lucid Reference Manual

0.1_rc1

Generated by Doxygen 1.5.1

Sun Dec 3 17:45:53 2006

Contents

Chapter 1

lucid Documentation

The lucid library combines a lot of useful functions i wrote for my projects. There are a lot of custom functions for strings, doubly-linked lists, bitmaps and flag lists, input/output, cryptogrpahic digests, and tcp connections.

Some of these functions, especially string and list functions, are completely self-contained and do not rely on libc. This makes integration in foreign projects as easy as possible.

On the other hand, functions for input/output or chroot are just wrappers around libc library functions, but may still be usefull to others as well.

The size of functions range from a few hundred bytes to about 30K.

1.1 Why another library?

This library was written for my own projects. As the number of foreign libraries i included grew, i have collected all functions i needed - and meanwhile most of them reimplemented - and made an own library.

1.2 Current Status

Actually 4 projects (libvserver, vcd, vstatd, vwrappers) are using lucid now, so an own shared library was necessary. The library contains a test suite for the most important functions. Functions not explicitely tested are either tested implicitely by other functions and a test may be written in the future, or the function is so simple that you can just hope it works ;)

1.3 Documentation

All function are documented with an inline source browser for easy learning and reference. To get an overview of function families please start at the Modules page. Experienced users may look up information in the Globals, Data Fields or File List.

Also take a look at the **Examples** (p. 273) and read the **License** (p. 274).

1.4 Bugs, Patches, Wishes

If you have found a bug in lucid that was not discovered by the test suite, or if you have any suggestion, wishes or patches for the future of lucid, please contact me at hollow[at]gentoo.org

Chapter 2

lucid Module Index

2.1 lucid Modules

Here is a list of all modules:

Chapter 3

lucid Data Structure Index

3.1 lucid Data Structures

Here are the data structures with brief descriptions:

Chapter 4

lucid File Index

4.1 lucid File List

Here is a list of all files with brief descriptions:

Chapter 5

lucid Page Index

5.1 lucid Related Pages

Here is a list of all related documentation pages:

Chapter 6

lucid Module Documentation

6.1 Internet address conversion

6.1.1 Detailed Description

The **addr_hton()** (p. 13) and **addr_ntoh()** (p. 14) functions convert from host- to network-byteorder, and vice versa, respectively.

The **addr_from_str()** (p. 14) function converts the Internet host address in standard numbers-and-dots notation pointed to by the string str into binary data and stores result in the ip/mask pair of pointers. **addr_from_str()** (p. 14) returns 0 if no argument was converted, 1 if ip was converted, 2 for mask and 3 for both.

The **addr_to_str()** (p. 15) function converts the Internet host address given in the ip/mask pair of pointers to a string in standard numbers-and-dots notation. The returned string is obtained by malloc and should be free(3)'d by the caller.

Functions

- uint32_t **addr_hton** (uint32_t addr)
  convert address from host to network byte order
- uint32_t **addr_ntoh** (uint32_t addr)
  convert address from network to host byte order
- int **addr_from_str** (const char *str, uint32_t *ip, uint32_t *mask)
  convert string to IP address and netmask
- char * **addr_to_str** (uint32_t ip, uint32_t mask)
  convert IP address and netmask to string

6.1.2 Function Documentation

6.1.2.1 uint32_t addr_hton (uint32_t *addr*)

convert address from host to network byte order

Parameters:

← ***addr*** address in host byte order

Returns:

address in network byte order

Definition at line 27 of file addr_hton.c.

Referenced by addr_from_str(), and addr_ntoh().

```
28 {
29         if (islitend())
30                 return (addr >> 24) |
31                        ((addr & 0xff0000) >> 8) |
32                        ((addr & 0xff00  ) << 8) |
33                        (addr << 24);
34         else
35                 return addr;
36 }
```

6.1.2.2 uint32_t addr_ntoh (uint32_t *addr*)

convert address from network to host byte order

Parameters:

← ***addr*** address in network byte order

Returns:

address in host byte order

Definition at line 20 of file addr_ntoh.c.

References addr_hton().

Referenced by addr_to_str().

```
21 {
22         return addr_hton(addr);
23 }
```

6.1.2.3 int addr_from_str (const char * *str*, uint32_t * *ip*, uint32_t * *mask*)

convert string to IP address and netmask

Parameters:

← ***str*** string in CIDR or netmask notation

→ ***ip*** pointer to store IP address

→ ***mask*** pointer to store netmask

Returns:

0 if no argument was converted, 1 if ip was converted, 2 for mask and 3 for both.

Definition at line 22 of file addr_from_str.c.

References _lucid_sscanf(), addr_hton(), str_index(), str_isdigit, str_isempty, str_len(), and str_toumax().

Referenced by tcp_connect(), and tcp_listen().

```
23 {
24         int rc = 0;
25         unsigned long long int cidr;
26 
27         union {
28                 uint8_t  b[4];
29                 uint32_t l;
30         } u;
31 
32         const char *p = str_index(str, '/', str_len(str));
33 
34         if (ip && (!p || p - str > 0)) {
35                 if (_lucid_sscanf(str, "%hhu.%hhu.%hhu.%hhu",
36                                   &u.b[0], &u.b[1], &u.b[2], &u.b[3]) == 4) {
37                         *ip = addr_hton(u.l);
38                         rc = 1;
39                 }
40         }
41 
42         if (!p || !mask)
43                 return rc;
44 
45         p++;
46 
47         /* CIDR notation */
48         if (!str_isempty(p) && str_isdigit(p)) {
49                 str_toumax(p, &cidr, 10, str_len(p));
50 
51                 if (cidr > 0 && cidr <= 32) {
52                         *mask = 0xffffffff & ~((1 << (32 - cidr)) - 1);
53                         rc   += 2;
54                 }
55         }
56 
57         if (!str_isempty(p)) {
58                 if (_lucid_sscanf(p, "%hhu.%hhu.%hhu.%hhu",
59                                   &u.b[0], &u.b[1], &u.b[2], &u.b[3]) == 4) {
60                         *mask = addr_hton(u.l);
61                         rc += 2;
62                 }
63         }
64 
65         return rc;
66 }
```

6.1.2.4 char* addr_to_str (uint32_t *ip*, uint32_t *mask*)

convert IP address and netmask to string

Parameters:

← ***ip*** IP adress to convert

← ***mask*** netmask to convert

Returns:

string in netmask notation (obtained with malloc(3))

Note:

The caller should free obtained memory using free(3)

See also:

malloc(3)
free(3)

Definition at line 21 of file addr_to_str.c.

References _lucid_asprintf(), and addr_ntoh().

```
22 {
23         char *buf;
24
25         ip   = addr_ntoh(ip);
26         mask = addr_ntoh(mask);
27
28         uint8_t *ipp   = (uint8_t *) &ip;
29         uint8_t *maskp = (uint8_t *) &mask;
30
31         if (mask)
32                 _lucid_asprintf(&buf, "%u.%u.%u.%u/%u.%u.%u.%u",
33                                       ipp[0],   ipp[1],   ipp[2],   ipp[3],
34                                       maskp[0], maskp[1], maskp[2], maskp[3]);
35
36         else
37                 _lucid_asprintf(&buf, "%u.%u.%u.%u", ipp[0],  ipp[1],  ipp[2],  ipp[3]);
38
39         return buf;
40 }
```

6.2 Argument vector conversion

6.2.1 Detailed Description

The command line arguments are the whitespace-separated tokens given in the shell command used to invoke the program; thus, in 'cat foo bar', the arguments are 'foo' and 'bar'. For the command 'cat foo bar', argc is 3 and argv has three elements, "cat", "foo" and "bar".

The **argv_from_str()** (p. 17) and **argv_to_str()** (p. 18) functions convert an argument vector (as in main()) from and to a string (as seen in the shell), respectively.

Functions

- int **argv_from_str** (char *str, char **const argv, int argc)
  convert string to argument vector
- char * **argv_to_str** (int argc, const char **const argv)
  convert argument vector to string

6.2.2 Function Documentation

6.2.2.1 int argv_from_str (char * *str*, char **const *argv*, int *argc*)

convert string to argument vector

Parameters:

← ***str*** space seperated string to convert
→ ***argv*** argument vector to fill
← ***argc*** size of argv

Returns:

number of arguments that (would) have been converted

Note:

the caller has to allocate memory for argv
this function modifies its first argument, use with caution

Definition at line 23 of file argv_from_str.c.

References char_isspace.

Referenced by exec_fork(), exec_fork_background(), exec_fork_pipe(), and exec_replace().

```
24 {
25         int i, argc = 0;
26
27         for (i = 0; i < max_argc; i++)
28                 argv[i] = NULL;
29
30         if (!str)
31                 return 0;
```

```
32
33         while (*str) {
34                 while (char_isspace(*str))
35                         *(str++) = '\0';
36
37                 if (*str && argc < max_argc - 1) {
38                         argv[argc] = str;
39                         argc++;
40
41                         while (*str && !char_isspace(*str))
42                                 str++;
43                 }
44
45                 else
46                         break;
47         }
48
49         return argc;
50 }
```

6.2.2.2 char* argv_to_str (int *argc*, const char **const *argv*)

convert argument vector to string

Parameters:

← ***argc*** size of argv

← ***argv*** argument vector to convert

Returns:

space seperated string (obtained by malloc(3))

Note:

The caller should free obtained memory using free(3)

See also:

malloc(3)
free(3)

Definition at line 22 of file argv_to_str.c.

References stralloc_t::len, stralloc_t::s, str_dupn(), stralloc_catm(), stralloc_free(), and stralloc_init().

```
23 {
24         int i;
25         size_t len;
26         char *str;
27         stralloc_t buf;
28
29         stralloc_init(&buf);
30
31         for (i = 0; i < argc; i++)
32                 stralloc_catm(&buf, argv[i], " ", 0);
33
34         if (buf.len < 1)
35                 len = 0;
```

```
36          else
37                  len = buf.len - 1;
38
39          str = str_dupn(buf.s, len);
40          stralloc_free(&buf);
41          return str;
42  }
```

6.3 Bitmap conversion

6.3.1 Detailed Description

The i2v and v2i family of functions convert between a bitmap and a bit index.

A bitmap is simply an integer with certain bits being 1 (enabled) and 0 (disabled).

These functions only return usable results if exactly one bit is enabled.

- Bit index to bitmap
  The resulting bitmask is a simple arithmetic left shift of 1 index times.
- Bitmap to bit index
  The resulting bit index is asimple arithmetic right shift until the map is empty.

These functions are mainly used by the flist family of functions.

Functions

- uint32_t **i2v32** (int index)
  convert bit index to 32 bit value
- uint64_t **i2v64** (int index)
  convert bit index to 64 bit value
- int **v2i32** (uint32_t val)
  convert 32 bit value to bit index
- int **v2i64** (uint64_t val)
  convert 64 bit value to bit index

6.3.2 Function Documentation

6.3.2.1 uint32_t i2v32 (int *index*)

convert bit index to 32 bit value

Parameters:

← ***index*** bit index (0-31)

Returns:

32 bit value

Definition at line 20 of file i2v32.c.

```
21 {
22         if (index < 0 || index > 31)
23                 return 0;
24
25         return (1UL << index);
26 }
```

6.3.2.2 uint64_t i2v64 (int *index*)

convert bit index to 64 bit value

Parameters:

← ***index*** bit index (0-63)

Returns:

64 bit value

Definition at line 20 of file i2v64.c.

```
21 {
22         if (index < 0 || index > 63)
23                 return 0;
24
25         return (1ULL << index);
26 }
```

6.3.2.3 int v2i32 (uint32_t *val*)

convert 32 bit value to bit index

Parameters:

← ***val*** 32 bit value

Returns:

bit index (0-31)

Definition at line 20 of file v2i32.c.

```
21 {
22         int index = 0;
23
24         if (val == 0)
25                 return -1;
26
27         while ((val = val >> 1))
28                 index++;
29
30         return index;
31 }
```

6.3.2.4 int v2i64 (uint64_t *val*)

convert 64 bit value to bit index

Parameters:

← ***val*** 64 bit value

Returns:

bit index (0-63)

Definition at line 20 of file v2i64.c.

```
21 {
22         int index = 0;
23
24         if (val == 0)
25                 return -1;
26
27         while ((val = val >> 1))
28                 index++;
29
30         return index;
31 }
```

6.4 Secure chroot wrappers

6.4.1 Detailed Description

The chroot system call changes the root directory of the current process. This directory will be used for pathnames beginning with /. The root directory is inherited by all children of the current process.

The chroot family of functions provide wrappers for other library functions to happen in a chroot while the caller still remains in the old root after the functions have returned.

One can break out of the chroot in many ways due to the nature of the chroot system call:

- This call changes an ingredient in the pathname resolution process and does nothing else.
- This call does not change the current working directory.
- This call does not close open file descriptors.

The main usage of these functions is to get a file descriptor, safe against symlink attacks, reffering to a directory inside a new root.

Functions

- int **chroot_fd** (int fd)
  chroot(2) to the directory pointed to by a filedescriptor
- int **chroot_mkdirp** (const char *root, const char *dir, mode_t mode)
  recursive mkdir(2) inside a secure chroot
- int **chroot_secure_chdir** (const char *root, const char *dir)
  symlink-attack safe chdir(2) in chroot(2)

6.4.2 Function Documentation

6.4.2.1 int chroot_fd (int *fd*)

chroot(2) to the directory pointed to by a filedescriptor

Parameters:

← ***fd*** file descriptor refering to a directory (fchdir(2))

Returns:

0 on success, -1 on error with errno set

See also:

Secure chroot wrappers (p. 23)
fchdir(2)

Definition at line 22 of file chroot_fd.c.

Referenced by chroot_mkdirp(), and chroot_secure_chdir().

```
23 {
24         if (fchdir(fd) == -1)
25                 return -1;
26
27         return chroot(".");
28 }
```

6.4.2.2 int chroot_mkdirp (const char * *root*, const char * *dir*, mode_t *mode*)

recursive mkdir(2) inside a secure chroot

Parameters:

← ***root*** new root path

← ***dir*** dir to be created in root

← ***mode*** file permissions

Returns:

0 on success, -1 on error with errno set

See also:

chroot_secure_chdir (p. 25)
mkdir(2)

Definition at line 28 of file chroot_mkdirp.c.

References chroot_fd(), mkdirp(), and open_read().

```
29 {
30         int orig_root, new_root;
31         int errno_orig;
32
33         if ((orig_root = open_read("/")) == -1)
34                 return -1;
35
36         if (chdir(root) == -1)
37                 return -1;
38
39         if ((new_root = open_read(".")) == -1)
40                 return -1;
41
42         /* check cwdfd */
43         if (chroot_fd(new_root) == -1)
44                 return -1;
45
46         /* now create the dir in the chroot */
47         if (mkdirp(dir, mode) == -1)
48                 goto err;
49
50         /* break out of the chroot */
51         chroot_fd(orig_root);
52
53         return 0;
54
```

```
55 err:
56         errno_orig = errno;
57         chroot_fd(orig_root);
58         errno = errno_orig;
59         return -1;
60 }
```

6.4.2.3 int chroot_secure_chdir (const char * *root*, const char * *dir*)

symlink-attack safe chdir(2) in chroot(2)

Parameters:

← ***root*** new root path

← ***dir*** dir to chdir(2) in root

Returns:

0 on success, -1 on error with errno set

See also:

Secure chroot wrappers (p. 23)
chdir(2)

Definition at line 27 of file chroot_secure_chdir.c.

References chroot_fd(), and open_read().

```
28 {
29         int orig_root, new_root;
30
31         if ((orig_root = open_read("/")) == -1)
32                 return -1;
33
34         if (chdir(root) == -1)
35                 return -1;
36
37         if ((new_root = open_read(".")) == -1)
38                 return -1;
39
40         int dirfd;
41         int errno_orig;
42
43         /* check cwdfd */
44         if (chroot_fd(new_root) == -1)
45                 return -1;
46
47         /* now go to dir in the chroot */
48         if (chdir(dir) == -1)
49                 goto err;
50
51         /* save a file descriptor of the target dir */
52         dirfd = open_read(".");
53
54         if (dirfd == -1)
55                 goto err;
56
57         /* break out of the chroot */
58         chroot_fd(orig_root);
59
```

```
60         /* now go to the saved target dir (but outside the chroot) */
61         if (fchdir(dirfd) == -1)
62                 goto err2;
63
64         close(dirfd);
65         return 0;
66
67 err2:
68         errno_orig = errno;
69         close(dirfd);
70         errno = errno_orig;
71 err:
72         errno_orig = errno;
73         chroot_fd(orig_root);
74         errno = errno_orig;
75         return -1;
76 }
```

6.5 Command execution wrappers

6.5.1 Detailed Description

The exec family of functions provide convenient wrappers around fork(2), execve(2), waitpid(2) and pipe(2).

These functions combine one or more of the above system calls in one function, thus allowing fast and simple process creation in applications.

Defines

- #define **EXEC_MAX_ARGV** 64
  maximum number of arguments that will be converted for execvp(2)

Functions

- int **exec_fork** (const char *fmt,...)
  fork, execvp and wait
- int **exec_fork_background** (const char *fmt,...)
  fork, execvp and ignore child
- int **exec_fork_pipe** (char **out, const char *fmt,...)
  pipe, fork, execvp and wait
- int **exec_replace** (const char *fmt,...)
  plain execvp

6.5.2 Define Documentation

6.5.2.1 #define EXEC_MAX_ARGV 64

maximum number of arguments that will be converted for execvp(2)

Definition at line 36 of file exec.h.

Referenced by exec_fork(), exec_fork_background(), exec_fork_pipe(), and exec_replace().

6.5.3 Function Documentation

6.5.3.1 int exec_fork (const char * *fmt*, ...)

fork, execvp and wait

Parameters:

← ***fmt*** format string passed to printf

← **...** variable number of arguments according to fmt

Returns:

status obtained by wait(2) or -1 with errno set

See also:

Formatted output conversion (p. 72)
execvp(2)

Definition at line 28 of file exec_fork.c.

References _lucid_vasprintf(), argv_from_str(), and EXEC_MAX_ARGV.

```
29 {
30         va_list ap;
31         va_start(ap, fmt);
32 
33         char *cmd;
34         _lucid_vasprintf(&cmd, fmt, ap);
35 
36         va_end(ap);
37 
38         int argc;
39         char *argv[EXEC_MAX_ARGV];
40 
41         argc = argv_from_str(cmd, argv, EXEC_MAX_ARGV);
42 
43         if (argc < 1) {
44                 free(cmd);
45                 return errno = EINVAL, -1;
46         }
47 
48         pid_t pid;
49         int status;
50 
51         switch ((pid = fork())) {
52         case -1:
53                 free(cmd);
54                 return -1;
55 
56         case 0:
57                 usleep(200);
58 
59                 execvp(argv[0], argv);
60 
61                 /* never get here */
62                 exit(errno);
63 
64         default:
65                 free(cmd);
66 
67                 if (waitpid(pid, &status, 0) == -1)
68                         return -1;
69         }
70 
71         return status;
72 }
```

6.5.3.2 int exec_fork_background (const char * *fmt*, ...)

fork, execvp and ignore child

Parameters:

← ***fmt*** format string passed to printf

← ... variable number of arguments according to fmt

Returns:

0 on success or -1 with errno set

See also:

Formatted output conversion (p. 72)
execvp(2)

Note:

this function closes file descriptors 0-100 before execvp

Definition at line 28 of file exec_fork_background.c.

References _lucid_vasprintf(), argv_from_str(), and EXEC_MAX_ARGV.

```
29 {
30         va_list ap;
31         va_start(ap, fmt);
32 
33         char *cmd;
34         _lucid_vasprintf(&cmd, fmt, ap);
35 
36         va_end(ap);
37 
38         int argc;
39         char *argv[EXEC_MAX_ARGV];
40 
41         argc = argv_from_str(cmd, argv, EXEC_MAX_ARGV);
42 
43         if (argc < 1) {
44                 free(cmd);
45                 return errno = EINVAL, -1;
46         }
47 
48         pid_t pid;
49         int i;
50 
51         switch ((pid = fork())) {
52         case -1:
53                 return -1;
54 
55         case 0:
56                 usleep(200);
57 
58                 for (i = 0; i < 100; i++)
59                         close(i);
60 
61                 execvp(argv[0], argv);
62 
63         default:
64                 signal(SIGCHLD, SIG_IGN);
65         }
66 
67         free(cmd);
68         return 0;
69 }
```

6.5.3.3 int exec_fork_pipe (char ** *out*, const char * *fmt*, ...)

pipe, fork, execvp and wait

Parameters:

→ ***out*** empty pointer to store combined stdout/stderr

← ***fmt*** format string passed to printf

← ***...*** variable number of arguments according to fmt

Returns:

status obtained by wait(2) or -1 with errno set

Note:

The caller should free obtained memory for out using free(3)

See also:

Formatted output conversion (p. 72)
malloc(3)
free(3)
execvp(2)

Definition at line 29 of file exec_fork_pipe.c.

References _lucid_vasprintf(), argv_from_str(), EXEC_MAX_ARGV, and io_read_eof().

```
30 {
31         va_list ap;
32         va_start(ap, fmt);
33 
34         char *cmd;
35         _lucid_vasprintf(&cmd, fmt, ap);
36 
37         va_end(ap);
38 
39         int argc;
40         char *argv[EXEC_MAX_ARGV];
41 
42         argc = argv_from_str(cmd, argv, EXEC_MAX_ARGV);
43 
44         if (argc < 1) {
45                 free(cmd);
46                 return errno = EINVAL, -1;
47         }
48 
49         int outfds[2];
50 
51         if (pipe(outfds) == -1)
52                 return -1;
53 
54         pid_t pid;
55         int status;
56 
57         switch ((pid = fork())) {
58         case -1:
59                 free(cmd);
60                 return -1;
61 
62         case 0:
63                 usleep(200);
64 
65                 close(outfds[0]);
66 
67                 dup2(outfds[1], STDOUT_FILENO);
```

```
68                  dup2(outfds[1], STDERR_FILENO);
69
70                  execvp(argv[0], argv);
71
72                  /* never get here */
73                  exit(errno);
74
75          default:
76                  free(cmd);
77
78                  close(outfds[1]);
79
80                  if (out && io_read_eof(outfds[0], out) == -1)
81                          return -1;
82
83                  close(outfds[0]);
84
85                  if (waitpid(pid, &status, 0) == -1)
86                          return -1;
87          }
88
89          return status;
90  }
```

6.5.3.4 int exec_replace (const char * *fmt*, ...)

plain execvp

Parameters:

← ***fmt*** format string passed to printf

← ***...*** variable number of arguments according to fmt

Returns:

only returns on error with errno set

See also:

Formatted output conversion (p. 72)
execvp(2)

Definition at line 27 of file exec_replace.c.

References _lucid_vasprintf(), argv_from_str(), and EXEC_MAX_ARGV.

```
28 {
29          va_list ap;
30          va_start(ap, fmt);
31
32          char *cmd;
33          _lucid_vasprintf(&cmd, fmt, ap);
34
35          va_end(ap);
36
37          int argc;
38          char *argv[EXEC_MAX_ARGV];
39
40          argc = argv_from_str(cmd, argv, EXEC_MAX_ARGV);
41
42          if (argc < 1) {
```

```
43                 free(cmd);
44                 return errno = EINVAL, -1;
45         }
46
47         execvp(argv[0], argv);
48
49         /* never get here */
50         free(cmd);
51         return -1;
52 }
```

6.6 Flag list conversion

6.6.1 Detailed Description

The flag list family of functions manages a list of possible values of a bitmap using strings as key into the list.

A bitmap is simply an integer with certain bits being 1 (enabled) and 0 (disabled).

The FLIST32_START and FLIST64_START macros provides a shortcut for list declaration and initialization; followed by one or more of FLIST32_NODE, FLIST32_NODE1, FLIST64_NODE and FLIST64_NODE1 to insert nodes into the list. The FLIST32_NODE1 and FLIST64_NODE1 macros are the same as FLIST32_NODE and FLIST64_NODE, respectively, except that they convert the bit index to a bitmap value before storing it in the list. The list should then be terminated using FLIST32_END or FLIST64_END, respectively.

The **flist32_getval()** (p. 35), **flist64_getval()** (p. 39), **flist32_getkey()** (p. 36) and **flist64_getkey()** (p. 39) functions provide lookup routines by key and value, respectively.

The **flist32_from_str()** (p. 37) and **flist64_from_str()** (p. 40) functions convert a string consisting of zero or more flag list keys seperated by a delimiter and optionally prefixed with a clear modifier to a bitmap/bitmask pair according to a given list.

The **flist32_to_str()** (p. 38) and **flist64_to_str()** (p. 41) functions convert a bitmap according to a given list to a string consisting of zero or more flag list keys seperated by a delimiter.

Data Structures

- struct **flist32_t**
  32 bit list object
- struct **flist64_t**
  64 bit list object

Defines

- #define **FLIST32_START**(LIST) const **flist32_t** LIST[] = {
  32 bit list initialization
- #define **FLIST32_NODE**(PREFIX, NAME) { #NAME, PREFIX ## _ ## NAME },
  32 bit list node
- #define **FLIST32_NODE1**(PREFIX, NAME) { #NAME, (1 << PREFIX ## _ ## NAME) },
  32 bit list node from index
- #define **FLIST32_END** { NULL, 0 } };
  32 bit list termination
- #define **FLIST64_START**(LIST) const **flist64_t** LIST[] = {
  64 bit list initialization

- #define **FLIST64_NODE**(PREFIX, NAME) { #NAME, PREFIX ## _ ## NAME },

  64 bit list node

- #define **FLIST64_NODE1**(PREFIX, NAME) { #NAME, (1 << PREFIX ## _ ## NAME) },

  64 bit list node from index

- #define **FLIST64_END** { NULL, 0 } };

  64 bit list termination

Functions

- uint32_t **flist32_getval** (const **flist32_t** list[], const char *key)

  get 32 bit value by key

- const char * **flist32_getkey** (const **flist32_t** list[], uint32_t val)

  get key from 32 bit value

- int **flist32_from_str** (const char *str, const **flist32_t** list[], uint32_t *flags, uint32_t *mask, char clmod, char delim)

  parse flag list string

- char * **flist32_to_str** (const **flist32_t** list[], uint32_t val, char delim)

  convert bit mask to flag list string

- uint64_t **flist64_getval** (const **flist64_t** list[], const char *key)

  get 64 bit value by key

- const char * **flist64_getkey** (const **flist64_t** list[], uint64_t val)

  get key from 64 bit value

- int **flist64_from_str** (const char *str, const **flist64_t** list[], uint64_t *flags, uint64_t *mask, char clmod, char delim)

  parse flag list string

- char * **flist64_to_str** (const **flist64_t** list[], uint64_t val, char delim)

  convert bit mask to flag list string

6.6.2 Define Documentation

6.6.2.1 #define FLIST32_START(LIST) const flist32_t LIST[] = {

32 bit list initialization

Definition at line 62 of file flist.h.

6.6.2.2 #define FLIST32_NODE(PREFIX, NAME) { #NAME, PREFIX ## _ ## NAME },

32 bit list node

Definition at line 65 of file flist.h.

6.6.2.3 #define FLIST32_NODE1(PREFIX, NAME) { #NAME, (1 << PREFIX ## _ ## NAME) },

32 bit list node from index

Definition at line 68 of file flist.h.

6.6.2.4 #define FLIST32_END { NULL, 0 } };

32 bit list termination

Definition at line 71 of file flist.h.

6.6.2.5 #define FLIST64_START(LIST) const flist64_t LIST[] = {

64 bit list initialization

Definition at line 140 of file flist.h.

6.6.2.6 #define FLIST64_NODE(PREFIX, NAME) { #NAME, PREFIX ## _ ## NAME },

64 bit list node

Definition at line 143 of file flist.h.

6.6.2.7 #define FLIST64_NODE1(PREFIX, NAME) { #NAME, (1 << PREFIX ## _ ## NAME) },

64 bit list node from index

Definition at line 146 of file flist.h.

6.6.2.8 #define FLIST64_END { NULL, 0 } };

64 bit list termination

Definition at line 149 of file flist.h.

6.6.3 Function Documentation

6.6.3.1 uint32_t flist32_getval (const flist32_t *list*[], const char * *key*)

get 32 bit value by key

Parameters:

← ***list*** list to use for conversion

← ***key*** key to look for

Returns:

32 bit value >= 1 if key found, 0 otherwise

Definition at line 23 of file flist32_getval.c.

References flist32_t::key, and str_cmp().

Referenced by flist32_from_str().

```
24 {
25         int i;
26
27         for (i = 0; list[i].key; i++) {
28                 if (str_cmp(list[i].key, key) == 0)
29                         return list[i].val;
30         }
31
32         return errno = ENOENT, 0;
33 }
```

6.6.3.2 const char* flist32_getkey (const flist32_t *list*[ ], uint32_t *val*)

get key from 32 bit value

Parameters:

← ***list*** list to use for conversion

← ***val*** 32 bit key to look for

Returns:

key if value was found, NULL otherwise

Note:

this functions does not reset the flags or mask argument to an empty bitmap, thus allowing incremental changes to the map.

Definition at line 23 of file flist32_getkey.c.

References flist32_t::key.

```
24 {
25         int i;
26
27         for (i = 0; list[i].key; i++) {
28                 if (list[i].val == val)
29                         return list[i].key;
30         }
31
32         return errno = ENOENT, NULL;
33 }
```

6.6.3.3 int flist32_from_str (const char * *str*, const flist32_t *list*[ ], uint32_t * *flags*, uint32_t * *mask*, char *clmod*, char *delim*)

parse flag list string

Parameters:

← ***str*** string to convert

← ***list*** list to use for conversion

→ ***flags*** pointer to a bit mask

→ ***mask*** pointer to a set mask

← ***clmod*** clear flag modifier

← ***delim*** flag delimiter

Returns:

0 on success, -1 on error with errno set

Definition at line 24 of file flist32_from_str.c.

References char_isspace, flist32_getval(), str_dup(), str_index(), str_isempty, and str_len().

```
27 {
28         char *p, *o, *buf;
29         int clear = 0;
30         uint32_t cur_flag;
31
32         if (str_isempty(str))
33                 return errno = EINVAL, -1;
34
35         buf = p = o = str_dup(str);
36
37         while (1) {
38                 p = str_index(p, delim, str_len(p));
39
40                 while (char_isspace(*o))
41                         o++;
42
43                 if (p) {
44                         if (o == p) {
45                                 p++;
46                                 continue;
47                         }
48
49                         *p++ = '\0';
50                 }
51
52                 if (*o == clmod)
53                         clear = 1;
54
55                 cur_flag = flist32_getval(list, o+clear);
56
57                 if (!cur_flag) {
58                         free(buf);
59                         return errno = ENOENT, -1;
60                 }
61
62                 if (clear) {
63                         *flags &= ~cur_flag;
64                         *mask  |=  cur_flag;
65                 } else {
```

```
66                          *flags |=  cur_flag;
67                          *mask  |=  cur_flag;
68                  }
69
70                  if (!p)
71                          break;
72                  else
73                          o = p;
74          }
75
76          free(buf);
77          return 0;
78  }
```

6.6.3.4 char* flist32_to_str (const flist32_t *list*[], uint32_t *val*, char *delim*)

convert bit mask to flag list string

Parameters:

← ***list*** list to use for conversion

← ***val*** bit mask

← ***delim*** flag delimiter

Returns:

flags list string (obtained with malloc(3))

Note:

the caller should free obtained memory using free(3)
this function ignores set bits if they do not appear in the list
if no flag was found or the bitmap was empty, an empty string is returned, not NULL

See also:

malloc(3)
free(3)

Definition at line 22 of file flist32_to_str.c.

References flist32_t::key, str_dupn(), stralloc_catf(), stralloc_free(), and stralloc_init().

```
23  {
24          int i;
25          size_t len;
26          char *str;
27          stralloc_t buf;
28
29          stralloc_init(&buf);
30
31          for (i = 0; list[i].key; i++)
32                  if (val & list[i].val)
33                          stralloc_catf(&buf, "%s%c", list[i].key, delim);
34
35          if (buf.len == 0)
36                  len = 0;
37          else
38                  len = buf.len - 1;
```

```
39
40          str = str_dupn(buf.s, len);
41          stralloc_free(&buf);
42          return str;
43 }
```

6.6.3.5 uint64_t flist64_getval (const flist64_t *list*[ ], const char * *key*)

get 64 bit value by key

Parameters:

← ***list*** list to use for conversion

← ***key*** key to look for

Returns:

64 bit value >= 1 if key was found, 0 otherwise

Definition at line 23 of file flist64_getval.c.

References flist64_t::key, and str_cmp().

Referenced by flist64_from_str().

```
24 {
25          int i;
26
27          for (i = 0; list[i].key; i++) {
28                  if (str_cmp(list[i].key, key) == 0)
29                          return list[i].val;
30          }
31
32          return errno = ENOENT, 0;
33 }
```

6.6.3.6 const char* flist64_getkey (const flist64_t *list*[ ], uint64_t *val*)

get key from 64 bit value

Parameters:

← ***list*** list to use for conversion

← ***val*** 64 bit key to look for

Returns:

key if value was found, NULL otherwise

Definition at line 23 of file flist64_getkey.c.

References flist64_t::key.

```
24 {
25          int i;
26
```

```
27         for (i = 0; list[i].key; i++) {
28                 if (list[i].val == val)
29                         return list[i].key;
30         }
31
32         return errno = ENOENT, NULL;
33 }
```

6.6.3.7 int flist64_from_str (const char * *str*, const flist64_t *list*[ ], uint64_t * *flags*, uint64_t * *mask*, char *clmod*, char *delim*)

parse flag list string

Parameters:

- ← ***str*** string to convert
- ← ***list*** list to use for conversion
- → ***flags*** pointer to a bit mask
- → ***mask*** pointer to a set mask
- ← ***clmod*** clear flag modifier
- ← ***delim*** flag delimiter

Returns:

0 on success, -1 on error with errno set

Note:

this functions does not reset the flags or mask argument to an empty bitmap, thus allowing incremental changes to the map.

Definition at line 24 of file flist64_from_str.c.

References char_isspace, flist64_getval(), str_dup(), str_index(), str_isempty, and str_len().

```
27 {
28         char *p, *o, *buf;
29         int clear = 0;
30         uint64_t cur_flag;
31
32         if (str_isempty(str))
33                 return errno = EINVAL, -1;
34
35         buf = p = o = str_dup(str);
36
37         while (1) {
38                 p = str_index(p, delim, str_len(p));
39
40                 while (char_isspace(*o))
41                         o++;
42
43                 if (p) {
44                         if (o == p) {
45                                 p++;
46                                 continue;
47                         }
48
49                         *p++ = '\0';
```

```
50                  }
51
52                  if (*o == clmod)
53                          clear = 1;
54
55                  cur_flag = flist64_getval(list, o+clear);
56
57                  if (!cur_flag) {
58                          free(buf);
59                          return errno = ENOENT, -1;
60                  }
61
62                  if (clear) {
63                          *flags &= ~cur_flag;
64                          *mask  |=  cur_flag;
65                  } else {
66                          *flags |=  cur_flag;
67                          *mask  |=  cur_flag;
68                  }
69
70                  if (!p)
71                          break;
72                  else
73                          o = p;
74          }
75
76          free(buf);
77          return 0;
78  }
```

6.6.3.8 char* flist64_to_str (const flist64_t *list*[ ], uint64_t *val*, char *delim*)

convert bit mask to flag list string

Parameters:

← ***list*** list to use for conversion

← ***val*** bit mask

← ***delim*** flag delimiter

Returns:

flags list string (obtained with malloc(3))

Note:

the caller should free obtained memory using free(3)
this function ignores set bits if they do not appear in the list
if no flag was found or the bitmap was empty, an empty string is returned, not NULL

See also:

malloc(3)
free(3)

Definition at line 22 of file flist64_to_str.c.

References flist64_t::key, str_dupn(), stralloc_catf(), stralloc_free(), and stralloc_init().

```
23 {
24         int i;
25         size_t len;
26         char *str;
27         stralloc_t buf;
28 
29         stralloc_init(&buf);
30 
31         for (i = 0; list[i].key; i++)
32                 if (val & list[i].val)
33                         stralloc_catf(&buf, "%s%c", list[i].key, delim);
34 
35         if (buf.len == 0)
36                 len = 0;
37         else
38                 len = buf.len - 1;
39 
40         str = str_dupn(buf.s, len);
41         stralloc_free(&buf);
42         return str;
43 }
```

6.7 Input/Output wrappers

6.7.1 Detailed Description

The io family of functions provide convenient wrappers around read(2).

The **io_read_eol()** (p. 43) function reads bytes until the end of the line, that is until one of the characters \r or \n is found, and stores them in the string pointed to by line.

The **io_read_eof()** (p. 44) function reads bytes until the end of the file is reached and stores them in the string pointed to by file.

The **io_read_len()** (p. 45) function reads an exact number of bytes from the file and stores them in the string pointed to by str.

Defines

- #define **CHUNKSIZE** 128

Functions

- int **io_read_eol** (int fd, char **line)
  read a line of input
- int **io_read_eof** (int fd, char **file)
  read until end of file
- int **io_read_len** (int fd, char **str, size_t len)
  read exact number of bytes

6.7.2 Define Documentation

6.7.2.1 #define CHUNKSIZE 128

bytes read at a time

Definition at line 40 of file io.h.

Referenced by io_read_eof(), and io_read_eol().

6.7.3 Function Documentation

6.7.3.1 int io_read_eol (int *fd*, char ** *line*)

read a line of input

Parameters:

← ***fd*** file descriptor to read from

→ ***line*** pointer to a string

Returns:

bytes on success, -1 on error with errno set

Note:

The caller should free obtained memory for line using free(3)

See also:

malloc(3)
free(3)
read(2)

Definition at line 24 of file io_read_eol.c.

References CHUNKSIZE, and str_dupn().

```
25 {
26         size_t chunks = 1, len = 0;
27         char *buf = malloc(chunks * CHUNKSIZE + 1);
28         char c;
29
30         for (;;) {
31                 switch(read(fd, &c, 1)) {
32                 case -1:
33                         return -1;
34
35                 case 0:
36                         goto out;
37
38                 default:
39                         if (c == '\n' || c == '\r')
40                                 goto out;
41
42                         if (len >= chunks * CHUNKSIZE) {
43                                 chunks++;
44                                 buf = realloc(buf, chunks * CHUNKSIZE + 1);
45                         }
46
47                         buf[len++] = c;
48                         break;
49                 }
50         }
51
52 out:
53         if (len > 0)
54                 *line = str_dupn(buf, len);
55
56         free(buf);
57         return len;
58 }
```

6.7.3.2 int io_read_eof (int *fd*, char ** *file*)

read until end of file

Parameters:

← ***fd*** file descriptor to read from

→ ***file*** pointer to a string

Returns:

bytes on success, -1 on error with errno set

Note:

The caller should free obtained memory for file using free(3)

See also:

malloc(3)
free(3)
read(2)

Definition at line 24 of file io_read_eof.c.

References CHUNKSIZE, and str_dupn().

Referenced by exec_fork_pipe().

```
25 {
26         int rc = -1;
27         size_t chunks = 1, len = 0;
28         char *buf = malloc(chunks * CHUNKSIZE + 1);
29
30         for (;;) {
31                 int bytes_read = read(fd, buf+len, CHUNKSIZE);
32
33                 if (bytes_read == -1)
34                         goto err;
35
36                 len += bytes_read;
37                 buf[len] = '\0';
38
39                 if (bytes_read == 0)
40                         goto out;
41
42                 if (bytes_read == CHUNKSIZE) {
43                         chunks++;
44                         buf = realloc(buf, chunks * CHUNKSIZE + 1);
45                 }
46         }
47
48 out:
49         if (len > 0)
50                 *file = str_dupn(buf, len);
51
52         rc = len;
53
54 err:
55         free(buf);
56         return rc;
57 }
```

6.7.3.3 int io_read_len (int *fd*, char ** *str*, size_t *len*)

read exact number of bytes

Parameters:

← ***fd*** file descriptor to read from

→ ***str*** pointer to a string

← ***len*** bytes to be read

Returns:

bytes read on success, -1 on error with errno set

Note:

The caller should free obtained memory for str using free(3)

See also:

malloc(3)
free(3)
read(2)

Definition at line 24 of file io_read_len.c.

References str_dupn().

```
25 {
26         char *buf = calloc(len + 1, sizeof(char));
27
28         ssize_t buflen = read(fd, buf, len);
29
30         if (buflen == -1)
31                 return -1;
32
33         if (buflen > 0)
34                 *str = str_dupn(buf, buflen);
35
36         free(buf);
37         return buflen;
38 }
```

6.8 Simple doubly linked lists

6.8.1 Detailed Description

The simplest kind of linked list is a singly-linked list, which has one link per node. This link points to the next node in the list, or to a null value or empty list if it is the final node; e.g. 12 -> 99 -> 37 -> NULL.

A more sophisticated kind of linked list is a doubly-linked list. Each node has two links: one points to the previous node, or points to a null value or empty list if it is the first node; and one points to the next, or points to a null value or empty list if it is the final node; e.g. NULL <- 26 <-> 56 <-> 46 -> NULL.

The list family of functions and macros provide routines to create a list, add, move or remove elements and iterate over the list.

Data Structures

- struct **list_head**

  list head

Defines

- #define **container_of**(ptr, type, member) ((type *)((char *)(ptr) - offsetof(type, member)))

  get container of list head

- #define **LIST_HEAD_INIT**(name) { &(name), &(name) }

  list head initialization

- #define **LIST_HEAD**(name) struct **list_head** name = LIST_HEAD_INIT(name)

  list head creation

- #define **list_entry**(ptr, type, member) container_of(ptr, type, member)

  get the struct for this entry

- #define **list_for_each**(pos, head) for (pos = (head) → next; pos != (head); pos = pos → next)

  iterate over a list

- #define **list_for_each_prev**(pos, head) for (pos = (head) → prev; pos != (head); pos = pos → prev)

  iterate over a list backwards

- #define **list_for_each_safe**(pos, n, head)

  iterate over a list safe against removal of list entry

- #define **list_for_each_entry**(pos, head, member)

  iterate over list of given type

- #define **list_for_each_entry_reverse**(pos, head, member)
  iterate backwards over list of given type.
- #define **list_for_each_entry_safe**(pos, n, head, member)
  iterate over list of given type safe against removal of list entry
- #define **list_for_each_entry_safe_reverse**(pos, n, head, member)
  iterate backwards over list of given type safe against removal of list entry

6.8.2 Define Documentation

6.8.2.1 #define container_of(ptr, type, member) ((type *)((char *)(ptr) - offsetof(type, member)))

get container of list head

Definition at line 43 of file list.h.

6.8.2.2 #define LIST_HEAD_INIT(name) { &(name), &(name) }

list head initialization

Definition at line 62 of file list.h.

6.8.2.3 #define LIST_HEAD(name) struct list_head name = LIST_HEAD_INIT(name)

list head creation

Definition at line 65 of file list.h.

6.8.2.4 #define list_entry(ptr, type, member) container_of(ptr, type, member)

get the struct for this entry

Parameters:

ptr the &struct **list_head** (p. 140) pointer

type the type of the struct this is embedded in

member the name of the list_struct within the struct

Definition at line 249 of file list.h.

6.8.2.5 #define list_for_each(pos, head) for (pos = (head) → next; pos != (head); pos = pos → next)

iterate over a list

Parameters:

pos the &struct **list_head** (p. 140) to use as a loop counter

head the head for your list

Definition at line 258 of file list.h.

6.8.2.6 #define list_for_each_prev(pos, head) for (pos = (head) → prev; pos != (head); pos = pos → prev)

iterate over a list backwards

Parameters:

pos the &struct **list_head** (p. 140) to use as a loop counter

head the head for your list

Definition at line 267 of file list.h.

6.8.2.7 #define list_for_each_safe(pos, n, head)

Value:

```
for (pos = (head)->next, n = pos->next; pos != (head); \
                pos = n, n = pos->next)
```

iterate over a list safe against removal of list entry

Parameters:

pos the &struct **list_head** (p. 140) to use as a loop counter

n another &struct **list_head** (p. 140) to use as temporary storage

head the head for your list

Definition at line 277 of file list.h.

6.8.2.8 #define list_for_each_entry(pos, head, member)

Value:

```
for (pos = list_entry((head)->next, typeof(*pos), member); \
             &pos->member != (head); \
             pos = list_entry(pos->member.next, typeof(*pos), member))
```

iterate over list of given type

Parameters:

pos the type * to use as a loop counter

head,: the head for your list

member the name of the list_struct within the struct

Definition at line 288 of file list.h.

6.8.2.9 #define list_for_each_entry_reverse(pos, head, member)

Value:

```
for (pos = list_entry((head)->prev, typeof(*pos), member); \
             &pos->member != (head); \
             pos = list_entry(pos->member.prev, typeof(*pos), member))
```

iterate backwards over list of given type.

Parameters:

pos the type * to use as a loop counter
head the head for your list
member the name of the list_struct within the struct

Definition at line 300 of file list.h.

6.8.2.10 #define list_for_each_entry_safe(pos, n, head, member)

Value:

```
for (pos = list_entry((head)->next, typeof(*pos), member), \
             n = list_entry(pos->member.next, typeof(*pos), member); \
             &pos->member != (head); \
             pos = n, n = list_entry(n->member.next, typeof(*n), member))
```

iterate over list of given type safe against removal of list entry

Parameters:

pos the type * to use as a loop counter
n another type * to use as temporary storage
head the head for your list
member the name of the list_struct within the struct

Definition at line 313 of file list.h.

6.8.2.11 #define list_for_each_entry_safe_reverse(pos, n, head, member)

Value:

```
for (pos = list_entry((head)->prev, typeof(*pos), member), \
             n = list_entry(pos->member.prev, typeof(*pos), member); \
             &pos->member != (head); \
             pos = n, n = list_entry(n->member.prev, typeof(*n), member))
```

iterate backwards over list of given type safe against removal of list entry

Parameters:

pos the type * to use as a loop counter
n another type * to use as temporary storage
head the head for your list
member the name of the list_struct within the struct

Definition at line 327 of file list.h.

6.9 Log system multiplexer

6.9.1 Detailed Description

The log system multiplexer allows the caller to send log messages to multiple destinations; currently: syslog(3), file, stderr.

An application can only open one connection to the multiplexer during runtime. Another call to **log_init()** (p. 53) will replace the previous connection.

log_init() (p. 53) opens a connection to the multiplexer for a program. The options argument is a pointer to a **log_options_t** (p. 141) structure used for the multiplexer configuration.

See also:

log_options_t (p. 141)
syslog(3)

Data Structures

- struct **log_options_t**
  multiplexer configuration data

Defines

- #define **LOG_TRACEME**
  simple trace helper

Functions

- void **log_init** (**log_options_t** *options)
  initialize log message mutliplexer
- void **log_internal** (int level, int strerr, const char *fmt, va_list ap)
- int **log_emerg** (const char *fmt,...)
  send EMERG level message to the multiplexer
- int **log_alert** (const char *fmt,...)
  send ALERT level message to the multiplexer
- int **log_crit** (const char *fmt,...)
  send CRIT level message to the multiplexer
- int **log_error** (const char *fmt,...)
  send ERR level message to the multiplexer
- int **log_warn** (const char *fmt,...)
  send WARNING level message to the multiplexer

- int **log_notice** (const char *fmt,...)
  send NOTICE level message to the multiplexer
- int **log_info** (const char *fmt,...)
  send INFO level message to the multiplexer
- int **log_debug** (const char *fmt,...)
  send DEBUG level message to the multiplexer
- void **log_emerg_and_die** (const char *fmt,...)
  send EMERG level message to the multiplexer and exit(2)
- void **log_alert_and_die** (const char *fmt,...)
  send ALERT level message to the multiplexer and exit(2)
- void **log_crit_and_die** (const char *fmt,...)
  send CRIT level message to the multiplexer and exit(2)
- void **log_error_and_die** (const char *fmt,...)
  send ERR level message to the multiplexer and exit(2)
- int **log_pemerg** (const char *fmt,...)
  send EMERG level message to the multiplexer and append strerror(errno)
- int **log_palert** (const char *fmt,...)
  send ALERT level message to the multiplexer and append strerror(errno)
- int **log_pcrit** (const char *fmt,...)
  send CRIT level message to the multiplexer and append strerror(errno)
- int **log_perror** (const char *fmt,...)
  send ERR level message to the multiplexer and append strerror(errno)
- int **log_pwarn** (const char *fmt,...)
  send WARNING level message to the multiplexer and append strerror(errno)
- int **log_pnotice** (const char *fmt,...)
  send NOTICE level message to the multiplexer and append strerror(errno)
- int **log_pinfo** (const char *fmt,...)
  send INFO level message to the multiplexer and append strerror(errno)
- int **log_pdebug** (const char *fmt,...)
  send DEBUG level message to the multiplexer and append strerror(errno)
- void **log_pemerg_and_die** (const char *fmt,...)
  send EMERG level message to the multiplexer, append strerror(errno) and exit(2)
- void **log_palert_and_die** (const char *fmt,...)
  send ALERT level message to the multiplexer, append strerror(errno) and exit(2)

- void **log_pcrit_and_die** (const char *fmt,...)
  send CRIT level message to the multiplexer, append strerror(errno) and exit(2)
- void **log_perror_and_die** (const char *fmt,...)
  send ERR level message to the multiplexer, append strerror(errno) and exit(2)
- void **log_close** (void)
  close connection to logging system

6.9.2 Define Documentation

6.9.2.1 #define LOG_TRACEME

Value:

```
log_debug("[trace] %s (%s:%d)", \
                        __FUNCTION__, basename(__FILE__), __LINE__);
```

simple trace helper

Definition at line 87 of file log.h.

6.9.3 Function Documentation

6.9.3.1 void log_init (log_options_t * *options*)

initialize log message mutliplexer

Parameters:

← ***options*** multiplexer configuration

See also:

log_options_t (p. 141)

Definition at line 28 of file log_init.c.

References _log_options, log_options_t::facility, log_options_t::fd, log_options_t::file, log_options_t::flags, log_options_t::ident, log_options_t::mask, log_options_t::stderr, str_cpyn(), str_len(), and log_options_t::syslog.

```
29 {
30         struct stat sb;
31 
32         if (options->file && (options->fd < 0 || fstat(options->fd, &sb) == -1))
33                 options->file = false;
34 
35         if (options->stderr && fstat(STDERR_FILENO, &sb) == -1)
36                 options->stderr = false;
37 
38         if (options->mask == 0)
39                 options->mask = LOG_UPTO(LOG_INFO);
```

```
40
41         if (!options->ident || str_len(options->ident) < 1)
42                 options->ident = "(none)";
43
44         options->flags &= ~LOG_PERROR;
45
46         if (options->syslog) {
47                 openlog(options->ident, options->flags, options->facility);
48                 setlogmask(options->mask);
49         }
50
51         _log_options = (log_options_t *)malloc(sizeof(log_options_t));
52
53         str_cpyn(_log_options, options, sizeof(log_options_t));
54 }
```

6.9.3.2 void log_internal (int *level*, int *strerr*, const char * *fmt*, va_list *ap*)

Definition at line 69 of file log_internal.c.

References _log_options, _lucid_asprintf(), _lucid_vasprintf(), log_options_t::facility, log_options_t::fd, log_options_t::file, log_options_t::stderr, and log_options_t::syslog.

```
70 {
71         char *buf, *msg;
72
73         if (!_log_options)
74                 return;
75
76         _lucid_vasprintf(&msg, fmt, ap);
77
78         if (strerr) {
79                 buf = msg;
80                 _lucid_asprintf(&msg, "%s: %s", buf, strerror(errno_orig));
81                 free(buf);
82         }
83
84         if (_log_options->syslog)
85                 syslog(_log_options->facility|level, "%s", msg);
86
87         if (_log_options->stderr)
88                 log_fd(STDERR_FILENO, level, msg);
89
90         if (_log_options->file)
91                 log_fd(_log_options->fd, level, msg);
92
93         free(msg);
94 }
```

6.9.3.3 int log_emerg (const char * *fmt*, ...)

send EMERG level message to the multiplexer

Parameters:

← ***fmt*** format string passed to printf

← ***...*** variable number of arguments according to fmt

Returns:

EXIT_FAILURE

See also:

Formatted output conversion (p. 72)

6.9.3.4 int log_alert (const char * *fmt*, ...)

send ALERT level message to the multiplexer

Parameters:

← ***fmt*** format string passed to printf
← ***...*** variable number of arguments according to fmt

Returns:

EXIT_FAILURE

See also:

Formatted output conversion (p. 72)

6.9.3.5 int log_crit (const char * *fmt*, ...)

send CRIT level message to the multiplexer

Parameters:

← ***fmt*** format string passed to printf
← ***...*** variable number of arguments according to fmt

Returns:

EXIT_FAILURE

See also:

Formatted output conversion (p. 72)

6.9.3.6 int log_error (const char * *fmt*, ...)

send ERR level message to the multiplexer

Parameters:

← ***fmt*** format string passed to printf
← ***...*** variable number of arguments according to fmt

Returns:

EXIT_FAILURE

See also:

Formatted output conversion (p. 72)

6.9.3.7 int log_warn (const char * *fmt*, ...)

send WARNING level message to the multiplexer

Parameters:

← ***fmt*** format string passed to printf

← **...** variable number of arguments according to fmt

Returns:

EXIT_SUCCESS

See also:

Formatted output conversion (p. 72)

6.9.3.8 int log_notice (const char * *fmt*, ...)

send NOTICE level message to the multiplexer

Parameters:

← ***fmt*** format string passed to printf

← **...** variable number of arguments according to fmt

Returns:

EXIT_SUCCESS

See also:

Formatted output conversion (p. 72)

6.9.3.9 int log_info (const char * *fmt*, ...)

send INFO level message to the multiplexer

Parameters:

← ***fmt*** format string passed to printf

← **...** variable number of arguments according to fmt

Returns:

EXIT_SUCCESS

See also:

Formatted output conversion (p. 72)

6.9.3.10 int log_debug (const char * *fmt*, ...)

send DEBUG level message to the multiplexer

Parameters:

← ***fmt*** format string passed to printf

← ***...*** variable number of arguments according to fmt

Returns:

EXIT_SUCCESS

See also:

Formatted output conversion (p. 72)

6.9.3.11 void log_emerg_and_die (const char * *fmt*, ...)

send EMERG level message to the multiplexer and exit(2)

Parameters:

← ***fmt*** format string passed to printf

← ***...*** variable number of arguments according to fmt

See also:

Formatted output conversion (p. 72)
exit(2)

6.9.3.12 void log_alert_and_die (const char * *fmt*, ...)

send ALERT level message to the multiplexer and exit(2)

Parameters:

← ***fmt*** format string passed to printf

← ***...*** variable number of arguments according to fmt

See also:

Formatted output conversion (p. 72)
exit(2)

6.9.3.13 void log_crit_and_die (const char * *fmt*, ...)

send CRIT level message to the multiplexer and exit(2)

Parameters:

← ***fmt*** format string passed to printf

← **...** variable number of arguments according to fmt

See also:

Formatted output conversion (p. 72)
exit(2)

6.9.3.14 void log_error_and_die (const char * *fmt*, ...)

send ERR level message to the multiplexer and exit(2)

Parameters:

← ***fmt*** format string passed to printf
← **...** variable number of arguments according to fmt

See also:

Formatted output conversion (p. 72)
exit(2)

6.9.3.15 int log_pemerg (const char * *fmt*, ...)

send EMERG level message to the multiplexer and append strerror(errno)

Parameters:

← ***fmt*** format string passed to printf
← **...** variable number of arguments according to fmt

Returns:

EXIT_FAILURE

See also:

Formatted output conversion (p. 72)

6.9.3.16 int log_palert (const char * *fmt*, ...)

send ALERT level message to the multiplexer and append strerror(errno)

Parameters:

← ***fmt*** format string passed to printf
← **...** variable number of arguments according to fmt

Returns:

EXIT_FAILURE

See also:

Formatted output conversion (p. 72)

6.9.3.17 int log_pcrit (const char * *fmt*, ...)

send CRIT level message to the multiplexer and append strerror(errno)

Parameters:

← ***fmt*** format string passed to printf

← ***...*** variable number of arguments according to fmt

Returns:

EXIT_FAILURE

See also:

Formatted output conversion (p. 72)

6.9.3.18 int log_perror (const char * *fmt*, ...)

send ERR level message to the multiplexer and append strerror(errno)

Parameters:

← ***fmt*** format string passed to printf

← ***...*** variable number of arguments according to fmt

Returns:

EXIT_FAILURE

See also:

Formatted output conversion (p. 72)

6.9.3.19 int log_pwarn (const char * *fmt*, ...)

send WARNING level message to the multiplexer and append strerror(errno)

Parameters:

← ***fmt*** format string passed to printf

← ***...*** variable number of arguments according to fmt

Returns:

EXIT_SUCCESS

See also:

Formatted output conversion (p. 72)

6.9.3.20 int log_pnotice (const char * *fmt*, ...)

send NOTICE level message to the multiplexer and append strerror(errno)

Parameters:

← ***fmt*** format string passed to printf

← ***...*** variable number of arguments according to fmt

Returns:

EXIT_SUCCESS

See also:

Formatted output conversion (p. 72)

6.9.3.21 int log_pinfo (const char * *fmt*, ...)

send INFO level message to the multiplexer and append strerror(errno)

Parameters:

← ***fmt*** format string passed to printf

← ***...*** variable number of arguments according to fmt

Returns:

EXIT_SUCCESS

See also:

Formatted output conversion (p. 72)

6.9.3.22 int log_pdebug (const char * *fmt*, ...)

send DEBUG level message to the multiplexer and append strerror(errno)

Parameters:

← ***fmt*** format string passed to printf

← ***...*** variable number of arguments according to fmt

Returns:

EXIT_SUCCESS

See also:

Formatted output conversion (p. 72)

6.9.3.23 void log_pemerg_and_die (const char * *fmt*, ...)

send EMERG level message to the multiplexer, append strerror(errno) and exit(2)

Parameters:

← ***fmt*** format string passed to printf
← **...** variable number of arguments according to fmt

See also:

Formatted output conversion (p. 72)
exit(2)

6.9.3.24 void log_palert_and_die (const char * *fmt*, ...)

send ALERT level message to the multiplexer, append strerror(errno) and exit(2)

Parameters:

← ***fmt*** format string passed to printf
← **...** variable number of arguments according to fmt

See also:

Formatted output conversion (p. 72)
exit(2)

6.9.3.25 void log_pcrit_and_die (const char * *fmt*, ...)

send CRIT level message to the multiplexer, append strerror(errno) and exit(2)

Parameters:

← ***fmt*** format string passed to printf
← **...** variable number of arguments according to fmt

See also:

Formatted output conversion (p. 72)
exit(2)

6.9.3.26 void log_perror_and_die (const char * *fmt*, ...)

send ERR level message to the multiplexer, append strerror(errno) and exit(2)

Parameters:

← ***fmt*** format string passed to printf
← **...** variable number of arguments according to fmt

See also:

Formatted output conversion (p. 72)
exit(2)

6.9.3.27 void log_close (void)

close connection to logging system

Definition at line 26 of file log_close.c.

References _log_options, log_options_t::fd, log_options_t::file, and log_options_t::syslog.

```
27 {
28         if (!_log_options)
29                 return;
30
31         if (_log_options->syslog)
32                 closelog();
33
34         if (_log_options->file)
35                 close(_log_options->fd);
36
37         free(_log_options);
38 }
```

6.10 Miscellaneous helpers

6.10.1 Detailed Description

The misc family of functions provide wrappers not fitting in any other module and not being worth an own category for each of them.

The **isdir()** (p. 63), **isfile()** (p. 64) and **islink()** (p. 64) functions wrap the stat(2) system call and checks if the path in the string pointed to by path is a directory, regular file or link, respectively.

The **mkdirp()** (p. 65) function creates any missing parent directories of the path in the string pointed to by path, before creating the directory itself. The **mkdirnamep()** (p. 65) function additionally calls dirname(3) on the path string before calling **mkdirp()** (p. 65).

The path_concat() function concatenates the strings pointed to by dirname and basename and checks the latter using **str_path_isdot()** (p. 109).

The **runlink()** (p. 67) function removes all files and directories in the path pointed to by the string path.

Functions

- int **isdir** (const char *path)
  check if given path is a directory
- int **isfile** (const char *path)
  check if given path is a regular file
- int **islink** (const char *path)
  check if given path is a symbolic link
- int **mkdirnamep** (const char *path, mode_t mode)
  recursive mkdir(2) with dirname(3)
- int **mkdirp** (const char *path, mode_t mode)
  recursive mkdir(2)
- int **runlink** (const char *path)
  recursive unlink(2) and rmdir(2)

6.10.2 Function Documentation

6.10.2.1 int isdir (const char * *path*)

check if given path is a directory

Parameters:

← ***path*** path to check

Returns:

1 on success, 0 otherwise

See also:

stat(2)

Definition at line 22 of file isdir.c.

```
23 {
24         struct stat stats;
25         return stat(path, &stats) == 0 && S_ISDIR(stats.st_mode);
26 }
```

6.10.2.2 int isfile (const char * *path*)

check if given path is a regular file

Parameters:

← ***path*** path to check

Returns:

1 on success, 0 otherwise

See also:

stat(2)

Definition at line 22 of file isfile.c.

```
23 {
24         struct stat stats;
25         return stat(path, &stats) == 0 && S_ISREG(stats.st_mode);
26 }
```

6.10.2.3 int islink (const char * *path*)

check if given path is a symbolic link

Parameters:

← ***path*** path to check

Returns:

1 on success, 0 otherwise

See also:

stat(2)

Definition at line 22 of file islink.c.

```
23 {
24         struct stat stats;
25         return stat(path, &stats) == 0 && S_ISLNK(stats.st_mode);
26 }
```

6.10.2.4 int mkdirnamep (const char * *path*, mode_t *mode*)

recursive mkdir(2) with dirname(3)

Parameters:

← ***path*** path to create

← ***mode*** file permissions

Returns:

0 on success, -1 on error with errno set

See also:

mkdir(2)
dirname(3)

Definition at line 25 of file mkdirnamep.c.

References mkdirp(), str_dup(), and str_isempty.

```
26 {
27         char *buf, *dname;
28         int rc, errno_orig;
29
30         if (str_isempty(path))
31                 return errno = EINVAL, -1;
32
33         buf   = str_dup(path);
34         dname = dirname(buf);
35
36         rc = mkdirp(dname, mode);
37
38         errno_orig = errno;
39         free(buf);
40         errno = errno_orig;
41
42         return rc;
43 }
```

6.10.2.5 int mkdirp (const char * *path*, mode_t *mode*)

recursive mkdir(2)

Parameters:

← ***path*** path to create

← ***mode*** file permissions

Returns:

0 on success, -1 on error with errno set

See also:

mkdir(2)

Definition at line 25 of file mkdirp.c.

References str_dup(), and str_isempty.

Referenced by chroot_mkdirp(), and mkdirnamep().

```
26 {
27         int errno_orig, rc = 0;
28         char *p, *buf, c;
29         struct stat sb;
30 
31         if (str_isempty(path))
32                 return errno = EINVAL, -1;
33 
34         buf = p = str_dup(path);
35 
36         do {
37                 c = 0;
38 
39                 while (*p) {
40                         if (*p == '/') {
41                                 do { ++p; } while (*p == '/');
42                                 c  = *p;
43                                 *p = '\0';
44                                 break;
45                         }
46 
47                         else
48                                 ++p;
49                 }
50 
51                 if (mkdir(buf, 0777) == -1) {
52                         if (errno != EEXIST || stat(buf, &sb) == -1)
53                                 goto err;
54 
55                         if (!S_ISDIR(sb.st_mode)) {
56                                 errno = ENOTDIR;
57                                 goto err;
58                         }
59 
60                         if (!c)
61                                 goto out;
62                 }
63 
64                 if (!c) {
65                         if (chmod(buf, mode) == -1)
66                                 goto err;
67 
68                         goto out;
69                 }
70 
71                 *p = c;
72         } while (1);
73 
74 err:
75         rc = -1;
76 out:
77         errno_orig = errno;
78         free(buf);
79         errno = errno_orig;
80         return rc;
81 }
```

6.10.2.6 int runlink (const char * *path*)

recursive unlink(2) and rmdir(2)

Parameters:

← ***path*** path to remove

Returns:

0 on success, -1 on error with errno set

See also:

unlink(2)
rmdir(2)

Definition at line 27 of file runlink.c.

References _lucid_asprintf(), and runlink().

Referenced by runlink().

```
28 {
29         struct stat sb;
30 
31         DIR *dp;
32         struct dirent *d;
33 
34         int status = 0;
35         char *p, *new_path;
36 
37         if (lstat(path, &sb) == -1) {
38                 if (errno == ENOENT)
39                         return 0;
40                 else
41                         return -1;
42         }
43 
44         if (S_ISDIR(sb.st_mode)) {
45                 if (!(dp = opendir(path)))
46                         return -1;
47 
48                 while ((d = readdir(dp))) {
49                         p = d->d_name;
50 
51                         if (p && p[0] == '.' && (!p[1] || (p[1] == '.' && !p[2])))
52                                 continue;
53 
54                         _lucid_asprintf(&new_path, "%s/%s", path, d->d_name);
55 
56                         if (runlink(new_path) == -1)
57                                 status = -1;
58 
59                         free(new_path);
60                 }
61 
62                 if (closedir(dp) == -1)
63                         return -1;
64 
65                 if (rmdir(path) == -1)
66                         return -1;
67 
68                 return status;
```

```
69          }
70
71          if (unlink(path) == -1)
72                  return -1;
73
74          return 0;
75 }
```

6.11 Create or open files

6.11.1 Detailed Description

The open family of functions provide wrappers around open(2) with different flags.

Functions

- int **open_append** (const char *filename)
  open file in append mode
- int **open_excl** (const char *filename)
  open file exclusively
- int **open_read** (const char *filename)
  open file for reading
- int **open_rw** (const char *filename)
  open file for reading and writing
- int **open_trunc** (const char *filename)
  open and truncate file for reading and writing
- int **open_write** (const char *filename)
  open file for writing

6.11.2 Function Documentation

6.11.2.1 int open_append (const char * *filename*)

open file in append mode

Parameters:

filename file to open

Returns:

filedescriptor on success, -1 otherwise with errno set

See also:

Create or open files (p. 69)

Definition at line 23 of file open_append.c.

```
24 {
25         return open(filename, O_WRONLY|O_NONBLOCK|O_APPEND|O_CREAT, 0666);
26 }
```

6.11.2.2 int open_excl (const char * *filename*)

open file exclusively

Parameters:

filename file to open

Returns:

filedescriptor on success, -1 otherwise with errno set

See also:

Create or open files (p. 69)

Definition at line 23 of file open_excl.c.

```
24 {
25         return open(filename, O_WRONLY|O_NONBLOCK|O_CREAT|O_EXCL, 0666);
26 }
```

6.11.2.3 int open_read (const char * *filename*)

open file for reading

Parameters:

filename file to open

Returns:

filedescriptor on success, -1 otherwise with errno set

See also:

Create or open files (p. 69)

Definition at line 23 of file open_read.c.

Referenced by chroot_mkdirp(), and chroot_secure_chdir().

```
24 {
25         return open(filename, O_RDONLY|O_NONBLOCK);
26 }
```

6.11.2.4 int open_rw (const char * *filename*)

open file for reading and writing

Parameters:

filename file to open

Returns:

filedescriptor on success, -1 otherwise with errno set

See also:

Create or open files (p. 69)

Definition at line 23 of file open_rw.c.

```
24 {
25         return open(filename, O_RDWR|O_NONBLOCK|O_CREAT, 0666);
26 }
```

6.11.2.5 int open_trunc (const char * *filename*)

open and truncate file for reading and writing

Parameters:

filename file to open

Returns:

filedescriptor on success, -1 otherwise with errno set

See also:

Create or open files (p. 69)

Definition at line 23 of file open_trunc.c.

```
24 {
25         return open(filename, O_WRONLY|O_NONBLOCK|O_CREAT|O_TRUNC, 0666);
26 }
```

6.11.2.6 int open_write (const char * *filename*)

open file for writing

Parameters:

filename file to open

Returns:

filedescriptor on success, -1 otherwise with errno set

See also:

Create or open files (p. 69)

Definition at line 23 of file open_write.c.

```
24 {
25         return open(filename, O_WRONLY|O_NONBLOCK|O_CREAT, 0666);
26 }
```

6.12 Formatted output conversion

6.12.1 Detailed Description

The functions in the printf() family produce output according to a format as described below.

The functions printf() and vprintf() write output to stdout, the standard output stream; dprintf() and vdprintf() write output to the file descriptor fd; asprintf() and vasprintf() allocate a string long enough to hold the output; snprintf() and vsnprintf() write to the character string str.

The functions vprintf(), vdprintf(), vasprintf() and vsnprintf() are equivalent to the functions printf(), dprintf(), asprintf() and snprintf(), respectively, except that they are called with a va_-list instead of a variable number of arguments. These functions do not call the va_end macro. Consequently, the value of ap is undefined after the call. The application should call va_end(ap) itself afterwards.

6.12.2 Format of the format string

The format string is composed of zero or more directives: ordinary characters (not %), which are copied unchanged to the output stream; and conversion specifications, each of which results in fetching zero or more subsequent arguments. Each conversion specification is introduced by the character %, and ends with a conversion specifier. In between there may be (in this order) zero or more flags, an optional minimum field width, an optional precision and an optional length modifier.

6.12.2.1 The flag characters

The character % is followed by zero or more of the following flags:

- #

  The value should be converted to an "alternate form". For o conversions, the first character of the output string is made zero (by prefixing a 0 if it was not zero already). For x conversions, the result has the string '0x' prepended to it. For other conversions, the result is undefined.

- 0

  The value should be zero padded. For d, i, o, u, x, and f conversions, the converted value is padded on the left with zeros rather than blanks. If the 0 and - flags both appear, the 0 flag is ignored. If a precision is given with a numeric conversion (d, i, o, u, x, and X), the 0 flag is ignored. For other conversions, the behavior is undefined.

- -

  The converted value is to be left adjusted on the field boundary. (The default is right justification.) Except for n conversions, the converted value is padded on the right with blanks, rather than on the left with blanks or zeros. A - overrides a 0 if both are given.

- ' '

  (a space) A blank should be left before a positive number (or empty string) produced by a signed conversion.

- +

  A sign (+ or -) should always be placed before a number produced by a signed conversion. By default a sign is used only for negative numbers. A + overrides a space if both are used.

6.12.2.2 The field width

An optional decimal digit string (with non-zero first digit) specifying a minimum field width. If the converted value has fewer characters than the field width, it will be padded with spaces on the left (or right, if the left-adjustment flag has been given). Instead of a decimal digit string one may write '*' to specify that the field width is given in the next argument, which must be of type int. A negative field width is taken as a '-' flag followed by a positive field width. In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is expanded to contain the conversion result.

6.12.2.3 The length modifier

Here, 'integer conversion' stands for d, i, o, u, or x conversion.

- hh

  A following integer conversion corresponds to a signed char or unsigned char argument, or a following n conversion corresponds to a pointer to a signed char argument.

- h

  A following integer conversion corresponds to a short int or unsigned short int argument, or a following n conversion corresponds to a pointer to a short int argument.

- l

  (ell) A following integer conversion corresponds to a long int or unsigned long int argument.

- ll

  (ell-ell). A following integer conversion corresponds to a long long int or unsigned long long int argument.

Note however that internally, hh, h, and l are handled as long int, ll as long long int, respectively.

6.12.2.4 The conversion specifier

A character that specifies the type of conversion to be applied. The conversion specifiers and their meanings are:

- d,i

  The int argument is converted to signed decimal notation. The precision, if any, gives the minimum number of digits that must appear; if the converted value requires fewer digits, it is padded on the left with zeros. The default precision is 1. When 0 is printed with an explicit precision 0, the output is empty.

- o,u,x,X

  The unsigned int argument is converted to unsigned octal (o), unsigned decimal (u), or unsigned hexadecimal (x) notation. The letters abcdef are used for x conversions. The precision, if any, gives the minimum number of digits that must appear; if the converted value requires fewer digits, it is padded on the left with zeros. The default precision is 1. When 0 is printed with an explicit precision 0, the output is empty.

- c

  The int argument is converted to an unsigned char, and the resulting character is written.

- s

  The const char * argument is expected to be a pointer to an array of character type (pointer to a string). Characters from the array are written up to (but not including) a terminating null byte ('\0'); if a precision is specified, no more than the number specified are written. If a precision is given, no null byte need be present; if the precision is not specified, or is greater than the size of the array, the array must contain a terminating null byte.

- p

  The void * pointer argument is printed in hexadecimal.

- n

  The number of characters written so far is stored into the integer indicated by the int * pointer argument. No argument is converted.

- %

  A '' is written. No argument is converted. The complete conversion specification is '%'.

6.12.3 Note on conformance

This printf implementation is not fully C99 or SUS compliant, though most common features are implemented in a completely self-contained way, to make integration within other applications as easy as possible.

See also:

fmt

Functions

- int **_lucid_vsnprintf** (char *str, int size, const char *fmt, va_list ap)
  write conversion to string using va_list

- int **_lucid_snprintf** (char *str, int size, const char *fmt,...)
  write conversion to string using variable number of arguments

- int **_lucid_vasprintf** (char **ptr, const char *fmt, va_list ap)
  write conversion to allocated string using va_list

- int **_lucid_asprintf** (char **ptr, const char *fmt,...)
  write conversion to allocated string using variable number of arguments

- int **_lucid_vdprintf** (int fd, const char *fmt, va_list ap)
  write conversion to file descriptor using va_list

- int **_lucid_dprintf** (int fd, const char *fmt,...)
  write conversion to file descriptor using variable number of arguments

- int **_lucid_vprintf** (const char *fmt, va_list ap)
  write conversion to stdout using va_list

- int **_lucid_printf** (const char *fmt,...)
  write conversion to stdout using variable number of arguments

6.12.4 Function Documentation

6.12.4.1 int _lucid_vsnprintf (char * *str*, int *size*, const char * *fmt*, va_list *ap*)

write conversion to string using va_list

Parameters:

→ ***str*** buffer to store conversion

← ***size*** size of str

← ***fmt*** format string

← ***ap*** variable number of arguments

Returns:

number of bytes (that would have been) written

Note:

Every conversion happens in this functions. All other printf functions are just convenient wrappers.

Definition at line 178 of file vsnprintf.c.

References EMIT, __printf_t::f, __printf_t::l, __printf_t::p, PFL_ALT, PFL_BLANK, PFL_LEFT, PFL_SIGN, PFL_SIGNED, PFL_UPPER, PFL_ZERO, PFR_CHAR, PFR_INT, PFR_LLONG, PFR_LONG, PFR_MAX, PFR_MIN, PFR_SHORT, PFS_CONV, PFS_FLAGS, PFS_MOD, PFS_NORMAL, PFS_PREC, PFS_WIDTH, __printf_t::s, str_index(), str_len(), and __printf_t::w.

Referenced by _lucid_snprintf(), and _lucid_vasprintf().

```
179 {
180         /* generic counter */
181         int i;
182 
183         /* generic pointer */
184         const char *p;
185 
186         /* keep track of string length */
187         int idx = 0;
188 
189         /* save pointer to start of current conversion */
190         const char *ccp = fmt;
191 
192         /* current character in format */
193         char c;
194 
195         /* current conversion data */
196         __printf_t f;
197 
198         /* arguments */
199         union {
200                 /* signed argument */
201                 signed long long int d;
202 
203                 /* unsigned argument */
204                 unsigned long long int u;
205 
206                 /* float argument */
```

```
207                 double f;
208
209                 /* character argument */
210                 int c;
211
212                 /* string argument */
213                 const char *s;
214
215                 /* pointer argument */
216                 void *p;
217
218                 /* number argument */
219                 int *n;
220         } arg;
221
222         /* base used for integer conversions */
223         int base;
224
225         /* number of consumed bytes in conversions */
226         int len;
227
228         /* don't consume original ap */
229         va_list ap;
230         va_copy(ap, _ap);
231
232         /* initialize conversion data */
233         f.f = 0;
234         f.l = PFR_INT;
235         f.p = -1;
236         f.s = PFS_NORMAL;
237         f.w = 0;
238
239         while ((c = *fmt++)) {
240                 switch (f.s) {
241                 case PFS_NORMAL:
242                         if (c == '%') {
243                                 f.f = 0;
244                                 f.l = PFR_INT;
245                                 f.p = -1;
246                                 f.s = PFS_FLAGS;
247                                 f.w = 0;
248                                 ccp = &c;
249                         }
250
251                         else
252                                 EMIT(c)
253
254                         break;
255
256                 case PFS_FLAGS:
257                         switch (c) {
258                         case '#':
259                                 f.f |= PFL_ALT;
260                                 break;
261
262                         case '0':
263                                 f.f |= PFL_ZERO;
264                                 break;
265
266                         case '-':
267                                 f.f &= ~PFL_ZERO; /* left overrides zero */
268                                 f.f |=  PFL_LEFT;
269                                 break;
270
271                         case ' ':
272                                 f.f |= PFL_BLANK;
273                                 break;
```

```
274
275                     case '+':
276                             f.f &= ~PFL_BLANK; /* sign overrides blank */
277                             f.f |=  PFL_SIGN;
278                             break;
279
280                     default:
281                             f.s = PFS_WIDTH;
282                             fmt--;
283                             break;
284                     }
285
286                     break;
287
288             case PFS_WIDTH:
289                     if (c == '-') {
290                                     f.f &= PFL_ZERO; /* left overrides zero */
291                                     f.f |= PFL_LEFT;
292                     }
293
294                     else if (c >= '0' && c <= '9')
295                             f.w = f.w * 10  + (c - '0');
296
297                     else if (c == '*') {
298                             f.w = va_arg(ap, int);
299
300                             if (f.w < 0) {
301                                     f.w = -f.w;
302                                     f.f &= PFL_ZERO; /* left overrides zero */
303                                     f.f |= PFL_LEFT;
304                             }
305                     }
306
307                     else if (c == '.') {
308                             f.p = 0;
309                             f.s = PFS_PREC;
310                     }
311
312                     else {
313                             f.s = PFS_MOD;
314                             fmt--;
315                     }
316
317                     break;
318
319             case PFS_PREC:
320                     if (c >= '0' && c <= '9')
321                             f.p = f.p * 10  + (c - '0');
322
323                     else if (c == '*') {
324                             f.p = va_arg(ap, int);
325
326                             if (f.p < 0)
327                                     f.p = 0;
328                     }
329
330                     else {
331                             f.s = PFS_MOD;
332                             fmt--;
333                     }
334
335                     break;
336
337             case PFS_MOD:
338                     switch (c) {
339                     case 'h':
340                             f.l--;
```

```
341                             break;
342
343                     case 'l':
344                             f.l++;
345                             break;
346
347                     default:
348                             f.s = PFS_CONV;
349                             fmt--;
350                             break;
351                     }
352
353                     break;
354
355             case PFS_CONV:
356                     f.s = PFS_NORMAL;
357
358                     if (f.l > PFR_MAX)
359                             f.l = PFR_MAX;
360
361                     if (f.l < PFR_MIN)
362                             f.l = PFR_MIN;
363
364                     switch (c) {
365                     case 'P':
366                             f.f |= PFL_UPPER;
367
368                     case 'p':
369                             base = 16;
370                             f.p  = (8 * sizeof(void *) + 3)/4;
371                             f.f |= PFL_ALT;
372
373                             arg.u = (unsigned long long int) (unsigned long int) va_arg(ap, void *);
374
375                             goto is_integer;
376
377                     case 'd':
378                     case 'i': /* signed conversion */
379                             base = 10;
380                             f.f |= PFL_SIGNED;
381
382                             switch (f.l) {
383                             case PFR_CHAR:
384                                     arg.d = (signed char) va_arg(ap, signed int);
385                                     break;
386
387                             case PFR_SHORT:
388                                     arg.d = (signed short int) va_arg(ap, signed int);
389                                     break;
390
391                             case PFR_INT:
392                                     arg.d = (signed int) va_arg(ap, signed int);
393                                     break;
394
395                             case PFR_LONG:
396                                     arg.d = (signed long int) va_arg(ap, signed long int);
397                                     break;
398
399                             case PFR_LLONG:
400                                     arg.d = (signed long long int) va_arg(ap, signed long long int);
401                                     break;
402
403                             default:
404                                     arg.d = (signed long long int) va_arg(ap, signed int);
405                                     break;
406                             }
407
```

```
408                             arg.u = (unsigned long long int) arg.d;
409
410                             goto is_integer;
411
412                     case 'o':
413                             base = 8;
414                             goto is_unsigned;
415
416                     case 'u':
417                             base = 10;
418                             goto is_unsigned;
419
420                     case 'X':
421                             f.f |= PFL_UPPER;
422
423                     case 'x':
424                             base = 16;
425                             goto is_unsigned;
426
427                     is_unsigned:
428                             switch (f.l) {
429                             case PFR_CHAR:
430                                     arg.u = (unsigned char) va_arg(ap, unsigned int);
431                                     break;
432
433                             case PFR_SHORT:
434                                     arg.u = (unsigned short int) va_arg(ap, unsigned int);
435                                     break;
436
437                             case PFR_INT:
438                                     arg.u = (unsigned int) va_arg(ap, unsigned int);
439                                     break;
440
441                             case PFR_LONG:
442                                     arg.u = (unsigned long int) va_arg(ap, unsigned long int);
443                                     break;
444
445                             case PFR_LLONG:
446                                     arg.u = (unsigned long long int) va_arg(ap, unsigned long long int);
447                                     break;
448
449                             default:
450                                     arg.u = (unsigned long long int) va_arg(ap, unsigned int);
451                                     break;
452                             }
453
454                     is_integer:
455                             len = __printf_int(str, size, arg.u, base, f);
456
457                             str += len;
458                             idx += len;
459                             break;
460
461                     case 'c': /* character conversion */
462                             arg.c = (char) va_arg(ap, int);
463                             EMIT(arg.c)
464                             break;
465
466                     case 's': /* string conversion */
467                             arg.s = va_arg(ap, const char *);
468                             arg.s = arg.s ? arg.s : "(null)";
469                             len   = str_len(arg.s);
470
471                     is_string:
472                             if (f.p != -1 && len > f.p)
473                                     len = f.p;
474
```

```
475                                 while (f.w-- > len && !(f.f & PFL_LEFT)) {
476                                         if (f.f & PFL_ZERO)
477                                                 EMIT('0')
478                                         else
479                                                 EMIT(' ')
480                                 }
481
482                                 for (i = len; i; i--)
483                                         EMIT(*arg.s++)
484
485                                 while (f.w-- > len && (f.f & PFL_LEFT))
486                                         EMIT(' ')
487
488                                 break;
489
490                         case 'n':
491                                 arg.n  = va_arg(ap, int *);
492                                 *arg.n = idx;
493
494                                 break;
495
496                         case '%':
497                                 EMIT(c)
498                                 break;
499
500                         default:
501                                 /* no padding for unknown conversion */
502                                 f.w =  0;
503                                 f.p = -1;
504
505                                 arg.s = ccp;
506                                 len   = str_len(arg.s);
507                                 fmt   = ccp + len;
508
509                                 p = str_index(arg.s + 1, '%', len - 1);
510
511                                 if (p != 0) {
512                                         len = p - arg.s - 1;
513                                         fmt = p - 1;
514                                 }
515
516                                 goto is_string;
517                         }
518
519                         break;
520                 }
521         }
522
523         va_end(ap);
524
525         return idx;
526 }
```

6.12.4.2 int _lucid_snprintf (char * *str*, int *size*, const char * *fmt*, ...)

write conversion to string using variable number of arguments

Parameters:

→ ***str*** buffer to store conversion

← ***size*** size of str

← ***fmt*** format string

← ***...*** variable number of arguments

Returns:

number of bytes (that would have been) written

Definition at line 20 of file snprintf.c.

References _lucid_vsnprintf().

```
21 {
22         va_list ap;
23         va_start(ap, fmt);
24
25         return _lucid_vsnprintf(str, size, fmt, ap);
26 }
```

6.12.4.3 int _lucid_vasprintf (char ** *ptr*, const char * *fmt*, va_list *ap*)

write conversion to allocated string using va_list

Parameters:

→ ***ptr*** pointer to string to store conversion

← ***fmt*** format string

← ***ap*** variable number of arguments

Returns:

number of bytes (that would have been) written

See also:

malloc(3)
free(3)

Definition at line 22 of file vasprintf.c.

References _lucid_vsnprintf().

Referenced by _lucid_asprintf(), _lucid_vdprintf(), exec_fork(), exec_fork_background(), exec_fork_pipe(), exec_replace(), log_internal(), and stralloc_catf().

```
23 {
24         va_list ap2;
25         int len;
26         char *buf;
27
28         /* don't consume the original ap, we'll need it again */
29         va_copy(ap2, ap);
30
31         /* get required size */
32         len = _lucid_vsnprintf(0, 0, fmt, ap2);
33
34         va_end(ap2);
35
36         /* if size is 0, no buffer is allocated
37         ** just set *ptr to NULL and return size */
38         if (len > 0) {
39                 if (!(buf = calloc(len + 1, sizeof(char))))
40                         return -1;
```

```
41
42                 _lucid_vsnprintf(buf, len, fmt, ap);
43
44                 *ptr = buf;
45         }
46
47         return len;
48 }
```

6.12.4.4 int _lucid_asprintf (char ** *ptr*, const char * *fmt*, ...)

write conversion to allocated string using variable number of arguments

Parameters:

→ ***ptr*** pointer to string to store conversion

← ***fmt*** format string

← ***...*** variable number of arguments

Returns:

number of bytes (that would have been) written

See also:

malloc(3)
free(3)

Definition at line 20 of file asprintf.c.

References _lucid_vasprintf().

Referenced by addr_to_str(), log_internal(), runlink(), and str_path_concat().

```
21 {
22         va_list ap;
23         va_start(ap, fmt);
24
25         return _lucid_vasprintf(ptr, fmt, ap);
26 }
```

6.12.4.5 int _lucid_vdprintf (int *fd*, const char * *fmt*, va_list *ap*)

write conversion to file descriptor using va_list

Parameters:

← ***fd*** open file descriptor

← ***fmt*** format string

← ***ap*** variable number of arguments

Returns:

number of bytes (that would have been) written

Definition at line 23 of file vdprintf.c.

References _lucid_vasprintf().

Referenced by _lucid_dprintf(), and _lucid_vprintf().

```
24 {
25         char *buf;
26         int buflen, len;
27
28         buflen = _lucid_vasprintf(&buf, fmt, ap);
29         len = write(fd, buf, buflen);
30         free(buf);
31
32         return len;
33 }
```

6.12.4.6 int _lucid_dprintf (int *fd*, const char * *fmt*, ...)

write conversion to file descriptor using variable number of arguments

Parameters:

← ***fd*** open file descriptor

← ***fmt*** format string

← ***...*** variable number of arguments

Returns:

number of bytes (that would have been) written

Definition at line 20 of file dprintf.c.

References _lucid_vdprintf().

```
21 {
22         va_list ap;
23         va_start(ap, fmt);
24
25         return _lucid_vdprintf(fd, fmt, ap);
26 }
```

6.12.4.7 int _lucid_vprintf (const char * *fmt*, va_list *ap*)

write conversion to stdout using va_list

Parameters:

← ***fmt*** format string

← ***ap*** variable number of arguments

Returns:

number of bytes (that would have been) written

Definition at line 20 of file vprintf.c.

References _lucid_vdprintf().

Referenced by _lucid_printf().

```
21 {
22         return _lucid_vdprintf(1, fmt, ap);
23 }
```

6.12.4.8 int _lucid_printf (const char * *fmt*, ...)

write conversion to stdout using variable number of arguments

Parameters:

← ***fmt*** format string

← ***...*** variable number of arguments

Returns:

number of bytes (that would have been) written

Definition at line 20 of file printf.c.

References _lucid_vprintf().

```
21 {
22         va_list ap;
23         va_start(ap, fmt);
24 
25         return _lucid_vprintf(fmt, ap);
26 }
```

6.13 Formatted input conversion

6.13.1 Detailed Description

The scanf() family of functions scans input according to format as described below. This format may contain conversion specifications; the results from such conversions, if any, are stored in the locations pointed to by the pointer arguments that follow format. Each pointer argument must be of a type that is appropriate for the value returned by the corresponding conversion specification.

If the number of conversion specifications in format exceeds the number of pointer arguments, the results are undefined. If the number of pointer arguments exceeds the number of conversion specifications, then the excess pointer arguments are evaluated, but are otherwise ignored.

The format string consists of a sequence of directives which describe how to process the sequence of input characters. If processing of a directive fails, no further input is read, and scanf() returns. A "failure" can be either of the following: input failure, meaning that input characters were unavailable, or matching failure, meaning that the input was inappropriate (see below).

A directive is one of the following:

- A sequence of white-space characters (space, tab, newline, etc; see isspace(3)). This directive matches any amount of white space, including none, in the input.

- An ordinary character (i.e., one other than white space or "). This character must exactly match the next character of input.

- A conversion specification, which commences with a " (percent) character. A sequence of characters from the input is converted according to this specification, and the result is placed in the corresponding pointer argument. If the next item of input does not match the the conversion specification, the conversion fails âĂŤ this is a matching failure.

6.13.2 Format of the format string

Each conversion specification in format begins with either the character " followed by:

- An optional '∗' assignment-suppression character: scanf() reads input as directed by the conversion specification, but discards the input. No corresponding pointer argument is required, and this specification is not included in the count of successful assignments returned by scanf().

- An optional decimal integer which specifies the maximum field width. Reading of characters stops either when this maximum is reached or when a non-matching character is found, whichever happens first. Most conversions discard initial whitespace characters (the exceptions are noted below), and these discarded characters don't count towards the maximum field width. String input conversions store a null terminator ('\0') to mark the end of the input; the maximum field width does not include this terminator.

- An optional type modifier character. For example, the l type modifier is used with integer conversions such as d to specify that the corresponding pointer argument refers to a long int rather than a pointer to an int.

- A conversion specifier that specifies the type of input conversion to be performed.

6.13.2.1 Conversions

The following type modifier characters can appear in a conversion specification:

- hh

  A following integer conversion corresponds to a signed char or unsigned char argument, or a following n conversion corresponds to a pointer to a signed char argument.

- h

  A following integer conversion corresponds to a short int or unsigned short int argument, or a following n conversion corresponds to a pointer to a short int argument.

- l

  (ell) A following integer conversion corresponds to a long int or unsigned long int argument.

- ll

  (ell-ell). A following integer conversion corresponds to a long long int or unsigned long long int argument.

The following conversion specifiers are available:

- %

  Matches a literal ". That is, %% in the format string matches a single input " character. No conversion is done, and assignment does not occur.

- d

  Matches an optionally signed decimal integer; the next pointer must be a pointer to int.

- i

  Matches an optionally signed integer; the next pointer must be a pointer to int. The integer is read in base 16 if it begins with 0x or 0X, in base 8 if it begins with 0, and in base 10 otherwise. Only characters that correspond to the base are used.

- o

  Matches an unsigned octal integer; the next pointer must be a pointer to unsigned int.

- u

  Matches an unsigned decimal integer; the next pointer must be a pointer to unsigned int.

- x,X

  Matches an unsigned hexadecimal integer; the next pointer must be a pointer to unsigned int.

- s

  Matches a sequence of non-white-space characters; the next pointer must be a pointer to character array that is long enough to hold the input sequence and the terminating null character ('\0'), which is added automatically. The input string stops at white space or at the maximum field width, whichever occurs first.

- c

  Matches a sequence of characters whose length is specified by the maximum field width (default 1); the next pointer must be a pointer to char, and there must be enough room for all the characters (no terminating null byte is added). The usual skip of leading white space is suppressed. To skip white space first, use an explicit space in the format.

- p

  Matches a pointer value (as printed by p in printf(3); the next pointer must be a pointer to a pointer to void.

- n

  Nothing is expected; instead, the number of characters consumed thus far from the input is stored through the next pointer, which must be a pointer to int. This is not a conversion, although it can be suppressed with the * assignment-suppression character.

Functions

- int **_lucid_vsscanf** (const char *str, const char *fmt, va_list ap)
  read conversion from string using va_list

- int **_lucid_sscanf** (const char *str, const char *fmt,...)
  read conversion from string using variable number of arguments

6.13.3 Function Documentation

6.13.3.1 int _lucid_vsscanf (const char * *str*, const char * *fmt*, va_list *ap*)

read conversion from string using va_list

Parameters:

← ***str*** source string

← ***fmt*** format string

→ ***ap*** variable number of arguments

Returns:

Number of converted arguments

Note:

Every conversion happens in this functions. All other scanf functions are just convenient wrappers.

Definition at line 59 of file vsscanf.c.

References char_isspace, __scanf_t::f, __scanf_t::l, __scanf_t::s, SFL_NOOP, SFL_WIDTH, SFR_CHAR, SFR_INT, SFR_LLONG, SFR_LONG, SFR_MAX, SFR_MIN, SFR_SHORT, SFS_CONV, SFS_EOF, SFS_ERR, SFS_FLAGS, SFS_MOD, SFS_NORMAL, SFS_WIDTH, str_len(), str_toumax(), and __scanf_t::w.

Referenced by _lucid_sscanf().

```
60 {
61         /* keep track of converted arguments */
62         int converted = 0;
63 
64         /* current character in format */
65         char c;
```

```
66
67          /* current conversion data */
68          __scanf_t f;
69
70          /* arguments */
71          union {
72                  /* unsigned argument */
73                  unsigned long long int u;
74
75                  /* string argument */
76                  char *s;
77          } arg;
78
79          /* base used for integer conversions */
80          int base;
81
82          /* number of bytes converted in str_toumax */
83          int len;
84
85          /* pointer for string conversion */
86          char *sp;
87
88          /* don't consume original ap */
89          va_list ap;
90          va_copy(ap, _ap);
91
92          /* initialize conversion data */
93          f.f = 0;
94          f.l = SFR_INT;
95          f.s = SFS_NORMAL;
96          f.w = str_len(str);
97
98          while ((c = *fmt++)) {
99                  switch (f.s) {
100                  case SFS_NORMAL:
101                          if (c == '%') {
102                                  f.f = 0;
103                                  f.l = SFR_INT;
104                                  f.s = SFS_FLAGS;
105                                  f.w = str_len(str);
106                          }
107
108                          else if (char_isspace(c))
109                                  while (char_isspace(*str))
110                                          str++;
111
112                          else if (*str == c)
113                                  str++;
114
115                          else
116                                  f.s = SFS_ERR;
117
118                          break;
119
120                  case SFS_FLAGS:
121                          switch (c) {
122                          case '*':
123                                  f.f |= SFL_NOOP;
124                                  break;
125
126                          case '0':
127                          case '1':
128                          case '2':
129                          case '3':
130                          case '4':
131                          case '5':
132                          case '6':
```

```
133                         case '7':
134                         case '8':
135                         case '9':
136                                 f.w  = (c - '0');
137                                 f.f |= SFL_WIDTH;
138                                 f.s  = SFS_WIDTH;
139                                 break;
140
141                         default:
142                                 f.s = SFS_MOD;
143                                 fmt--;
144                                 break;
145                         }
146
147                         break;
148
149                 case SFS_WIDTH:
150                         if (c >= '0' && c <= '9')
151                                 f.w = f.w * 10  + (c - '0');
152
153                         else {
154                                 f.s = SFS_MOD;
155                                 fmt--;
156                         }
157
158                         break;
159
160                 case SFS_MOD:
161                         switch (c) {
162                         case 'h':
163                                 f.l--;
164                                 break;
165
166                         case 'l':
167                                 f.l++;
168                                 break;
169
170                         default:
171                                 f.s = SFS_CONV;
172                                 fmt--;
173                                 break;
174                         }
175
176                         break;
177
178                 case SFS_CONV:
179                         f.s = SFS_NORMAL;
180
181                         if (f.l > SFR_MAX)
182                                 f.l = SFR_MAX;
183
184                         if (f.l < SFR_MIN)
185                                 f.l = SFR_MIN;
186
187                         switch (c) {
188                         case 'd':
189                                 base = 10;
190                                 goto scan_int;
191
192                         case 'i': /* signed conversion */
193                                 base = 0;
194                                 goto scan_int;
195
196                         case 'o':
197                                 base = 8;
198                                 goto scan_int;
199
```

```
200                     case 'u':
201                             base = 10;
202                             goto scan_int;
203
204                     case 'X':
205                     case 'x':
206                             base = 16;
207                             goto scan_int;
208
209                     scan_int:
210                             while (char_isspace(*str))
211                                     str++;
212
213                             if (!*str) {
214                                     f.s = SFS_EOF;
215                                     break;
216                             }
217
218                             len = str_toumax(str, &arg.u, base, f.w);
219
220                             if (len <= 0) {
221                                     f.s = SFS_ERR;
222                                     break;
223                             }
224
225                             str += len;
226                             converted++;
227
228                             if (!(f.f & SFL_NOOP)) {
229                                     switch (f.l) {
230                                     case SFR_CHAR:
231                                             *va_arg(ap, unsigned char *) = arg.u;
232                                             break;
233
234                                     case SFR_SHORT:
235                                             *va_arg(ap, unsigned short int *) = arg.u;
236                                             break;
237
238                                     case SFR_INT:
239                                             *va_arg(ap, unsigned int *) = arg.u;
240                                             break;
241
242                                     case SFR_LONG:
243                                             *va_arg(ap, unsigned long int *) = arg.u;
244                                             break;
245
246                                     case SFR_LLONG:
247                                             *va_arg(ap, unsigned long long int *) = arg.u;
248                                             break;
249
250                                     default:
251                                             *va_arg(ap, unsigned long long int *) = arg.u;
252                                             break;
253                                     }
254                             }
255
256                             break;
257
258                     case 'c': /* character conversion */
259                             /* default width = 1 */
260                             f.w = (f.f & SFL_WIDTH) ? f.w : 1;
261
262                             if ((f.f & SFL_NOOP)) {
263                                     while (f.w--) {
264                                             if (!*str) {
265                                                     f.s = SFS_EOF;
266                                                     break;
```

```
267                                             }
268
269                                             str++;
270                                     }
271                             }
272
273                             else {
274                                     arg.s = va_arg(ap, char *);
275
276                                     while (f.w--) {
277                                             if (!*str) {
278                                                     f.s = SFS_EOF;
279                                                     break;
280                                             }
281
282                                             *arg.s++ = *str++;
283                                     }
284                             }
285
286                             if (f.s != SFS_EOF && !(f.f & SFL_NOOP))
287                                     converted++;
288
289                             break;
290
291                     case 's': /* string conversion */
292                             if (!(f.f & SFL_NOOP)) {
293                                     while (f.w-- && !char_isspace(*str)) {
294                                             if (!*str) {
295                                                     f.s = SFS_EOF;
296                                                     break;
297                                             }
298
299                                             str++;
300                                     }
301                             }
302
303                             else {
304                                     sp = arg.s = va_arg(ap, char *);
305
306                                     while (f.w-- && !char_isspace(*str)) {
307                                             if (!*str) {
308                                                     f.s = SFS_EOF;
309                                                     break;
310                                             }
311
312                                             *sp++ = *str++;
313                                     }
314
315                                     if (f.s != SFS_EOF)
316                                             *sp = '\0';
317                             }
318
319                             if (f.s != SFS_EOF && !(f.f & SFL_NOOP))
320                                     converted++;
321
322                             break;
323
324                     case 'P':
325                     case 'p': /* pointer conversion */
326                             while (char_isspace(*str))
327                                     str++;
328
329                             if (!*str) {
330                                     f.s = SFS_EOF;
331                                     break;
332                             }
333
```

```
334                                 len = str_toumax(str, &arg.u, 0, f.w);
335
336                                 if (len <= 0) {
337                                         f.s = SFS_ERR;
338                                         break;
339                                 }
340
341                                 if (!(f.f & SFL_NOOP))
342                                         *va_arg(ap, void **) = (void *) (unsigned long int) arg.u;
343
344                                 str += len;
345                                 converted++;
346
347                                 break;
348
349                         case 'n':
350                                 *va_arg(ap, int *) = converted;
351
352                                 break;
353
354                         case '%':
355                                 if (*str == '%')
356                                         str++;
357                                 else
358                                         f.s = SFS_ERR;
359
360                                 break;
361
362                         default:
363                                 f.s = SFS_ERR;
364                                 break;
365                         }
366
367                         break;
368
369                 case SFS_EOF:
370                         converted = converted ? converted : -1;
371
372                 case SFS_ERR:
373                         va_end(ap);
374                         return converted;
375                 }
376         }
377
378         if (f.s == SFS_EOF)
379                 converted = converted ? converted : -1;
380
381         va_end(ap);
382         return converted;
383 }
```

6.13.3.2 int _lucid_sscanf (const char * *str*, const char * *fmt*, ...)

read conversion from string using variable number of arguments

Parameters:

← ***str*** source string
← ***fmt*** format string
→ ***...*** variable number of arguments

Returns:

Number of converted arguments

Definition at line 20 of file sscanf.c.

References _lucid_vsscanf().

Referenced by addr_from_str().

```
21 {
22         va_list ap;
23         va_start(ap, fmt);
24 
25         return _lucid_vsscanf(str, fmt, ap);
26 }
```

6.14 String classification and conversion

6.14.1 Detailed Description

The char family of functions check whether ch, which must have the value of an unsigned char, falls into a certain character class.

The str_check family of functions extend the classification of single characters to strings. The **str_check()** (p. 103) function checks the string pointed to by str for a set of allowed character classes. As soon as a character is found that is not allowed checking stops and 0 is returned.

The **str_cmp()** (p. 104) function compares the string pointed to by str1 to the string pointed to by str2. It returns an integer less than, equal to, or greater than zero if str1 is found, respectively, to be less than, to match, or be greater than str2.

The strcpy() function copies the string pointed to by src (including the terminating '\0' character) to the array pointed to by dst. The strings may not overlap, and the destination string dst must be large enough to receive the copy. The strncpy() function is similar, except that not more than n bytes of src are copied. Thus, if there is no null byte among the first n bytes of src, the result will not be null-terminated.

The **str_dup()** (p. 105) function returns a pointer to a new string which is a duplicate of the string str. The **str_dupn()** (p. 106) function is similar, but only copies at most n characters. If s is longer than n, only n characters are copied, and a terminating null byte is added.

The **str_index()** (p. 106) returns a pointer to the first occurence of the character c in the string pointed to by str.

The **str_len()** (p. 107) function calculates the length of the string str, not including the terminating '\0' character.

The **str_path_concat()** (p. 108) function concatenates the directory name pointed to by dirname and file name pointed to by basename and checks that the latter does not contain any dot entries.

The **str_path_isabs()** (p. 108) and **str_path_isdot()** (p. 109) functions check if the file path pointed to by str is absolute or contains dots, respectively.

The **str_toupper()** (p. 110) and **str_tolower()** (p. 110) functions map lower-case to upper case and vice-versa, respectively.

The **str_zero()** (p. 107) function sets the first n bytes of the byte area starting at s to zero (bytes containing '\0').

The **str_toumax()** (p. 111) function converts the string pointed to by str to an unsigned long long int val using base as conversion base.

Defines

- #define **char_isascii**(ch) ((unsigned int)(ch) < 128u)
  check for an ASCII character
- #define **char_isblank**(ch) (ch == ' ' || ch == '\t')
  check for a blank character (space, horizontal tab)
- #define **char_iscntrl**(ch) ((unsigned int)(ch) < 32u || ch == 127)
  check for an ASCII control character

- #define **char_isdigit**(ch) ((unsigned int)(ch - '0') < 10u)
  check for a digit character (0-9)
- #define **char_isgraph**(ch) ((unsigned int)(ch - '!') < 94u)
  check for graphable characters (excluding space)
- #define **char_islower**(ch) ((unsigned int)(ch - 'a') < 26u)
  check for a lower-case character
- #define **char_isprint**(ch) ((unsigned int)(ch - ' ') < 95u)
  check for a printable character (including space)
- #define **char_isspace**(ch) ((unsigned int)(ch - '\t') < 5u || ch == ' ')
  check for a whitespace character (\t, \n, \v, \f, \r)
- #define **char_isupper**(ch) ((unsigned int)(ch - 'A') < 26u)
  check for an upper-case character
- #define **char_isxdigit**(ch)
  check for a hexadecimal character
- #define **char_isalpha**(ch) (char_islower(ch) || char_isupper(ch))
  check for an upper- or lower-case character
- #define **char_isalnum**(ch) (char_isalpha(ch) || char_isdigit(ch))
  check for an upper-, lower-case or digit character
- #define **char_ispunct**(ch)
  check for a punctuation character
- #define **char_tolower**(ch) do { if (char_isupper(ch)) ch += 32; } while(0)
  convert character to lower-case
- #define **char_toupper**(ch) do { if (char_islower(ch)) ch -= 32; } while(0)
  convert character to upper-case
- #define **CC_ALNUM** (1 << 1)
  class for alpha-numerical characters
- #define **CC_ALPHA** (1 << 2)
  class for upper- or lower-case characters
- #define **CC_ASCII** (1 << 3)
  class for ASCII characters
- #define **CC_BLANK** (1 << 4)
  class for blank characters
- #define **CC_CNTRL** (1 << 5)
  class for ASCII control characters

- #define **CC_DIGIT** (1 << 6)
  class for digit characters
- #define **CC_GRAPH** (1 << 7)
  class for graphable characters
- #define **CC_LOWER** (1 << 8)
  class for lower-case characters
- #define **CC_PRINT** (1 << 9)
  class for printable characters
- #define **CC_PUNCT** (1 << 10)
  class for punctuation characters
- #define **CC_SPACE** (1 << 11)
  class for white space characters
- #define **CC_UPPER** (1 << 12)
  class for upper-case characters
- #define **CC_XDIGIT** (1 << 13)
  class for hexadecimal characters
- #define **str_isempty**(str) (!str || str_check(str, CC_SPACE))
  check if string is empty
- #define **str_isalnum**(str) str_check(str, CC_ALNUM)
  check string for alpha-numerical characters
- #define **str_isalpha**(str) str_check(str, CC_ALPHA)
  check string for upper- or lower-case characters
- #define **str_isascii**(str) str_check(str, CC_ASCII)
  check string for ASCII characters
- #define **str_isdigit**(str) str_check(str, CC_DIGIT)
  check string for digit characters
- #define **str_isgraph**(str) str_check(str, CC_GRAPH)
  check string for graphable characters
- #define **str_islower**(str) str_check(str, CC_LOWER)
  check string for lower-case characters
- #define **str_isprint**(str) str_check(str, CC_PRINT)
  check string for printable characters
- #define **str_isupper**(str) str_check(str, CC_UPPER)

check string for upper-case characters

- #define **str_isxdigit**(str) str_check(str, CC_XDIGIT)
  check string for hexadecimal characters

Functions

- int **str_check** (const char *str, int allowed)
  check string against classes of allowed characters
- int **str_cmp** (const char *str1, const char *str2)
  compare two strings
- int **str_cpy** (char *dst, const char *src)
  copy a string
- int **str_cpyn** (void *dst, const void *src, int n)
  copy a string
- char * **str_dup** (const char *str)
  duplicate a string
- char * **str_dupn** (const char *str, int n)
  duplicate a string
- char * **str_index** (const char *str, int c, int n)
  scan string for character
- int **str_len** (const char *str)
  calculate the length of a string
- void **str_zero** (void *str, int n)
  write zero-valued bytes
- char * **str_path_concat** (const char *dirname, const char *basename)
  concatenate dirname and basename
- int **str_path_isabs** (const char *str)
  check if path is absolute and contains no dot entries or ungraphable characters
- int **str_path_isdot** (const char *str)
  check if given path contains . or .. entries
- char * **str_tolower** (char *str)
  convert string to lower-case
- char * **str_toupper** (char *str)
  convert string to upper-case

- int **str_toumax** (const char *str, unsigned long long int *val, int base, int n)
  convert string to integer

6.14.2 Define Documentation

6.14.2.1 #define char_isascii(ch) ((unsigned int)(ch) < 128u)

check for an ASCII character
Definition at line 75 of file str.h.
Referenced by str_check().

6.14.2.2 #define char_isblank(ch) (ch == ' ' || ch == '\t')

check for a blank character (space, horizontal tab)
Definition at line 78 of file str.h.
Referenced by str_check().

6.14.2.3 #define char_iscntrl(ch) ((unsigned int)(ch) < 32u || ch == 127)

check for an ASCII control character
Definition at line 81 of file str.h.
Referenced by str_check().

6.14.2.4 #define char_isdigit(ch) ((unsigned int)(ch - '0') < 10u)

check for a digit character (0-9)
Definition at line 84 of file str.h.
Referenced by str_check().

6.14.2.5 #define char_isgraph(ch) ((unsigned int)(ch - '!') < 94u)

check for graphable characters (excluding space)
Definition at line 87 of file str.h.
Referenced by str_check().

6.14.2.6 #define char_islower(ch) ((unsigned int)(ch - 'a') < 26u)

check for a lower-case character
Definition at line 90 of file str.h.
Referenced by str_check().

6.14.2.7 #define char_isprint(ch) ((unsigned int)(ch - ' ') < 95u)

check for a printable character (including space)

Definition at line 93 of file str.h.

Referenced by str_check().

6.14.2.8 #define char_isspace(ch) ((unsigned int)(ch - '\t') < 5u || ch == ' ')

check for a whitespace character (\t, \n, \v, \f, \r)

Definition at line 96 of file str.h.

Referenced by _lucid_vsscanf(), argv_from_str(), flist32_from_str(), flist64_from_str(), str_check(), and str_toumax().

6.14.2.9 #define char_isupper(ch) ((unsigned int)(ch - 'A') < 26u)

check for an upper-case character

Definition at line 99 of file str.h.

Referenced by str_check().

6.14.2.10 #define char_isxdigit(ch)

Value:

```
(char_isdigit(ch) || \
                        (unsigned int)(ch - 'a') < 6u || \
                        (unsigned int)(ch - 'A') < 6u)
```

check for a hexadecimal character

Definition at line 102 of file str.h.

Referenced by str_check().

6.14.2.11 #define char_isalpha(ch) (char_islower(ch) || char_isupper(ch))

check for an upper- or lower-case character

Definition at line 108 of file str.h.

Referenced by str_check().

6.14.2.12 #define char_isalnum(ch) (char_isalpha(ch) || char_isdigit(ch))

check for an upper-, lower-case or digit character

Definition at line 111 of file str.h.

Referenced by str_check().

6.14.2.13 #define char_ispunct(ch)

Value:

```
(char_isprint(ch) && \
                        !char_isalnum(ch) && \
                        !char_isspace(ch))
```

check for a punctuation character

Definition at line 114 of file str.h.

Referenced by str_check().

6.14.2.14 #define char_tolower(ch) do { if (char_isupper(ch)) ch += 32; } while(0)

convert character to lower-case

Definition at line 120 of file str.h.

Referenced by str_tolower().

6.14.2.15 #define char_toupper(ch) do { if (char_islower(ch)) ch -= 32; } while(0)

convert character to upper-case

Definition at line 123 of file str.h.

Referenced by str_toupper().

6.14.2.16 #define CC_ALNUM (1 << 1)

class for alpha-numerical characters

Definition at line 127 of file str.h.

Referenced by str_check().

6.14.2.17 #define CC_ALPHA (1 << 2)

class for upper- or lower-case characters

Definition at line 130 of file str.h.

Referenced by str_check().

6.14.2.18 #define CC_ASCII (1 << 3)

class for ASCII characters

Definition at line 133 of file str.h.

Referenced by str_check().

6.14.2.19 #define CC_BLANK (1 << 4)

class for blank characters

Definition at line 136 of file str.h.

Referenced by str_check().

6.14.2.20 #define CC_CNTRL (1 << 5)

class for ASCII control characters

Definition at line 139 of file str.h.

Referenced by str_check().

6.14.2.21 #define CC_DIGIT (1 << 6)

class for digit characters

Definition at line 142 of file str.h.

Referenced by str_check().

6.14.2.22 #define CC_GRAPH (1 << 7)

class for graphable characters

Definition at line 145 of file str.h.

Referenced by str_check().

6.14.2.23 #define CC_LOWER (1 << 8)

class for lower-case characters

Definition at line 148 of file str.h.

Referenced by str_check().

6.14.2.24 #define CC_PRINT (1 << 9)

class for printable characters

Definition at line 151 of file str.h.

Referenced by str_check().

6.14.2.25 #define CC_PUNCT (1 << 10)

class for punctuation characters

Definition at line 154 of file str.h.

Referenced by str_check().

6.14.2.26 #define CC_SPACE (1 << 11)

class for white space characters

Definition at line 157 of file str.h.

Referenced by str_check().

6.14.2.27 #define CC_UPPER (1 << 12)

class for upper-case characters

Definition at line 160 of file str.h.

Referenced by str_check().

6.14.2.28 #define CC_XDIGIT (1 << 13)

class for hexadecimal characters

Definition at line 163 of file str.h.

Referenced by str_check().

6.14.2.29 #define str_isempty(str) (!str || str_check(str, CC_SPACE))

check if string is empty

Definition at line 176 of file str.h.

Referenced by addr_from_str(), flist32_from_str(), flist64_from_str(), mkdirnamep(), mkdirp(), str_path_concat(), str_path_isabs(), and str_path_isdot().

6.14.2.30 #define str_isalnum(str) str_check(str, CC_ALNUM)

check string for alpha-numerical characters

Definition at line 179 of file str.h.

6.14.2.31 #define str_isalpha(str) str_check(str, CC_ALPHA)

check string for upper- or lower-case characters

Definition at line 182 of file str.h.

6.14.2.32 #define str_isascii(str) str_check(str, CC_ASCII)

check string for ASCII characters

Definition at line 185 of file str.h.

6.14.2.33 #define str_isdigit(str) str_check(str, CC_DIGIT)

check string for digit characters

Definition at line 188 of file str.h.

Referenced by addr_from_str().

6.14.2.34 #define str_isgraph(str) str_check(str, CC_GRAPH)

check string for graphable characters

Definition at line 191 of file str.h.

Referenced by str_path_isabs().

6.14.2.35 #define str_islower(str) str_check(str, CC_LOWER)

check string for lower-case characters

Definition at line 194 of file str.h.

6.14.2.36 #define str_isprint(str) str_check(str, CC_PRINT)

check string for printable characters

Definition at line 197 of file str.h.

6.14.2.37 #define str_isupper(str) str_check(str, CC_UPPER)

check string for upper-case characters

Definition at line 200 of file str.h.

6.14.2.38 #define str_isxdigit(str) str_check(str, CC_XDIGIT)

check string for hexadecimal characters

Definition at line 203 of file str.h.

6.14.3 Function Documentation

6.14.3.1 int str_check (const char * *str*, int *allowed*)

check string against classes of allowed characters

Parameters:

← ***str*** string to check

← ***allowed*** allowed classes of characters (multiple classes by ORing)

Returns:

1 if all characters are valid, 0 otherwise

Definition at line 20 of file str_check.c.

References CC_ALNUM, CC_ALPHA, CC_ASCII, CC_BLANK, CC_CNTRL, CC_DIGIT, CC_GRAPH, CC_LOWER, CC_PRINT, CC_PUNCT, CC_SPACE, CC_UPPER, CC_XDIGIT, char_isalnum, char_isalpha, char_isascii, char_isblank, char_iscntrl, char_isdigit, char_isgraph, char_islower, char_isprint, char_ispunct, char_isspace, char_isupper, char_isxdigit, and str_len().

```
21 {
22         int i, n;
23
24         if (!str)
25                 return 1;
26
27         n = str_len(str);
28
29         for (i = 0; i < n; i++) {
30                 if (allowed & CC_ALNUM  && char_isalnum (str[i])) continue;
31                 if (allowed & CC_ALPHA  && char_isalpha (str[i])) continue;
32                 if (allowed & CC_ASCII  && char_isascii (str[i])) continue;
33                 if (allowed & CC_BLANK  && char_isblank (str[i])) continue;
34                 if (allowed & CC_CNTRL  && char_iscntrl (str[i])) continue;
35                 if (allowed & CC_DIGIT  && char_isdigit (str[i])) continue;
36                 if (allowed & CC_GRAPH  && char_isgraph (str[i])) continue;
37                 if (allowed & CC_LOWER  && char_islower (str[i])) continue;
38                 if (allowed & CC_PRINT  && char_isprint (str[i])) continue;
39                 if (allowed & CC_PUNCT  && char_ispunct (str[i])) continue;
40                 if (allowed & CC_SPACE  && char_isspace (str[i])) continue;
41                 if (allowed & CC_UPPER  && char_isupper (str[i])) continue;
42                 if (allowed & CC_XDIGIT && char_isxdigit(str[i])) continue;
43
44                 return 0;
45         }
46
47         return 1;
48 }
```

6.14.3.2 int str_cmp (const char * *str1*, const char * *str2*)

compare two strings

Parameters:

← ***str1*** first string

← ***str2*** second string

Returns:

An integer greater than, equal to, or less than 0, if the string pointed to by str1 is greater than, equal to, or less than the string pointed to by str2, respectively.

Definition at line 20 of file str_cmp.c.

Referenced by flist32_getval(), flist64_getval(), and str_path_isdot().

```
21 {
22         while (*str1 && *str2 && *str1 == *str2)
23                 str1++, str2++;
24
25         return *str1 - *str2;
26 }
```

6.14.3.3 int str_cpy (char * *dst*, const char * *src*)

copy a string

Parameters:

→ ***dst*** destination string

← ***src*** source string

Returns:

Number of bytes that have been copied.

Definition at line 20 of file str_cpy.c.

References str_cpyn(), and str_len().

```
21 {
22         return str_cpyn(dst, src, str_len(src));
23 }
```

6.14.3.4 int str_cpyn (void * *dst*, const void * *src*, int *n*)

copy a string

Parameters:

→ ***dst*** destination string

← ***src*** source string

← ***n*** copy first n bytes

Returns:

Number of bytes that have been copied.

Definition at line 20 of file str_cpyn.c.

Referenced by log_init(), str_cpy(), str_dupn(), stralloc_catb(), stralloc_copyb(), and whirlpool_finalize().

```
21 {
22         char *d = dst;
23         const char *s = src;
24 
25         while (n--)
26                 *d++ = *s++;
27 
28         return d - (char *) dst;
29 }
```

6.14.3.5 char* str_dup (const char * *str*)

duplicate a string

Parameters:

← ***str*** source string

Returns:

A pointer to the duplicated string, or NULL if insufficient memory was available.

Definition at line 20 of file str_dup.c.

References str_dupn(), and str_len().

Referenced by flist32_from_str(), flist64_from_str(), mkdirnamep(), mkdirp(), str_path_isabs(), and str_path_isdot().

```
21 {
22         return str_dupn(str, str_len(str));
23 }
```

6.14.3.6 char* str_dupn (const char * *str*, int *n*)

duplicate a string

Parameters:

← ***str*** source string

← ***n*** string length

Returns:

A pointer to the duplicated string, or NULL if insufficient memory was available.

Definition at line 22 of file str_dupn.c.

References str_cpyn().

Referenced by argv_to_str(), flist32_to_str(), flist64_to_str(), io_read_eof(), io_read_eol(), io_read_len(), str_dup(), and whirlpool_digest().

```
23 {
24         char *buf = calloc(n + 1, sizeof(char));
25
26         if (buf)
27                 str_cpyn(buf, str, n);
28
29         return buf;
30 }
```

6.14.3.7 char* str_index (const char * *str*, int *c*, int *n*)

scan string for character

Parameters:

← ***str*** string to scan

← ***c*** character to look for

← ***n*** scan first n bytes

Returns:

A pointer to the matched character or NULL if the character is not found.

Definition at line 20 of file str_index.c.

Referenced by _lucid_vsnprintf(), addr_from_str(), flist32_from_str(), flist64_from_str(), str_path_isabs(), and str_path_isdot().

```
21 {
22         char *p = (char *) str;
23
24         for (; n; p++, n--)
25                 if (*p == c)
26                         return p;
27
28         return 0;
29 }
```

6.14.3.8 int str_len (const char * *str*)

calculate the length of a string

Parameters:

← ***str*** source string

Returns:

number of characters in str

Definition at line 20 of file str_len.c.

Referenced by _lucid_vsnprintf(), _lucid_vsscanf(), addr_from_str(), flist32_from_str(), flist64_from_str(), log_init(), str_check(), str_cpy(), str_dup(), str_path_isabs(), str_path_isdot(), stralloc_cats(), stralloc_copys(), and whirlpool_digest().

```
21 {
22         int i = 0;
23
24         while (*str++)
25                 i++;
26
27         return i;
28 }
```

6.14.3.9 void str_zero (void * *str*, int *n*)

write zero-valued bytes

Parameters:

→ ***str*** destination string

← ***n*** write first n bytes

Definition at line 20 of file str_zero.c.

Referenced by tcp_connect(), tcp_listen(), whirlpool_finalize(), and whirlpool_init().

```
21 {
22         char *p = str;
23
24         while (n--)
25                 *p++ = 0;
26 }
```

6.14.3.10 char* str_path_concat (const char * *dirname*, const char * *basename*)

concatenate dirname and basename

Parameters:

← ***dirname*** directory part

← ***basename*** basename part

Returns:

A pointer to the newly allocated string or NULL if insufficent memory was available.

Definition at line 22 of file str_path_concat.c.

References _lucid_asprintf(), str_isempty, and str_path_isdot().

```
23 {
24         char *path = 0;
25
26         if (str_isempty(dirname) || str_path_isdot(basename))
27                 return 0;
28
29         _lucid_asprintf(&path, "%s/%s", dirname, basename);
30
31         return path;
32 }
```

6.14.3.11 int str_path_isabs (const char * *str*)

check if path is absolute and contains no dot entries or ungraphable characters

Parameters:

← ***str*** path to check

Returns:

1 if str is an absolute pathname, 0 otherwise

Note:

this function does not check if the path exists

Definition at line 22 of file str_path_isabs.c.

References str_dup(), str_index(), str_isempty, str_isgraph, str_len(), and str_path_isdot().

```
23 {
24         char *buf, *p, *o;
25
26         if (str_isempty(str))
27                 return 0;
28
29         if (*str != '/')
30                 return 0;
31
32         if (str_path_isdot(str))
33                 return 0;
34
35         buf = p = o = str_dup(str);
36
37         while (1) {
38                 p = str_index(p, '/', str_len(p));
39
40                 if (p)
41                         *p++ = '\0';
42
43                 if (!str_isgraph(o)) {
44                         free(buf);
45                         return 0;
46                 }
47
48                 if (!p)
49                         break;
50                 else
51                         o = p;
52         }
53
54         free(buf);
55         return 1;
56 }
```

6.14.3.12 int str_path_isdot (const char * *str*)

check if given path contains . or .. entries

Parameters:

← ***str*** path to check

Returns:

1 if str has dot entries, 0 otherwise

Definition at line 22 of file str_path_isdot.c.

References str_cmp(), str_dup(), str_index(), str_isempty, and str_len().

Referenced by str_path_concat(), and str_path_isabs().

```
23 {
24         char *buf, *p, *o;
25
26         if (str_isempty(str))
27                 return 0;
28
29         buf = p = o = str_dup(str);
30
31         while (1) {
32                 p = str_index(p, '/', str_len(p));
```

```
33
34                  if (p)
35                          *p++ = '\0';
36
37                  if (str_cmp(o, ".") == 0 || str_cmp(o, "..") == 0) {
38                          free(buf);
39                          return 1;
40                  }
41
42                  if (!p)
43                          break;
44                  else
45                          o = p;
46          }
47
48          free(buf);
49          return 0;
50  }
```

6.14.3.13 char* str_tolower (char * *str*)

convert string to lower-case

Parameters:

→ ***str*** string to convert

Returns:

pointer to str

Definition at line 20 of file str_tolower.c.

References char_tolower.

```
21 {
22          char *p = str;
23
24          while (*p) {
25                  char_tolower(*p);
26                  p++;
27          }
28
29          return str;
30 }
```

6.14.3.14 char* str_toupper (char * *str*)

convert string to upper-case

Parameters:

→ ***str*** string to convert

Returns:

pointer to str

Definition at line 20 of file str_toupper.c.

References char_toupper.

```
21 {
22         char *p = str;
23
24         while (*p) {
25                 char_toupper(*p);
26                 p++;
27         }
28
29         return str;
30 }
```

6.14.3.15 int str_toumax (const char * *str*, unsigned long long int * *val*, int *base*, int *n*)

convert string to integer

Parameters:

← ***str*** source string

→ ***val*** destination integer

← ***base*** conversion base

← ***n*** convert first n bytes

Returns:

Number of bytes read from str

Definition at line 36 of file str_toumax.c.

References char_isspace.

Referenced by _lucid_vsscanf(), and addr_from_str().

```
37 {
38         char c;
39         const char *p = str;
40         int d, minus = 0;
41         unsigned long long int v = 0;
42
43         while (n && char_isspace((unsigned char) *p)) {
44                 p++;
45                 n--;
46         }
47
48         /* Single optional + or - */
49         if (n) {
50                 c = *p;
51
52                 if (c == '-' || c == '+') {
53                         minus = (c == '-');
54                         p++;
55                         n--;
56                 }
57         }
58
59         if (base == 0) {
```

```
60                 if (n >= 2 && p[0] == '0' && (p[1] == 'x' || p[1] == 'X')) {
61                         n -= 2;
62                         p += 2;
63                         base = 16;
64                 }
65
66                 else if (n >= 1 && p[0] == '0') {
67                         n--;
68                         p++;
69                         base = 8;
70                 }
71
72                 else {
73                         base = 10;
74                 }
75         }
76
77         else if (base == 16) {
78                 if (n >= 2 && p[0] == '0' && (p[1] == 'x' || p[1] == 'X')) {
79                         n -= 2;
80                         p += 2;
81                 }
82         }
83
84         while (n && (d = char_todigit(*p)) >= 0 && d < base) {
85                 v = v * base + d;
86                 n--;
87                 p++;
88         }
89
90         if (p - str > 0)
91                 *val = minus ? -v : v;
92
93         return p - str;
94 }
```

6.15 Dynamic string allocator

6.15.1 Detailed Description

A stralloc variable holds a byte string in dynamically allocated space. String contents are unrestricted; in particular, strings may contain \0. String length is limited only by memory and by the size of an unsigned int.

A stralloc structure has three components: sa.s is a pointer to the first byte of the string, or 0 if space is not allocated; sa.len is the number of bytes in the string, or undefined if space is not allocated; sa.a is the number of bytes allocated for the string, or undefined if space is not allocated.

Applications are expected to use sa.s and sa.len directly.

The **stralloc_ready()** (p. 115) function makes sure that sa has enough space allocated to hold len bytes. The **stralloc_readyplus()** (p. 115) function is like **stralloc_ready()** (p. 115) except that, if sa is already allocated, stralloc_readyplus adds the current length of sa to len.

The **stralloc_copyb()** (p. 116) function copies the string pointed to by src into dst, allocating space if necessary. The **stralloc_copys()** (p. 117) function copies a \0-terminated string from src into dst, without the \0. It is the same as stralloc_copyb(&dst,buf,str_len(buf)). stralloc_copy copies the string stored in src into dst. It is the same as stralloc_copyb(&dst,src.s,src.len); src must already be allocated.

The **stralloc_catb()** (p. 118) adds the string pointed to by src to the end of the string stored in dst, allocating space if necessary. If dst is unallocated, stralloc_catb is the same as stralloc_copyb. The **stralloc_cats()** (p. 119) function is analogous to stralloc_copys, and stralloc_cat is analogous to stralloc_copy. The **stralloc_catf()** (p. 118) and **stralloc_catm()** (p. 119) functions are analogous to **stralloc_cats()** (p. 119) except that they take a formatted conversion or variable number of arguments, respectively, and appends these to the string stored in dst.

Data Structures

- struct **stralloc_t**

  dynamic string allocator tracking data

Functions

- void **stralloc_init** (**stralloc_t** *sa)

  initialize dynamic string allocator

- void **stralloc_zero** (**stralloc_t** *sa)

  truncate string length to zero

- int **stralloc_ready** (**stralloc_t** *sa, size_t len)

  ensure that enough memory has been allocated

- int **stralloc_readyplus** (**stralloc_t** *sa, size_t len)

  ensure that enough memory has been allocated

- void **stralloc_free** (**stralloc_t** *sa)

  deallocate all memory

- int **stralloc_copyb** (**stralloc_t** *dst, const char *src, size_t len)
  copy a static string to a dynamic one
- int **stralloc_copys** (**stralloc_t** *dst, const char *src)
  copy a static string to a dynamic one
- int **stralloc_copy** (**stralloc_t** *dst, const **stralloc_t** *src)
  copy one dynamic string to another
- int **stralloc_catb** (**stralloc_t** *dst, const char *src, size_t len)
  concatenate a dynamic string and a static one
- int **stralloc_catf** (**stralloc_t** *dst, const char *fmt,...)
  concatenate a dynamic string and a static one using formatted conversion
- int **stralloc_catm** (**stralloc_t** *dst,...)
  concatenate a dynamic string and multiple static ones
- int **stralloc_cats** (**stralloc_t** *dst, const char *src)
  concatenate a dynamic string and a static one
- int **stralloc_cat** (**stralloc_t** *dst, const **stralloc_t** *src)
  concatenate two dynamic strings
- int **stralloc_cmp** (const **stralloc_t** *a, const **stralloc_t** *b)
  compare two dynamic strings

6.15.2 Function Documentation

6.15.2.1 void stralloc_init (stralloc_t * *sa*)

initialize dynamic string allocator

Parameters:

→ ***sa*** string to initialize

Definition at line 23 of file stralloc_init.c.

References stralloc_t::a, stralloc_t::len, and stralloc_t::s.

Referenced by argv_to_str(), flist32_to_str(), flist64_to_str(), and whirlpool_digest().

```
24 {
25         sa->s   = NULL;
26         sa->len = sa->a = 0;
27 }
```

6.15.2.2 void stralloc_zero (stralloc_t * *sa*)

truncate string length to zero

Parameters:

→ ***sa*** string to truncate

Definition at line 21 of file stralloc_zero.c.

References stralloc_t::len.

```
22 {
23         sa->len = 0;
24 }
```

6.15.2.3 int stralloc_ready (stralloc_t * *sa*, size_t *len*)

ensure that enough memory has been allocated

Parameters:

← ***sa*** string to check

← ***len*** minimum length that has to be available

Returns:

0 on success, -1 on error with errno set

Definition at line 23 of file stralloc_ready.c.

References stralloc_t::a, and stralloc_t::s.

Referenced by stralloc_copyb(), and stralloc_readyplus().

```
24 {
25         size_t wanted = len + (len >> 3) + 30;
26         char *tmp;
27
28         if (!sa->s || sa->a < len) {
29                 if (!(tmp = realloc(sa->s, wanted)))
30                         return -1;
31
32                 sa->a = wanted;
33                 sa->s = tmp;
34         }
35
36         return 0;
37 }
```

6.15.2.4 int stralloc_readyplus (stralloc_t * *sa*, size_t *len*)

ensure that enough memory has been allocated

Parameters:

← ***sa*** string to check

← ***len*** additional length that has to be available

Returns:

0 on success, -1 on error with errno set

Definition at line 23 of file stralloc_readyplus.c.

References stralloc_t::len, stralloc_t::s, and stralloc_ready().

Referenced by stralloc_catb().

```
24 {
25         if (sa->s) {
26                 if (sa->len + len < len)
27                         return errno = EINVAL, -1;
28
29                 return stralloc_ready(sa, sa->len + len);
30         }
31
32         return stralloc_ready(sa, len);
33 }
```

6.15.2.5 void stralloc_free (stralloc_t * *sa*)

deallocate all memory

Parameters:

→ ***sa*** string to initialize

Definition at line 23 of file stralloc_free.c.

References stralloc_t::s.

Referenced by argv_to_str(), flist32_to_str(), flist64_to_str(), and whirlpool_digest().

```
24 {
25         if (sa->s)
26                 free(sa->s);
27
28         sa->s = 0;
29 }
```

6.15.2.6 int stralloc_copyb (stralloc_t * *dst*, const char * *src*, size_t *len*)

copy a static string to a dynamic one

Parameters:

→ ***dst*** dynamic destination string

← ***src*** static source string

← ***len*** copy at most len bytes

Returns:

0 on success, -1 on error with errno set

Definition at line 22 of file stralloc_copyb.c.

References stralloc_t::len, stralloc_t::s, str_cpyn(), and stralloc_ready().

Referenced by stralloc_copy(), and stralloc_copys().

```
23 {
24         if (stralloc_ready(dst, len) == -1)
25                 return -1;
26
27         str_cpyn(dst->s, src, len);
28         dst->len = len;
29         return 0;
30 }
```

6.15.2.7 int stralloc_copys (stralloc_t * *dst*, const char * *src*)

copy a static string to a dynamic one

Parameters:

→ ***dst*** dynamic destination string

← ***src*** static source string

Returns:

0 on success, -1 on error with errno set

Definition at line 22 of file stralloc_copys.c.

References str_len(), and stralloc_copyb().

```
23 {
24         return stralloc_copyb(dst, src, str_len(src));
25 }
```

6.15.2.8 int stralloc_copy (stralloc_t * *dst*, const stralloc_t * *src*)

copy one dynamic string to another

Parameters:

→ ***dst*** dynamic destination string

← ***src*** dynamic source string

Returns:

0 on success, -1 on error with errno set

Definition at line 21 of file stralloc_copy.c.

References stralloc_t::len, stralloc_t::s, and stralloc_copyb().

```
22 {
23         return stralloc_copyb(dst, src->s, src->len);
24 }
```

6.15.2.9 int stralloc_catb (stralloc_t * *dst*, const char * *src*, size_t *len*)

concatenate a dynamic string and a static one

Parameters:

→ ***dst*** dynamic destination string

← ***src*** static source string

← ***len*** append at most len bytes

Returns:

0 on success, -1 on error with errno set

Definition at line 22 of file stralloc_catb.c.

References stralloc_t::len, stralloc_t::s, str_cpyn(), and stralloc_readyplus().

Referenced by stralloc_cat(), and stralloc_cats().

```
23 {
24         if (stralloc_readyplus(dst, len) == -1)
25                 return -1;
26
27         str_cpyn(dst->s + dst->len, src, len);
28         dst->len += len;
29         return 0;
30 }
```

6.15.2.10 int stralloc_catf (stralloc_t * *dst*, const char * *fmt*, ...)

concatenate a dynamic string and a static one using formatted conversion

Parameters:

→ ***dst*** dynamic destination string

← ***fmt*** format string

← ***...*** variable number of arguments

Returns:

0 on success, -1 on error with errno set

Definition at line 25 of file stralloc_catf.c.

References _lucid_vasprintf(), and stralloc_cats().

Referenced by flist32_to_str(), flist64_to_str(), and whirlpool_digest().

```
26 {
27         va_list ap;
28         char *buf;
29         int rc;
30
31         va_start(ap, fmt);
32         _lucid_vasprintf(&buf, fmt, ap);
33         va_end(ap);
```

```
34
35         rc = stralloc_cats(dst, buf);
36
37         free(buf);
38
39         return rc;
40 }
```

6.15.2.11 int stralloc_catm (stralloc_t * *dst*, ...)

concatenate a dynamic string and multiple static ones

Parameters:

→ ***dst*** dynamic destination string

← ***...*** variable number of source strings

Returns:

0 on success, -1 on error with errno set

Note:

the last argument must be NULL

Definition at line 23 of file stralloc_catm.c.

References stralloc_cats().

Referenced by argv_to_str().

```
24 {
25         va_list ap;
26         char *s;
27
28         va_start(ap, dst);
29
30         while ((s = va_arg(ap, char *))) {
31                 if (stralloc_cats(dst, s) == -1) {
32                         va_end(ap);
33                         return -1;
34                 }
35         }
36
37         va_end(ap);
38         return 0;
39 }
```

6.15.2.12 int stralloc_cats (stralloc_t * *dst*, const char * *src*)

concatenate a dynamic string and a static one

Parameters:

→ ***dst*** dynamic destination string

← ***src*** static source string

Returns:

0 on success, -1 on error with errno set

Definition at line 22 of file stralloc_cats.c.

References str_len(), and stralloc_catb().

Referenced by stralloc_catf(), and stralloc_catm().

```
23 {
24         return stralloc_catb(dst, src, str_len(src));
25 }
```

6.15.2.13 int stralloc_cat (stralloc_t * *dst*, const stralloc_t * *src*)

concatenate two dynamic strings

Parameters:

→ ***dst*** dynamic destination string

← ***src*** dynamic source string

Returns:

0 on success, -1 on error with errno set

Definition at line 21 of file stralloc_cat.c.

References stralloc_t::len, stralloc_t::s, and stralloc_catb().

```
22 {
23         return stralloc_catb(dst, src->s, src->len);
24 }
```

6.15.2.14 int stralloc_cmp (const stralloc_t * *a*, const stralloc_t * *b*)

compare two dynamic strings

Parameters:

← ***a*** first string

← ***b*** second string

Returns:

An integer greater than, equal to, or less than 0, if the string pointed to by a is greater than, equal to, or less than the string pointed to by b, respectively.

Definition at line 21 of file stralloc_cmp.c.

References stralloc_t::len, and stralloc_t::s.

```
22 {
23         size_t i, j;
24
25         for (i = 0;; ++i) {
26                 if (i == a->len)
27                         return i == b->len ? 0 : -1;
28
29                 if (i == b->len)
30                         return 1;
31
32                 if ((j = ((unsigned char)(a->s[i]) - (unsigned char)(b->s[i]))))
33                         return j;
34         }
35
36         return j;
37 }
```

6.16 TCP socket wrappers

6.16.1 Detailed Description

The tcp family of functions provide wrappers around connect(2) and listen(2) taking an IP address in the string pointed to by ip and the port number as arguments.

Functions

- int **tcp_listen** (const char *ip, int port, int backlog)
  listen for incomming connections
- int **tcp_connect** (const char *ip, int port)
  connect to TCP socket

6.16.2 Function Documentation

6.16.2.1 int tcp_listen (const char * *ip*, int *port*, int *backlog*)

listen for incomming connections

Parameters:

ip IP to listen on

port port to listen on

backlog queue backlog

Returns:

filedescriptor for the newly allocated socket, -1 on error with errno set

Definition at line 27 of file tcp_listen.c.

References addr_from_str(), and str_zero().

```
28 {
29         int fd;
30         struct sockaddr_in inaddr;
31 
32         if (port < 1)
33                 return errno = EINVAL, -1;
34 
35         str_zero(&inaddr, sizeof(inaddr));
36         inaddr.sin_family = AF_INET;
37         inaddr.sin_port   = htons(port);
38 
39         if (addr_from_str(ip, &inaddr.sin_addr.s_addr, 0) == 0)
40                 return errno = EINVAL, -1;
41 
42         if ((fd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
43                 return -1;
44 
45         if (bind(fd, (struct sockaddr *) &inaddr, sizeof(struct sockaddr_in)) == -1) {
46                 close(fd);
47                 return -1;
```

```
48         }
49
50         if (listen(fd, backlog) == -1) {
51                 close(fd);
52                 return -1;
53         }
54
55         return fd;
56 }
```

6.16.2.2 int tcp_connect (const char * *ip*, int *port*)

connect to TCP socket

Parameters:

ip IP to connect to

port port to connect to

Returns:

filedescriptor for the newly allocated connection, -1 on error with errno set

Definition at line 27 of file tcp_connect.c.

References addr_from_str(), and str_zero().

```
28 {
29         int fd;
30         struct sockaddr_in inaddr;
31
32         if (port < 1)
33                 return errno = EINVAL, -1;
34
35         str_zero(&inaddr, sizeof(inaddr));
36         inaddr.sin_family = AF_INET;
37         inaddr.sin_port   = htons(port);
38
39         if (addr_from_str(ip, &inaddr.sin_addr.s_addr, 0) == 0)
40                 return errno = EINVAL, -1;
41
42         if ((fd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
43                 return -1;
44
45         if (connect(fd, (struct sockaddr *) &inaddr, sizeof(struct sockaddr_in)) == -1) {
46                 close(fd);
47                 return -1;
48         }
49
50         return fd;
51 }
```

6.17 Whirlpool hash function

6.17.1 Detailed Description

WHIRLPOOL is a cryptographic hash function designed after the Square block cipher. WHIRLPOOL is a Miyaguchi-Preneel construction based on a substantially modified Advanced Encryption Standard (AES). Given a message less than 2^256 bits in length, it returns a 512-bit message digest.

The **whirlpool_init()** (p. 129) function initializes the hash context pointed to by context. After initialization input can be added to the transform routine using **whirlpool_add()** (p. 131). Once all bytes have been added the transform has to be finished by calling whilrpool_finalize().

An application should not directly use the internal **whirlpool_transform()** (p. 126) function, but always use **whirlpool_add()** (p. 131).

The **whirlpool_digest()** (p. 133) function combines the procedure explained above for a single string and returns the digest in hexadecimal notation.

Data Structures

- struct **whirlpool_t**
  dynamic whirlpool state data

Defines

- #define **DIGESTBYTES** 64
  number of bytes in the digest
- #define **DIGESTBITS** (8*DIGESTBYTES)
  number of bits in the digest
- #define **WBLOCKBYTES** 64
  number of bytes in the input buffer
- #define **WBLOCKBITS** (8*WBLOCKBYTES)
  number of bits in the input buffer
- #define **LENGTHBYTES** 32
  number of hashed bytes
- #define **LENGTHBITS** (8*LENGTHBYTES)
  number of hashed bits

Functions

- void **whirlpool_transform** (**whirlpool_t** *const context)
  internal transform routine

- void **whirlpool_init** (**whirlpool_t** *const context)

  initialize whirlpool state context

- void **whirlpool_finalize** (**whirlpool_t** *const context, unsigned char *const result)

  finalize whirlpool transformation

- void **whirlpool_add** (**whirlpool_t** *const context, const unsigned char *const src, unsigned long bits)

  add bytes to the transform routine

- char * **whirlpool_digest** (const char *str)

  create digest from string

6.17.2 Define Documentation

6.17.2.1 #define DIGESTBYTES 64

number of bytes in the digest

Definition at line 46 of file whirlpool.h.

Referenced by whirlpool_digest(), and whirlpool_finalize().

6.17.2.2 #define DIGESTBITS (8*DIGESTBYTES)

number of bits in the digest

Definition at line 49 of file whirlpool.h.

Referenced by whirlpool_add().

6.17.2.3 #define WBLOCKBYTES 64

number of bytes in the input buffer

Definition at line 53 of file whirlpool.h.

Referenced by whirlpool_finalize().

6.17.2.4 #define WBLOCKBITS (8*WBLOCKBYTES)

number of bits in the input buffer

Definition at line 56 of file whirlpool.h.

6.17.2.5 #define LENGTHBYTES 32

number of hashed bytes

Definition at line 60 of file whirlpool.h.

Referenced by whirlpool_finalize(), and whirlpool_init().

6.17.2.6 #define LENGTHBITS (8*LENGTHBYTES)

number of hashed bits

Definition at line 63 of file whirlpool.h.

6.17.3 Function Documentation

6.17.3.1 void whirlpool_transform (whirlpool_t *const *context*)

internal transform routine

Parameters:

← ***context*** whirlpool state context

Definition at line 27 of file whirlpool_transform.c.

References whirlpool_t::buf, whirlpool_t::hash, and R.

Referenced by whirlpool_add(), and whirlpool_finalize().

```
28 {
29         int i, r;
30         uint64_t K[8];
31         uint64_t block[8];
32         uint64_t state[8];
33         uint64_t L[8];
34         uint8_t *buf = context->buf;
35 
36         /* map the buffer to a block */
37         for (i = 0; i < 8; i++, buf += 8) {
38                 block[i] = (((uint64_t)buf[0]        ) << 56) ^
39                            (((uint64_t)buf[1] & 0xffL) << 48) ^
40                            (((uint64_t)buf[2] & 0xffL) << 40) ^
41                            (((uint64_t)buf[3] & 0xffL) << 32) ^
42                            (((uint64_t)buf[4] & 0xffL) << 24) ^
43                            (((uint64_t)buf[5] & 0xffL) << 16) ^
44                            (((uint64_t)buf[6] & 0xffL) <<  8) ^
45                            (((uint64_t)buf[7] & 0xffL)      );
46         }
47 
48         /* compute and apply K^0 to the cipher state */
49         state[0] = block[0] ^ (K[0] = context->hash[0]);
50         state[1] = block[1] ^ (K[1] = context->hash[1]);
51         state[2] = block[2] ^ (K[2] = context->hash[2]);
52         state[3] = block[3] ^ (K[3] = context->hash[3]);
53         state[4] = block[4] ^ (K[4] = context->hash[4]);
54         state[5] = block[5] ^ (K[5] = context->hash[5]);
55         state[6] = block[6] ^ (K[6] = context->hash[6]);
56         state[7] = block[7] ^ (K[7] = context->hash[7]);
57 
58         /* iterate over all rounds */
59         for (r = 1; r <= R; r++) {
60                 /* compute K^r from K^{r-1} */
61                 L[0] = C0[(int)(K[0] >> 56)       ] ^
62                        C1[(int)(K[7] >> 48) & 0xff] ^
63                        C2[(int)(K[6] >> 40) & 0xff] ^
64                        C3[(int)(K[5] >> 32) & 0xff] ^
65                        C4[(int)(K[4] >> 24) & 0xff] ^
66                        C5[(int)(K[3] >> 16) & 0xff] ^
67                        C6[(int)(K[2] >>  8) & 0xff] ^
68                        C7[(int)(K[1]      ) & 0xff] ^
```

```
69                      rc[r];
70
71              L[1] = C0[(int)(K[1] >> 56)       ] ^
72                     C1[(int)(K[0] >> 48) & 0xff] ^
73                     C2[(int)(K[7] >> 40) & 0xff] ^
74                     C3[(int)(K[6] >> 32) & 0xff] ^
75                     C4[(int)(K[5] >> 24) & 0xff] ^
76                     C5[(int)(K[4] >> 16) & 0xff] ^
77                     C6[(int)(K[3] >>  8) & 0xff] ^
78                     C7[(int)(K[2]      ) & 0xff];
79
80              L[2] = C0[(int)(K[2] >> 56)       ] ^
81                     C1[(int)(K[1] >> 48) & 0xff] ^
82                     C2[(int)(K[0] >> 40) & 0xff] ^
83                     C3[(int)(K[7] >> 32) & 0xff] ^
84                     C4[(int)(K[6] >> 24) & 0xff] ^
85                     C5[(int)(K[5] >> 16) & 0xff] ^
86                     C6[(int)(K[4] >>  8) & 0xff] ^
87                     C7[(int)(K[3]      ) & 0xff];
88
89              L[3] = C0[(int)(K[3] >> 56)       ] ^
90                     C1[(int)(K[2] >> 48) & 0xff] ^
91                     C2[(int)(K[1] >> 40) & 0xff] ^
92                     C3[(int)(K[0] >> 32) & 0xff] ^
93                     C4[(int)(K[7] >> 24) & 0xff] ^
94                     C5[(int)(K[6] >> 16) & 0xff] ^
95                     C6[(int)(K[5] >>  8) & 0xff] ^
96                     C7[(int)(K[4]      ) & 0xff];
97
98              L[4] = C0[(int)(K[4] >> 56)       ] ^
99                     C1[(int)(K[3] >> 48) & 0xff] ^
100                     C2[(int)(K[2] >> 40) & 0xff] ^
101                     C3[(int)(K[1] >> 32) & 0xff] ^
102                     C4[(int)(K[0] >> 24) & 0xff] ^
103                     C5[(int)(K[7] >> 16) & 0xff] ^
104                     C6[(int)(K[6] >>  8) & 0xff] ^
105                     C7[(int)(K[5]      ) & 0xff];
106
107              L[5] = C0[(int)(K[5] >> 56)       ] ^
108                     C1[(int)(K[4] >> 48) & 0xff] ^
109                     C2[(int)(K[3] >> 40) & 0xff] ^
110                     C3[(int)(K[2] >> 32) & 0xff] ^
111                     C4[(int)(K[1] >> 24) & 0xff] ^
112                     C5[(int)(K[0] >> 16) & 0xff] ^
113                     C6[(int)(K[7] >>  8) & 0xff] ^
114                     C7[(int)(K[6]      ) & 0xff];
115
116              L[6] = C0[(int)(K[6] >> 56)       ] ^
117                     C1[(int)(K[5] >> 48) & 0xff] ^
118                     C2[(int)(K[4] >> 40) & 0xff] ^
119                     C3[(int)(K[3] >> 32) & 0xff] ^
120                     C4[(int)(K[2] >> 24) & 0xff] ^
121                     C5[(int)(K[1] >> 16) & 0xff] ^
122                     C6[(int)(K[0] >>  8) & 0xff] ^
123                     C7[(int)(K[7]      ) & 0xff];
124
125              L[7] = C0[(int)(K[7] >> 56)       ] ^
126                     C1[(int)(K[6] >> 48) & 0xff] ^
127                     C2[(int)(K[5] >> 40) & 0xff] ^
128                     C3[(int)(K[4] >> 32) & 0xff] ^
129                     C4[(int)(K[3] >> 24) & 0xff] ^
130                     C5[(int)(K[2] >> 16) & 0xff] ^
131                     C6[(int)(K[1] >>  8) & 0xff] ^
132                     C7[(int)(K[0]      ) & 0xff];
133
134              K[0] = L[0];
135              K[1] = L[1];
```

```
136                 K[2] = L[2];
137                 K[3] = L[3];
138                 K[4] = L[4];
139                 K[5] = L[5];
140                 K[6] = L[6];
141                 K[7] = L[7];
142
143                 /* apply the r-th round transformation */
144                 L[0] = C0[(int)(state[0] >> 56)       ] ^
145                        C1[(int)(state[7] >> 48) & 0xff] ^
146                        C2[(int)(state[6] >> 40) & 0xff] ^
147                        C3[(int)(state[5] >> 32) & 0xff] ^
148                        C4[(int)(state[4] >> 24) & 0xff] ^
149                        C5[(int)(state[3] >> 16) & 0xff] ^
150                        C6[(int)(state[2] >>  8) & 0xff] ^
151                        C7[(int)(state[1]      ) & 0xff] ^
152                        K[0];
153
154                 L[1] = C0[(int)(state[1] >> 56)       ] ^
155                        C1[(int)(state[0] >> 48) & 0xff] ^
156                        C2[(int)(state[7] >> 40) & 0xff] ^
157                        C3[(int)(state[6] >> 32) & 0xff] ^
158                        C4[(int)(state[5] >> 24) & 0xff] ^
159                        C5[(int)(state[4] >> 16) & 0xff] ^
160                        C6[(int)(state[3] >>  8) & 0xff] ^
161                        C7[(int)(state[2]      ) & 0xff] ^
162                        K[1];
163
164                 L[2] = C0[(int)(state[2] >> 56)       ] ^
165                        C1[(int)(state[1] >> 48) & 0xff] ^
166                        C2[(int)(state[0] >> 40) & 0xff] ^
167                        C3[(int)(state[7] >> 32) & 0xff] ^
168                        C4[(int)(state[6] >> 24) & 0xff] ^
169                        C5[(int)(state[5] >> 16) & 0xff] ^
170                        C6[(int)(state[4] >>  8) & 0xff] ^
171                        C7[(int)(state[3]      ) & 0xff] ^
172                        K[2];
173
174                 L[3] = C0[(int)(state[3] >> 56)       ] ^
175                        C1[(int)(state[2] >> 48) & 0xff] ^
176                        C2[(int)(state[1] >> 40) & 0xff] ^
177                        C3[(int)(state[0] >> 32) & 0xff] ^
178                        C4[(int)(state[7] >> 24) & 0xff] ^
179                        C5[(int)(state[6] >> 16) & 0xff] ^
180                        C6[(int)(state[5] >>  8) & 0xff] ^
181                        C7[(int)(state[4]      ) & 0xff] ^
182                        K[3];
183
184                 L[4] = C0[(int)(state[4] >> 56)       ] ^
185                        C1[(int)(state[3] >> 48) & 0xff] ^
186                        C2[(int)(state[2] >> 40) & 0xff] ^
187                        C3[(int)(state[1] >> 32) & 0xff] ^
188                        C4[(int)(state[0] >> 24) & 0xff] ^
189                        C5[(int)(state[7] >> 16) & 0xff] ^
190                        C6[(int)(state[6] >>  8) & 0xff] ^
191                        C7[(int)(state[5]      ) & 0xff] ^
192                        K[4];
193
194                 L[5] = C0[(int)(state[5] >> 56)       ] ^
195                        C1[(int)(state[4] >> 48) & 0xff] ^
196                        C2[(int)(state[3] >> 40) & 0xff] ^
197                        C3[(int)(state[2] >> 32) & 0xff] ^
198                        C4[(int)(state[1] >> 24) & 0xff] ^
199                        C5[(int)(state[0] >> 16) & 0xff] ^
200                        C6[(int)(state[7] >>  8) & 0xff] ^
201                        C7[(int)(state[6]      ) & 0xff] ^
202                        K[5];
```

```
203
204                 L[6] = C0[(int)(state[6] >> 56)       ] ^
205                        C1[(int)(state[5] >> 48) & 0xff] ^
206                        C2[(int)(state[4] >> 40) & 0xff] ^
207                        C3[(int)(state[3] >> 32) & 0xff] ^
208                        C4[(int)(state[2] >> 24) & 0xff] ^
209                        C5[(int)(state[1] >> 16) & 0xff] ^
210                        C6[(int)(state[0] >>  8) & 0xff] ^
211                        C7[(int)(state[7]      ) & 0xff] ^
212                        K[6];
213
214                 L[7] = C0[(int)(state[7] >> 56)       ] ^
215                        C1[(int)(state[6] >> 48) & 0xff] ^
216                        C2[(int)(state[5] >> 40) & 0xff] ^
217                        C3[(int)(state[4] >> 32) & 0xff] ^
218                        C4[(int)(state[3] >> 24) & 0xff] ^
219                        C5[(int)(state[2] >> 16) & 0xff] ^
220                        C6[(int)(state[1] >>  8) & 0xff] ^
221                        C7[(int)(state[0]      ) & 0xff] ^
222                        K[7];
223
224                 state[0] = L[0];
225                 state[1] = L[1];
226                 state[2] = L[2];
227                 state[3] = L[3];
228                 state[4] = L[4];
229                 state[5] = L[5];
230                 state[6] = L[6];
231                 state[7] = L[7];
232         }
233
234         /* apply the Miyaguchi-Preneel compression function */
235         context->hash[0] ^= state[0] ^ block[0];
236         context->hash[1] ^= state[1] ^ block[1];
237         context->hash[2] ^= state[2] ^ block[2];
238         context->hash[3] ^= state[3] ^ block[3];
239         context->hash[4] ^= state[4] ^ block[4];
240         context->hash[5] ^= state[5] ^ block[5];
241         context->hash[6] ^= state[6] ^ block[6];
242         context->hash[7] ^= state[7] ^ block[7];
243 }
```

6.17.3.2 void whirlpool_init (whirlpool_t *const *context*)

initialize whirlpool state context

Parameters:

← ***context*** whirlpool state context

Definition at line 25 of file whirlpool_init.c.

References whirlpool_t::bits, whirlpool_t::buf, whirlpool_t::hash, whirlpool_t::len, LENGTHBYTES, whirlpool_t::pos, and str_zero().

Referenced by whirlpool_digest().

```
26 {
27         int i;
28
29         str_zero(context->len, LENGTHBYTES);
30
31         context->bits = context->pos = 0;
```

```
32         context->buf[0]  = 0;
33
34         for (i = 0; i < 8; i++)
35                 context->hash[i] = 0L;
36 }
```

6.17.3.3 void whirlpool_finalize (whirlpool_t *const *context*, unsigned char *const *result*)

finalize whirlpool transformation

Parameters:

← ***context*** whirlpool state context

→ ***result*** string to store digest

Definition at line 25 of file whirlpool_finalize.c.

References whirlpool_t::bits, whirlpool_t::buf, DIGESTBYTES, whirlpool_t::hash, whirlpool_t::len, LENGTHBYTES, whirlpool_t::pos, str_cpyn(), str_zero(), WBLOCKBYTES, and whirlpool_transform().

Referenced by whirlpool_digest().

```
26 {
27         int i;
28         uint8_t *buf    = context->buf;
29         uint8_t *len    = context->len;
30         int bits        = context->bits;
31         int pos         = context->pos;
32         uint8_t *digest = result;
33
34         /* append a '1'-bit */
35         buf[pos] |= 0x80U >> (bits & 7);
36         pos++; /* all remaining bits on the current uint8_t are set to zero. */
37
38         /* pad with zero bits to complete (N*WBLOCKBITS - LENGTHBITS) bits */
39         if (pos > WBLOCKBYTES - LENGTHBYTES) {
40                 if (pos < WBLOCKBYTES)
41                         str_zero(&buf[pos], WBLOCKBYTES - pos);
42
43                 /* process data block */
44                 whirlpool_transform(context);
45
46                 /* reset buffer */
47                 pos = 0;
48         }
49
50         if (pos < WBLOCKBYTES - LENGTHBYTES)
51                 str_zero(&buf[pos], (WBLOCKBYTES - LENGTHBYTES) - pos);
52
53         pos = WBLOCKBYTES - LENGTHBYTES;
54
55         /* append bit length of hashed data */
56         str_cpyn(&buf[WBLOCKBYTES - LENGTHBYTES], len, LENGTHBYTES);
57
58         /* process data block */
59         whirlpool_transform(context);
60
61         /* return the completed message digest */
62         for (i = 0; i < DIGESTBYTES/8; i++) {
63                 digest[0] = (uint8_t)(context->hash[i] >> 56);
```

```
64                 digest[1] = (uint8_t)(context->hash[i] >> 48);
65                 digest[2] = (uint8_t)(context->hash[i] >> 40);
66                 digest[3] = (uint8_t)(context->hash[i] >> 32);
67                 digest[4] = (uint8_t)(context->hash[i] >> 24);
68                 digest[5] = (uint8_t)(context->hash[i] >> 16);
69                 digest[6] = (uint8_t)(context->hash[i] >>  8);
70                 digest[7] = (uint8_t)(context->hash[i]      );
71                 digest += 8;
72         }
73
74         context->bits = bits;
75         context->pos  = pos;
76 }
```

6.17.3.4 void whirlpool_add (whirlpool_t *const *context*, const unsigned char *const *src*, unsigned long *bits*)

add bytes to the transform routine

Parameters:

- ← ***context*** whirlpool state context
- ← ***src*** source string
- ← ***bits*** number of bits in the source string

Definition at line 24 of file whirlpool_add.c.

References whirlpool_t::bits, whirlpool_t::buf, DIGESTBITS, whirlpool_t::len, whirlpool_t::pos, and whirlpool_transform().

Referenced by whirlpool_digest().

```
26 {
27         int i;
28         uint32_t b, carry;
29         int srcpos = 0; /* index of leftmost source uint8_t containing data (1 to 8 bits). */
30
31         int gap      = (8 - ((int)srcbits & 7)) & 7; /* space on src[srcpos]. */
32         int rem      = context->bits & 7; /* occupied bits on buf[pos]. */
33
34         uint8_t *buf = context->buf;
35         uint8_t *len = context->len;
36         int bits     = context->bits;
37         int pos      = context->pos;
38
39         /* tally the length of the added data */
40         uint64_t value = srcbits;
41
42         for (i = 31, carry = 0; i >= 0 && (carry != 0 || value != 0ULL); i--) {
43                 carry += len[i] + ((uint32_t)value & 0xff);
44                 len[i] = (uint8_t)carry;
45                 carry >>= 8;
46                 value >>= 8;
47         }
48
49         /* process data in chunks of 8 bits */
50         while (srcbits > 8) {
51                 /* take a byte from the source */
52                 b = ((src[srcpos] << gap) & 0xff) |
53                                 ((src[srcpos + 1] & 0xff) >> (8 - gap));
54
```

```
55                /* process this byte */
56                buf[pos++] |= (uint8_t)(b >> rem);
57                bits += 8 - rem; /* bits = 8*pos; */
58
59                if (bits == DIGESTBITS) {
60                        /* process data block */
61                        whirlpool_transform(context);
62
63                        /* reset buf */
64                        bits = pos = 0;
65                }
66
67                buf[pos] = b << (8 - rem);
68                bits += rem;
69
70                /* proceed to remaining data */
71                srcbits -= 8;
72                srcpos++;
73        }
74
75        /* now 0 <= srcbits <= 8;
76         * furthermore, all data (if any is left) is in src[srcpos].
77         */
78        if (srcbits > 0) {
79                b = (src[srcpos] << gap) & 0xff; /* bits are left-justified on b. */
80
81                /* process the remaining bits */
82                buf[pos] |= b >> rem;
83        }
84
85        else
86                b = 0;
87
88        /* all remaining data fits on buf[pos],
89         * and there still remains some space.
90         */
91        if (rem + srcbits < 8)
92                bits += srcbits;
93
94        else {
95                /* buf[pos] is full */
96                pos++;
97                bits += 8 - rem; /* bits = 8*pos; */
98                srcbits -= 8 - rem;
99
100                /* now 0 <= srcbits < 8;
101                 * furthermore, all data (if any is left) is in src[srcpos].
102                 */
103                if (bits == DIGESTBITS) {
104                        /* process data block */
105                        whirlpool_transform(context);
106
107                        /* reset buf */
108                        bits = pos = 0;
109                }
110
111                buf[pos] = b << (8 - rem);
112                bits += (int)srcbits;
113        }
114
115        context->bits = bits;
116        context->pos  = pos;
117 }
```

6.17.3.5 char* whirlpool_digest (const char * *str*)

create digest from string

Parameters:

← ***str*** source string

Returns:

digest string (memory obtained by malloc(3))

Note:

The caller should free obtained memory using free(3)

See also:

malloc(3)
free(3)

Definition at line 24 of file whirlpool_digest.c.

References DIGESTBYTES, stralloc_t::len, stralloc_t::s, str_dupn(), str_len(), stralloc_catf(), stralloc_free(), stralloc_init(), whirlpool_add(), whirlpool_finalize(), and whirlpool_init().

```
25 {
26         whirlpool_t ctx;
27         stralloc_t sa;
28         char *buf;
29         uint8_t digest[DIGESTBYTES];
30         int i;
31
32         whirlpool_init(&ctx);
33         whirlpool_add(&ctx, (const unsigned char * const) str, str_len(str)*8);
34         whirlpool_finalize(&ctx, digest);
35
36         stralloc_init(&sa);
37
38         for (i = 0; i < DIGESTBYTES; i++)
39                 stralloc_catf(&sa, "%02X", digest[i]);
40
41         buf = str_dupn(sa.s, sa.len);
42
43         stralloc_free(&sa);
44
45         return buf;
46 }
```

Chapter 7

lucid Data Structure Documentation

7.1 __printf_t Struct Reference

7.1.1 Detailed Description

Definition at line 51 of file vsnprintf.c.

Data Fields

- unsigned int **f**
- int **l**
- int **p**
- int **s**
- unsigned int **w**

7.1.2 Field Documentation

7.1.2.1 unsigned int __printf_t::f

Definition at line 52 of file vsnprintf.c.

Referenced by _lucid_vsnprintf().

7.1.2.2 int __printf_t::l

Definition at line 53 of file vsnprintf.c.

Referenced by _lucid_vsnprintf().

7.1.2.3 int __printf_t::p

Definition at line 54 of file vsnprintf.c.

Referenced by _lucid_vsnprintf().

7.1.2.4 int __printf_t::s

Definition at line 55 of file vsnprintf.c.

Referenced by _lucid_vsnprintf().

7.1.2.5 unsigned int __printf_t::w

Definition at line 56 of file vsnprintf.c.

Referenced by _lucid_vsnprintf().

The documentation for this struct was generated from the following file:

- printf/**vsnprintf.c**

7.2 __scanf_t Struct Reference

7.2.1 Detailed Description

Definition at line 48 of file vsscanf.c.

Data Fields

- int **f**
- int **l**
- int **s**
- int **w**

7.2.2 Field Documentation

7.2.2.1 int __scanf_t::f

Definition at line 49 of file vsscanf.c.

Referenced by _lucid_vsscanf().

7.2.2.2 int __scanf_t::l

Definition at line 50 of file vsscanf.c.

Referenced by _lucid_vsscanf().

7.2.2.3 int __scanf_t::s

Definition at line 51 of file vsscanf.c.

Referenced by _lucid_vsscanf().

7.2.2.4 int __scanf_t::w

Definition at line 52 of file vsscanf.c.

Referenced by _lucid_vsscanf().

The documentation for this struct was generated from the following file:

- scanf/**vsscanf.c**

7.3 flist32_t Struct Reference

`#include <flist.h>`

7.3.1 Detailed Description

32 bit list object

Definition at line 56 of file flist.h.

Data Fields

- const char * **key**
- const uint32_t **val**

7.3.2 Field Documentation

7.3.2.1 const char* flist32_t::key

Node key (must be unique)

Definition at line 57 of file flist.h.

Referenced by flist32_getkey(), flist32_getval(), and flist32_to_str().

7.3.2.2 const uint32_t flist32_t::val

Node value (32-bit)

Definition at line 58 of file flist.h.

The documentation for this struct was generated from the following file:

- **flist.h**

7.4 flist64_t Struct Reference

`#include <flist.h>`

7.4.1 Detailed Description

64 bit list object

Definition at line 134 of file flist.h.

Data Fields

- const char * **key**
- const uint64_t **val**

7.4.2 Field Documentation

7.4.2.1 const char* flist64_t::key

Node key (must be unique)

Definition at line 135 of file flist.h.

Referenced by flist64_getkey(), flist64_getval(), and flist64_to_str().

7.4.2.2 const uint64_t flist64_t::val

Node value (64-bit)

Definition at line 136 of file flist.h.

The documentation for this struct was generated from the following file:

- **flist.h**

7.5 list_head Struct Reference

`#include <list.h>`

Collaboration diagram for list_head:

7.5.1 Detailed Description

list head

Definition at line 57 of file list.h.

Data Fields

- **list_head * next**
- **list_head * prev**

7.5.2 Field Documentation

7.5.2.1 struct list_head* list_head::next

Definition at line 58 of file list.h.

7.5.2.2 struct list_head * list_head::prev

Definition at line 58 of file list.h.

The documentation for this struct was generated from the following file:

- **list.h**

7.6 log_options_t Struct Reference

`#include <log.h>`

7.6.1 Detailed Description

multiplexer configuration data

- The string pointed to by ident is prepended to every message, and is typically set to the program name.
- The file argument enables (1) or disables (0) multiplexing to the file destination.
- The stderr argument enables (1) or disables (0) multiplexing to the stderr destination.
- The time argument enables (1) or disables (0) output of the time string in multiplexed messages; only used for the file and stderr destinations.
- The syslog argument enables (1) or disables (0) multiplexing to the syslog(3) destination.
- The flags argument specifies flags which control the operation of the multiplexer.
- The facility argument is used to specify what type of program is logging the message; only used for the syslog(3) destination.
- The mask argument is the lower level bound of messages being multiplexed.

Definition at line 63 of file log.h.

Data Fields

- char * **ident**
- bool **file**
- int **fd**
- bool **stderr**
- bool **time**
- bool **syslog**
- int **flags**
- int **facility**
- int **mask**

7.6.2 Field Documentation

7.6.2.1 char* log_options_t::ident

program identifier

Definition at line 64 of file log.h.

Referenced by log_init().

7.6.2.2 bool log_options_t::file

file destination switch

Definition at line 65 of file log.h.

Referenced by log_close(), log_init(), and log_internal().

7.6.2.3 int log_options_t::fd

file descriptor

Definition at line 66 of file log.h.

Referenced by log_close(), log_init(), and log_internal().

7.6.2.4 bool log_options_t::stderr

stderr destination switch

Definition at line 67 of file log.h.

Referenced by log_init(), and log_internal().

7.6.2.5 bool log_options_t::time

time output switch

Definition at line 68 of file log.h.

7.6.2.6 bool log_options_t::syslog

syslog destination switch

Definition at line 69 of file log.h.

Referenced by log_close(), log_init(), and log_internal().

7.6.2.7 int log_options_t::flags

control flags

Definition at line 70 of file log.h.

Referenced by log_init().

7.6.2.8 int log_options_t::facility

program facility

Definition at line 71 of file log.h.

Referenced by log_init(), and log_internal().

7.6.2.9 int log_options_t::mask

lower level bound

Definition at line 72 of file log.h.

Referenced by log_init().

The documentation for this struct was generated from the following file:

- **log.h**

7.7 stralloc_t Struct Reference

`#include <stralloc.h>`

7.7.1 Detailed Description

dynamic string allocator tracking data

This struct is used to keep track of the dynamic string state, i.e. its contents, its length and its additionaly allocated memory.

Definition at line 69 of file stralloc.h.

Data Fields

- char * **s**
- size_t **len**
- size_t **a**

7.7.2 Field Documentation

7.7.2.1 char* stralloc_t::s

pointer to dynamic string

Definition at line 70 of file stralloc.h.

Referenced by argv_to_str(), stralloc_cat(), stralloc_catb(), stralloc_cmp(), stralloc_copy(), stralloc_copyb(), stralloc_free(), stralloc_init(), stralloc_ready(), stralloc_readyplus(), and whirlpool_digest().

7.7.2.2 size_t stralloc_t::len

current length of s

Definition at line 71 of file stralloc.h.

Referenced by argv_to_str(), stralloc_cat(), stralloc_catb(), stralloc_cmp(), stralloc_copy(), stralloc_copyb(), stralloc_init(), stralloc_readyplus(), stralloc_zero(), and whirlpool_digest().

7.7.2.3 size_t stralloc_t::a

additional free memory in s

Definition at line 72 of file stralloc.h.

Referenced by stralloc_init(), and stralloc_ready().

The documentation for this struct was generated from the following file:

- **stralloc.h**

7.8 whirlpool_t Struct Reference

`#include <whirlpool.h>`

7.8.1 Detailed Description

dynamic whirlpool state data

This struct is used to keep track of the whirlpool transform, i.e. its hashing state, input buffer, number of hashed bits, etc.

Definition at line 71 of file whirlpool.h.

Data Fields

- uint8_t **len** [LENGTHBYTES]
- uint8_t **buf** [WBLOCKBYTES]
- int **bits**
- int **pos**
- uint64_t **hash** [DIGESTBYTES/8]

7.8.2 Field Documentation

7.8.2.1 uint8_t whirlpool_t::len[LENGTHBYTES]

global number of hashed bits

Definition at line 72 of file whirlpool.h.

Referenced by whirlpool_add(), whirlpool_finalize(), and whirlpool_init().

7.8.2.2 uint8_t whirlpool_t::buf[WBLOCKBYTES]

buffer of data to hash

Definition at line 73 of file whirlpool.h.

Referenced by whirlpool_add(), whirlpool_finalize(), whirlpool_init(), and whirlpool_transform().

7.8.2.3 int whirlpool_t::bits

current number of bits on the buffer

Definition at line 74 of file whirlpool.h.

Referenced by whirlpool_add(), whirlpool_finalize(), and whirlpool_init().

7.8.2.4 int whirlpool_t::pos

current (possibly incomplete) byte slot on the buffer

Definition at line 75 of file whirlpool.h.

Referenced by whirlpool_add(), whirlpool_finalize(), and whirlpool_init().

7.8.2.5 uint64_t whirlpool_t::hash[DIGESTBYTES/8]

the hashing state

Definition at line 76 of file whirlpool.h.

Referenced by whirlpool_finalize(), whirlpool_init(), and whirlpool_transform().

The documentation for this struct was generated from the following file:

- **whirlpool.h**

Chapter 8

lucid File Documentation

8.1 addr.h File Reference

`#include <stdint.h>`

Include dependency graph for addr.h:

This graph shows which files directly or indirectly include this file:

Functions

- uint32_t **addr_hton** (uint32_t addr)
  convert address from host to network byte order
- uint32_t **addr_ntoh** (uint32_t addr)
  convert address from network to host byte order
- int **addr_from_str** (const char *str, uint32_t *ip, uint32_t *mask)
  convert string to IP address and netmask
- char * **addr_to_str** (uint32_t ip, uint32_t mask)
  convert IP address and netmask to string

8.2 addr/addr_from_str.c File Reference

```
#include "addr.h"
#include "scanf.h"
#include "str.h"
```

Include dependency graph for addr_from_str.c:

Functions

- int **addr_from_str** (const char *str, uint32_t *ip, uint32_t *mask)
  convert string to IP address and netmask

8.3 addr/addr_hton.c File Reference

```
#include "addr.h"
```

Include dependency graph for addr_hton.c:

Functions

- uint32_t **addr_hton** (uint32_t addr)
  convert address from host to network byte order

8.4 addr/addr_ntoh.c File Reference

```
#include "addr.h"
```

Include dependency graph for addr_ntoh.c:

Functions

- uint32_t **addr_ntoh** (uint32_t addr)
  convert address from network to host byte order

8.5 addr/addr_to_str.c File Reference

```
#include "addr.h"
#include "printf.h"
```

Include dependency graph for addr_to_str.c:

Functions

- char * **addr_to_str** (uint32_t ip, uint32_t mask)
  convert IP address and netmask to string

8.6 argv.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- int **argv_from_str** (char *str, char **const argv, int argc)
  convert string to argument vector
- char * **argv_to_str** (int argc, const char **const argv)
  convert argument vector to string

8.7 argv/argv_from_str.c File Reference

```
#include <stdlib.h>
#include "str.h"
#include "argv.h"
```

Include dependency graph for argv_from_str.c:

Functions

- int **argv_from_str** (char *str, char **const argv, int max_argc)
  convert string to argument vector

8.8 argv/argv_to_str.c File Reference

```
#include "argv.h"
#include "str.h"
#include "stralloc.h"
```

Include dependency graph for argv_to_str.c:

Functions

- char * **argv_to_str** (int argc, const char **const argv)
  convert argument vector to string

8.9 bitmap.h File Reference

`#include <stdint.h>`

Include dependency graph for bitmap.h:

This graph shows which files directly or indirectly include this file:

Functions

- uint32_t **i2v32** (int index)
  convert bit index to 32 bit value
- uint64_t **i2v64** (int index)
  convert bit index to 64 bit value
- int **v2i32** (uint32_t val)
  convert 32 bit value to bit index
- int **v2i64** (uint64_t val)
  convert 64 bit value to bit index

8.10 bitmap/i2v32.c File Reference

```
#include "bitmap.h"
```

Include dependency graph for i2v32.c:

Functions

- uint32_t **i2v32** (int index)
  convert bit index to 32 bit value

8.11 bitmap/i2v64.c File Reference

```
#include "bitmap.h"
```

Include dependency graph for i2v64.c:

Functions

- uint64_t **i2v64** (int index)
  convert bit index to 64 bit value

8.12 bitmap/v2i32.c File Reference

```
#include "bitmap.h"
```

Include dependency graph for v2i32.c:

Functions

- int **v2i32** (uint32_t val)
  convert 32 bit value to bit index

8.13 bitmap/v2i64.c File Reference

```
#include "bitmap.h"
```

Include dependency graph for v2i64.c:

Functions

- int **v2i64** (uint64_t val)
  convert 64 bit value to bit index

8.14 chroot.h File Reference

`#include <sys/types.h>`

Include dependency graph for chroot.h:

This graph shows which files directly or indirectly include this file:

Functions

- int **chroot_fd** (int fd)
  chroot(2) to the directory pointed to by a filedescriptor
- int **chroot_mkdirp** (const char *root, const char *dir, mode_t mode)
  recursive mkdir(2) inside a secure chroot
- int **chroot_secure_chdir** (const char *root, const char *dir)
  symlink-attack safe chdir(2) in chroot(2)

8.15 chroot/chroot_fd.c File Reference

```
#include <unistd.h>
```

```
#include "chroot.h"
```

Include dependency graph for chroot_fd.c:

Functions

- int **chroot_fd** (int fd)
  chroot(2) to the directory pointed to by a filedescriptor

8.16 chroot/chroot_mkdirp.c File Reference

```
#include <unistd.h>
#include <errno.h>
#include "chroot.h"
#include "misc.h"
#include "open.h"
```

Include dependency graph for chroot_mkdirp.c:

Functions

- int **chroot_mkdirp** (const char *root, const char *dir, mode_t mode)
  recursive mkdir(2) inside a secure chroot

8.17 chroot/chroot_secure_chdir.c File Reference

```
#include <unistd.h>
#include <errno.h>
#include "chroot.h"
#include "open.h"
```

Include dependency graph for chroot_secure_chdir.c:

Functions

- int **chroot_secure_chdir** (const char *root, const char *dir)
  symlink-attack safe chdir(2) in chroot(2)

8.18 doxygen/examples.h File Reference

8.19 doxygen/license.h File Reference

8.20 doxygen/main.h File Reference

8.21 exec.h File Reference

This graph shows which files directly or indirectly include this file:

Defines

- #define **EXEC_MAX_ARGV** 64
  maximum number of arguments that will be converted for execvp(2)

Functions

- int **exec_fork** (const char *fmt,...)
  fork, execvp and wait
- int **exec_fork_background** (const char *fmt,...)
  fork, execvp and ignore child
- int **exec_fork_pipe** (char **out, const char *fmt,...)
  pipe, fork, execvp and wait
- int **exec_replace** (const char *fmt,...)
  plain execvp

8.22 exec/exec_fork.c File Reference

```
#include <unistd.h>
#include <stdlib.h>
#include <stdarg.h>
#include <errno.h>
#include <sys/wait.h>
#include "argv.h"
#include "exec.h"
#include "printf.h"
```

Include dependency graph for exec_fork.c:

Functions

- int **exec_fork** (const char *fmt,...)
  fork, execvp and wait

8.23 exec/exec_fork_background.c File Reference

```
#include <unistd.h>
#include <stdlib.h>
#include <stdarg.h>
#include <errno.h>
#include <signal.h>
#include "argv.h"
#include "exec.h"
#include "printf.h"
```

Include dependency graph for exec_fork_background.c:

Functions

- int **exec_fork_background** (const char *fmt,...)
  fork, execvp and ignore child

8.24 exec/exec_fork_pipe.c File Reference

```
#include <unistd.h>
#include <stdlib.h>
#include <stdarg.h>
#include <errno.h>
#include <sys/wait.h>
#include "argv.h"
#include "exec.h"
#include "io.h"
#include "printf.h"
```

Include dependency graph for exec_fork_pipe.c:

Functions

- int **exec_fork_pipe** (char **out, const char *fmt,...)
  pipe, fork, execvp and wait

8.25 exec/exec_replace.c File Reference

```
#include <unistd.h>
#include <stdlib.h>
#include <stdarg.h>
#include <errno.h>
#include "argv.h"
#include "exec.h"
#include "printf.h"
```

Include dependency graph for exec_replace.c:

Functions

- int **exec_replace** (const char *fmt,...)
  plain execvp

8.26 flist.h File Reference

`#include <stdint.h>`

Include dependency graph for flist.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- struct **flist32_t**
  32 bit list object
- struct **flist64_t**
  64 bit list object

Defines

- #define **FLIST32_START**(LIST) const **flist32_t** LIST[] = {
  32 bit list initialization
- #define **FLIST32_NODE**(PREFIX, NAME) { #NAME, PREFIX ## _ ## NAME },
  32 bit list node
- #define **FLIST32_NODE1**(PREFIX, NAME) { #NAME, (1 << PREFIX ## _ ## NAME) },
  32 bit list node from index
- #define **FLIST32_END** { NULL, 0 } };
  32 bit list termination
- #define **FLIST64_START**(LIST) const **flist64_t** LIST[] = {
  64 bit list initialization
- #define **FLIST64_NODE**(PREFIX, NAME) { #NAME, PREFIX ## _ ## NAME },
  64 bit list node
- #define **FLIST64_NODE1**(PREFIX, NAME) { #NAME, (1 << PREFIX ## _ ## NAME) },
  64 bit list node from index
- #define **FLIST64_END** { NULL, 0 } };
  64 bit list termination

Functions

- uint32_t **flist32_getval** (const **flist32_t** list[], const char *key)
  get 32 bit value by key
- const char * **flist32_getkey** (const **flist32_t** list[], uint32_t val)
  get key from 32 bit value
- int **flist32_from_str** (const char *str, const **flist32_t** list[], uint32_t *flags, uint32_t *mask, char clmod, char delim)
  parse flag list string
- char * **flist32_to_str** (const **flist32_t** list[], uint32_t val, char delim)
  convert bit mask to flag list string
- uint64_t **flist64_getval** (const **flist64_t** list[], const char *key)
  get 64 bit value by key
- const char * **flist64_getkey** (const **flist64_t** list[], uint64_t val)
  get key from 64 bit value
- int **flist64_from_str** (const char *str, const **flist64_t** list[], uint64_t *flags, uint64_t *mask, char clmod, char delim)
  parse flag list string
- char * **flist64_to_str** (const **flist64_t** list[], uint64_t val, char delim)
  convert bit mask to flag list string

8.27 flist/flist32_from_str.c File Reference

```
#include <stdlib.h>
#include <errno.h>
#include "flist.h"
#include "str.h"
```

Include dependency graph for flist32_from_str.c:

Functions

- int **flist32_from_str** (const char *str, const **flist32_t** list[], uint32_t *flags, uint32_t *mask, char clmod, char delim)

  parse flag list string

8.28 flist/flist32_getkey.c File Reference

```
#include <stdlib.h>
#include <errno.h>
#include "flist.h"
```

Include dependency graph for flist32_getkey.c:

Functions

- const char * **flist32_getkey** (const **flist32_t** list[], uint32_t val)
  get key from 32 bit value

8.29 flist/flist32_getval.c File Reference

```
#include <errno.h>
#include "flist.h"
#include "str.h"
```

Include dependency graph for flist32_getval.c:

Functions

- uint32_t **flist32_getval** (const **flist32_t** list[], const char *key)
  get 32 bit value by key

8.30 flist/flist32_to_str.c File Reference

```
#include "flist.h"
#include "str.h"
#include "stralloc.h"
```

Include dependency graph for flist32_to_str.c:

Functions

- char * **flist32_to_str** (const **flist32_t** list[], uint32_t val, char delim)
  convert bit mask to flag list string

8.31 flist/flist64_from_str.c File Reference

```
#include <stdlib.h>
#include <errno.h>
#include "flist.h"
#include "str.h"
```

Include dependency graph for flist64_from_str.c:

Functions

- int **flist64_from_str** (const char *str, const **flist64_t** list[], uint64_t *flags, uint64_t *mask, char clmod, char delim)

  parse flag list string

8.32 flist/flist64_getkey.c File Reference

```
#include <stdlib.h>
#include <errno.h>
#include "flist.h"
```

Include dependency graph for flist64_getkey.c:

Functions

- const char * **flist64_getkey** (const **flist64_t** list[], uint64_t val)
  get key from 64 bit value

8.33 flist/flist64_getval.c File Reference

```
#include <errno.h>
#include "flist.h"
#include "str.h"
```

Include dependency graph for flist64_getval.c:

Functions

- uint64_t **flist64_getval** (const **flist64_t** list[], const char *key)
  get 64 bit value by key

8.34 flist/flist64_to_str.c File Reference

```
#include "flist.h"
#include "str.h"
#include "stralloc.h"
```

Include dependency graph for flist64_to_str.c:

Functions

- char * **flist64_to_str** (const **flist64_t** list[], uint64_t val, char delim)
  convert bit mask to flag list string

8.35 io.h File Reference

This graph shows which files directly or indirectly include this file:

Defines

- #define **CHUNKSIZE** 128

Functions

- int **io_read_eol** (int fd, char **line)
  read a line of input
- int **io_read_eof** (int fd, char **file)
  read until end of file
- int **io_read_len** (int fd, char **str, size_t len)
  read exact number of bytes

8.36 io/io_read_eof.c File Reference

```
#include <stdlib.h>
#include <unistd.h>
#include "io.h"
#include "str.h"
```

Include dependency graph for io_read_eof.c:

Functions

- int **io_read_eof** (int fd, char **file)
  read until end of file

8.37 io/io_read_eol.c File Reference

```
#include <stdlib.h>
#include <unistd.h>
#include "io.h"
#include "str.h"
```

Include dependency graph for io_read_eol.c:

Functions

- int **io_read_eol** (int fd, char **line)
  read a line of input

8.38 io/io_read_len.c File Reference

```
#include <stdlib.h>
#include <unistd.h>
#include "io.h"
#include "str.h"
```

Include dependency graph for io_read_len.c:

Functions

- int **io_read_len** (int fd, char **str, size_t len)
  read exact number of bytes

8.39 list.h File Reference

`#include <stddef.h>`

Include dependency graph for list.h:

Data Structures

- struct **list_head**
  list head

Defines

- #define **container_of**(ptr, type, member) ((type *)((char *)(ptr) - offsetof(type, member)))
  get container of list head
- #define **LIST_HEAD_INIT**(name) { &(name), &(name) }
  list head initialization
- #define **LIST_HEAD**(name) struct **list_head** name = LIST_HEAD_INIT(name)
  list head creation
- #define **list_entry**(ptr, type, member) container_of(ptr, type, member)
  get the struct for this entry
- #define **list_for_each**(pos, head) for (pos = (head) → next; pos != (head); pos = pos → next)
  iterate over a list
- #define **list_for_each_prev**(pos, head) for (pos = (head) → prev; pos != (head); pos = pos → prev)
  iterate over a list backwards
- #define **list_for_each_safe**(pos, n, head)
  iterate over a list safe against removal of list entry
- #define **list_for_each_entry**(pos, head, member)
  iterate over list of given type
- #define **list_for_each_entry_reverse**(pos, head, member)
  iterate backwards over list of given type.
- #define **list_for_each_entry_safe**(pos, n, head, member)
  iterate over list of given type safe against removal of list entry
- #define **list_for_each_entry_safe_reverse**(pos, n, head, member)

iterate backwards over list of given type safe against removal of list entry

8.40 log.h File Reference

`#include <stdbool.h>`

`#include <stdarg.h>`

`#include <libgen.h>`

Include dependency graph for log.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- struct **log_options_t**
  multiplexer configuration data

Defines

- #define **LOG_TRACEME**
  simple trace helper

Functions

- void **log_init** (**log_options_t** *options)
  initialize log message mutliplexer
- void **log_internal** (int level, int strerr, const char *fmt, va_list ap)
- int **log_emerg** (const char *fmt,...)
  send EMERG level message to the multiplexer
- int **log_alert** (const char *fmt,...)
  send ALERT level message to the multiplexer
- int **log_crit** (const char *fmt,...)
  send CRIT level message to the multiplexer
- int **log_error** (const char *fmt,...)
  send ERR level message to the multiplexer
- int **log_warn** (const char *fmt,...)
  send WARNING level message to the multiplexer
- int **log_notice** (const char *fmt,...)
  send NOTICE level message to the multiplexer

- int **log_info** (const char *fmt,...)
  send INFO level message to the multiplexer
- int **log_debug** (const char *fmt,...)
  send DEBUG level message to the multiplexer
- void **log_emerg_and_die** (const char *fmt,...)
  send EMERG level message to the multiplexer and exit(2)
- void **log_alert_and_die** (const char *fmt,...)
  send ALERT level message to the multiplexer and exit(2)
- void **log_crit_and_die** (const char *fmt,...)
  send CRIT level message to the multiplexer and exit(2)
- void **log_error_and_die** (const char *fmt,...)
  send ERR level message to the multiplexer and exit(2)
- int **log_pemerg** (const char *fmt,...)
  send EMERG level message to the multiplexer and append strerror(errno)
- int **log_palert** (const char *fmt,...)
  send ALERT level message to the multiplexer and append strerror(errno)
- int **log_pcrit** (const char *fmt,...)
  send CRIT level message to the multiplexer and append strerror(errno)
- int **log_perror** (const char *fmt,...)
  send ERR level message to the multiplexer and append strerror(errno)
- int **log_pwarn** (const char *fmt,...)
  send WARNING level message to the multiplexer and append strerror(errno)
- int **log_pnotice** (const char *fmt,...)
  send NOTICE level message to the multiplexer and append strerror(errno)
- int **log_pinfo** (const char *fmt,...)
  send INFO level message to the multiplexer and append strerror(errno)
- int **log_pdebug** (const char *fmt,...)
  send DEBUG level message to the multiplexer and append strerror(errno)
- void **log_pemerg_and_die** (const char *fmt,...)
  send EMERG level message to the multiplexer, append strerror(errno) and exit(2)
- void **log_palert_and_die** (const char *fmt,...)
  send ALERT level message to the multiplexer, append strerror(errno) and exit(2)
- void **log_pcrit_and_die** (const char *fmt,...)

send CRIT level message to the multiplexer, append strerror(errno) and exit(2)

- void **log_perror_and_die** (const char *fmt,...)

  send ERR level message to the multiplexer, append strerror(errno) and exit(2)

- void **log_close** (void)

  close connection to logging system

8.41 log/log_close.c File Reference

```
#include <unistd.h>
#include <stdlib.h>
#include <syslog.h>
#include "log.h"
```

Include dependency graph for log_close.c:

Functions

- void **log_close** (void)
  close connection to logging system

Variables

- **log_options_t** * **_log_options**

8.41.1 Variable Documentation

8.41.1.1 log_options_t* _log_options

Definition at line 26 of file log_init.c.

Referenced by log_close(), log_init(), and log_internal().

8.42 log/log_init.c File Reference

```
#include <unistd.h>
#include <stdlib.h>
#include <syslog.h>
#include <sys/stat.h>
#include "log.h"
#include "str.h"
```

Include dependency graph for log_init.c:

Functions

- void **log_init** (**log_options_t** *options)
  initialize log message mutliplexer

Variables

- **log_options_t** * **_log_options** = NULL

8.42.1 Variable Documentation

8.42.1.1 log_options_t* _log_options = NULL

Definition at line 26 of file log_init.c.

8.43 log/log_internal.c File Reference

```
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <stdarg.h>
#include <syslog.h>
#include <string.h>
#include <time.h>
#include "log.h"
#include "printf.h"
#include "str.h"
```

Include dependency graph for log_internal.c:

Defines

- #define **LOGFUNC**(name, level, rc)
- #define **LOGFUNCDIE**(name, level)
- #define **LOGPFUNC**(name, level, rc)
- #define **LOGPFUNCDIE**(name, level)

Functions

- void **log_internal** (int level, int strerr, const char *fmt, va_list ap)

Variables

- **log_options_t** * **_log_options**

8.43.1 Define Documentation

8.43.1.1 #define LOGFUNC(name, level, rc)

Value:

```
int log_ ## name (const char *fmt, ...) { \
        va_list ap; va_start(ap, fmt); \
        log_internal(level, 0, fmt, ap); \
        va_end(ap); \
        return rc; \
}
```

Definition at line 96 of file log_internal.c.

8.43.1.2 #define LOGFUNCDIE(name, level)

Value:

```
void log_ ## name ## _and_die(const char *fmt, ...) { \
        va_list ap; va_start(ap, fmt); \
        log_internal(level, 0, fmt, ap); \
        va_end(ap); \
        exit(EXIT_FAILURE); \
}
```

Definition at line 113 of file log_internal.c.

8.43.1.3 #define LOGPFUNC(name, level, rc)

Value:

```
int log_p ## name (const char *fmt, ...) { \
        errno_orig = errno; \
        va_list ap; va_start(ap, fmt); \
        log_internal(level, 1, fmt, ap); \
        va_end(ap); \
        return rc; \
}
```

Definition at line 126 of file log_internal.c.

8.43.1.4 #define LOGPFUNCDIE(name, level)

Value:

```
void log_p ## name ## _and_die(const char *fmt, ...) { \
        errno_orig = errno; \
        va_list ap; va_start(ap, fmt); \
        log_internal(level, 1, fmt, ap); \
        va_end(ap); \
        exit(EXIT_FAILURE); \
}
```

Definition at line 144 of file log_internal.c.

8.43.2 Variable Documentation

8.43.2.1 log_options_t* _log_options

Definition at line 26 of file log_init.c.

8.44 misc.h File Reference

`#include <sys/types.h>`

Include dependency graph for misc.h:

This graph shows which files directly or indirectly include this file:

Functions

- int **isdir** (const char *path)
  check if given path is a directory
- int **isfile** (const char *path)
  check if given path is a regular file
- int **islink** (const char *path)
  check if given path is a symbolic link
- int **mkdirnamep** (const char *path, mode_t mode)
  recursive mkdir(2) with dirname(3)
- int **mkdirp** (const char *path, mode_t mode)
  recursive mkdir(2)
- int **runlink** (const char *path)
  recursive unlink(2) and rmdir(2)

8.45 misc/isdir.c File Reference

```
#include <sys/stat.h>
#include "misc.h"
```

Include dependency graph for isdir.c:

Functions

- int **isdir** (const char *path)
  check if given path is a directory

8.46 misc/isfile.c File Reference

```
#include <sys/stat.h>
```

```
#include "misc.h"
```

Include dependency graph for isfile.c:

Functions

- int **isfile** (const char *path)
  check if given path is a regular file

8.47 misc/islink.c File Reference

```
#include <sys/stat.h>
#include "misc.h"
```

Include dependency graph for islink.c:

Functions

- int **islink** (const char *path)
  check if given path is a symbolic link

8.48 misc/mkdirnamep.c File Reference

```
#include <stdlib.h>
#include <errno.h>
#include <libgen.h>
#include "misc.h"
#include "str.h"
```

Include dependency graph for mkdirnamep.c:

Functions

- int **mkdirnamep** (const char *path, mode_t mode)
  recursive mkdir(2) with dirname(3)

8.49 misc/mkdirp.c File Reference

```
#include <stdlib.h>
#include <errno.h>
#include <sys/stat.h>
#include "misc.h"
#include "str.h"
```

Include dependency graph for mkdirp.c:

Functions

- int **mkdirp** (const char *path, mode_t mode)
  recursive mkdir(2)

8.50 misc/runlink.c File Reference

```
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <dirent.h>
#include <sys/stat.h>
#include "misc.h"
#include "printf.h"
```

Include dependency graph for runlink.c:

Functions

- int **runlink** (const char *path)
  recursive unlink(2) and rmdir(2)

8.51 open.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- int **open_append** (const char *filename)
  open file in append mode
- int **open_excl** (const char *filename)
  open file exclusively
- int **open_read** (const char *filename)
  open file for reading
- int **open_rw** (const char *filename)
  open file for reading and writing
- int **open_trunc** (const char *filename)
  open and truncate file for reading and writing
- int **open_write** (const char *filename)
  open file for writing

8.52 open/open_append.c File Reference

```
#include <fcntl.h>
```

```
#include "open.h"
```

Include dependency graph for open_append.c:

Functions

- int **open_append** (const char *filename)
  open file in append mode

8.53 open/open_excl.c File Reference

```
#include <fcntl.h>
```

```
#include "open.h"
```

Include dependency graph for open_excl.c:

Functions

- int **open_excl** (const char *filename)
  open file exclusively

8.54 open/open_read.c File Reference

```
#include <fcntl.h>
```

```
#include "open.h"
```

Include dependency graph for open_read.c:

Functions

- int **open_read** (const char *filename)
  open file for reading

8.55 open/open_rw.c File Reference

```
#include <fcntl.h>
```

```
#include "open.h"
```

Include dependency graph for open_rw.c:

Functions

- int **open_rw** (const char *filename)
  open file for reading and writing

8.56 open/open_trunc.c File Reference

```
#include <fcntl.h>
```

```
#include "open.h"
```

Include dependency graph for open_trunc.c:

Functions

- int **open_trunc** (const char *filename)
  open and truncate file for reading and writing

8.57 open/open_write.c File Reference

```
#include <fcntl.h>
```

```
#include "open.h"
```

Include dependency graph for open_write.c:

Functions

- int **open_write** (const char *filename)
  open file for writing

8.58 printf.h File Reference

`#include <stdarg.h>`

Include dependency graph for printf.h:

This graph shows which files directly or indirectly include this file:

Functions

- int **_lucid_vsnprintf** (char *str, int size, const char *fmt, va_list ap)
  write conversion to string using va_list
- int **_lucid_snprintf** (char *str, int size, const char *fmt,...)
  write conversion to string using variable number of arguments
- int **_lucid_vasprintf** (char **ptr, const char *fmt, va_list ap)
  write conversion to allocated string using va_list
- int **_lucid_asprintf** (char **ptr, const char *fmt,...)
  write conversion to allocated string using variable number of arguments
- int **_lucid_vdprintf** (int fd, const char *fmt, va_list ap)
  write conversion to file descriptor using va_list
- int **_lucid_dprintf** (int fd, const char *fmt,...)
  write conversion to file descriptor using variable number of arguments
- int **_lucid_vprintf** (const char *fmt, va_list ap)
  write conversion to stdout using va_list
- int **_lucid_printf** (const char *fmt,...)
  write conversion to stdout using variable number of arguments

8.59 printf/asprintf.c File Reference

```
#include "printf.h"
```

Include dependency graph for asprintf.c:

Functions

- int **_lucid_asprintf** (char **ptr, const char *fmt,...)
  write conversion to allocated string using variable number of arguments

8.60 printf/dprintf.c File Reference

```
#include "printf.h"
```

Include dependency graph for dprintf.c:

Functions

- int **_lucid_dprintf** (int fd, const char *fmt,...)
  write conversion to file descriptor using variable number of arguments

8.61 printf/printf.c File Reference

```
#include "printf.h"
```

Include dependency graph for printf.c:

Functions

- int **_lucid_printf** (const char *fmt,...)
  write conversion to stdout using variable number of arguments

8.62 printf/snprintf.c File Reference

```
#include "printf.h"
```

Include dependency graph for snprintf.c:

Functions

- int **_lucid_snprintf** (char *str, int size, const char *fmt,...)
  write conversion to string using variable number of arguments

8.63 printf/vasprintf.c File Reference

```
#include <stdlib.h>
```

```
#include "printf.h"
```

Include dependency graph for vasprintf.c:

Functions

- int **_lucid_vasprintf** (char **ptr, const char *fmt, va_list ap)
  write conversion to allocated string using va_list

8.64 printf/vdprintf.c File Reference

```
#include <unistd.h>
#include <stdlib.h>
#include "printf.h"
```

Include dependency graph for vdprintf.c:

Functions

- int **_lucid_vdprintf** (int fd, const char *fmt, va_list ap)
  write conversion to file descriptor using va_list

8.65 printf/vprintf.c File Reference

```
#include "printf.h"
```

Include dependency graph for vprintf.c:

Functions

- int **_lucid_vprintf** (const char *fmt, va_list ap)
  write conversion to stdout using va_list

8.66 printf/vsnprintf.c File Reference

```
#include "printf.h"
#include "str.h"
```

Include dependency graph for vsnprintf.c:

Data Structures

- struct **__printf_t**

Defines

- #define **PFR_MIN** PFR_CHAR
- #define **PFR_MAX** PFR_LLONG
- #define **EMIT**(C) { if (idx < size) { *str++ = C; } idx++; }

Enumerations

- enum **__printf_flags** {
  PFL_ALT = 0x01, **PFL_ZERO** = 0x02, **PFL_LEFT** = 0x04, **PFL_BLANK** = 0x08,
  PFL_SIGN = 0x10, **PFL_UPPER** = 0x20, **PFL_SIGNED** = 0x40 }
- enum **__printf_rank** {
  PFR_CHAR, **PFR_SHORT**, **PFR_INT**, **PFR_LONG**,
  PFR_LLONG }
- enum **__printf_state** {
  PFS_NORMAL, **PFS_FLAGS**, **PFS_WIDTH**, **PFS_PREC**,
  PFS_MOD, **PFS_CONV** }

Functions

- int **_lucid_vsnprintf** (char *str, int size, const char *fmt, va_list _ap)
  write conversion to string using va_list

8.66.1 Define Documentation

8.66.1.1 #define PFR_MIN PFR_CHAR

Definition at line 39 of file vsnprintf.c.

Referenced by _lucid_vsnprintf().

8.66.1.2 #define PFR_MAX PFR_LLONG

Definition at line 40 of file vsnprintf.c.

Referenced by _lucid_vsnprintf().

8.66.1.3 #define EMIT(C) { if (idx < size) { *str++ = C; } idx++; }

Definition at line 59 of file vsnprintf.c.

Referenced by _lucid_vsnprintf().

8.66.2 Enumeration Type Documentation

8.66.2.1 enum __printf_flags

Enumerator:

PFL_ALT

PFL_ZERO

PFL_LEFT

PFL_BLANK

PFL_SIGN

PFL_UPPER

PFL_SIGNED

Definition at line 21 of file vsnprintf.c.

```
21                     {
22         PFL_ALT    = 0x01,
23         PFL_ZERO   = 0x02,
24         PFL_LEFT   = 0x04,
25         PFL_BLANK  = 0x08,
26         PFL_SIGN   = 0x10,
27         PFL_UPPER  = 0x20,
28         PFL_SIGNED = 0x40,
29 };
```

8.66.2.2 enum __printf_rank

Enumerator:

PFR_CHAR

PFR_SHORT

PFR_INT

PFR_LONG

PFR_LLONG

Definition at line 31 of file vsnprintf.c.

```
31                  {
32         PFR_CHAR,
33         PFR_SHORT,
34         PFR_INT,
35         PFR_LONG,
36         PFR_LLONG,
37 };
```

8.66.2.3 enum __printf_state

Enumerator:

PFS_NORMAL

PFS_FLAGS

PFS_WIDTH

PFS_PREC

PFS_MOD

PFS_CONV

Definition at line 42 of file vsnprintf.c.

```
42                   {
43         PFS_NORMAL,
44         PFS_FLAGS,
45         PFS_WIDTH,
46         PFS_PREC,
47         PFS_MOD,
48         PFS_CONV,
49 };
```

8.67 scanf.h File Reference

`#include <stdarg.h>`

Include dependency graph for scanf.h:

This graph shows which files directly or indirectly include this file:

Functions

- int **_lucid_vsscanf** (const char *str, const char *fmt, va_list ap)
  read conversion from string using va_list
- int **_lucid_sscanf** (const char *str, const char *fmt,...)
  read conversion from string using variable number of arguments

8.68 scanf/sscanf.c File Reference

```
#include "scanf.h"
```

Include dependency graph for sscanf.c:

Functions

- int **_lucid_sscanf** (const char *str, const char *fmt,...)
  read conversion from string using variable number of arguments

8.69 scanf/vsscanf.c File Reference

```
#include "scanf.h"
```

```
#include "str.h"
```

Include dependency graph for vsscanf.c:

Data Structures

- struct **__scanf_t**

Defines

- #define **SFR_MIN** SFR_CHAR
- #define **SFR_MAX** SFR_LLONG

Enumerations

- enum **__scanf_flags** { **SFL_NOOP** = 0x01, **SFL_WIDTH** = 0x02 }
- enum **__scanf_rank** {
  SFR_CHAR, **SFR_SHORT**, **SFR_INT**, **SFR_LONG**,
  SFR_LLONG }
- enum **__scanf_state** {
  SFS_NORMAL, **SFS_FLAGS**, **SFS_WIDTH**, **SFS_MOD**,
  SFS_CONV, **SFS_STORE**, **SFS_EOF**, **SFS_ERR** }

Functions

- int **_lucid_vsscanf** (const char *str, const char *fmt, va_list _ap)
  read conversion from string using va_list

8.69.1 Define Documentation

8.69.1.1 #define SFR_MIN SFR_CHAR

Definition at line 34 of file vsscanf.c.

Referenced by _lucid_vsscanf().

8.69.1.2 #define SFR_MAX SFR_LLONG

Definition at line 35 of file vsscanf.c.

Referenced by _lucid_vsscanf().

8.69.2 Enumeration Type Documentation

8.69.2.1 enum __scanf_flags

Enumerator:

SFL_NOOP
SFL_WIDTH

Definition at line 21 of file vsscanf.c.

```
21                  {
22         SFL_NOOP  = 0x01,
23         SFL_WIDTH = 0x02,
24 };
```

8.69.2.2 enum __scanf_rank

Enumerator:

SFR_CHAR
SFR_SHORT
SFR_INT
SFR_LONG
SFR_LLONG

Definition at line 26 of file vsscanf.c.

```
26                  {
27         SFR_CHAR,
28         SFR_SHORT,
29         SFR_INT,
30         SFR_LONG,
31         SFR_LLONG,
32 };
```

8.69.2.3 enum __scanf_state

Enumerator:

SFS_NORMAL
SFS_FLAGS
SFS_WIDTH
SFS_MOD
SFS_CONV
SFS_STORE
SFS_EOF
SFS_ERR

Definition at line 37 of file vsscanf.c.

```
37                  {
38         SFS_NORMAL,
39         SFS_FLAGS,
40         SFS_WIDTH,
41         SFS_MOD,
42         SFS_CONV,
43         SFS_STORE,
44         SFS_EOF,
45         SFS_ERR,
46 };
```

8.70 str.h File Reference

This graph shows which files directly or indirectly include this file:

Defines

- #define **char_isascii**(ch) ((unsigned int)(ch) < 128u)
  check for an ASCII character
- #define **char_isblank**(ch) (ch == ' ' || ch == '\t')
  check for a blank character (space, horizontal tab)
- #define **char_iscntrl**(ch) ((unsigned int)(ch) < 32u || ch == 127)
  check for an ASCII control character
- #define **char_isdigit**(ch) ((unsigned int)(ch - '0') < 10u)
  check for a digit character (0-9)
- #define **char_isgraph**(ch) ((unsigned int)(ch - '!') < 94u)
  check for graphable characters (excluding space)
- #define **char_islower**(ch) ((unsigned int)(ch - 'a') < 26u)
  check for a lower-case character
- #define **char_isprint**(ch) ((unsigned int)(ch - ' ') < 95u)
  check for a printable character (including space)
- #define **char_isspace**(ch) ((unsigned int)(ch - '\t') < 5u || ch == ' ')
  check for a whitespace character (\t, \n, \v, \f, \r)
- #define **char_isupper**(ch) ((unsigned int)(ch - 'A') < 26u)
  check for an upper-case character
- #define **char_isxdigit**(ch)
  check for a hexadecimal character
- #define **char_isalpha**(ch) (char_islower(ch) || char_isupper(ch))
  check for an upper- or lower-case character
- #define **char_isalnum**(ch) (char_isalpha(ch) || char_isdigit(ch))
  check for an upper-, lower-case or digit character
- #define **char_ispunct**(ch)
  check for a punctuation character
- #define **char_tolower**(ch) do { if (char_isupper(ch)) ch += 32; } while(0)
  convert character to lower-case

- #define **char_toupper**(ch) do { if (char_islower(ch)) ch -= 32; } while(0)
  convert character to upper-case
- #define **CC_ALNUM** (1 << 1)
  class for alpha-numerical characters
- #define **CC_ALPHA** (1 << 2)
  class for upper- or lower-case characters
- #define **CC_ASCII** (1 << 3)
  class for ASCII characters
- #define **CC_BLANK** (1 << 4)
  class for blank characters
- #define **CC_CNTRL** (1 << 5)
  class for ASCII control characters
- #define **CC_DIGIT** (1 << 6)
  class for digit characters
- #define **CC_GRAPH** (1 << 7)
  class for graphable characters
- #define **CC_LOWER** (1 << 8)
  class for lower-case characters
- #define **CC_PRINT** (1 << 9)
  class for printable characters
- #define **CC_PUNCT** (1 << 10)
  class for punctuation characters
- #define **CC_SPACE** (1 << 11)
  class for white space characters
- #define **CC_UPPER** (1 << 12)
  class for upper-case characters
- #define **CC_XDIGIT** (1 << 13)
  class for hexadecimal characters
- #define **str_isempty**(str) (!str || str_check(str, CC_SPACE))
  check if string is empty
- #define **str_isalnum**(str) str_check(str, CC_ALNUM)
  check string for alpha-numerical characters
- #define **str_isalpha**(str) str_check(str, CC_ALPHA)
  check string for upper- or lower-case characters

- #define **str_isascii**(str) str_check(str, CC_ASCII)
  check string for ASCII characters
- #define **str_isdigit**(str) str_check(str, CC_DIGIT)
  check string for digit characters
- #define **str_isgraph**(str) str_check(str, CC_GRAPH)
  check string for graphable characters
- #define **str_islower**(str) str_check(str, CC_LOWER)
  check string for lower-case characters
- #define **str_isprint**(str) str_check(str, CC_PRINT)
  check string for printable characters
- #define **str_isupper**(str) str_check(str, CC_UPPER)
  check string for upper-case characters
- #define **str_isxdigit**(str) str_check(str, CC_XDIGIT)
  check string for hexadecimal characters

Functions

- int **str_check** (const char *str, int allowed)
  check string against classes of allowed characters
- int **str_cmp** (const char *str1, const char *str2)
  compare two strings
- int **str_cpy** (char *dst, const char *src)
  copy a string
- int **str_cpyn** (void *dst, const void *src, int n)
  copy a string
- char * **str_dup** (const char *str)
  duplicate a string
- char * **str_dupn** (const char *str, int n)
  duplicate a string
- char * **str_index** (const char *str, int c, int n)
  scan string for character
- int **str_len** (const char *str)
  calculate the length of a string
- void **str_zero** (void *str, int n)

write zero-valued bytes

- char * **str_path_concat** (const char *dirname, const char *basename)
  concatenate dirname and basename
- int **str_path_isabs** (const char *str)
  check if path is absolute and contains no dot entries or ungraphable characters
- int **str_path_isdot** (const char *str)
  check if given path contains . or .. entries
- char * **str_tolower** (char *str)
  convert string to lower-case
- char * **str_toupper** (char *str)
  convert string to upper-case
- int **str_toumax** (const char *str, unsigned long long int *val, int base, int n)
  convert string to integer

8.71 str/str_check.c File Reference

```
#include "str.h"
```

Include dependency graph for str_check.c:

Functions

- int **str_check** (const char *str, int allowed)
  check string against classes of allowed characters

8.72 str/str_cmp.c File Reference

```
#include "str.h"
```

Include dependency graph for str_cmp.c:

Functions

- int **str_cmp** (const char *str1, const char *str2)
  compare two strings

8.73 str/str_cpy.c File Reference

```
#include "str.h"
```

Include dependency graph for str_cpy.c:

Functions

- int **str_cpy** (char *dst, const char *src)
  copy a string

8.74 str/str_cpyn.c File Reference

```
#include "str.h"
```

Include dependency graph for str_cpyn.c:

Functions

- int **str_cpyn** (void *dst, const void *src, int n)
  copy a string

8.75 str/str_dup.c File Reference

```
#include "str.h"
```

Include dependency graph for str_dup.c:

Functions

- char * **str_dup** (const char *str)
  duplicate a string

8.76 str/str_dupn.c File Reference

```
#include <stdlib.h>
```

```
#include "str.h"
```

Include dependency graph for str_dupn.c:

Functions

- char * **str_dupn** (const char *str, int n)
  duplicate a string

8.77 str/str_index.c File Reference

```
#include "str.h"
```

Include dependency graph for str_index.c:

Functions

- char * **str_index** (const char *str, int c, int n)
  scan string for character

8.78 str/str_len.c File Reference

```
#include "str.h"
```

Include dependency graph for str_len.c:

Functions

- int **str_len** (const char *str)
  calculate the length of a string

8.79 str/str_path_concat.c File Reference

```
#include "printf.h"
```

```
#include "str.h"
```

Include dependency graph for str_path_concat.c:

Functions

- char * **str_path_concat** (const char *dirname, const char *basename)
  concatenate dirname and basename

8.80 str/str_path_isabs.c File Reference

```
#include <stdlib.h>
```

```
#include "str.h"
```

Include dependency graph for str_path_isabs.c:

Functions

- int **str_path_isabs** (const char *str)
  check if path is absolute and contains no dot entries or ungraphable characters

8.81 str/str_path_isdot.c File Reference

```
#include <stdlib.h>
```

```
#include "str.h"
```

Include dependency graph for str_path_isdot.c:

Functions

- int **str_path_isdot** (const char *str)
  check if given path contains . or .. entries

8.82 str/str_tolower.c File Reference

```
#include "str.h"
```

Include dependency graph for str_tolower.c:

Functions

- char * **str_tolower** (char *str)
  convert string to lower-case

8.83 str/str_toumax.c File Reference

```
#include "str.h"
```

Include dependency graph for str_toumax.c:

Functions

- int **str_toumax** (const char *str, unsigned long long int *val, int base, int n)
  convert string to integer

8.84 str/str_toupper.c File Reference

```
#include "str.h"
```

Include dependency graph for str_toupper.c:

Functions

- char * **str_toupper** (char *str)
  convert string to upper-case

8.85 str/str_zero.c File Reference

```
#include "str.h"
```

Include dependency graph for str_zero.c:

Functions

- void **str_zero** (void *str, int n)
  write zero-valued bytes

8.86 stralloc.h File Reference

`#include <sys/types.h>`

Include dependency graph for stralloc.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- struct **stralloc_t**
  dynamic string allocator tracking data

Functions

- void **stralloc_init** (**stralloc_t** *sa)
  initialize dynamic string allocator
- void **stralloc_zero** (**stralloc_t** *sa)
  truncate string length to zero
- int **stralloc_ready** (**stralloc_t** *sa, size_t len)
  ensure that enough memory has been allocated
- int **stralloc_readyplus** (**stralloc_t** *sa, size_t len)
  ensure that enough memory has been allocated
- void **stralloc_free** (**stralloc_t** *sa)
  deallocate all memory
- int **stralloc_copyb** (**stralloc_t** *dst, const char *src, size_t len)
  copy a static string to a dynamic one
- int **stralloc_copys** (**stralloc_t** *dst, const char *src)
  copy a static string to a dynamic one
- int **stralloc_copy** (**stralloc_t** *dst, const **stralloc_t** *src)
  copy one dynamic string to another
- int **stralloc_catb** (**stralloc_t** *dst, const char *src, size_t len)
  concatenate a dynamic string and a static one
- int **stralloc_catf** (**stralloc_t** *dst, const char *fmt,...)
  concatenate a dynamic string and a static one using formatted conversion

- int **stralloc_catm** (**stralloc_t** *dst,...)
  concatenate a dynamic string and multiple static ones
- int **stralloc_cats** (**stralloc_t** *dst, const char *src)
  concatenate a dynamic string and a static one
- int **stralloc_cat** (**stralloc_t** *dst, const **stralloc_t** *src)
  concatenate two dynamic strings
- int **stralloc_cmp** (const **stralloc_t** *a, const **stralloc_t** *b)
  compare two dynamic strings

8.87 stralloc/stralloc_cat.c File Reference

```
#include "stralloc.h"
```

Include dependency graph for stralloc_cat.c:

Functions

- int **stralloc_cat** (**stralloc_t** *dst, const **stralloc_t** *src)
  concatenate two dynamic strings

8.88 stralloc/stralloc_catb.c File Reference

```
#include "str.h"
```

```
#include "stralloc.h"
```

Include dependency graph for stralloc_catb.c:

Functions

- int **stralloc_catb** (**stralloc_t** *dst, const char *src, size_t len)
  concatenate a dynamic string and a static one

8.89 stralloc/stralloc_catf.c File Reference

```
#include <stdarg.h>
#include <stdlib.h>
#include "printf.h"
#include "stralloc.h"
```

Include dependency graph for stralloc_catf.c:

Functions

- int **stralloc_catf** (**stralloc_t** *dst, const char *fmt,...)
  concatenate a dynamic string and a static one using formatted conversion

8.90 stralloc/stralloc_catm.c File Reference

```
#include <stdarg.h>
```

```
#include "stralloc.h"
```

Include dependency graph for stralloc_catm.c:

Functions

- int **stralloc_catm** (**stralloc_t** *dst,...)
  concatenate a dynamic string and multiple static ones

8.91 stralloc/stralloc_cats.c File Reference

```
#include "str.h"
#include "stralloc.h"
```

Include dependency graph for stralloc_cats.c:

Functions

- int **stralloc_cats** (**stralloc_t** *dst, const char *src)
  concatenate a dynamic string and a static one

8.92 stralloc/stralloc_cmp.c File Reference

```
#include "stralloc.h"
```

Include dependency graph for stralloc_cmp.c:

Functions

- int **stralloc_cmp** (const **stralloc_t** *a, const **stralloc_t** *b)
  compare two dynamic strings

8.93 stralloc/stralloc_copy.c File Reference

```
#include "stralloc.h"
```

Include dependency graph for stralloc_copy.c:

Functions

- int **stralloc_copy** (**stralloc_t** *dst, const **stralloc_t** *src)
  copy one dynamic string to another

8.94 stralloc/stralloc_copyb.c File Reference

```
#include "str.h"
#include "stralloc.h"
```

Include dependency graph for stralloc_copyb.c:

Functions

- int **stralloc_copyb** (**stralloc_t** *dst, const char *src, size_t len)
  copy a static string to a dynamic one

8.95 stralloc/stralloc_copys.c File Reference

```
#include "str.h"
#include "stralloc.h"
```

Include dependency graph for stralloc_copys.c:

Functions

- int **stralloc_copys** (**stralloc_t** *dst, const char *src)
  copy a static string to a dynamic one

8.96 stralloc/stralloc_free.c File Reference

```
#include <stdlib.h>
```

```
#include "stralloc.h"
```

Include dependency graph for stralloc_free.c:

Functions

- void **stralloc_free** (**stralloc_t** *sa)
  deallocate all memory

8.97 stralloc/stralloc_init.c File Reference

```
#include <stdlib.h>
```

```
#include "stralloc.h"
```

Include dependency graph for stralloc_init.c:

Functions

- void **stralloc_init** (**stralloc_t** *sa)
  initialize dynamic string allocator

8.98 stralloc/stralloc_ready.c File Reference

```
#include <stdlib.h>
```

```
#include "stralloc.h"
```

Include dependency graph for stralloc_ready.c:

Functions

- int **stralloc_ready** (**stralloc_t** *sa, size_t len)
  ensure that enough memory has been allocated

8.99 stralloc/stralloc_readyplus.c File Reference

```
#include <errno.h>
#include "stralloc.h"
```

Include dependency graph for stralloc_readyplus.c:

Functions

- int **stralloc_readyplus** (**stralloc_t** *sa, size_t len)
  ensure that enough memory has been allocated

8.100 stralloc/stralloc_zero.c File Reference

```
#include "stralloc.h"
```

Include dependency graph for stralloc_zero.c:

Functions

- void **stralloc_zero** (**stralloc_t** *sa)

  truncate string length to zero

8.101 tcp.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- int **tcp_listen** (const char *ip, int port, int backlog)
  listen for incomming connections
- int **tcp_connect** (const char *ip, int port)
  connect to TCP socket

8.102 tcp/tcp_connect.c File Reference

```
#include <unistd.h>
#include <errno.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include "addr.h"
#include "str.h"
#include "tcp.h"
```

Include dependency graph for tcp_connect.c:

Functions

- int **tcp_connect** (const char *ip, int port)
  connect to TCP socket

8.103 tcp/tcp_listen.c File Reference

```
#include <unistd.h>
#include <errno.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include "addr.h"
#include "str.h"
#include "tcp.h"
```

Include dependency graph for tcp_listen.c:

Functions

- int **tcp_listen** (const char *ip, int port, int backlog)
  listen for incomming connections

8.104 whirlpool.h File Reference

`#include <stdint.h>`

Include dependency graph for whirlpool.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- struct **whirlpool_t**
  dynamic whirlpool state data

Defines

- #define **DIGESTBYTES** 64
  number of bytes in the digest
- #define **DIGESTBITS** (8*DIGESTBYTES)
  number of bits in the digest
- #define **WBLOCKBYTES** 64
  number of bytes in the input buffer
- #define **WBLOCKBITS** (8*WBLOCKBYTES)
  number of bits in the input buffer
- #define **LENGTHBYTES** 32
  number of hashed bytes
- #define **LENGTHBITS** (8*LENGTHBYTES)
  number of hashed bits

Functions

- void **whirlpool_transform** (**whirlpool_t** *const context)
  internal transform routine
- void **whirlpool_init** (**whirlpool_t** *const context)
  initialize whirlpool state context
- void **whirlpool_finalize** (**whirlpool_t** *const context, unsigned char *const result)
  finalize whirlpool transformation

- void **whirlpool_add** (**whirlpool_t** *const context, const unsigned char *const src, unsigned long bits)

  add bytes to the transform routine

- char * **whirlpool_digest** (const char *str)

  create digest from string

8.105 whirlpool/whirlpool_add.c File Reference

```
#include "whirlpool.h"
```

Include dependency graph for whirlpool_add.c:

Functions

- void **whirlpool_add** (**whirlpool_t** *const context, const unsigned char *const src, unsigned long srcbits)

  add bytes to the transform routine

8.106 whirlpool/whirlpool_digest.c File Reference

```
#include <stdlib.h>
#include "str.h"
#include "stralloc.h"
#include "whirlpool.h"
```

Include dependency graph for whirlpool_digest.c:

Functions

- char * **whirlpool_digest** (const char *str)
  create digest from string

8.107 whirlpool/whirlpool_finalize.c File Reference

```
#include "str.h"
```

```
#include "whirlpool.h"
```

Include dependency graph for whirlpool_finalize.c:

Functions

- void **whirlpool_finalize** (**whirlpool_t** *const context, unsigned char *const result)
  finalize whirlpool transformation

8.108 whirlpool/whirlpool_init.c File Reference

```
#include "str.h"
```

```
#include "whirlpool.h"
```

Include dependency graph for whirlpool_init.c:

Functions

- void **whirlpool_init** (**whirlpool_t** *const context)
  initialize whirlpool state context

8.109 whirlpool/whirlpool_internal.h File Reference

This graph shows which files directly or indirectly include this file:

Defines

- #define **ONE8** 0xffU
- #define **ONE16** 0xffffU
- #define **ONE32** 0xffffffffU
- #define **ONE64** 0xffffffffffffffffULL
- #define **T8**(x) ((x) & ONE8)
- #define **T16**(x) ((x) & ONE16)
- #define **T32**(x) ((x) & ONE32)
- #define **T64**(x) ((x) & ONE64)
- #define **U8TO32_BIG**(c) (((u32)T8(*(c)) << 24) | ((u32)T8(*((c) + 1)) << 16) | ((u32)T8(*((c) + 2)) << 8) | ((u32)T8(*((c) + 3))))
- #define **U8TO32_LITTLE**(c) (((u32)T8(*(c))) | ((u32)T8(*((c) + 1)) << 8) | (u32)T8(*((c) + 2)) << 16) | ((u32)T8(*((c) + 3)) << 24))
- #define **U32TO8_BIG**(c, v) do { u32 x = (v); u8 *d = (c); d[0] = T8(x >> 24); d[1] = T8(x >> 16); d[2] = T8(x >> 8); d[3] = T8(x); } while (0)
- #define **U32TO8_LITTLE**(c, v) do { u32 x = (v); u8 *d = (c); d[0] = T8(x); d[1] = T8(x >> 8); d[2] = T8(x >> 16); d[3] = T8(x >> 24); } while (0)
- #define **ROTL32**(v, n) (T32((v) << (n)) | ((v) >> (32 - (n))))
- #define **ROTR64**(v, n) (((v) >> (n)) | T64((v) << (64 - (n))))
- #define **R** 10

8.109.1 Define Documentation

8.109.1.1 #define ONE8 0xffU

Definition at line 21 of file whirlpool_internal.h.

8.109.1.2 #define ONE16 0xffffU

Definition at line 22 of file whirlpool_internal.h.

8.109.1.3 #define ONE32 0xffffffffU

Definition at line 25 of file whirlpool_internal.h.

8.109.1.4 #define ONE64 0xffffffffffffffffULL

Definition at line 30 of file whirlpool_internal.h.

8.109.1.5 #define T8(x) ((x) & ONE8)

Definition at line 32 of file whirlpool_internal.h.

8.109.1.6 #define T16(x) ((x) & ONE16)

Definition at line 33 of file whirlpool_internal.h.

8.109.1.7 #define T32(x) ((x) & ONE32)

Definition at line 34 of file whirlpool_internal.h.

8.109.1.8 #define T64(x) ((x) & ONE64)

Definition at line 35 of file whirlpool_internal.h.

8.109.1.9 #define U8TO32_BIG(c) (((u32)T8(*(c)) << 24) | ((u32)T8(*((c) + 1)) << 16) | ((u32)T8(*((c) + 2)) << 8) | ((u32)T8(*((c) + 3))))

Definition at line 41 of file whirlpool_internal.h.

8.109.1.10 #define U8TO32_LITTLE(c) (((u32)T8(*(c))) | ((u32)T8(*((c) + 1)) << 8) | (u32)T8(*((c) + 2)) << 16) | ((u32)T8(*((c) + 3)) << 24))

Definition at line 47 of file whirlpool_internal.h.

8.109.1.11 #define U32TO8_BIG(c, v) do { u32 x = (v); u8 *d = (c); d[0] = T8(x >> 24); d[1] = T8(x >> 16); d[2] = T8(x >> 8); d[3] = T8(x); } while (0)

Definition at line 53 of file whirlpool_internal.h.

8.109.1.12 #define U32TO8_LITTLE(c, v) do { u32 x = (v); u8 *d = (c); d[0] = T8(x); d[1] = T8(x >> 8); d[2] = T8(x >> 16); d[3] = T8(x >> 24); } while (0)

Definition at line 59 of file whirlpool_internal.h.

8.109.1.13 #define ROTL32(v, n) (T32((v) << (n)) | ((v) >> (32 - (n))))

Definition at line 71 of file whirlpool_internal.h.

8.109.1.14 #define ROTR64(v, n) (((v) >> (n)) | T64((v) << (64 - (n))))

Definition at line 72 of file whirlpool_internal.h.

8.109.1.15 #define R 10

Definition at line 77 of file whirlpool_internal.h.

Referenced by whirlpool_transform().

8.110 whirlpool/whirlpool_tables.h File Reference

```
#include <stdint.h>
```

```
#include "whirlpool_internal.h"
```

Include dependency graph for whirlpool_tables.h:

This graph shows which files directly or indirectly include this file:

8.111 whirlpool/whirlpool_transform.c File Reference

```
#include <stdint.h>
#include "whirlpool.h"
#include "whirlpool_tables.h"
```

Include dependency graph for whirlpool_transform.c:

Functions

- void **whirlpool_transform** (**whirlpool_t** *const context)
  internal transform routine

Chapter 9

lucid Page Documentation

9.1 Examples

To be done ...

9.2 License

9.2.1 GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

9.2.1.1 Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

9.2.1.2 TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means

either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be

distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right

claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Index