, [43](#)
  - OPT\_DASH, [44](#)
  - OPT\_DONE, [44](#)
  - OPT\_LDASH, [44](#)
  - OPT\_NOOPT, [44](#)
  - OPT\_UNKNOWN, [44](#)
  - UNKNOWN, [43](#)
  - USAGE, [43](#)
- any
  - sys::io::query::contains\_ftor, [80](#)
- argCount
  - nih, [57](#)
- BATTERY
  - anonymous\_namespace{powerd++.cpp}, [43](#)
- bmatch
  - nih::Options< OptionT, DefCount >, [130](#)
- c\_str
  - sys::env::Var, [165](#)
  - sys::sc\_error< Domain >, [139](#)
- Callback
  - anonymous\_namespace{libloadplay.cpp}::Callback< FunctionArgs >, [76](#), [77](#)
- CELSIUS
  - anonymous\_namespace{clas.cpp}, [24](#)
- celsius
  - clas, [48](#)
- clas, [47](#)
  - celsius, [48](#)
  - freq, [48](#)
  - ival, [50](#)
  - load, [50](#)
  - range, [51](#)
  - samples, [51](#)
  - temperature, [51](#)
- clk
  - timing::Cycle, [83](#)
- close
  - sys::io::file< own, Features ... >, [95](#)
- CMD
  - anonymous\_namespace{loadplay.cpp}, [33](#)
- CNT\_SAMPLES
  - anonymous\_namespace{powerd++.cpp}, [44](#)
- constants, [52](#)
  - TJMAX\_SOURCES, [53](#)
- contains\_t
  - sys::io, [67](#)
- contains\_v
  - sys::io, [68](#)
- countof
  - utility, [72](#)
- cptime\_t
  - types, [71](#)
- daemon
  - libloadplay.cpp, [179](#)
- debug
  - anonymous\_namespace{libloadplay.cpp}, [29](#)
- dprintf
  - anonymous\_namespace{libloadplay.cpp}, [29](#)

- ECLARG
  - errors, [54](#)
- ECONFLICT
  - errors, [55](#)
- EDAEMON
  - errors, [55](#)
- EEXCEPT
  - errors, [55](#)
- EEXEC
  - errors, [55](#)
- EFILE
  - errors, [55](#)
- EFORBIDDEN
  - errors, [55](#)
- EFREQ
  - errors, [55](#)
- EIVAL
  - errors, [55](#)
- ELOAD
  - errors, [55](#)
- EMODE
  - errors, [55](#)
- Emulator
  - anonymous\_namespace{libloadplay.cpp}::Emulator, [85](#)
- ENOFREQ
  - errors, [55](#)
- eof
  - sys::io::file\_feature< FileT,... >, [98](#)
- EOUTOFRANGE
  - errors, [55](#)
- EPID
  - errors, [55](#)
- ERANGFMT
  - errors, [55](#)
- erase
  - sys::env::Var, [165](#)
- error
  - sys::io::file\_feature< FileT,... >, [98](#)
- errors, [53](#)
  - ECLARG, [54](#)
  - ECONFLICT, [55](#)
  - EDAEMON, [55](#)
  - EEXCEPT, [55](#)
  - EEXEC, [55](#)
  - EFILE, [55](#)
  - EFORBIDDEN, [55](#)
  - EFREQ, [55](#)
  - EIVAL, [55](#)
  - ELOAD, [55](#)
  - EMODE, [55](#)
  - ENOFREQ, [55](#)
  - EOUTOFRANGE, [55](#)
  - EPID, [55](#)
  - ERANGFMT, [55](#)
  - ESAMPLES, [55](#)
  - ESIGNAL, [55](#)
  - ESYSCTL, [55](#)
  - ETEMPERATURE, [55](#)
  - EWOPEN, [55](#)
  - Exit, [54](#)
  - ExitStr, [55](#)
  - fail, [55](#)
  - LENGTH, [55](#)
  - OK, [54](#)
  - errors::Exception, [54](#), [173](#)
  - ESAMPLES
    - errors, [55](#)
  - ESIGNAL
    - errors, [55](#)
  - ESYSCTL
    - errors, [55](#)
  - ETEMPERATURE
    - errors, [55](#)
  - EWOPEN
    - errors, [55](#)
  - execute
    - anonymous\_namespace{loadplay.cpp}, [33](#)
  - Exit
    - errors, [54](#)
  - ExitStr
    - errors, [55](#)
  - FAHRENHEIT
    - anonymous\_namespace{clas.cpp}, [24](#)
  - fail
    - anonymous\_namespace{libloadplay.cpp}, [29](#)
    - errors, [55](#)
  - feature
    - sys::io, [67](#)
  - FEATURES
    - anonymous\_namespace{libloadplay.cpp}, [32](#)
    - anonymous\_namespace{loadrec.cpp}, [38](#)
  - file
    - sys::io::file< link, Features ... >, [90](#)
    - sys::io::file< own, Features ... >, [94](#), [95](#)
  - file\_feature
    - sys::io::file\_feature< FileT,... >, [98](#)
  - FILE\_IN
    - anonymous\_namespace{loadplay.cpp}, [33](#)
  - FILE\_OUT
    - anonymous\_namespace{loadplay.cpp}, [33](#)
  - FILE\_OUTPUT
    - anonymous\_namespace{loadrec.cpp}, [37](#)
  - FILE\_PID
    - anonymous\_namespace{loadrec.cpp}, [37](#)
    - anonymous\_namespace{powerd++.cpp}, [44](#)
  - filename
    - anonymous\_namespace{loadplay.cpp}, [33](#)
  - FLAG\_FOREGROUND
    - anonymous\_namespace{powerd++.cpp}, [44](#)
  - FLAG\_VERBOSE
    - anonymous\_namespace{loadrec.cpp}, [37](#)
    - anonymous\_namespace{powerd++.cpp}, [44](#)
  - flush
    - sys::io::file\_feature< FileT, write, Tail ... >, [107](#)
  - Frame

- anonymous\_namespace{libloadplay.cpp}::Report::Frame, anonymous\_namespace{powerd++.cpp}, 44
  - 113
- frame
  - anonymous\_namespace{libloadplay.cpp}::Report, 137
- freq
  - clas, 48
- FREQ\_MAX
  - anonymous\_namespace{powerd++.cpp}, 43
- FREQ\_MAX\_AC
  - anonymous\_namespace{powerd++.cpp}, 43
- FREQ\_MAX\_BATT
  - anonymous\_namespace{powerd++.cpp}, 43
- FREQ\_MIN
  - anonymous\_namespace{powerd++.cpp}, 43
- FREQ\_MIN\_AC
  - anonymous\_namespace{powerd++.cpp}, 43
- FREQ\_MIN\_BATT
  - anonymous\_namespace{powerd++.cpp}, 43
- FREQ\_RANGE
  - anonymous\_namespace{powerd++.cpp}, 44
- FREQ\_RANGE\_AC
  - anonymous\_namespace{powerd++.cpp}, 44
- FREQ\_RANGE\_BATT
  - anonymous\_namespace{powerd++.cpp}, 44
- FREQ\_TRACKING
  - version, 75
- get
  - anonymous\_namespace{libloadplay.cpp}::SysctlValue, 158, 159
  - nih::Options< OptionT, DefCount >, 130, 131
  - sys::ctl::Sysctl< 0 >, 150, 151
  - sys::ctl::Sysctl< MibDepth >, 146, 147
  - sys::io::file< link, Features ... >, 91
  - sys::io::file< own, Features ... >, 96
- getc
  - sys::io::file\_feature< FileT, read, Tail ... >, 101
- geteuid
  - libloadplay.cpp, 179
- getMib
  - anonymous\_namespace{libloadplay.cpp}::Sysctls, 154
- gets
  - sys::io::file\_feature< FileT, read, Tail ... >, 101
- GHZ
  - anonymous\_namespace{clas.cpp}, 24
- HITEMP\_RANGE
  - anonymous\_namespace{powerd++.cpp}, 44
- Hold
  - anonymous\_namespace{libloadplay.cpp}::Hold< T >, 117
- HZ
  - anonymous\_namespace{clas.cpp}, 24
- ifile
  - anonymous\_namespace{libloadplay.cpp}, 28
- IGNORE
  - init
    - anonymous\_namespace{powerd++.cpp}, 44
  - init\_loads
    - anonymous\_namespace{powerd++.cpp}, 44
  - is\_superset\_of\_t
    - sys::io, 67
  - is\_superset\_of\_v
    - sys::io, 69
  - ival
    - clas, 50
  - IVAL\_DURATION
    - anonymous\_namespace{loadrec.cpp}, 37
  - IVAL\_POLL
    - anonymous\_namespace{loadrec.cpp}, 37
    - anonymous\_namespace{powerd++.cpp}, 44
  - KELVIN
    - anonymous\_namespace{clas.cpp}, 24
  - KHZ
    - anonymous\_namespace{clas.cpp}, 24
  - LENGTH
    - anonymous\_namespace{powerd++.cpp}, 43
    - errors, 55
  - libloadplay.cpp
    - daemon, 179
    - geteuid, 179
    - pidfile\_close, 179
    - pidfile\_fileno, 179
    - pidfile\_open, 180
    - pidfile\_remove, 180
    - pidfile\_write, 180
    - sysctl, 180
    - sysctlbyname, 181
    - sysctlnametomib, 183
  - link
    - sys::io, 68
  - load
    - clas, 50
  - loadplay.cpp
    - main, 185
  - loadrec.cpp
    - main, 187
  - LoadrecBits
    - version, 74
  - Main
    - anonymous\_namespace{libloadplay.cpp}::Main, 119
  - main
    - loadplay.cpp, 185
    - loadrec.cpp, 187
    - powerd++.cpp, 197
  - make\_Once
    - sys::ctl, 60
  - make\_Options
    - nih, 57
  - make\_Sysctl



- sys::ctl, 61
- match
  - nih::Options< OptionT, DefCount >, 131
- Max
  - utility::Max< T >, 121
- MHZ
  - anonymous\_namespace{clas.cpp}, 24
- mib\_t
  - anonymous\_namespace{libloadplay.cpp}::mib\_t, 122, 123
- mibs
  - anonymous\_namespace{libloadplay.cpp}::SysctlIs, 155
- MILLISECOND
  - anonymous\_namespace{clas.cpp}, 24
- Min
  - utility::Min< T >, 125
- MODE\_AC
  - anonymous\_namespace{powerd++.cpp}, 43
- MODE\_BATT
  - anonymous\_namespace{powerd++.cpp}, 43
- MODE\_UNKNOWN
  - anonymous\_namespace{powerd++.cpp}, 44
- mtx
  - anonymous\_namespace{libloadplay.cpp}::SysctlValue, 163
- ncpu
  - anonymous\_namespace{libloadplay.cpp}::Emulator, 85
- nih, 56
  - argCount, 57
  - make\_Options, 57
- nih::enum\_has\_members< OptionT, class >, 87
- nih::Options< OptionT, DefCount >, 128
  - bmatch, 130
  - get, 130, 131
  - match, 131
  - offset, 132
  - operator OptionT, 132
  - operator(), 132
  - operator[], 132
  - opt\_dash, 134
  - opt\_ldash, 134
  - opt\_noopt, 134
  - opt\_unknown, 134
  - Options, 130
  - removePath, 133
  - show, 133
  - usage, 134
- nih::Parameter, 56, 190
- OE
  - anonymous\_namespace{loadplay.cpp}, 32
  - anonymous\_namespace{loadrec.cpp}, 35
  - anonymous\_namespace{powerd++.cpp}, 43
- offset
  - nih::Options< OptionT, DefCount >, 132
- ofile
  - anonymous\_namespace{libloadplay.cpp}, 28
  - anonymous\_namespace{loadrec.cpp}, 35
- OK
  - errors, 54
- Once
  - sys::ctl::Once< T, SysctlT >, 127
- ONLINE
  - anonymous\_namespace{powerd++.cpp}, 43
- operator bool
  - sys::io::file\_feature< FileT,... >, 98
- operator char const \*
  - sys::env::Var, 165
- operator int
  - sys::sc\_error< Domain >, 139
- operator int \*
  - anonymous\_namespace{libloadplay.cpp}::mib\_t, 123
- operator int const \*
  - anonymous\_namespace{libloadplay.cpp}::mib\_t, 123
- operator OptionT
  - nih::Options< OptionT, DefCount >, 132
- operator T
  - sys::ctl::Sync< T, SysctlT >, 144
- operator T const &
  - sys::ctl::Once< T, SysctlT >, 128
  - utility::Max< T >, 121
  - utility::Min< T >, 125
  - utility::Sum< T >, 142
- operator<
  - anonymous\_namespace{libloadplay.cpp}::mib\_t, 123
- operator<<
  - anonymous\_namespace{libloadplay.cpp}, 30
- operator()
  - anonymous\_namespace{libloadplay.cpp}::Callback< FunctionArgs >, 77
  - anonymous\_namespace{libloadplay.cpp}::Emulator, 85
  - nih::Options< OptionT, DefCount >, 132
  - sys::io::query::contains\_ftor, 80
  - timing::Cycle, 82
  - utility::Formatter< BufSize >, 112
- operator+=
  - utility::Sum< T >, 142
- operator=
  - anonymous\_namespace{libloadplay.cpp}::SysctlValue, 160
  - sys::ctl::Sync< T, SysctlT >, 144
  - sys::env::Var, 165
  - sys::io::file< link, Features ... >, 91
  - sys::io::file< own, Features ... >, 96
  - utility::Max< T >, 121
  - utility::Min< T >, 126
- operator==
  - anonymous\_namespace{libloadplay.cpp}::mib\_t, 124
- operator""\_fmt

- utility::literals, 74
- operator""\_FREQ\_TRACKING
  - version::literals, 75
- operator""\_r
  - anonymous\_namespace{libloadplay.cpp}, 30
- operator[]
  - anonymous\_namespace{libloadplay.cpp}::Report::Frame, 113, 115
  - anonymous\_namespace{libloadplay.cpp}::Sysctl, 154
  - nih::Options< OptionT, DefCount >, 132
  - sys::env::Vars, 167
- OPT\_DASH
  - anonymous\_namespace{loadplay.cpp}, 33
  - anonymous\_namespace{loadrec.cpp}, 37
  - anonymous\_namespace{powerd++.cpp}, 44
- opt\_dash
  - nih::Options< OptionT, DefCount >, 134
- OPT\_DONE
  - anonymous\_namespace{loadplay.cpp}, 33
  - anonymous\_namespace{loadrec.cpp}, 37
  - anonymous\_namespace{powerd++.cpp}, 44
- OPT\_LDASH
  - anonymous\_namespace{loadplay.cpp}, 33
  - anonymous\_namespace{loadrec.cpp}, 37
  - anonymous\_namespace{powerd++.cpp}, 44
- opt\_ldash
  - nih::Options< OptionT, DefCount >, 134
- OPT\_NOOPT
  - anonymous\_namespace{loadplay.cpp}, 33
  - anonymous\_namespace{loadrec.cpp}, 37
  - anonymous\_namespace{powerd++.cpp}, 44
- opt\_noopt
  - nih::Options< OptionT, DefCount >, 134
- OPT\_UNKNOWN
  - anonymous\_namespace{loadplay.cpp}, 33
  - anonymous\_namespace{loadrec.cpp}, 37
  - anonymous\_namespace{powerd++.cpp}, 44
- opt\_unknown
  - nih::Options< OptionT, DefCount >, 134
- Options
  - nih::Options< OptionT, DefCount >, 130
- own
  - sys::io, 68
- ownership
  - sys::io, 68
- PARAMETERS
  - anonymous\_namespace{loadplay.cpp}, 34
  - anonymous\_namespace{loadrec.cpp}, 38
  - anonymous\_namespace{powerd++.cpp}, 47
- PERCENT
  - anonymous\_namespace{clas.cpp}, 24
- pfh
  - sys::pid::Pidfile, 136
- Pidfile
  - sys::pid::Pidfile, 135
- pidfile\_close
  - libloadplay.cpp, 179
- pidfile\_fileno
  - libloadplay.cpp, 179
- pidfile\_open
  - libloadplay.cpp, 180
- pidfile\_remove
  - libloadplay.cpp, 180
- pidfile\_write
  - libloadplay.cpp, 180
- powerd++.cpp
  - main, 197
- print
  - sys::io::file\_feature< FileT, write, Tail ... >, 108
- print\_sysctls
  - anonymous\_namespace{loadrec.cpp}, 37
- printf
  - sys::io::file\_feature< FileT, write, Tail ... >, 108, 109
- putc
  - sys::io::file\_feature< FileT, write, Tail ... >, 109
- range
  - clas, 51
- RANKINE
  - anonymous\_namespace{clas.cpp}, 24
- read
  - sys::io, 68
  - sys::io::file\_feature< FileT, read, Tail ... >, 101, 102
- read\_args
  - anonymous\_namespace{loadrec.cpp}, 37
  - anonymous\_namespace{powerd++.cpp}, 44
- registerOnSet
  - anonymous\_namespace{libloadplay.cpp}::SysctlValue, 160, 161
- release
  - sys::io::file< own, Features ... >, 96
- removePath
  - nih::Options< OptionT, DefCount >, 133
- Report
  - anonymous\_namespace{libloadplay.cpp}::Report, 137
- rewind
  - sys::io::file\_feature< FileT, seek, Tail ... >, 105
- run
  - anonymous\_namespace{loadrec.cpp}, 37
- samples
  - clas, 51
- SCALAR
  - anonymous\_namespace{clas.cpp}, 24
- scanf
  - sys::io::file\_feature< FileT, read, Tail ... >, 103
- SECOND
  - anonymous\_namespace{clas.cpp}, 24
- seek
  - sys::io, 68
  - sys::io::file\_feature< FileT, seek, Tail ... >, 105
- set
  - anonymous\_namespace{libloadplay.cpp}::SysctlValue, 161, 162

- sys::ctl::Sysctl< 0 >, [151](#), [152](#)
  - sys::ctl::Sysctl< MibDepth >, [147](#), [148](#)
- set\_mode
  - anonymous\_namespace{powerd++.cpp}, [45](#)
- show
  - nih::Options< OptionT, DefCount >, [133](#)
- Signal
  - sys::sig::Signal, [140](#)
- signal\_recv
  - anonymous\_namespace{powerd++.cpp}, [45](#)
- size
  - anonymous\_namespace{libloadplay.cpp}::SysctlValue, [162](#), [163](#)
  - sys::ctl::Sysctl< 0 >, [152](#)
  - sys::ctl::Sysctl< MibDepth >, [148](#)
- sprintf
  - utility, [72](#)
- sprintf\_safe
  - utility, [72](#)
- src/clas.hpp, [167](#)
- src/constants.hpp, [169](#)
- src/Cycle.hpp, [171](#)
- src/errors.hpp, [172](#)
- src/libloadplay.cpp, [174](#)
- src/loadplay.cpp, [183](#)
- src/loadrec.cpp, [185](#)
- src/Options.hpp, [187](#)
- src/powerd++.cpp, [191](#)
- src/sys/env.hpp, [197](#)
- src/sys/error.hpp, [199](#)
- src/sys/io.hpp, [200](#)
- src/sys/pidfile.hpp, [203](#)
- src/sys/signal.hpp, [205](#)
- src/sys/sysctl.hpp, [206](#)
- src/types.hpp, [208](#)
- src/utility.hpp, [209](#)
- src/version.hpp, [211](#)
- Standard I/O File Access, [22](#)
- str
  - sys::env::Var, [166](#)
- strcmp
  - anonymous\_namespace{libloadplay.cpp}, [30](#)
- Sum
  - utility::Sum< T >, [141](#)
- Sync
  - sys::ctl::Sync< T, SysctlT >, [143](#)
- sys, [58](#)
- sys::ctl, [58](#)
  - make\_Once, [60](#)
  - make\_Sysctl, [61](#)
  - sysctl\_get, [61](#)
  - sysctl\_raw, [62](#)
  - sysctl\_set, [62](#)
  - SysctlOnce, [60](#)
  - SysctlSync, [60](#)
- sys::ctl::error, [60](#), [208](#)
- sys::ctl::Once< T, SysctlT >, [126](#)
  - Once, [127](#)
  - operator T const &, [128](#)
- sys::ctl::Sync< T, SysctlT >, [142](#)
  - operator T, [144](#)
  - operator=, [144](#)
  - Sync, [143](#)
- sys::ctl::Sysctl< 0 >, [149](#)
  - get, [150](#), [151](#)
  - set, [151](#), [152](#)
  - size, [152](#)
  - Sysctl, [149](#), [150](#)
- sys::ctl::Sysctl< MibDepth >, [144](#)
  - get, [146](#), [147](#)
  - set, [147](#), [148](#)
  - size, [148](#)
  - Sysctl, [146](#)
- sys::env, [63](#)
- sys::env::error, [63](#), [198](#)
- sys::env::Var, [163](#)
  - c\_str, [165](#)
  - erase, [165](#)
  - operator char const \*, [165](#)
  - operator=, [165](#)
  - str, [166](#)
  - Var, [164](#)
- sys::env::Vars, [166](#)
  - operator[], [167](#)
- sys::io, [63](#)
  - contains\_t, [67](#)
  - contains\_v, [68](#)
  - feature, [67](#)
  - is\_superset\_of\_t, [67](#)
  - is\_superset\_of\_v, [69](#)
  - link, [68](#)
  - own, [68](#)
  - ownership, [68](#)
  - read, [68](#)
  - seek, [68](#)
  - write, [68](#)
- sys::io::contains, [65](#), [202](#)
- sys::io::contains< set< Set ... >, Value >, [77](#)
- sys::io::enable\_if< bool, T >, [86](#)
- sys::io::enable\_if< false, T >, [86](#)
- sys::io::file, [65](#), [202](#)
- sys::io::file< link, Features ... >, [88](#)
  - file, [90](#)
  - get, [91](#)
  - operator=, [91](#)
- sys::io::file< own, Features ... >, [92](#)
  - close, [95](#)
  - file, [94](#), [95](#)
  - get, [96](#)
  - operator=, [96](#)
  - release, [96](#)
- sys::io::file\_feature< FileT, read, Tail ... >, [99](#)
  - getc, [101](#)
  - gets, [101](#)
  - read, [101](#), [102](#)
  - scanf, [103](#)

- sys::io::file\_feature< FileT, seek, Tail ... >, 103
  - rewind, 105
  - seek, 105
  - tell, 105
- sys::io::file\_feature< FileT, write, Tail ... >, 106
  - flush, 107
  - print, 108
  - printf, 108, 109
  - putc, 109
  - write, 110
- sys::io::file\_feature< FileT,... >, 97
  - eof, 98
  - error, 98
  - file\_feature, 98
  - operator bool, 98
- sys::io::is\_superset\_of, 66, 203
- sys::io::is\_superset\_of< LSetT, set< RSet ... > >, 117
- sys::io::query, 66, 203
- sys::io::query::contains\_ftor, 79
  - all, 79
  - any, 80
  - operator(), 80
- sys::io::set, 66, 202
- sys::pid, 69
- sys::pid::error, 69, 205
- sys::pid::Pidfile, 135
  - pfh, 136
  - Pidfile, 135
  - write, 136
- sys::sc\_error< Domain >, 138
  - c\_str, 139
  - operator int, 139
- sys::sig, 69
- sys::sig::error, 70, 206
- sys::sig::Signal, 139
  - Signal, 140
- Sysctl
  - sys::ctl::Sysctl< 0 >, 149, 150
  - sys::ctl::Sysctl< MibDepth >, 146
- sysctl
  - libloadplay.cpp, 180
- sysctl\_fail
  - anonymous\_namespace{power++.cpp}, 46
- sysctl\_get
  - sys::ctl, 61
- sysctl\_raw
  - sys::ctl, 62
- sysctl\_set
  - sys::ctl, 62
- sysctlbyname
  - libloadplay.cpp, 181
- sysctlnametomib
  - libloadplay.cpp, 183
- SysctlOnce
  - sys::ctl, 60
- sysctls
  - anonymous\_namespace{libloadplay.cpp}::Sysctls, 155
- SysctlSync
  - sys::ctl, 60
- SysctlValue
  - anonymous\_namespace{libloadplay.cpp}::SysctlValue, 157, 158
- SysctlValue::get< std::string >
  - anonymous\_namespace{libloadplay.cpp}, 31
- tell
  - sys::io::file\_feature< FileT, seek, Tail ... >, 105
- temperature
  - clas, 51
- THZ
  - anonymous\_namespace{clas.cpp}, 24
- timing, 70
- timing::Cycle, 81
  - clk, 83
  - operator(), 82
- TJMAX\_SOURCES
  - constants, 53
- to\_value
  - utility, 73
- types, 70
  - cptime\_t, 71
- Unit
  - anonymous\_namespace{clas.cpp}, 24
- unit
  - anonymous\_namespace{clas.cpp}, 24
- UnitStr
  - anonymous\_namespace{clas.cpp}, 24
- UNKNOWN
  - anonymous\_namespace{clas.cpp}, 24
  - anonymous\_namespace{power++.cpp}, 43
- update\_freq
  - anonymous\_namespace{power++.cpp}, 46
- update\_loads
  - anonymous\_namespace{power++.cpp}, 46
- USAGE
  - anonymous\_namespace{loadplay.cpp}, 33
  - anonymous\_namespace{loadrec.cpp}, 37
  - anonymous\_namespace{power++.cpp}, 43
- usage
  - nih::Options< OptionT, DefCount >, 134
- utility, 71
  - countof, 72
  - sprintf, 72
  - sprintf\_safe, 72
  - to\_value, 73
- utility::Formatter< BufSize >, 111
  - operator(), 112
- utility::literals, 73
  - operator""\_fmt, 74
- utility::Max< T >, 120
  - Max, 121
  - operator T const &, 121
  - operator=, 121
- utility::Min< T >, 124
  - Min, 125

- operator T const &, [125](#)
  - operator=, [126](#)
- utility::Sum< T >, [140](#)
  - operator T const &, [142](#)
  - operator+=, [142](#)
  - Sum, [141](#)
- value
  - anonymous\_namespace{libloadplay.cpp}::SysctlValue, [163](#)
- Var
  - sys::env::Var, [164](#)
- verbose
  - anonymous\_namespace{loadrec.cpp}, [37](#)
  - anonymous\_namespace{powerd++.cpp}, [46](#)
- version, [74](#)
  - FREQ\_TRACKING, [75](#)
  - LoadrecBits, [74](#)
- version::literals, [75](#)
  - operator""\_FREQ\_TRACKING, [75](#)
- warn
  - anonymous\_namespace{libloadplay.cpp}, [31](#)
- write
  - sys::io, [68](#)
  - sys::io::file\_feature< FileT, write, Tail ... >, [110](#)
  - sys::pid::Pidfile, [136](#)