sd_bus_default

- Description
- Reference ownership
- Return Value
  - Errors
- History
- See Also

systemd

sd_bus_default

3

sd_bus_default

sd_bus_default_user

sd_bus_default_system

sd_bus_open

sd_bus_open_with_description

sd_bus_open_user

sd_bus_open_user_with_description

sd_bus_open_user_machine

sd_bus_open_system

sd_bus_open_system_with_description

sd_bus_open_system_remote

sd_bus_open_system_machine

Acquire a connection to a system or user bus

#include <systemd/sd-bus.h>

int

sd_bus_default

sd_bus **

bus

int

sd_bus_default_user

sd_bus **

bus

int

sd_bus_default_system

sd_bus **

bus

int

sd_bus_open

sd_bus **

bus

int

sd_bus_open_with_description

sd_bus **

bus

const char *

description

int

sd_bus_open_user

sd_bus **

bus

int

sd_bus_open_user_with_description

sd_bus **

bus

const char *

description

int

sd_bus_open_user_machine

sd_bus **

bus

const char *

machine

int

sd_bus_open_system

sd_bus **

bus

int

sd_bus_open_system_with_description

sd_bus **

bus

const char *

description

int

sd_bus_open_system_remote

sd_bus **

bus

const char *

host

int

sd_bus_open_system_machine

sd_bus **

bus

const char *

machine

Description

`sd_bus_default()` acquires a bus connection object to the user bus when invoked from within a user slice (any session under `user-*.slice`, e.g.: `user@1000.service`), or to the system bus otherwise. The connection object is associated with the calling thread. Each time the function is invoked from the same thread, the same object is returned, but its reference count is increased by one, as long as at least one reference is kept. When the last reference to the connection is dropped (using the `sd_bus_unref(3)` call), the connection is terminated. Note that the connection is not automatically terminated when the associated thread ends. It is important to drop the last reference to the bus connection explicitly before the thread ends, as otherwise, the connection will leak. Also, queued but unread or unwritten messages keep the bus referenced, see below.

`sd_bus_default_user()` returns a user bus connection object associated with the calling thread. `sd_bus_default_system()` is similar, but connects to the system bus. Note that `sd_bus_default()` is identical to these two calls, depending on the execution context.

`sd_bus_open()` creates a new, independent bus connection to the user bus when invoked in user context, or the system bus otherwise. `sd_bus_open_user()` is similar, but connects only to the user bus. `sd_bus_open_system()` does the same, but connects to the system bus. In contrast to `sd_bus_default()`, `sd_bus_default_user()`, and `sd_bus_default_system()`, these calls return new, independent connection objects that are not associated with the invoking thread and are not shared between multiple invocations. It is recommended to share connections per thread to efficiently make use the available resources. Thus, it is recommended to use `sd_bus_default()`, `sd_bus_default_user()` and `sd_bus_default_system()` to connect to the user or system buses.

`sd_bus_open_with_description()`, `sd_bus_open_user_with_description()`, and `sd_bus_open_system_with_description()` are similar to `sd_bus_open()`, `sd_bus_open_user()`, and `sd_bus_open_system()`, but allow a description string to be set, see `sd_bus_set_description(3)`. `description` may be `NULL`, in which case this function is equivalent to `sd_bus_open()`. This description string is used in log messages about the bus object, and including a "name" for the bus makes them easier to understand. Some messages are emitted during bus initialization, hence using this function is preferable to setting the description later with `sd_bus_open_with_description()`. The argument is copied internally and will not be referenced after the function returns.

If the `$DBUS_SESSION_BUS_ADDRESS` environment variable is set (cf. `environ(7)`), it will be used as the address of the user bus. This variable can contain multiple addresses separated by `;`. If this variable is not set, a suitable default for the default user D-Bus instance will be used.

If the `$DBUS_SYSTEM_BUS_ADDRESS` environment variable is set, it will be used as the address of the system bus. This variable uses the same syntax as `$DBUS_SESSION_BUS_ADDRESS`. If this variable is not set, a suitable default for the default system D-Bus instance will be used.

`sd_bus_open_system_remote()` connects to the system bus on the specified host using `ssh(1)`. `host` consists of an optional user name followed by the `@` symbol, and the hostname, optionally followed by a `:` and a port, optionally followed by a `/` and a machine name. If the machine name is given, a connection is created to the system bus in the specified container on the remote machine, and otherwise a connection to the system bus on the specified host is created.

Note that entering a container is a privileged operation, and will likely only work for the root user on the remote machine.

`sd_bus_open_system_machine()` connects to the system bus in the specified `machine`, where `machine` is the name of a local container, possibly prefixed by a user name and a separating `@`. If the container name is specified as the special string `.host` the connection is made to the local system. This is useful to connect to the local system bus as specific user, e.g. `foobar@.host` to connect to the local system bus as local user `foobar`. If the `@` syntax is used either the left-hand side or the right-hand side may be omitted (but not both) in which case the local user name or `.host` is implied. If the `@` syntax is not used the connection is always made as root user. See `sd_bus_set_address(3)` for a description of the address syntax, and `machinectl(1)` for more information about the "machine" concept. Note that connections into local containers are only available to privileged processes at this time.

`sd_bus_open_user_machine()` is similar to `sd_bus_open_system_machine()`, but connects to the user bus of the root user, or if the `@` syntax is used, of the specified user.

These calls allocate a bus connection object and initiate the connection to a well-known bus of some form. An alternative to using these high-level calls is to create an unconnected bus object with `sd_bus_new(3)` and to connect it with `sd_bus_start(3)`.

Reference ownership

The functions `sd_bus_open()`, `sd_bus_open_user()`, `sd_bus_open_user_machine()`, `sd_bus_open_system()`, `sd_bus_open_system_remote()`, and `sd_bus_open_system_machine()` return a new connection object and the caller owns the sole reference. When not needed anymore, this reference should be destroyed with `sd_bus_unref(3)`.

The functions `sd_bus_default()`, `sd_bus_default_user()` and `sd_bus_default_system()` do not necessarily create a new object, but increase the connection reference of an existing connection object by one. Use `sd_bus_unref(3)` to drop the reference.

Queued but unwritten/unread messages keep a reference to their bus connection object. For this reason, even if an application dropped all references to a bus connection, it might not get destroyed right away. Until all incoming queued messages are read, and until all outgoing unwritten messages are written, the bus object will stay alive. `sd_bus_flush()` may be used to write all outgoing queued messages so they drop their references. To flush the unread incoming messages, use `sd_bus_close()`, which will also close the bus connection. When using the default bus logic, it is a good idea to first invoke `sd_bus_flush()` followed by `sd_bus_close()` when a thread or process terminates, and thus its bus connection object should be freed.

Normally, slot objects (as created by `sd_bus_add_match(3)` and similar calls) keep a reference to their bus connection object, too. Thus, as long as a bus slot object remains referenced its bus object will remain allocated too. Optionally, bus slot objects may be placed in "floating" mode. When in floating mode the life cycle of the bus slot object is bound to the bus object, i.e. when the bus object is freed the bus slot object is automatically unreferenced too. The floating state of a slot object may be controlled explicitly with `sd_bus_slot_set_floating(3)`, though usually floating bus slot objects are created by passing `NULL` as the `slot` parameter of `sd_bus_add_match()` and related calls, thus indicating that the caller is not directly interested in referencing and managing the bus slot object.

The life cycle of the default bus connection should be the responsibility of the code that creates/owns the thread the default bus connection object is associated with. Library code should neither call `sd_bus_flush()` nor `sd_bus_close()` on default bus objects unless it does so in its own private, self-allocated thread. Library code should not use the default bus object in other threads unless it is clear that the program using it will life cycle the bus connection object and flush and close it before exiting from the thread. In libraries where it is not clear that the calling program will

life cycle the bus connection object, it is hence recommended to use `sd_bus_open_system()` instead of `sd_bus_default_system()` and related calls.

Return Value

On success, these calls return 0 or a positive integer. On failure, these calls return a negative errno-style error code.

Errors

Returned errors may indicate the following problems:

`-EINVAL`
The specified parameters are invalid.

`-ENOMEDIUM`
The requested bus type is not available because of invalid environment (for example the user session bus is not available because `$XDG_RUNTIME_DIR` is not set).

`-ENOMEM`
Memory allocation failed.

`-ESOCKTNOSUPPORT`
The protocol version required to connect to the selected bus is not supported.

In addition, other connection-related errors may be returned. See `sd_bus_send(3)`.

History

`sd_bus_default()`, `sd_bus_default_user()`, `sd_bus_default_system()`, `sd_bus_open()`, `sd_bus_open_user()`, `sd_bus_open_system()`, `sd_bus_open_system_remote()`, and `sd_bus_open_system_machine()` were added in version 221.

`sd_bus_open_with_description()`, `sd_bus_open_user_with_description()`, and `sd_bus_open_system_with_description()` were added in version 239.

`sd_bus_open_user_machine()` was added in version 248.

See Also

`systemd(1)`, `sd-bus(3)`, `sd_bus_new(3)`, `sd_bus_ref(3)`, `sd_bus_unref(3)`, `sd_bus_close(3)`, `ssh(1)`, `systemd-machined.service(8)`, `machinectl(1)`

□□□ #2
□ □□□ □□□ 2026-10-31 14:48:59 UTC
□ □□□ □□□ 2026-10-31 14:49:00 UTC