NAME

febug_start(), **febug_end()**, **febug_register_type()**, **febug_wrap()**, **febug_unwrap()** — User-space debugfs ABI wrapper library

SYNOPSIS

```
#include <libfebug.h>
cc -lfebug ...

#define FEBUG_DONT   0
#define FEBUG_SOCKET "/var/run/febug.sock"
#define FEBUG_SIGNUM SIGUSR2

getenv("FEBUG_DONT");
getenv("FEBUG_SOCKET");

int febug_global_controlled_socket = -1;

void febug_start();

void febug_start_path(const char * path);

void febug_debug_handler(int);

void febug_register_type(uint64_t type, void (*formatter)(int, size_t));

void febug_wrap(uint64_t type, const void * data, const char * name, ...);

void febug_wrap_signal(uint64_t type, const void * data, uint8_t signal,
    const char * name, ...);

void febug_wrap_signalv(uint64_t type, const void * data, uint8_t signal,
    const char * name, va_list ap);

void febug_unwrap(const void * data);

void febug_end();
```

DESCRIPTION

Simplifies writing programs debuggable with febug(8) by presenting a high-level interface to febug-abi(5).

There are three compile-time macros that allow customising **libfebug** behaviour:

- FEBUG_DONT — If non-zero, all symbols become **static**, functions turn into no-ops, and therefore no symbols from libfebug.a|.so are imported at link-time; this is intended as a way to easily disable febug(8) integration completely on release builds.
- FEBUG_SIGNUM — The signal to request from febug(8) when using **febug_wrap**(). Defaults to SIGUSR2.
- FEBUG_SOCKET — The path to connect to febug(8) on. Defaults to /var/run/febug.sock.

There are two environment variables that allow a user to customise its behaviour:

- FEBUG_DONT — If set, don't try to connect to febug(8), so all library functions become no-ops.
- FEBUG_SOCKET — If set, use its value instead of the built-in FEBUG_SOCKET to connect to febug(8).

To be debugged, a program needs to, first, call **febug_start_path**() (likely via **febug_start**(), which simply passes FEBUG_SOCKET thereto) to connect to febug(8), which, if successful, will set *febug_global_controlled_socket* appropriately.

The program needs to install **febug_debug_handler**() (or a wrapper around it) as the signal handler for FEBUG_SIGNUM (and any other signals, if different ones are explicitly requested); if notifications are disabled (by requesting SIGKILL), some event loop that answers on *febug_global_controlled_socket* must be in place. It's a no-op if *febug_global_controlled_socket* is **-1**.

The program should register handlers for types of variables it wishes to handle by calling **febug_register_type**() — those type numbers should be consistent across the program, lest the wrong handler is called. If no handler was registered for a type, **febug_debug_handler**() will instead return a generic "not found" message. The handler takes the write end of the pipe as the first argument, and the variable ID as the second; it shouldn't close the pipe, as that is done by **febug_debug_handler**() regardless, and the program would then run the risk of closing another file with the same descriptor simultaneously opened by another thread. It's a no-op if *febug_global_controlled_socket* is **-1**.

At any time, when the program wishes to expose a variable, it can call **febug_wrap_signalv**() (likely via **febug_wrap_signal**() (likely via **febug_wrap**(), which passes FEBUG_SIGNUM thereto)), which will send a **febug_message** with the specified type and signal numbers, ID equal to the data pointer, and name formatted according to printf(3). It's a no-op if *febug_global_controlled_socket* is **-1**.

When the variable goes out of scope, the program should call **febug_unwrap**() to send a **stop_febug_message** with the same data pointer as it did **febug_wrap**(), to prevent reading random data that might no longer be mapped, or make sense. It's a no-op if *febug_global_controlled_socket* is **-1**.

When it wishes to stop being debugged, the program may call **febug_end**() which will shut and reset *febug_global_controlled_socket*, if any, and deallocate the type→handler map. The program may omit this if it'd be the last thing it did before exiting, since the kernel will close all file descriptors and free all mappings anyway.

EXAMPLES

The following program sorts a string with qsort(3) but waits half a second between each comparison; the string can be inspected via a febug(8) mount:

```
// SPDX-License-Identifier: MIT


#define _POSIX_C_SOURCE 200809L

#include <libfebug.h>

#include <errno.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>


#define CSTRING_FEBUG_TP 420
static void cstring_febug_formatter(int fd, size_t data) {
  const char * str = (const char *)data;
  dprintf(fd, "%s\n", str);
}


static int char_comp(const void * lhs, const void * rhs) {
```

```
  const struct timespec half_second = {0, 500 * 1000 * 1000};
  nanosleep(&half_second, 0);

  return *(const char *)lhs - *(const char *)rhs;
}


int main() {
  febug_start();
  febug_register_type(CSTRING_FEBUG_TP, cstring_febug_formatter);


  struct sigaction handler;
  memset(&handler, 0, sizeof(handler));
  handler.sa_handler = febug_debug_handler;
  if(sigaction(FEBUG_SIGNUM, &handler, 0) == -1) {
    fprintf(stderr, "sigaction: %s\n", strerror(errno));
    return 1;
  }


  {
    __attribute__((__cleanup__(febug_unwrap))) char data[] =
        "JVLOkgsYmhCyEFxouKzDNajivGlpWqbdBwnfTAXQcreRHPIUSMtZQWERTYUIOPqwertyuiop"
        "1234567890";
    febug_wrap(CSTRING_FEBUG_TP, data, "cool_data");

    qsort(data, strlen(data), 1, char_comp);
  }

  sleep(2);

  febug_end();
}
```

SEE ALSO

`febug-abi`(5) — the ABI wrapped by this library.
`libfebug++`(3) and `libfebug.rs`(3) — equivalent C++ and Rust libraries.

SPECIAL THANKS

To all who support further development, in particular:

- ThePhD
- Embark Studios
- Jasper Bekkers

REPORTING BUGS

febug tracker:
https://todo.sr.ht/~nabijaczleweli/febug

febug mailing list: ⟨`~nabijaczleweli/febug@lists.sr.ht`⟩, archived at
https://lists.sr.ht/~nabijaczleweli/febug

NAME

febug::controlled_socket, **febug::wrapper**, **febug::formatters**, **febug::debug_handler()** — User-space debugfs ABI wrapper library for C++

SYNOPSIS

```
#include <libfebug.hpp>
c++ -lfebug++ ...

#define FEBUG_DONT   0
#define FEBUG_SOCKET "/var/run/febug.sock"
#define FEBUG_SIGNUM SIGUSR2

getenv("FEBUG_DONT");
getenv("FEBUG_SOCKET");

struct febug::controlled_socket;
const febug::controlled_socket febug::global_controlled_socket;

struct febug::wrapper;

febug::wrapper::wrapper(const T & data, const char * name, ...);

febug::wrapper::wrapper(const T & data, uint8_t signal, const char * name,
    ...);

febug::wrapper::wrapper(const T & data, uint8_t signal, const char * name,
    va_list ap);

std::map<size_t, void (*)(int, size_t)> febug::formatters;

void febug::debug_handler(int);
```

DESCRIPTION

Simplifies writing C++ programs debuggable with `febug`(8) by presenting a high-level interface to `febug-abi`(5).

There are three compile-time macros that allow customising **libfebug++** behaviour:

- `FEBUG_DONT` If non-zero, all symbols become **static**, functions turn into no-ops, and therefore no symbols from `libfebug++.a|.so` are imported at link-time; this is intended as a way to easily disable `febug`(8) integration completely on release builds.
- `FEBUG_SIGNUM` The signal to request from `febug`(8) when using **febug_wrap**(). Defaults to `SIGUSR2`.
- `FEBUG_SOCKET` The path to connect to `febug`(8) on. Defaults to `/var/run/febug.sock`.

There are two environment variables that allow a user to customise its behaviour:

- `FEBUG_DONT` If set, don't try to connect to `febug`(8), so all library functions become no-ops.
- `FEBUG_SOCKET` If set, use its value instead of the built-in `FEBUG_SOCKET` to connect to `febug`(8).

The *febug::controlled_socket* structure is defined as follows:

```
struct febug::controlled_socket {
    int fd = -1;
    inline operator int() const noexcept { return this->fd; }
    controlled_socket(const char * path = FEBUG_SOCKET) noexcept;
    ~controlled_socket() noexcept;
};
```

There is a global instance at *febug::global_controlled_socket* which, if *path* (FEBUG_SOCKET) isn't the null pointer, attempts to connect to febug(8) and will set *fd*, if successful. Similarly, destroying it will hang up and reset *fd*.

The program needs to install **febug::debug_handler**() (or a wrapper around it) as the signal handler for FEBUG_SIGNUM (and any other signals, if different ones are explicitly requested); if notifications are disabled (by requesting SIGKILL), some event loop that answers on *febug::global_controlled_socket* must be in place. It's a no-op if *febug::global_controlled_socket* is **-1**.

The program should register handlers for types of variables it wishes to handle by adding entries to *febug::formatters* — the key is typeid(std::decay_t<T>).hash_code(), so if this yields different results for two types that should have the same handler, multiple entries need to be registered. If no handler was registered for a type, **febug::debug_handler**() will instead return a generic "not found" message. The handler takes the write end of the pipe as the first argument, and the variable ID as the second; it shouldn't close the pipe, as that is done by **febug::debug_handler**() regardless, and the program would then run the risk of closing another file with the same descriptor simultaneously opened by another thread.

The *febug::wrapper* structure is defined as follows:

```
template <class T>
struct febug::wrapper {
    const T * data;
    wrapper(const T & data, const char * name, ...) noexcept;
    wrapper(const T & data, uint8_t signal, const char * name, ...) noexcept;
    wrapper(const T & data, uint8_t signal, const char * name, va_list ap) noexc
    ~wrapper() noexcept;
};
```

If the program wishes to debug a variable, it should construct a *febug::wrapper* referencing it; the constructor will send a **febug_message** with the type corresponding to typeid(std::decay_t<T>).hash_code(), ID corresponding to the pointer to *data*, signal being either specified or defaulting to FEBUG_SIGNUM, and name formatted according to printf(3). The destructor will send a **stop_febug_message**. Both become no-ops if *febug::global_controlled_socket* is **-1**.

EXAMPLES

The following program sorts a *std::vector<int>* with **std::sort**() but waits a second between each comparison; the vector and the amount of comparisons can be inspected via a febug(8) mount:

```
// SPDX-License-Identifier: MIT


#include <libfebug.hpp>

#include <algorithm>
#include <cstring>
#include <errno.h>
#include <unistd.h>
#include <vector>


int main() {
  if(febug::global_controlled_socket != -1) {
    febug::formatters.emplace(
        typeid(std::vector<int>).hash_code(), [](int retpipe, std::size_t vid) {
```

```
        const std::vector<int> & data = *(const std::vector<int> *)vid;
        for(auto num : data)
          dprintf(retpipe, "%d ", num);
        write(retpipe, "\n", 1);
      });
  febug::formatters.emplace(
      typeid(std::size_t).hash_code(), [](int retpipe, std::size_t vid) {
        const std::size_t & data = *(const std::size_t *)vid;
        dprintf(retpipe, "%zu\n", data);
      });
}


struct sigaction handler {};
handler.sa_handler = febug::debug_handler;
if(sigaction(FEBUG_SIGNUM, &handler, nullptr) == -1)
  std::fprintf(stderr, "sigaction: %s\n", std::strerror(errno));


{
  std::vector<int> data{-1, -2, -3, 0, 1, 2, 3, -1, -2, -3, 0, 1, 2, 3,
                        -1, -2, -3, 0, 1, 2, 3, -1, -2, -3, 0, 1, 2, 3,
                        -1, -2, -3, 0, 1, 2, 3, -1, -2, -3, 0, 1, 2, 3};
  std::size_t comparisons_done{};
  febug::wrapper data_w{data, "cool_data"};
  febug::wrapper comparisons_done_w{comparisons_done, "comparisons"};

  std::sort(data.begin(), data.end(), [&](auto lhs, auto rhs) {
    sleep(1);
    ++comparisons_done;
    return lhs < rhs;
  });
}

  sleep(2);
}
```

SEE ALSO

`febug-abi`(5) — the ABI wrapped by this library.
`libfebug`(3) and `libfebug,rs`(3) — equivalent C and Rust libraries.

SPECIAL THANKS

To all who support further development, in particular:

- ThePhD
- Embark Studios
- Jasper Bekkers

REPORTING BUGS

febug tracker:

https://todo.sr.ht/~nabijaczleweli/febug

febug mailing list: ⟨~nabijaczleweli/febug@lists.sr.ht⟩, archived at
https://lists.sr.ht/~nabijaczleweli/febug

NAME

febug::start(), **febug::Wrapper**, **febug::Wrappable**, **febug::StaticWrappable**, **febug::GLOBAL_CONTROLLED_SOCKET**, **febug::FORMATTERS** — User-space debugfs ABI wrapper library for Rust

SYNOPSIS

```
[dependencies]
febug = "0.2.0"
```

```
env!("FEBUG_DONT")?
env!("FEBUG_SOCKET") = "/var/run/febug.sock"
env!("FEBUG_SIGNUM") = SIGUSR2
```

```
env::var("FEBUG_DONT");
env::var("FEBUG_SOCKET");
```

static GLOBAL_CONTROLLED_SOCKET:
AtomicI32 = **-1**;

fn febug::start();

fn febug::start_raw(*path: &[u8]*);

extern "C" fn debug_handler(*_: c_int*);

bool fn febug::install_handler();

bool fn febug::install_handler_signal(*signal: u8*);

fn febug::end();

static febug::FORMATTERS:
Lazy<Mutex<BTreeMap<TypeId, fn(&mut File, usize)>>>;

trait febug::StaticWrappable: *'static*;

Wrapper<Self> fn febug::StaticWrappable::wrap(*&self*, *name: Arguments<'_>*);

Wrapper<Self> fn febug::StaticWrappable::wrap_signal(*&self*, *signal: u8*, *name: Arguments<'_>*);

trait febug::Wrappable;

Wrapper<Self> fn febug::Wrappable::wrap(*&self*, *tp: u64*, *name: Arguments<'_>*);

Wrapper<Self> fn febug::Wrappable::wrap_signal(*&self*, *tp: u64*, *signal: u8*, *name: Arguments<'_>*);

struct febug::Wrapper<T> { / ... */ }*;

Wrapper<T> fn febug::Wrapper::new(*&self*, *tp: u64*, *data:argument*, *name: fmt::Arguments*, *name: Arguments<'_>*);

Wrapper<T> fn febug::Wrapper::new_signal(*&self*, *tp: u64*, *data:argument*, *signal: u8*, *name: fmt::Arguments*, *name: Arguments<'_>*);

struct febug::abi::FebugMessage;
struct febug::abi::StopFebugMessage;
struct febug::abi::AttnFebugMessage;

DESCRIPTION

Simplifies writing Rust programs debuggable with febug(8) by presenting a high-level interface to febug-abi(5).

There are three compile-time environment variables that allow customising **libfebug.rs** behaviour:

- FEBUG_DONT — If set, all functions turn into no-ops; this is intended as a way to easily disable febug(8) integration completely on release builds.
- FEBUG_SIGNUM — The signal to request from febug(8) when using **febug_wrap**(). Defaults to SIGUSR2.
- FEBUG_SOCKET — The path to connect to febug(8) on. Defaults to /var/run/febug.sock.

There are two environment variables that allow a user to customise its behaviour:

- FEBUG_DONT — If set, don't try to connect to febug(8), so all library functions become no-ops.
- FEBUG_SOCKET — If set, use its value instead of the built-in FEBUG_SOCKET to connect to febug(8).

To be debugged, a program needs to, first, call **febug::start_raw**() (likely via **febug::start**(), which simply passes b"/var/run/febug.sock" thereto) to connect to febug(8), which, if successful, will set *febug::GLOBAL_CONTROLLED_SOCKET* to the connection's file descriptor.

The program needs to install **febug::debug_handler**() (or a wrapper around it) as the signal handler for FEBUG_SIGNUM (and any other signals, if different ones are explicitly requested); if notifications are disabled (by requesting SIGKILL), some event loop that answers on *febug::GLOBAL_CONTROLLED_SOCKET* must be in place. It's a no-op if *febug::GLOBAL_CONTROLLED_SOCKET* is **-1**. A convenience **febug::install_handler**() function is provided, doing just that, and returning **true** if the handler was installed.

The program should register handlers for types of variables it wishes to handle by adding entries to *febug::FORMATTERS*. If no handler was registered for a type, or the lock was poisoned, **febug::debug_handler**() will write a generic "not found" message. The key is *std::any::TypeId* corresponding to the debugged type; if your type is not *'static*, lie here. The handler takes the write end of the pipe as the first argument, and the variable ID as the second. It's a no-op if *febug::GLOBAL_CONTROLLED_SOCKET* is **-1**.

At any time, when the program wishes to expose a variable, it can construct a *febug::Wrapper* (likely via one of the convenience *febug::StaticWrappable* or *febug::Wrappable* traits), which will send a **febug_message** with the specified type and signal numbers (defaulting to SIGUSR2), ID equal to the address of the data argument, and name formatted. It's a no-op if *febug::GLOBAL_CONTROLLED_SOCKET* is **-1**.

When dropped, *febug::Wrapper* will send a **stop_febug_message**. It's a no-op if *febug::GLOBAL_CONTROLLED_SOCKET* is **-1**.

When it wishes to stop being debugged, the program may call **febug::end**() which will shut and reset *febug::GLOBAL_CONTROLLED_SOCKET*. The program may omit this if it'd be the last thing it did before exiting, since the kernel will close all file descriptors and free all mappings anyway.

EXAMPLES

The following program spawns 10 threads, each successive one sorting a longer subsection of a *String*, but waits a quarter-second between each comparison; the *String* for each thread and the amount of comparisons can be inspected via a febug(8) mount:

```
// SPDX-License-Identifier: MIT


extern crate febug;

use febug::StaticWrappable;
use std::time::Duration;
use std::any::TypeId;
use std::io::Write;
use std::thread;


fn main() {
    febug::start();
    if febug::install_handler() {
        febug::FORMATTERS.lock()
            .unwrap()
            .insert(TypeId::of::<String>(), |of, vid| {
                let data = unsafe { &*(vid as *const String) };

                let _ = of.write_all(data.as_bytes());
                let _ = of.write_all(b"\n");
            });
    }


    let threads = (0..10)
        .map(|i| {
            thread::spawn(move || {
                let mut sorteing = "The quick red fox jumps over the lazy brown \
                                    dog... tHE QUICK RED FOX JUMPS OVER THE \
                                    LAZY BROWN DOG!!"
                                       [0..(i + 1) * 10]
                    .to_string();
                let _sorteing_w = sorteing.wrap(format_args!("cool_data_{}", i));

                unsafe { sorteing.as_bytes_mut() }.sort_unstable_by(|a, b| {
                    thread::sleep(Duration::from_millis(250));
                    a.cmp(b)
                });

                thread::sleep(Duration::from_secs(2));
            })
        })
        .collect::<Vec<_>>();
    for t in threads {
        let _ = t.join();
    }
}
```

SEE ALSO

`febug-abi`(5) — the ABI wrapped by this library.
`libfebug`(3) and `libfebug++`(3) — equivalent C and C++ libraries.

SPECIAL THANKS

To all who support further development, in particular:

- ThePhD
- Embark Studios
- Jasper Bekkers

REPORTING BUGS

febug tracker:
https://todo.sr.ht/~nabijaczleweli/febug

febug mailing list: ⟨`~nabijaczleweli/febug@lists.sr.ht`⟩, archived at
https://lists.sr.ht/~nabijaczleweli/febug

NAME

`struct febug_message`, **`struct stop_febug_message`**, **`struct attn_febug_message`** — User-space debugfs ABI

SYNOPSIS

`#include <febug-abi.h>`

`struct febug_message`;
`struct stop_febug_message`;
`struct attn_febug_message`;

DESCRIPTION

The febug ABI consists of two messages sent from the program wishing to be debugged to `febug`(8), and one sent from `febug`(8) to the program.

To be debugged, the program must create a socket with `socket(AF_UNIX, SOCK_SEQPACKET, 0)` and `connect`(2) to the appropriate end-point (`/var/run/febug.sock`, conventionally). The program *must* then wait to receive a place-holder **`attn_febug_message`** before continuing, to ensure that the other end of its socket is correctly configured to receive credentials. Sending the first message will pass along client credentials; sending the null message is a valid way to trigger this. After `febug`(8) receives credentials, a directory corresponding to the debugged process' PID will be created in the filesystem.

All messages must be sent in a single `send`(2) or `sendmsg`(2) call, specifying the exact size of the message, as that's what's used to differentiate between different messages. `febug`(8) will ignore messages (whose sizes) it does not recognise.

Afterward, for each variable of interest, the process should send a 4096-byte **`febug_message`**, defined as follows:

```
struct [[packed]] febug_message {
    uint64_t variable_id;
    uint64_t variable_type;
    uint8_t  signal;
    char     name[/* Enough to bring the overall size to 4096. */];
};
```

Wherein

variable_id	is the locally unique identifier of the variable (e.g. a pointer to that variable).
variable_type	is arbitrary user data, passed back to the program verbatim (e.g. a function pointer to a formatter).
signal	is the signal to send to the program when a variable is to be read (see below).
name	is the globally unique name of this variable — a NUL terminator is respected but not required if the name truly spans to the final byte. If the same as one of the already-present variables, it will be overridden.

When `febug`(8) receives a **`febug_message`**, it creates a file under the process' directory. When that file is opened, `febug`(8) will:

1. send the process an **`attn_febug_message`** with a single file descriptor via `SCM_RIGHTS` auxilliary data (see `cmsg`(3)) representing the write end of a pipe.
2. `kill`(2) the process with the signal from the *signal* field if it wasn't `SIGKILL`.

Note, that the sent file descriptor *must* be closed by the program when it's done serialising the variable, and therefore, if the process opts not to receive a signal, it *must* handle the message through some other mechanism.

`attn_febug_message` is 16 bytes, and defined as follows:

```
struct [[packed]] attn_febug_message {
    uint64_t variable_id;
    uint64_t variable_type;
};
```

Both fields correspond to the ones sent in the **`febug_message`** that installed the variable.

The process may receive any number of **`attn_febug_message`**s until it sends an 8-byte **`stop_febug_message`**, defined as follows:

```
struct [[packed]] stop_febug_message {
    uint64_t variable_id;
};
```

Upon receipt, the corresponding variable, if any, is unlinked.

When the process' end of the socket is closed, all extant variables are freed, and the process' directory is removed.

SEE ALSO

`libfebug`(3), `libfebug++`(3), and `libfebug.rs`(3) — libraries that wrap this ABI.

SPECIAL THANKS

To all who support further development, in particular:

- ThePhD
- Embark Studios
- Jasper Bekkers

REPORTING BUGS

febug tracker:

https://todo.sr.ht/~nabijaczleweli/febug

febug mailing list: ⟨`~nabijaczleweli/febug@lists.sr.ht`⟩, archived at

https://lists.sr.ht/~nabijaczleweli/febug

NAME

`febug` — User-space debugfs filesystem driver

SYNOPSIS

febug_start() [**-h**| **--help**] [**-V**| **--version**] [**-d**] [*librefuse options*] *mountpoint*

DESCRIPTION

Mounts a filesystem at *mountpoint* that allows programs to register themselves and expose variables to be (relatively) non-intrusively inspected at run-time, inspired by Linux's *debugfs:*

https://www.kernel.org/doc/html/latest/filesystems/debugfs.html

filesystem.

See febug-abi(8) for implementation details, and the **EXAMPLES** section for an example debug session.

OPTIONS

-h, **--help** and **-V**, **--version** are self-explanatory.

-d enables debug output from both **febug_start()** and **librefuse**.

febug_start() passes all arguments (which have to, therefore, include *mountpoint*) to fuse_main(3), with **-f** (foreground) and **-o default_permissions** appended. If run with effective UID of **0**, it also appends **-o allow_other**.

ENVIRONMENT

FEBUG_SOCKET the socket at which to listen for programs, or /var/run/febug.sock by default.

EXAMPLES

```
# service febug start
$ mount |grep febug
/dev/puffs on /var/run/febug type puffs|refuse:febug (nodev, nosuid)
$ ./out/examples/vector-sort &
[1] 1409
$ LD_LIBRARY_PATH=out ./out/examples/string-qsort &
[2] 1410
$ ls /var/run/febug/
1409 1410
$ ls -l /var/run/febug/
dr-xr-x--- 4 nabijaczleweli users 0 Jan 15 19:52 1409
dr-xr-x--- 3 nabijaczleweli users 0 Jan 15 19:52 1410
$ ls /var/run/febug/1409/
comparisons cool_data
$ cat /var/run/febug/1409/*
24
-3 -2 -3 -2 -3 -2 3 -1 -2 -3 0 1 2 3 -1 -2 -3 0 1 2 3 -1 -2 -3 0 1 2 3 -1 2 1 0 1 2 3
$ cat /var/run/febug/1409/*
45
-3 -2 -3 -2 -3 -2 -3 -2 -2 -3 -3 -2 -1 3 -1 1 0 0 1 2 3 2 -1 3 0 1 2 3 -1 2 1 0 1 2 3
$ grep . /var/run/febug/*/*
/var/run/febug/1409/comparisons:71
/var/run/febug/1409/cool_data:-3 -3 -3 -3 -3 -3 -2 -2 -2 -2 -2 -2 -1 3 -1 1 0 0 1 2 3
/var/run/febug/1410/cool_data:3012987654ACEFOLJKODNIEMIGHBPPbdWwnfTpXQcreRlVvUSitZQWj
$ kill %1
$ ls /var/run/febug/
1410
```

SEE ALSO

`febug-abi`(5) — the ABI used to connect with this filesystem.

`libfebug`(3), `libfebug++`(3), `libfebug.rs`(3) — libraries wrapping said ABI.

SPECIAL THANKS

To all who support further development, in particular:

- ThePhD
- Embark Studios
- Jasper Bekkers

REPORTING BUGS

febug tracker:
https://todo.sr.ht/~nabijaczleweli/febug

febug mailing list: ⟨`~nabijaczleweli/febug@lists.sr.ht`⟩, archived at
https://lists.sr.ht/~nabijaczleweli/febug