Name ar.h, ar – archive file format

Synopsis

```
#include <ar.h>
```

Description The archive command `ar` is used to combine several files into one. Archives are used mainly as libraries to be searched by the link editor `ld`.

Each archive begins with the archive magic string.

```
#define  ARMAG   "!<arch>\n"    /* magic string */
#define  SARMAG   8             /* length of magic string */
```

Following the archive magic string are the archive file members. Each file member is preceded by a file member header which is of the following format:

```
#define  ARFMAG   "`\n"         /* header trailer string */

struct  ar_hdr                  /* file member header */
{
    char    ar_name[16];        /* '/' terminated file member name */
    char    ar_date[12];        /* file member date */
    char    ar_uid[6]           /* file member user identification */
    char    ar_gid[6]           /* file member group identification */
    char    ar_mode[8]          /* file member mode (octal) */
    char    ar_size[10];        /* file member size */
    char    ar_fmag[2];         /* header trailer string */
};
```

All information in the file member headers is in printable ASCII. The numeric information contained in the headers is stored as decimal numbers (except for *ar_mode* which is in octal). Thus, if the archive contains only printable files, the archive itself is printable.

If the file member name is 15 characters or less, the *ar_name* field contains the name directly, and is terminated by a slash (`/`) and padded with blanks on the right. If the member's name is longer than 15 characters, *ar_name* contains a slash (`/`) followed by a decimal representation of the name's offset in the archive string table described below.

The *ar_date* field is the modification date of the file at the time of its insertion into the archive. Common format archives can be moved from system to system as long as the portable archive command `ar` is used.

Each archive file member begins on an even byte boundary; a newline is inserted between files if necessary. Nevertheless, the size given reflects the actual size of the file exclusive of padding.

Notice there is no provision for empty areas in an archive file.

Each archive that contains object files (see `a.out`(4)) includes an archive symbol table. This symbol table is used by the link editor `ld` to determine which archive members must be loaded during the link edit process. The archive symbol table (if it exists) is always the first file in the archive (but is never listed) and is automatically created and/or updated by `ar`.

The archive symbol table comes in 32 and 64–bit formats. These formats differ only in the width of the integer word used to represent the number of symbols and offsets into the archive. The 32–bit format can be used with archives smaller than 4GB, while the 64–bit format is required for larger archives. The `ar` command selects the symbol table format to use based on the size of the archive it is creating, and will use the smaller format when possible.

The 32–bit archive symbol table has a zero length name, so `ar_name` contains the string "`/`" padded with 15 blank characters on the right. A 64–bit archive symbol table sets `ar_name` to "`/SYM64/`", padded with 9 blank characters to the right.

All integer words in a 32–bit symbol table have four bytes, while all integer words in a 64–bit symbol table have eight bytes. Both formats use the machine-independent encoding shown below. All machines use the encoding described here for the symbol table, even if the machine's natural byte order is different.

```
                    0    1    2    3
0x01020304          01   02   03   04

                    0    1    2    3    4    5    6    7
0x0102030405060708  01   02   03   04   05   06   07   08
```

The contents of an archive symbol table file are as follows, where *wordsize* is 4 bytes for a 32–bit symbol table and 8 bytes for a 64–bit symbol table.

1. The number of symbols. Length: *wordsize* bytes.
2. The array of offsets into the archive file. Length: *wordsize* bytes * "the number of ✘symbols".
3. The name string table. Length: *ar_size* – *wordsize* bytes * ("the number of ✘symbols" + 1).

As an example, the following 32–bit symbol table defines 4 symbols. The archive member at file offset 114 defines *name*. The archive member at file offset 122 defines *object*. The archive member at file offset 426 defines `function` and the archive member at file offset 434 defines *name2*.

Example Symbol Table

```
Offset     +0   +1   +2   +3
          ___________________
 0       |         4         | 4 offset entries
         |___________________|
 4       |        114        | name
         |___________________|
 8       |        122        | object
         |___________________|
12       |        426        | function
         |___________________|
16       |        434        | name2
         |___________________|
20       |  n | a  | m  | e  |
         |____|____|____|____|
```

```
24       | \0 | o  | b  | j  |
         |____|____|____|____|
28       |  e | c  | t  | \0 |
         |____|____|____|____|
32       |  f | u  | n  | c  |
         |____|____|____|____|
36       |  t | i  | o  | n  |
         |____|____|____|____|
40       | \0 | n  | a  | m  |
         |____|____|____|____|
44       |  e | 2  | \0 |    |
         |____|____|____|____|
```

The same example, using a 64–bit symbol table would be rendered as follows. The archive member at file offset 134 defines *name*. The archive member at file offset 142 defines *object*. The archive member at file offset 446 defines `function` and the archive member at file offset 454 defines *name2*.

```
Offset     +0   +1   +2   +3   +4   +5   +6   +7

          _______________________________________
 0       |                   4                   | 4 offset entries
         |_______________________________________|
 8       |                  134                  | name
         |_______________________________________|
16       |                  142                  | object
         |_______________________________________|
24       |                  446                  | function
         |_______________________________________|
32       |                  454                  | name2
         |_______________________________________|
40       |  n | a  | m  | e  | \0 | o  | b  | j  |
         |____|____|____|____|____|____|____|____|
48       |  e | c  | t  | \0 |  f | u  | n  | c  |
         |____|____|____|____|____|____|____|____|
56       |  t | i  | o  | n  | \0 | n  | a  | m  |
         |____|____|____|____|____|____|____|____|
64       |  e | 2  | \0 |    |
         |____|____|____|____|
```

The symbol string table contains exactly as many null terminated strings as there are elements in the offsets array. Each offset from the array is associated with the corresponding name from the string table (in order). The names in the string table are all the defined global symbols found in the common object files in the archive. Each offset is the location of the archive header for the associated symbol.

If some archive member's name is more than 15 bytes long, a special archive member contains a table of file names, each followed by a slash and a new-line. This string table member, if present, will precede all "✘normal" archive members. The special archive symbol table is not a

"✘normal" member, and must be first if it exists. The `ar_name` entry of the string table's member header holds a zero length name `ar_name[0]=='/'`, followed by one trailing slash (`ar_name[1]=='/'`), followed by blanks (`ar_name[2]==' '`, etc.). Offsets into the string table begin at zero. Example *ar_name* values for short and long file names appear below.

```
Offset   +0   +1   +2   +3   +4   +5   +6   +7   +8   +9
        _____________________________________________
 0     | f  | i  | l  | e  | _  | n  | a  | m  | e  | _  |
       |____|____|____|____|____|____|____|____|____|____|
10     | s  | a  | m  | p  | l  | e  | /  | \n | l  | o  |
       |____|____|____|____|____|____|____|____|____|____|
20     | n  | g  | e  | r  | f  | i  | l  | e  | n  | a  |
       |____|____|____|____|____|____|____|____|____|____|
30     | m  | e  | x  | a  | m  | p  | l  | e  | /  | \n |
       |____|____|____|____|____|____|____|____|____|____|
```

```
   Member Name                          ar_name
_____________________________________________________________
short-name           | short-name/  | Not in string table
                     |              |
file_name_sample     | /0           | Offset 0 in string table
                     |              |
longerfilenamexample | /18          | Offset 18 in string table
_____________________|______________|_____________________________
```

See Also `ar`(1), `ld`(1), `strip`(1), `a.out`(4)

Notes The `strip` utility will remove all archive symbol entries from the header. The archive symbol entries must be restored with the `-ts` options of the `ar` command before the archive can be used with the link editor `ld`.

The maximum size of a single file within an archive is limited to 4GB by the size of the `ar_size` field in the archive member structure. An archive can therefore exceed 4GB in size, but no single member within an archive can be larger than 4GB.

The maximum user ID for an individual file within an archive is limited to 6 characters by the `ar_uid` field of the archive member header. Any file with a user ID greater than 999999 is set to user ID "nobody" (60001).

The maximum group ID for an individual file within an archive is limited to 6 characters by the `ar_gid` field of the archive member header. Any file with a group ID greater than 999999 is set to group ID "nobody" (60001).