

sd_bus_error



- [Description](#)
 - [Reference ownership](#)
 - [Return Value](#)
 - [Errors](#)
 - [Examples](#)
 - [History](#)
 - [See Also](#)
-

systemd

sd_bus_error

3

sd_bus_error

SD_BUS_ERROR_MAKE_CONST

SD_BUS_ERROR_NULL

sd_bus_error_free

sd_bus_error_set

sd_bus_error_setf

sd_bus_error_setfv

sd_bus_error_set_const

sd_bus_error_set_errno

sd_bus_error_set_errnof

sd_bus_error_set_errnofv

sd_bus_error_get_errno

sd_bus_error_copy

sd_bus_error_move

sd_bus_error_is_set

sd_bus_error_has_name

sd_bus_error_has_names_sentinel

sd_bus_error_has_names

sd-bus error handling

```
#include <systemd/sd-bus.h>
```

```
typedef struct { const char *name; const char *message; ... } sd_bus_error;
```

```
SD_BUS_ERROR_MAKE_CONST(name, message)
```

```
SD_BUS_ERROR_NULL
```

void

sd_bus_error_free

sd_bus_error *

e

int

sd_bus_error_set

sd_bus_error *

e

const char *

name

const char *

message

int

sd_bus_error_setf

sd_bus_error *

e

const char *

name

const char *

format

...

int

sd_bus_error_setfv

sd_bus_error *

e

const char *

name

const char *

format

va_list

ap

int

sd_bus_error_set_const

sd_bus_error *

e

const char *

name

const char *

message

int

sd_bus_error_set_errno

sd_bus_error *

e

int

error

int

sd_bus_error_set_errnof

sd_bus_error *

e

int

error

const char *

format

...

int

sd_bus_error_set_errnofv

sd_bus_error *

e

int

error

const char *

format

va_list

ap

int

sd_bus_error_get_errno

const sd_bus_error *

e

int

sd_bus_error_copy

sd_bus_error *

dst

const sd_bus_error *

e

int

sd_bus_error_move

sd_bus_error *

dst

sd_bus_error *

e

int

sd_bus_error_is_set

const sd_bus_error *

e

int

```
sd_bus_error_has_name
```

```
const sd_bus_error *
```

```
e
```

```
const char *
```

```
name
```

```
int
```

```
sd_bus_error_has_names_sentinel
```

```
const sd_bus_error *
```

```
e
```

```
...
```

```
#define sd_bus_error_has_names(e, ...) sd_bus_error_has_names_sentinel(e, ..., NULL)
```

Description

The `sd_bus_error` structure carries information about a D-Bus error condition, or lack thereof. The functions described below may be used to set and query fields in this structure.

- The `name` field contains a short identifier of an error. It should follow the rules for error names described in the D-Bus specification, subsection [Valid D-Bus Names](#). A number of common, standardized error names are described in `sd-bus-errors(3)`, but additional domain-specific errors may be defined by applications.
- The `message` field usually contains a human-readable string describing the details, but might be `NULL`.

An unset `sd_bus_error` structure should have both fields initialized to `NULL`, and signifies lack of an error, i.e. success. Assign `SD_BUS_ERROR_NULL` to the structure in order to initialize both fields to `NULL`. When no longer necessary, resources held by the `sd_bus_error` structure should be destroyed with `sd_bus_error_free()`.

`sd_bus_error_set()` sets an error structure to the specified name and message strings. The strings will be copied into internal, newly allocated memory. It is essential to free the contents again when they are not required anymore (see above). Do not use this call on error structures that have already been set. If you intend to reuse an error structure, free the old data stored in it with `sd_bus_error_free()` first.

`sd_bus_error_set()` will return an `errno`-like value (see `errno(3)`) determined from the specified error name `name`. If `name` is `NULL`, it is assumed that no error occurred, and `0` is returned. If `name` is nonnull, a negative value is always returned. If `e` is `NULL`, no error structure is initialized, but `name` is still converted into an `errno`-style value.

Various well-known D-Bus errors are converted to well-known `errno` counterparts, and the other ones to `-EIO`. See `sd-bus-errors(3)` for a list of well-known error names. Additional error mappings may be defined with `sd_bus_error_add_map(3)`.

`sd_bus_error_set()` is designed to be conveniently used in a `return` statement. If `message` is `NULL`, no message is set. This call can fail if no memory may be allocated for the name and message strings, in which case an `SD_BUS_ERROR_NO_MEMORY` error will be set instead and `-ENOMEM` returned.

`sd_bus_error_setf()` and `sd_bus_error_setfv()` are similar to `sd_bus_error_set()`, but take a `printf(3)` format string and corresponding arguments to generate the message field.

`sd_bus_error_setf()` uses variadic arguments, and `sd_bus_error_setfv()` accepts the arguments as a `va_arg(3)` parameter list.

`sd_bus_error_set_const()` is similar to `sd_bus_error_set()`, but the string parameters are not copied internally, and must hence remain constant and valid for the lifetime of `e`. Use this call to avoid memory allocations when setting error structures. Since this call does not allocate memory, it will not fail with an out-of-memory condition as `sd_bus_error_set()` may, as described above.

Alternatively, the `SD_BUS_ERROR_MAKE_CONST()` macro may be used to generate a literal, constant bus error structure on-the-fly.

`sd_bus_error_set_errno()` will immediately return `0` if the specified error parameter `error` is `0`. Otherwise, it will set `name` from an `errno`-like value that is converted to a D-Bus error.

`strerror_r(3)` will be used to set `message`. Well-known D-Bus error names will be used for `name` if applicable, otherwise a name in the `System.Error.` namespace will be generated. The sign of the specified error number is ignored and the absolute value is used implicitly. If the specified error `error` is non-zero, the call always returns a negative value, for convenient usage in `return` statements. This call might fail due to lack of memory, in which case an `SD_BUS_ERROR_NO_MEMORY` error is set instead, and `-ENOMEM` is returned.

`sd_bus_error_set_errnof()` and `sd_bus_error_set_errnofv()` are similar to `sd_bus_error_set_errno()`, but in addition to `error`, take a `printf(3)` format string and corresponding arguments. The message field will be generated from `format` and the arguments. `sd_bus_error_set_errnof()` uses variadic arguments, and `sd_bus_error_set_errnofv()` accepts the arguments as a `va_arg(3)` parameter list.

`sd_bus_error_get_errno()` converts the `name` field of an error structure to an `errno`-like (positive) value using the same rules as `sd_bus_error_set()`. If `e` is `NULL`, `0` will be returned.

`sd_bus_error_copy()` will initialize `dst` using the values in `e`, if `e` has been set with an error value before. Otherwise, it will return immediately. If the strings in `e` were set using `sd_bus_error_set_const()`, they will be shared. Otherwise, they will be copied. Before this call, `dst` must be unset, i.e. either freshly initialized with `NULL` or reset using `sd_bus_error_free()`.

`sd_bus_error_copy()` generally returns `0` or a negative `errno`-like value based on the input parameter `e`: `0` if it was unset and a negative integer if it was set to some error, similarly to `sd_bus_error_set()`. It may however also return an error generated internally, for example `-ENOMEM` if a memory allocation fails.

`sd_bus_error_move()` is similar to `sd_bus_error_copy()`, but will move any error information from `e` into `dst`, resetting the former. This function cannot fail, as no new memory is allocated. Note that if `e` is not set, `dst` is initialized to `SD_BUS_ERROR_NULL`. Moreover, if `dst` is `NULL` no operation is executed on it and resources held by `e` are freed and reset. Returns a converted `errno`-like, non-positive error value.

`sd_bus_error_is_set()` will return a non-zero value if `e` is non-`NULL` and an error has been set, `false` otherwise.

`sd_bus_error_has_name()` will return a non-zero value if `e` is non-`NULL` and an error with the same `name` has been set, `false` otherwise.

`sd_bus_error_has_names_sentinel()` is similar to `sd_bus_error_has_name()`, but takes multiple names to check against. The list must be terminated with `NULL`. `sd_bus_error_has_names()` is a macro wrapper around `sd_bus_error_has_names_sentinel()` that adds the `NULL` sentinel automatically.

`sd_bus_error_free()` will destroy resources held by `e`. The parameter itself will not be deallocated, and must be `free(3)`d by the caller if necessary. The function may also be called safely on unset errors (error structures with both fields set to `NULL`), in which case it performs no operation. This call will reset the error structure after freeing the data, so that all fields are set to `NULL`. The structure may be reused afterwards.

Reference ownership

`sd_bus_error` is not reference-counted. Users should destroy resources held by it by calling `sd_bus_error_free()`. Usually, error structures are allocated on the stack or passed in as function parameters, but they may also be allocated dynamically, in which case it is the duty of the caller to `free(3)` the memory held by the structure itself after freeing its contents with `sd_bus_error_free()`.

Return Value

The functions `sd_bus_error_set()`, `sd_bus_error_setf()`, and `sd_bus_error_set_const()` always return `0` when the specified error value is `NULL`, and a negative `errno`-like value corresponding to the `name` parameter otherwise. The functions `sd_bus_error_set_errno()`, `sd_bus_error_set_ennof()` and `sd_bus_error_set_ennofv()`, return `0` when the specified error value is `0`, and a negative `errno`-like value corresponding to the `error` parameter otherwise. If an error occurs internally, one

of the negative error values listed below will be returned. This allows those functions to be conveniently used in a `return` statement, see the example below.

`sd_bus_error_get_errno()` returns `false` when `e` is `NULL`, and a positive `errno` value mapped from `e->name` otherwise.

`sd_bus_error_copy()` and `sd_bus_error_move()` return a negative error value converted from the source error, and zero if the error has not been set. This allows those functions to be conveniently used in a `return` statement, see the example below.

`sd_bus_error_is_set()` returns a non-zero value when `e` and the name field are non-`NULL`, zero otherwise.

`sd_bus_error_has_name()`, `sd_bus_error_has_names()`, and `sd_bus_error_has_names_sentinel()` return a non-zero value when `e` is non-`NULL` and the name field is equal to one of the given names, zero otherwise.

Errors

Return value may indicate the following problems in the invocation of the function itself:

`-EINVAL`

Error was already set in the `sd_bus_error` structure when one the error-setting functions was called.

`-ENOMEM`

Memory allocation failed.

On success, `sd_bus_error_set()`, `sd_bus_error_setf()`, `sd_bus_error_set_const()`, `sd_bus_error_set_errno()`, `sd_bus_error_set_errnof()`, `sd_bus_error_set_errnofv()`, `sd_bus_error_copy()`, and `sd_bus_error_move()` will return a negative converted `errno`-style value, or `0` if the error parameter is `NULL` or unset. D-Bus errors are converted to the integral `errno`-style value, and the mapping mechanism is extensible, see the discussion above. This effectively means that almost any negative `errno`-style value can be returned.

Examples

Using the negative return value to propagate an error

History

`sd_bus_error_free()`, `sd_bus_error_set()`, `sd_bus_error_setf()`, `sd_bus_error_set_const()`, `sd_bus_error_set_errno()`, `sd_bus_error_set_errnof()`, `sd_bus_error_set_errnofv()`, `sd_bus_error_get_errno()`, `sd_bus_error_copy()`, `sd_bus_error_is_set()`, and `sd_bus_error_has_name()` were added in version 221.

`sd_bus_error_move()` was added in version 240.

`sd_bus_error_has_names_sentinel()` was added in version 247.

`sd_bus_error_setfv()` was added in version 252.

See Also

`systemd(1)`, `sd-bus(3)`, `sd-bus-errors(3)`, `sd_bus_error_add_map(3)`, `errno(3)`, `strerror_r(3)`

ⓘ ⓘ #2

ⓘ ⓘ ⓘ 2026-10-31 14:49:53 UTC

ⓘ ⓘ ⓘ 2026-10-31 14:49:54 UTC