sd_is_fifo

- Description
- Return Value
- Notes
- History
- See Also

systemd

sd_is_fifo

3

sd_is_fifo

sd_is_socket

sd_is_socket_inet

sd_is_socket_unix

sd_is_socket_sockaddr

sd_is_mq

sd_is_special

Check the type of a file descriptor

#include <systemd/sd-daemon.h>

int

sd_is_fifo

int

fd

const char *

path

int

sd_is_socket

int

fd

int

family

int

type

int

listening

int

sd_is_socket_inet

int

fd

int

family

int

type

int

listening

uint16_t

port

int

sd_is_socket_sockaddr

int

fd

int

type

const struct sockaddr *

addr

unsigned

addr_len

int

listening

int

sd_is_socket_unix

int

fd

int

type

int

listening

const char *

path

size_t

length

int

sd_is_mq

int

fd

const char *

path

int

sd_is_special

int

fd

const char *

path

Description

`sd_is_fifo()` may be called to check whether the specified file descriptor refers to a FIFO or pipe. If the `path` parameter is not `NULL`, it is checked whether the FIFO is bound to the specified file system path.

`sd_is_socket()` may be called to check whether the specified file descriptor refers to a socket. If the `family` parameter is not `AF_UNSPEC`, it is checked whether the socket is of the specified family (`AF_UNIX`, `AF_INET`, …). If the `type` parameter is not 0, it is checked whether the socket is of the specified type (`SOCK_STREAM`, `SOCK_DGRAM`, …). If the `listening` parameter is positive, it is checked whether the socket is in accepting mode, i.e. `listen()` has been called for it. If `listening` is 0, it is checked whether the socket is not in this mode. If the parameter is negative, no such check is made. The `listening` parameter should only be used for stream sockets and should be set to a negative value otherwise.

`sd_is_socket_inet()` is similar to `sd_is_socket()`, but optionally checks the IPv4 or IPv6 port number the socket is bound to, unless `port` is zero. For this call `family` must be passed as either `AF_UNSPEC`, `AF_INET`, or `AF_INET6`.

`sd_is_socket_sockaddr()` is similar to `sd_is_socket_inet()`, but checks if the socket is bound to the address specified by `addr`. The family specified by `addr` must be either `AF_INET` or `AF_INET6` and

`addr_len` must be large enough for that family. If `addr` specifies a non-zero port, it is also checked if the socket is bound to this port. In addition, for IPv6, if `addr` specifies non-zero sin6_flowinfo or sin6_scope_id, it is checked if the socket has the same values.

`sd_is_socket_unix()` is similar to `sd_is_socket()` but optionally checks the `AF_UNIX` path the socket is bound to, unless the `path` parameter is `NULL`. For normal file system `AF_UNIX` sockets, set the `length` parameter to 0. For Linux abstract namespace sockets, set the `length` to the size of the address, including the initial 0 byte, and set the `path` to the initial 0 byte of the socket address.

`sd_is_mq()` may be called to check whether the specified file descriptor refers to a POSIX message queue. If the `path` parameter is not `NULL`, it is checked whether the message queue is bound to the specified name.

`sd_is_special()` may be called to check whether the specified file descriptor refers to a special file. If the `path` parameter is not `NULL`, it is checked whether the file descriptor is bound to the specified filename. Special files in this context are character device nodes and files in `/proc/` or `/sys/`.

Return Value

On failure, these calls return a negative errno-style error code. If the file descriptor is of the specified type and bound to the specified address, a positive return value is returned, otherwise zero.

Notes

Internally, these functions use a combination of `fstat(2)` and `getsockname(2)` to check the file descriptor type and where it is bound to.

History

`sd_is_special()` was added in version 209.

`sd_is_socket_sockaddr()` was added in version 233.

See Also

`systemd(1)`, `sd-daemon(3)`, `sd_listen_fds(3)`, `systemd.service(5)`, `systemd.socket(5)`, `ip(7)`, `ipv6(7)`, `unix(7)`, `fifo(7)`, `mq_overview(7)`, `socket(7)`.

□□□ #2
□ □□□ □□□ 2026-10-31 14:57:39 UTC
□ □□□ □□□ 2026-10-31 14:57:40 UTC