echoclient

Generated by Doxygen 1.8.1.2

Sat Aug 31 2013 19:05:24

Contents

Chapter 1

File Index

1.1 File List

Here is a list of all files with brief descriptions:

Chapter 2

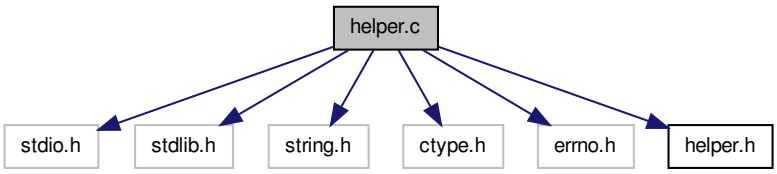
File Documentation

2.1 helper.c File Reference

Implementation of helper functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <errno.h>
#include "helper.h"
```

Include dependency graph for helper.c:



Macros

- #define MAX_BUFFER_SIZE 1024

  Maximum character buffer size.

Functions

- char * get_errmsg (void)

  Gets the global error message.
- void set_errmsg (const char *buffer)

  Sets the global error message.
- void set_errno_errmsg (const char *buffer)

  Sets the global error message based on errno.
- void mk_errmsg (const char *buffer, char **error_msg)

  Makes an error message.

- void mk_errno_errmsg (const char *buffer, char **error_msg)

  Makes an error message based on errno.
- char * trim_line_ending (char *buffer)

  Trims CR and LF characters from the end of a string.
- char * trim_right (char *buffer)

  Trims trailing whitespace from a string.
- char * trim_left (char *buffer)

  Trims leading whitespace from a string.
- char * trim (char *buffer)

  Trims leading and trailing whitespace from a string.

Variables

- static char helper_error_msg [MAX_BUFFER_SIZE] = {0}

  Global error message string.

2.1.1 Detailed Description

Implementation of helper functions.

Author

Paul Griffiths

Copyright

2.1.2 Macro Definition Documentation

2.1.2.1 #define MAX_BUFFER_SIZE 1024

Maximum character buffer size.

2.1.3 Function Documentation

2.1.3.1 char* get_errmsg (void)

Gets the global error message.

Returns

A pointer to the global error message.

2.1.3.2 void mk_errmsg (const char * *buffer*, char ** *error_msg*)

Makes an error message.

This function provides a thread-safe way for a function to set an error message.

Parameters

buffer	A buffer containing the error message.
error_msg	A pointer to a char pointer which will refer to the error message. This should be free()d by the called.

2.1.3.3 void mk_errno_errmsg (const char * *buffer,* char ** *error_msg*)

Makes an error message based on errno.

This function provides a thread-safe way for a function to set an error message, with the usual caveat that errno itself is not threadsafe.

Parameters

buffer	A buffer containing the error message.
error_msg	A pointer to a char pointer which will refer to the error message. This should be free()d by the called.

2.1.3.4 void set_errmsg (const char * *buffer*)

Sets the global error message.

Uses a statically allocated buffer, so this is not thread-safe.

Parameters

buffer	A buffer containing the error message.

2.1.3.5 void set_errno_errmsg (const char * *buffer*)

Sets the global error message based on errno.

Uses a statically allocated buffer, so this is not thread-safe.

Parameters

buffer	A buffer containing the error message.

2.1.3.6 char* trim (char * *buffer*)

Trims leading and trailing whitespace from a string.

Parameters

buffer	The string to trim.

Returns

A pointer to the passed buffer.

2.1.3.7 char* trim_left (char * *buffer*)

Trims leading whitespace from a string.

Parameters

buffer	The string to trim.

Returns

A pointer to the passed buffer.

2.1.3.8 char* trim_line_ending (char * *buffer*)

Trims CR and LF characters from the end of a string.

Parameters

buffer	The string to trim.

Returns

A pointer to the passed buffer.

2.1.3.9 char* trim_right (char * *buffer*)

Trims trailing whitespace from a string.

Parameters

buffer	The string to trim.

Returns

A pointer to the passed buffer.

2.1.4 Variable Documentation

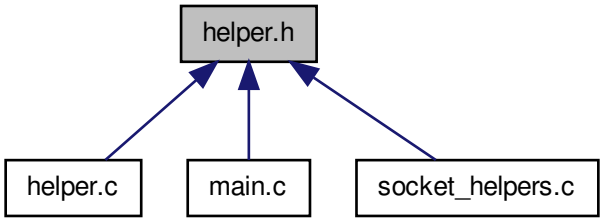
2.1.4.1 char helper_error_msg[MAX_BUFFER_SIZE] = {0} [static]

Global error message string.

2.2 helper.h File Reference

Interface to helper functions.

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



Macros

- #define TRUE 1

  TRUE identifier.
- #define FALSE 0

  FALSE identifier.
- #define ERROR_RETURN (-1)

  Generic function return failure code.
- #define DPRINTF(arg) printf arg

  Calls printf() only when DEBUG is defined.
- #define DFPRINTF(arg) fprintf arg

  Calls fprintf() only when DEBUG is defined.

Functions

- char * get_errmsg (void)

  Gets the global error message.
- void set_errmsg (const char *buffer)

  Sets the global error message.
- void set_errno_errmsg (const char *buffer)

  Sets the global error message based on errno.
- void mk_errmsg (const char *buffer, char **error_msg)

  Makes an error message.
- void mk_errno_errmsg (const char *buffer, char **error_msg)

  Makes an error message based on errno.
- char * trim_line_ending (char *buffer)

  Trims CR and LF characters from the end of a string.
- char * trim_right (char *buffer)

  Trims trailing whitespace from a string.
- char * trim_left (char *buffer)

  Trims leading whitespace from a string.
- char * trim (char *buffer)

  Trims leading and trailing whitespace from a string.

2.2.1 Detailed Description

Interface to helper functions. Interface to helper functions.

Author

Paul Griffiths

Copyright

2.2.2 Macro Definition Documentation

2.2.2.1 #define DFPRINTF(*arg*) fprintf arg

Calls fprintf() only when DEBUG is defined.

Parameters

arg	The normal parameters to fprintf()

2.2.2.2 #define DPRINTF(*arg*) printf arg

Calls printf() only when DEBUG is defined.

Parameters

arg	The normal parameters to printf()

2.2.2.3 #define ERROR_RETURN (-1)

Generic function return failure code.

Provided for visibility when returning with error.

2.2.2.4 #define FALSE 0

FALSE identifier.

2.2.2.5 #define TRUE 1

TRUE identifier.

2.2.3 Function Documentation

2.2.3.1 char* get_errmsg (void)

Gets the global error message.

Returns

A pointer to the global error message.

2.2.3.2 void mk_errmsg (const char * *buffer,* char ** *error_msg*)

Makes an error message.

This function provides a thread-safe way for a function to set an error message.

Parameters

buffer	A buffer containing the error message.
error_msg	A pointer to a char pointer which will refer to the error message. This should be free()d by the called.

2.2.3.3 void mk_errno_errmsg (const char * *buffer,* char ** *error_msg*)

Makes an error message based on errno.

This function provides a thread-safe way for a function to set an error message, with the usual caveat that errno itself is not threadsafe.

Parameters

buffer	A buffer containing the error message.
error_msg	A pointer to a char pointer which will refer to the error message. This should be free()d by the called.

2.2.3.4 void set_errmsg (const char * *buffer*)

Sets the global error message.

Uses a statically allocated buffer, so this is not thread-safe.

Parameters

buffer	A buffer containing the error message.

2.2.3.5 void set_errno_errmsg (const char * *buffer*)

Sets the global error message based on errno.

Uses a statically allocated buffer, so this is not thread-safe.

Parameters

buffer	A buffer containing the error message.

2.2.3.6 char* trim (char * *buffer*)

Trims leading and trailing whitespace from a string.

Parameters

buffer	The string to trim.

Returns

A pointer to the passed buffer.

2.2.3.7 char* trim_left (char * *buffer*)

Trims leading whitespace from a string.

Parameters

buffer	The string to trim.

Returns

A pointer to the passed buffer.

2.2.3.8 char* trim_line_ending (char * *buffer*)

Trims CR and LF characters from the end of a string.

Parameters

buffer	The string to trim.

Returns

A pointer to the passed buffer.

2.2.3.9 char* trim_right (char * *buffer*)

Trims trailing whitespace from a string.

Parameters

buffer	The string to trim.

Returns

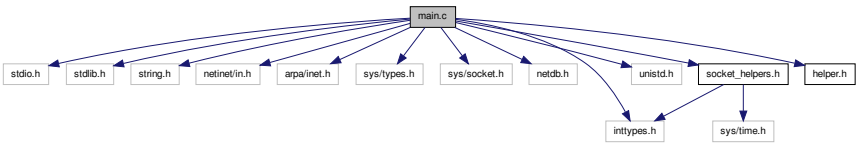
A pointer to the passed buffer.

2.3 main.c File Reference

Main function for echoclient.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <inttypes.h>
#include <unistd.h>
#include "socket_helpers.h"
#include "helper.h"
```

Include dependency graph for main.c:



Macros

- #define MAX_BUFFER_LEN 1024

  Maximum character buffer size.

Functions

- void run_echo_client (const int c_sock)

  Runs the echo client.
- int connect_with_command_line_args (int argc, char **argv)

  Attempts to connect to a service specified in cmdline args.
- int main (int argc, char **argv)

  Main function.

2.3.1 Detailed Description

Main function for echoclient. Main function for echoclient.

Author

Paul Griffiths

Copyright

Copyright 2013 Paul Griffiths. Distributed under the terms of the GNU General Public License. http://www.gnu.org/licenses/

2.3.2 Macro Definition Documentation

2.3.2.1 #define MAX_BUFFER_LEN 1024

Maximum character buffer size.

2.3.3 Function Documentation

2.3.3.1 int connect_with_command_line_args (int *argc*, char ** *argv*)

Attempts to connect to a service specified in cmdline args.

Expects `argc` to be `3`, with `argv[1]` specifying a hostname or IP address, and `argv[2]` specifying a valid port.

Parameters

argc	Number of command line arguments, passed from main()
argv	Command line arguments, passed from main()

Returns

The file descriptor of the connected socket on success, or -1 on failure.

2.3.3.2 int main (int *argc*, char ** *argv*)

Main function.

Connects to an echo server and runs the echo client.

Returns

Exit status.

2.3.3.3 void run_echo_client (const int *c_sock*)

Runs the echo client.

Parameters

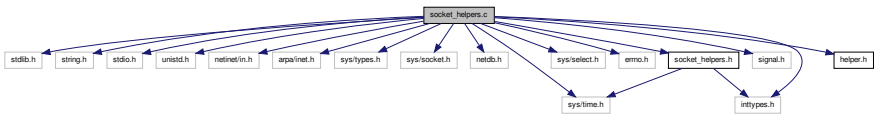
c_sock	File descriptor of the connected socket to use.

2.4 socket_helpers.c File Reference

Implementation of socket helper functions.

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <sys/time.h>
#include <sys/select.h>
#include <errno.h>
#include <inttypes.h>
#include <signal.h>
#include "socket_helpers.h"
#include "helper.h"
```

Include dependency graph for socket_helpers.c:



Macros

- #define MAX_BUFFER_SIZE 1024

  Maximum character buffer size.

Functions

- ssize_t socket_readline (const int socket, char ∗buffer, const size_t max_len)

  Reads a \n terminated line from a socket.
- ssize_t socket_readline_timeout (const int socket, char ∗buffer, const size_t max_len, struct timeval ∗time_-out)

  Reads a \n terminated line from a socket with timeout.
- ssize_t socket_writeline (const int socket, const char ∗buffer, const size_t max_len)

  Writes a line to a socket.
- uint16_t port_from_string (const char ∗port_str)

  Extracts a valid TCP/UDP port from a string.
- int conn_socket_from_string (const char ∗host, const char ∗port)

  Creates a connected sock from a hostname and port.
- void ignore_sigpipe (void)

  Ignores the SIGPIPE signal.

2.4.1 Detailed Description

Implementation of socket helper functions.

Author

Paul Griffiths

Copyright

2.4.2 Macro Definition Documentation

2.4.2.1 #define MAX_BUFFER_SIZE 1024

Maximum character buffer size.

2.4.3 Function Documentation

2.4.3.1 int conn_socket_from_string (const char * *host*, const char * *port*)

Creates a connected sock from a hostname and port.

Parameters

host	A string containing the hostname to which to connect.
port	A string containing the port to which to connect.
error_msg	A pointer to a char pointer which may point to an error message on failure. Set this to NULL to avoid setting an error message.

Returns

The file descriptor of the connected socket on success, or -1 on failure.

2.4.3.2 void ignore_sigpipe (void)

Ignores the SIGPIPE signal.

The write() system call will, when writing to a closed socket, elicit an RST (reset) flag. A second write() system call will trigger a SIGPIPE signal to be raised. The default action of SIGPIPE is to terminate the program, with no error message, which is not desirable. If we want to do anything special when SIGPIPE is triggered, we could set up a handler, but if we don't, then ignoring SIGPIPE is fine, provided our socket functions respond appropriately to the condition (write() will return EPIPE after an ignored SIGPIPE signal).

2.4.3.3 uint16_t port_from_string (const char * *port_str*)

Extracts a valid TCP/UDP port from a string.

Parameters

port_str	The string from which to extract

Returns

The port number on success, or zero if `port_str` does not contain a valid TCP/UDP port (port 0 is reserved and cannot be used).

2.4.3.4 ssize_t socket_readline (const int *socket*, char * *buffer*, const size_t *max_len*)

Reads a \n terminated line from a socket.

The function will not overwrite the buffer, so `max_len` should be the size of the whole buffer, and function will at most write `max_len - 1` characters plus the terminating \0. Any terminating CR or LF characters will be stripped.

Parameters

socket	File description of the socket
buffer	The buffer into which to read
max_len	The maximum number of characters to read, including the terminating \0.
error_msg	A pointer to a char pointer which may point to an error message on failure. Set this to NULL to avoid setting an error message.

Returns

The number of characters read, or -1 on encountering an error.

2.4.3.5 ssize_t socket_readline_timeout (const int *socket*, char * *buffer*, const size_t *max_len*, struct timeval * *time_out*)

Reads a \n terminated line from a socket with timeout.

Behaves the same as socket_readline(), except it will time out if no input is available on the socket after the specified time. Any terminating CR or LF characters will be stripped.

Parameters

socket	File description of the socket
buffer	The buffer into which to read
max_len	The maximum number of characters to read, including the terminating \0.
time_out	A pointer to a `timeval` struct containing the timeout period. Note that some implementations of `select()` may alter this variable, so the calling function should consider it unusable after return. In addition, on such an implementation, the value will specify the cumulative timeout period over the entire read line operation, rather than resetting after reading each character.
error_msg	A pointer to a char pointer which may point to an error message on failure. Set this to NULL to avoid setting an error message.

Returns

The number of characters read, or -1 on encountering an error.

2.4.3.6 ssize_t socket_writeline (const int *socket*, const char * *buffer*, const size_t *max_len*)

Writes a line to a socket.

The function adds a network-standard terminating CRLF, so the provided string should not normally end in any newline characters.

Parameters

socket	File description of the socket
buffer	The buffer from which to write.
max_len	The maximum number of characters to write to the buffer. Due to the addition of CRLF, `max_len + 2` characters may actually be written.
error_msg	A pointer to a char pointer which may point to an error message on failure. Set this to NULL to avoid setting an error message.

Returns

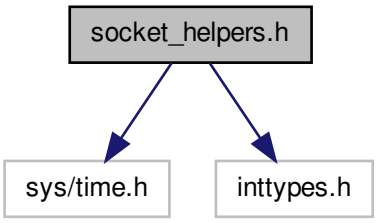
The number of characters written, or -1 on encountering an error.

2.5 socket_helpers.h File Reference

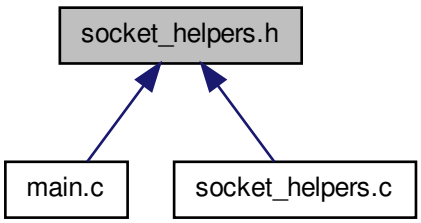
Interface to socket helper functions.

```
#include <sys/time.h>
#include <inttypes.h>
```

Include dependency graph for socket_helpers.h:



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



Functions

- ssize_t socket_readline (const int l_socket, char *buffer, const size_t max_len)

  Reads a \n terminated line from a socket.
- ssize_t socket_readline_timeout (const int l_socket, char *buffer, const size_t max_len, struct timeval *time_out)

  Reads a \n terminated line from a socket with timeout.
- ssize_t socket_writeline (const int l_socket, const char *buffer, const size_t max_len)

  Writes a line to a socket.
- uint16_t port_from_string (const char *port_str)

  Extracts a valid TCP/UDP port from a string.

- int conn_socket_from_string (const char *host, const char *port)

  Creates a connected sock from a hostname and port.
- void ignore_sigpipe (void)

  Ignores the SIGPIPE signal.

2.5.1 Detailed Description

Interface to socket helper functions.

Author

Paul Griffiths

Copyright

2.5.2 Function Documentation

2.5.2.1 int conn_socket_from_string (const char * *host*, const char * *port*)

Creates a connected sock from a hostname and port.

Parameters

host	A string containing the hostname to which to connect.
port	A string containing the port to which to connect.
error_msg	A pointer to a char pointer which may point to an error message on failure. Set this to NULL to avoid setting an error message.

Returns

The file descriptor of the connected socket on success, or -1 on failure.

2.5.2.2 void ignore_sigpipe (void)

Ignores the SIGPIPE signal.

The write() system call will, when writing to a closed socket, elicit an RST (reset) flag. A second write() system call will trigger a SIGPIPE signal to be raised. The default action of SIGPIPE is to terminate the program, with no error message, which is not desirable. If we want to do anything special when SIGPIPE is triggered, we could set up a handler, but if we don't, then ignoring SIGPIPE is fine, provided our socket functions respond appropriately to the condition (write() will return EPIPE after an ignored SIGPIPE signal).

2.5.2.3 uint16_t port_from_string (const char * *port_str*)

Extracts a valid TCP/UDP port from a string.

Parameters

port_str	The string from which to extract

Returns

The port number on success, or zero if `port_str` does not contain a valid TCP/UDP port (port 0 is reserved and cannot be used).

2.5.2.4 ssize_t socket_readline (const int *socket*, char * *buffer*, const size_t *max_len*)

Reads a \n terminated line from a socket.

The function will not overwrite the buffer, so `max_len` should be the size of the whole buffer, and function will at most write `max_len - 1` characters plus the terminating \0. Any terminating CR or LF characters will be stripped.

Parameters

socket	File description of the socket
buffer	The buffer into which to read
max_len	The maximum number of characters to read, including the terminating \0.
error_msg	A pointer to a char pointer which may point to an error message on failure. Set this to NULL to avoid setting an error message.

Returns

The number of characters read, or -1 on encountering an error.

2.5.2.5 ssize_t socket_readline_timeout (const int *socket*, char * *buffer*, const size_t *max_len*, struct timeval * *time_out*)

Reads a \n terminated line from a socket with timeout.

Behaves the same as socket_readline(), except it will time out if no input is available on the socket after the specified time. Any terminating CR or LF characters will be stripped.

Parameters

socket	File description of the socket
buffer	The buffer into which to read
max_len	The maximum number of characters to read, including the terminating \0.
time_out	A pointer to a `timeval` struct containing the timeout period. Note that some implementations of `select()` may alter this variable, so the calling function should consider it unusable after return. In addition, on such an implementation, the value will specify the cumulative timeout period over the entire read line operation, rather than resetting after reading each character.
error_msg	A pointer to a char pointer which may point to an error message on failure. Set this to NULL to avoid setting an error message.

Returns

The number of characters read, or -1 on encountering an error.

2.5.2.6 ssize_t socket_writeline (const int *socket*, const char * *buffer*, const size_t *max_len*)

Writes a line to a socket.

The function adds a network-standard terminating CRLF, so the provided string should not normally end in any newline characters.

Parameters

socket	File description of the socket
buffer	The buffer from which to write.
max_len	The maximum number of characters to write to the buffer. Due to the addition of CRLF, `max_len + 2` characters may actually be written.
error_msg	A pointer to a char pointer which may point to an error message on failure. Set this to NULL to avoid setting an error message.

Returns

The number of characters written, or -1 on encountering an error.

Index