sd_journal_get_data

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

systemd

sd_journal_get_data

3

sd_journal_get_data

sd_journal_get_data_to_fd

sd_journal_enumerate_data

sd_journal_enumerate_available_data

sd_journal_restart_data

SD_JOURNAL_FOREACH_DATA

sd_journal_set_data_threshold

sd_journal_get_data_threshold

SD_JOURNAL_DATA_SKIP_FIELD

Read data fields from the current journal entry

#include <systemd/sd-journal.h>

int

sd_journal_get_data

sd_journal *

j

const char *

field

const void **

data

size_t *

length

int

sd_journal_get_data_to_fd

sd_journal *

j

const char *

field

sd_journal_data_flags_t

flags

int

fd

size_t *

ret_size

int

sd_journal_enumerate_data

sd_journal *

j

const void **

data

size_t *

length

int

sd_journal_enumerate_available_data

sd_journal *

j

const void **

data

size_t *

length

void

sd_journal_restart_data

sd_journal *

j

SD_JOURNAL_FOREACH_DATA

sd_journal *

j

const void *

data

size_t

length

int

sd_journal_set_data_threshold

sd_journal *

j

size_t

sz

int

sd_journal_get_data_threshold

sd_journal *

j

size_t *

sz

Description

`sd_journal_get_data()` gets the data object associated with a specific field from the current journal entry. It takes four arguments: the journal context object, a string with the field name to request, plus a pair of optional pointers to pointer/size variables where the data object and its size shall be stored in. Either pointer may be `NULL`, in which case the corresponding value is not returned (this is supported since version 261). The field name should be an entry field name. Well-known field names are listed in `systemd.journal-fields(7)`, but any field can be specified. The returned data is in a read-only memory map and is only valid until the next invocation of `sd_journal_get_data()`, `sd_journal_enumerate_data()`, `sd_journal_enumerate_available_data()`, or when the read pointer is altered. Note that the data returned will be prefixed with the field name and `=`.

`sd_journal_get_data_to_fd()` is similar to `sd_journal_get_data()`, but writes the result data to the file descriptor specified by the `fd` argument. If the `ret_size` argument is non-`NULL`, the number of bytes written to the file descriptor is stored there. Data is written starting at the `fd`'s current offset, and the `fd` is never rewound. On failure, the `fd` may already contain partially written data.

`sd_journal_enumerate_data()` may be used to iterate through all fields of the current entry. On each invocation the data for the next field is returned. The order of these fields is not defined. The data

returned is in the same format as with `sd_journal_get_data()` and also follows the same life-time semantics.

`sd_journal_enumerate_available_data()` is similar to `sd_journal_enumerate_data()`, but silently skips any fields which may be valid, but are too large or not supported by current implementation.

`sd_journal_restart_data()` resets the data enumeration index to the beginning of the entry. The next invocation of `sd_journal_enumerate_data()` will return the first field of the entry again.

Note that the `SD_JOURNAL_FOREACH_DATA()` macro may be used as a handy wrapper around `sd_journal_restart_data()` and `sd_journal_enumerate_available_data()`.

Note that these functions will not work before `sd_journal_next(3)` (or related call) has been called at least once, in order to position the read pointer at a valid entry.

`sd_journal_set_data_threshold()` may be used to change the data field size threshold for data returned by `sd_journal_enumerate_fields()`. This threshold is a hint only: it indicates that the client program is interested only in the initial parts of the data fields, up to the threshold in size — but the library might still return larger data objects. That means applications should not rely exclusively on this setting to limit the size of the data fields returned, but need to apply an explicit size limit on the returned data as well. This threshold defaults to 64K by default. To retrieve the complete data fields this threshold should be turned off by setting it to 0, so that the library always returns the complete data objects. It is recommended to set this threshold as low as possible since this relieves the library from having to decompress large compressed data objects in full.

`sd_journal_get_data_threshold()` returns the currently configured data field size threshold.

The `flags` argument is a bitmask of zero or more of the following flags:

`SD_JOURNAL_DATA_SKIP_FIELD`
By default, the result data is prefixed with the field name and `=`, matching the format returned by `sd_journal_get_data()`. When this flag is specified, the prefix is omitted and only the field's value is written to the file descriptor.

Return Value

`sd_journal_get_data()` returns 0 on success or a negative errno-style error code. `sd_journal_enumerate_data()` and `sd_journal_enumerate_available_data()` return a positive integer if the next field has been read, 0 when no more fields remain, or a negative errno-style error code. `sd_journal_restart_data()` does not return anything. `sd_journal_set_data_threshold()` and `sd_journal_get_threshold()` return 0 on success or a negative errno-style error code.

Errors

Returned errors may indicate the following problems:

`-EINVAL`
One of the required parameters is `NULL` or invalid. For `sd_journal_get_data()`, only the journal context object and field name are required non-`NULL`. For `sd_journal_enumerate_data()` and `sd_journal_enumerate_available_data()`, `ret_data` and `ret_size` are required as well.

`-ECHILD`
The journal object was created in a different process, library or module instance.

`-EADDRNOTAVAIL`
The read pointer is not positioned at a valid entry; `sd_journal_next(3)` or a related call has not been called at least once.

`-ENOENT`
The current entry does not include the specified field.

`-ENOMEM`
Memory allocation failed.

`-ENOBUFS`
A compressed entry is too large.

`-E2BIG`
The data field is too large for this computer architecture (e.g. above 4 GB on a 32-bit architecture).

`-EPROTONOSUPPORT`
The journal is compressed with an unsupported method or the journal uses an unsupported feature.

`-EBADMSG`
The journal is corrupted (possibly just the entry being iterated over).

`-EIO`
An I/O error was reported by the kernel.

`-EBADF`
An invalid file descriptor is passed to `sd_journal_get_data_to_fd()`.

Notes

Examples

See `sd_journal_next(3)` for a complete example how to use `sd_journal_get_data()`.

Use the `SD_JOURNAL_FOREACH_DATA()` macro to iterate through all fields of the current journal entry:

```
…
int print_fields(sd_journal *j) {
  const void *data;
  size_t length;
  SD_JOURNAL_FOREACH_DATA(j, data, length)
    printf("%.*s\n", (int) length, data);
}
…
```

History

`sd_journal_get_data()`, `sd_journal_enumerate_data()`, `sd_journal_restart_data()`, and `SD_JOURNAL_FOREACH_DATA()` were added in version 187.

`sd_journal_set_data_threshold()` and `sd_journal_get_data_threshold()` were added in version 196.

`sd_journal_enumerate_available_data()` was added in version 246.

`sd_journal_get_data_to_fd()` was added in version 263.

See Also

`systemd(1)`, `systemd.journal-fields(7)`, `sd-journal(3)`, `sd_journal_open(3)`, `sd_journal_next(3)`, `sd_journal_get_realtime_usec(3)`, `sd_journal_query_unique(3)`

□□□ #2
□ □□□ □□□ 2026-10-31 14:57:48 UTC
□ □□□ □□□ 2026-10-31 14:57:49 UTC