sd-id128

- Description
- History
- See Also

systemd

sd-id128

3

sd-id128

SD_ID128_ALLF

SD_ID128_CONST_STR

SD_ID128_FORMAT_STR

SD_ID128_FORMAT_VAL

SD_ID128_MAKE

SD_ID128_MAKE_STR

SD_ID128_MAKE_UUID_STR

SD_ID128_NULL

SD_ID128_UUID_FORMAT_STR

sd_id128_equal

sd_id128_string_equal

sd_id128_in_set

sd_id128_in_set_sentinel

sd_id128_in_setv

sd_id128_is_allf

sd_id128_is_null

sd_id128_t

APIs for processing 128-bit IDs

#include <systemd/sd-id128.h>

`SD_ID128_ALLF`

`SD_ID128_NULL`

`SD_ID128_CONST_STR(id)`

`SD_ID128_FORMAT_STR`

`SD_ID128_FORMAT_VAL(id)`

`SD_ID128_MAKE(v0, v1, v2, v3, v4, v5, v6, v7, v8, v9, vA, vB, vC, vD, vE, vF)`

`SD_ID128_MAKE_STR(v0, v1, v2, v3, v4, v5, v6, v7, v8, v9, vA, vB, vC, vD, vE, vF)`

`SD_ID128_MAKE_UUID_STR(v0, v1, v2, v3, v4, v5, v6, v7, v8, v9, vA, vB, vC, vD, vE, vF)`

`SD_ID128_UUID_FORMAT_STR`

int

sd_id128_equal

sd_id128_t

a

sd_id128_t

b

int

sd_id128_string_equal

const char *

a

sd_id128_t

b

int

sd_id128_is_null

sd_id128_t

id

int

sd_id128_is_allf

sd_id128_t

id

int

sd_id128_in_setv

sd_id128_t

id

va_list

ap

int

sd_id128_in_set_sentinel

sd_id128_t

id

...

SD_ID128_NULL

int

sd_id128_in_set

sd_id128_t

id

...

pkg-config --cflags --libs libsystemd

Description

`sd-id128.h` is part of `libsystemd(3)` and provides APIs to generate, convert, and compare 128-bit ID values. The 128-bit ID values processed and generated by these APIs are a generalization of OSF UUIDs as defined by RFC 4122 but use a simpler string format. These functions impose no structure on the used IDs, much unlike OSF UUIDs or Microsoft GUIDs, but are mostly compatible with those types of IDs.

A 128-bit ID is implemented as the following union type:

```
typedef union sd_id128 {
  uint8_t bytes[16];
  uint64_t qwords[2];
} sd_id128_t;
```

This union type allows accessing the 128-bit ID as 16 separate bytes or two 64-bit words. It is generally safer to access the ID components by their 8-bit array to avoid endianness issues. This union is intended to be passed by value (as opposed to pass-by-reference) and may be directly manipulated by clients.

A couple of macros are defined to denote and decode 128-bit IDs:

`SD_ID128_MAKE()` is used to write a constant ID in source code. A commonly used idiom is to assign a name to an ID using this macro:

```
#define SD_MESSAGE_COREDUMP SD_ID128_MAKE(fc,2e,22,bc,6e,e6,47,b6,b9,07,29,ab,34,a2,50,b1)
```

`SD_ID128_NULL` defines an ID consisting of only `NUL` bytes (i.e. all bits off).

`SD_ID128_ALLF` defines an ID consisting of only `0xFF` bytes (i.e. all bits on).

`SD_ID128_MAKE_STR()` is similar to `SD_ID128_MAKE()`, but creates a `const char*` expression that can be conveniently used in message formats and such:

```
#include <stdio.h>
#define SD_MESSAGE_COREDUMP_STR
SD_ID128_MAKE_STR(fc,2e,22,bc,6e,e6,47,b6,b9,07,29,ab,34,a2,50,b1)

int main(int argc, char **argv) {
  puts("Match for coredumps: MESSAGE_ID=" SD_MESSAGE_COREDUMP_STR);
}
```

`SD_ID128_CONST_STR()` converts constant IDs into constant strings for output. The following example code will output the string "fc2e22bc6ee647b6b90729ab34a250b1":

```
int main(int argc, char *argv[]) {
  puts("Match for coredumps: %s", SD_ID128_CONST_STR(SD_MESSAGE_COREDUMP));
}
```

`SD_ID128_FORMAT_STR` and `SD_ID128_FORMAT_VAL()` is used to format an ID in a `printf(3)` format string, as shown in the following example:

```
int main(int argc, char *argv[]) {
  sd_id128_t id;
  id = SD_ID128_MAKE(ee,89,be,71,bd,6e,43,d6,91,e6,c5,5d,eb,03,02,07);
  printf("The ID encoded in this C file is " SD_ID128_FORMAT_STR ".\n",
SD_ID128_FORMAT_VAL(id));
  return 0;
}
```

`SD_ID128_UUID_FORMAT_STR` and `SD_ID128_MAKE_UUID_STR()` are similar to `SD_ID128_FORMAT_STR` and `SD_ID128_MAKE_STR()`, but include separating hyphens to conform to the "UUID canonical representation". They format the string based on RFC4122 Variant 1 rules, i.e. converting from Big Endian byte order. This matches behaviour of most other Linux userspace infrastructure. It's probably best to avoid UUIDs of other variants, in order to avoid unnecessary ambiguities. All 128-bit IDs generated by the sd-id128 APIs strictly conform to Variant 1 Version 4 UUIDs, as per RFC 4122.

`sd_id128_equal()` compares two 128-bit IDs:

```
int main(int argc, char *argv[]) {
  sd_id128_t a, b, c;
  a = SD_ID128_MAKE(ee,89,be,71,bd,6e,43,d6,91,e6,c5,5d,eb,03,02,07);
  b = SD_ID128_MAKE(f2,28,88,9c,5f,09,44,15,9d,d7,04,77,58,cb,e7,3e);
  c = a;
```

```
  assert(sd_id128_equal(a, c));
  assert(!sd_id128_equal(a, b));
  return 0;
}
```

`sd_id128_string_equal()` is similar to `sd_id128_equal()`, but the first ID is formatted as `const char*`. The same restrictions apply as to the first argument of `sd_id128_from_string()`.

`sd_id128_is_null()` checks if an ID consists of only `NUL` bytes:

```
assert(sd_id128_is_null(SD_ID128_NULL));
```

Similarly, `sd_id128_is_allf()` checks if an ID consists of only `0xFF` bytes (all bits on):

```
assert(sd_id128_is_allf(SD_ID128_ALLF));
```

`sd_id128_in_set_sentinel()` takes a list of IDs and returns true if the first argument is equal to any of the subsequent arguments. The argument list is terminated by an `SD_ID128_NULL` sentinel, which must be present.

`sd_id128_in_set()` is a convenience function that takes a list of IDs and returns true if the first argument is equal to any of the subsequent arguments:

```
int main(int argc, char *argv[]) {
  sd_id12_t a = SD_ID128_MAKE(ee,89,be,71,bd,6e,43,d6,91,e6,c5,5d,eb,03,02,07);
  assert(sd_id128_in_set(a, a));
  assert(sd_id128_in_set(a, a, a));
  assert(!sd_id128_in_set(a));
  assert(!sd_id128_in_set(a,
                          SD_ID128_MAKE(f2,28,88,9c,5f,09,44,15,9d,d7,04,77,58,cb,e7,3e)
                          SD_ID128_MAKE(2f,88,28,5f,9c,44,09,9d,d7,15,77,04,bc,85,7e,e3)
                          SD_ID128_ALLF));
  return 0;
}
```

`sd_id128_in_set()` is defined as a macro over `sd_id128_in_set_sentinel()`, adding the `SD_ID128_NULL` sentinel automatically. Since `sd_id128_in_set_sentinel()` uses `SD_ID128_NULL` as the sentinel, `SD_ID128_NULL` cannot be otherwise placed in the argument list.

`sd_id128_in_setv()` is similar to `sd_id128_in_set_sentinel()`, but takes a struct varargs argument.

New randomized IDs may be generated with `systemd-id128(1)`'s `new` command.

See `sd_id128_to_string(3)`, `sd_id128_randomize(3)` and `sd_id128_get_machine(3)` for information about other implemented functions.

History

`sd_id128_equal()`, `sd_id128_string_equal()`, `sd_id128_is_null()`, `sd_id128_is_allf()`, `sd_id128_in_setv()`, `sd_id128_in_set_sentinel()`, and `sd_id128_in_set()` were added in version 252.

See Also

`systemd(1)`, `sd_id128_to_string(3)`, `sd_id128_randomize(3)`, `sd_id128_get_machine(3)`, `printf(3)`, `journalctl(1)`, `sd-journal(3)`, `pkg-config(1)`, `machine-id(5)`

□□□ #1
□ □□□ □□□ 2026-10-31 14:48:36 UTC
□ □□□ □□□ 2026-10-31 14:48:36 UTC