



Linux Ubuntu 26.04 Manual Pages on command 'dladdr1.3'

\$ man dladdr1.3

dladdr(3) Library Functions Manual dladdr(3)

NAME

dladdr, dladdr1 - translate address to symbolic information

LIBRARY

Dynamic linking library (libdl, -ldl)

SYNOPSIS

```
#define _GNU_SOURCE
#include <dlfcn.h>

int dladdr(const void *addr, Dl_info *info);

int dladdr1(const void *addr, Dl_info *info, void **extra_info,
            int flags);
```

DESCRIPTION

The function `dladdr()` determines whether the address specified in `addr` is located in one of the shared objects loaded by the calling application. If it is, then `dladdr()` returns information about the shared object and symbol that overlaps `addr`. This information is returned in a `Dl_info` structure:

```
typedef struct {
    const char *dli_fname; /* Pathname of shared object that
                           contains address */
    void *dli_fbase; /* Base address at which shared
                    object is loaded */
    const char *dli_sname; /* Name of symbol whose definition
                           overlaps addr */
};
```

```
void      *dli_saddr; /* Exact address of symbol named
                        in dli_sname */
```

```
} Dl_info;
```

If no symbol matching `addr` could be found, then `dli_sname` and `dli_saddr` are set to `NULL`.

The function `dladdr1()` is like `dladdr()`, but returns additional information via the argument `extra_info`. The information returned depends on the value specified in `flags`, which can have one of the following values:

RTLD_DL_LINKMAP

Obtain a pointer to the link map for the matched file. The `extra_info` argument points to a pointer to a `link_map` structure (i.e., `struct link_map **`), defined in `<link.h>` as:

```
struct link_map {
    ElfW(Addr) l_addr; /* Difference between the
                        address in the ELF file and
                        the address in memory */
    char      *l_name; /* Absolute pathname where
                        object was found */
    ElfW(Dyn) *l_ld; /* Dynamic section of the
                     shared object */
    struct link_map *l_next, *l_prev;
                /* Chain of loaded objects */
    /* Plus additional fields private to the
       implementation */
};
```

RTLD_DL_SYMENT

Obtain a pointer to the ELF symbol table entry of the matching symbol. The `extra_info` argument is a pointer to a symbol pointer: `const ElfW(Sym) **`. The `ElfW()` macro definition turns its argument into the name of an ELF data type suitable for the hardware architecture. For example, on a 64-bit platform, `ElfW(Sym)` yields the data type name `Elf64_Sym`, which is defined in `<elf.h>` as:

```
typedef struct {
    Elf64_Word st_name; /* Symbol name */
    unsigned char st_info; /* Symbol type and binding */
    /* ... other fields ... */
};
```

```

    unsigned char st_other; /* Symbol visibility */

    Elf64_Section st_shndx; /* Section index */

    Elf64_Addr st_value; /* Symbol value */

    Elf64_Xword st_size; /* Symbol size */

} Elf64_Sym;

```

The `st_name` field is an index into the string table.

The `st_info` field encodes the symbol's type and binding. The type can be extracted using the macro `ELF64_ST_TYPE(st_info)` (or `ELF32_ST_TYPE()` on 32-bit platforms), which yields one of the following values:

Value	Description
<code>STT_NOTYPE</code>	Symbol type is unspecified
<code>STT_OBJECT</code>	Symbol is a data object
<code>STT_FUNC</code>	Symbol is a code object
<code>STT_SECTION</code>	Symbol associated with a section
<code>STT_FILE</code>	Symbol's name is filename
<code>STT_COMMON</code>	Symbol is a common data object
<code>STT_TLS</code>	Symbol is thread-local data object
<code>STT_GNU_IFUNC</code>	Symbol is indirect code object

The symbol binding can be extracted from the `st_info` field using the macro `ELF64_ST_BIND(st_info)` (or `ELF32_ST_BIND()` on 32-bit platforms), which yields one of the following values:

Value	Description
<code>STB_LOCAL</code>	Local symbol
<code>STB_GLOBAL</code>	Global symbol
<code>STB_WEAK</code>	Weak symbol
<code>STB_GNU_UNIQUE</code>	Unique symbol

The `st_other` field contains the symbol's visibility, which can be extracted using the macro `ELF64_ST_VISIBILITY(st_info)` (or `ELF32_ST_VISIBILITY()` on 32-bit platforms), which yields one of the following values:

Value	Description
<code>STV_DEFAULT</code>	Default symbol visibility rules
<code>STV_INTERNAL</code>	Processor-specific hidden class
<code>STV_HIDDEN</code>	Symbol unavailable in other modules

SEE ALSO

[dl_iterate_phdr\(3\)](#), [dlopen\(3\)](#), [dlsym\(3\)](#), [ld.so\(8\)](#)

Linux man-pages 6.17

2026-02-08

[dladdr\(3\)](#)