CFNetwork

Generated by Doxygen 1.8.11

Contents

Chapter 1

License

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 CFNetwork Namespace Reference

`CFNetwork` is a collection of utilities that simplifies the process of developing an application that will make use of the network.

Classes

- class Connection

  An object-oriented encapsulation for network connections.
- class InvalidArgument

  The `InvalidArgument` exception can be thrown by methods in the `CFNetwork` namespace when an invalid argument is provided.
- class Socket

  An object-oriented encapsulation for sockets.
- class UnexpectedError

  The `UnexpectedError` exception can be thrown by methods in the `CFNetwork` namespace when an unexpected error is encountered.

Enumerations

- enum ConnectionFlow { ConnectionFlow::Inbound, **Inbound**, ConnectionFlow::Outbound, **Outbound** }

  The `ConnectionFlow` enum is responsible for communicating whether or not a given `Connection` is setup for outbound connectivity or was received inbound from a `Socket` object.
- enum SocketFamily { SocketFamily::IPv4, **IPv4** = AF_INET, SocketFamily::IPv6, **IPv6** = AF_INET6 }

  The `SocketFamily` enum is responsible for communicating which address family that a `Socket` object is using.

Functions

- struct sockaddr_storage parseAddress (const std::string &addr)

  Dynamically parse a `std::string` into a `sockaddr_storage` structure that is capable of being used in socket operations.

Variables

- const int MAX_BYTES = 8192

  The maximum number of bytes that should be contained within all buffers in this namespace's classes.

6.1.1 Detailed Description

`CFNetwork` is a collection of utilities that simplifies the process of developing an application that will make use of the network.

6.1.2 Enumeration Type Documentation

6.1.2.1 enum CFNetwork::ConnectionFlow [strong]

The `ConnectionFlow` enum is responsible for communicating whether or not a given `Connection` is setup for outbound connectivity or was received inbound from a `Socket` object.

Enumerator

Inbound Represents an inbound `Connection`.

Outbound Represents an outbound `Connection`.

6.1.2.2 enum CFNetwork::SocketFamily [strong]

The `SocketFamily` enum is responsible for communicating which address family that a `Socket` object is using.

Enumerator

IPv4 Refers to the `AF_INET` socket family.

IPv6 Refers to the `AF_INET6` socket family.

6.1.3 Function Documentation

6.1.3.1 struct sockaddr_storage CFNetwork::parseAddress (const std::string & *addr*)

Dynamically parse a `std::string` into a `sockaddr_storage` structure that is capable of being used in socket operations.

The `struct sockaddr_storage` can be reinterpret cast into any of the following structures (after checking the `ss_family` attribute):

- `struct sockaddr`
- `struct sockaddr_in`
- `struct sockaddr_in6`

Exceptions

InvalidArgument	on failure or when an unexpected address family is encountered.

Returns

`struct sockaddr_storage` containing the relevant information.

Chapter 7

Class Documentation

7.1 CFNetwork::Connection Class Reference

An object-oriented encapsulation for network connections.

`#include <Connection.hpp>`

Public Member Functions

- Connection (const std::string &addr, int port)

  `Connection` *Constructor (outbound).*
- Connection (const std::string &laddr, const std::string &raddr, int port, int socket)

  `Connection` *Constructor (inbound).*
- ~Connection ()

  `Connection` *Destructor.*
- int getDescriptor () const

  Fetches the file descriptor of the `Connection` *instance.*
- SocketFamily getFamily () const

  Fetches the address family of the `Connection` *instance.*
- ConnectionFlow getFlow () const

  Fetches the flow type of the `Connection` *instance.*
- const std::string & getListen () const

  Fetches the listening address of the `Connection` *instance.*
- int getPort () const

  Fetches the port of the `Connection` *instance.*
- const std::string & getRemote () const

  Fetches the remote address of the `Connection` *instance.*
- std::string read () const

  Attempts to read data from the internal file descriptor.
- bool valid () const

  Determines if the file descriptor is considered valid for read, write, or any other operations.
- void write (std::string data, bool newline=true) const

  Attempts to write the provided data to the internal file descriptor.

Protected Attributes

- SocketFamily family = SocketFamily::IPv4

  Used to describe the socket family type of a `Connection`.
- ConnectionFlow flow = ConnectionFlow::Inbound

  Used to describe the connection flow direction of a `Connection`.
- std::string listen = ""

  Holds the listening address associated with an inbound `Connection`.
- int port = 0

  Holds the listening port for an inbound `Connection` or the outbound port for an outbound `Connection`.
- std::string remote = "0.0.0.0"

  Holds the remote address of a `Connection`.
- int socket = -1

  Holds the file descriptor associated with a `Connection`.

7.1.1 Detailed Description

An object-oriented encapsulation for network connections.

The `Connection` object is responsible for communication between two network endpoints. The object can be setup by accepting an incoming connection on a `Socket` object, or by explicitly making an outgoing connection to a given address and port.

The `Connection` object is not copyable or assignable since it contains resources that do not lend themselves well to duplication.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 CFNetwork::Connection::Connection (const std::string & *addr*, int *port*)

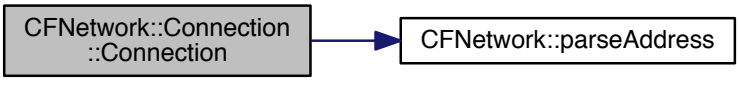
`Connection` Constructor (outbound).

Allows for constructing a `Connection` object to an outbound endpoint.

Parameters

addr	The address of the remote endpoint
port	The port of the remote endpoint

Here is the call graph for this function:



7.1.2.2 CFNetwork::Connection::Connection (const std::string & *laddr,* const std::string & *raddr,* int *port,* int *socket*)

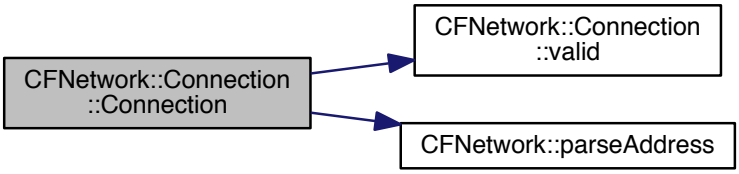
`Connection` Constructor (inbound).

Allows for constructing a `Connection` object from an inbound client file descriptor that was accepted by a listening socket.

Parameters

laddr	The address of the local listening socket
raddr	The address of the remote client
port	The port of the listening socket that received the client
socket	The file descriptor for the client

Here is the call graph for this function:

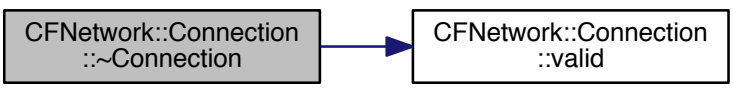


7.1.2.3 CFNetwork::Connection::~Connection ( )

`Connection` Destructor.

Upon destruction of a `Connection` object, close its associated file descriptor (if still valid).

Here is the call graph for this function:



7.1.3 Member Function Documentation

7.1.3.1 int CFNetwork::Connection::getDescriptor ( ) const

Fetches the file descriptor of the `Connection` instance.

The internal file descriptor can be used to perform more advanced actions that this class doesn't accommodate for.

Returns

`int` representing a file descriptor.

7.1.3.2 SocketFamily CFNetwork::Connection::getFamily ( ) const

Fetches the address family of the `Connection` instance.

See also

`SocketFamily` for more information on socket families.

Returns

`SocketFamily` value describing the address family.

7.1.3.3 ConnectionFlow CFNetwork::Connection::getFlow ( ) const

Fetches the flow type of the `Connection` instance.

See also

`ConnectionFlow` for more information on flow types.

Returns

`ConnectionFlow` value describing the flow type.

7.1.3.4 const std::string & CFNetwork::Connection::getListen ( ) const

Fetches the listening address of the `Connection` instance.

This method will produce a `std::string` of an IPv4/IPv6 address only (no IP addresses will be reverse resolved into hostnames).

In the context of an outbound `Connection`, the resulting value will be an empty `std::string`.

Returns

`std::string` containing the listening address.

7.1.3.5 int CFNetwork::Connection::getPort ( ) const

Fetches the port of the `Connection` instance.

If the `Connection` represents an inbound client, the port will be that of the originating `Socket` listening port. For outbound connections, the port will be the original value provided during construction.

Returns

`int` representing the port.

7.1.3.6 const std::string & CFNetwork::Connection::getRemote ( ) const

Fetches the remote address of the `Connection` instance.

This method will produce a `std::string` of an IPv4/IPv6 address only (no IP addresses will be reverse resolved into hostnames).

Returns

`std::string` containing the remote peer's IP address.

7.1.3.7 std::string CFNetwork::Connection::read ( ) const

Attempts to read data from the internal file descriptor.

Performs a blocking read on the internal file descriptor up to `MAX_BYTES - 1`. If there were zero bytes read then the `Connection` will be invalidated due to being reset by the remote peer.

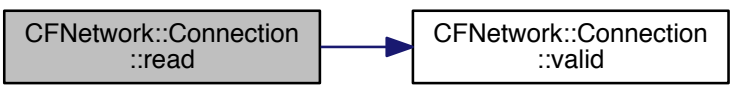
Exceptions

InvalidArgument	if the `Connection` is invalid.
UnexpectedError	if the `Connection` was reset by peer.

Returns

`std::string` containing the data that was read.

Here is the call graph for this function:



7.1.3.8 bool CFNetwork::Connection::valid ( ) const

Determines if the file descriptor is considered valid for read, write, or any other operations.

A file descriptor is considered invalid if a call requesting its flags fails with the return value of `-1` or `errno` is set to `EBADF` (the provided argument is not an open file descriptor). If neither case is satisfied, the file descriptor is considered valid.

See also

`fcntl()` for more information regarding this procedure's test.

Returns

`true` if the file descriptor is valid, `false` otherwise.

7.1.3.9 void CFNetwork::Connection::write (std::string *data,* bool *newline* = `true`) const

Attempts to write the provided data to the internal file descriptor.

An optional newline character is inserted into the provided data by default, however this can be avoided using the appropriate parameter for this method.

Exceptions

InvalidArgument	if the internal file descriptor is considered invalid.

Parameters

data	`std::string` containing the contents to write
newline	Whether or not a newline character should be included

Here is the call graph for this function:

The documentation for this class was generated from the following files:

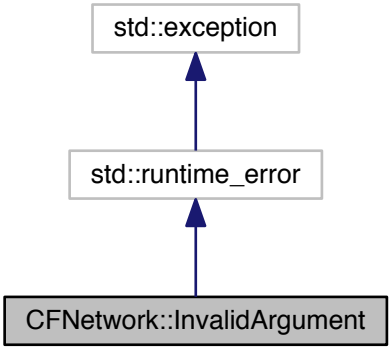
- Connection.hpp
- Connection.cpp

7.2 CFNetwork::InvalidArgument Class Reference

The `InvalidArgument` exception can be thrown by methods in the `CFNetwork` namespace when an invalid argument is provided.

`#include <CFNetwork.hpp>`

Inheritance diagram for CFNetwork::InvalidArgument:



7.2.1 Detailed Description

The `InvalidArgument` exception can be thrown by methods in the `CFNetwork` namespace when an invalid argument is provided.

This is a non-critical exception, and can safely be caught.

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

- CFNetwork.hpp

7.3 CFNetwork::Socket Class Reference

An object-oriented encapsulation for sockets.

`#include <Socket.hpp>`

Public Member Functions

- Socket (const std::string &addr, int port)

  `Socket` *Constructor.*
- ~Socket ()

  `Socket` *Destructor.*
- std::shared_ptr< Connection > accept () const

  Accepts an incoming client and creates a `Connection` *object for it.*
- int getDescriptor () const

  Fetches the file descriptor of the `Socket` *instance.*
- SocketFamily getFamily () const

  Fetches the address family of the `Socket` *instance.*
- const std::string & getHost () const

  Fetches the listening address of the associated `Socket`*.*
- int getPort () const

  Fetches the port of the `Socket` *instance.*
- bool valid () const

  Determines if the file descriptor is considered valid for read, write, or any other operations.

Protected Attributes

- SocketFamily family = SocketFamily::IPv4

  Used to describe the socket family type of a `Socket`*.*
- std::string host = "0.0.0.0"

  Holds the listening address associated with a `Socket`*.*
- int port = 0

  Holds the listening port associated with a `Socket`*.*
- int socket = -1

  Holds the file descriptor associated with a `Socket`*.*

7.3.1 Detailed Description

An object-oriented encapsulation for sockets.

The `Socket` object is responsible for preparations in order to ultimately accept connections on a given listening address and port number.

The `Socket` object is not copyable or assignable since it contains resources that do not lend themselves well to duplication.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 CFNetwork::Socket::Socket (const std::string & *addr*, int *port*)

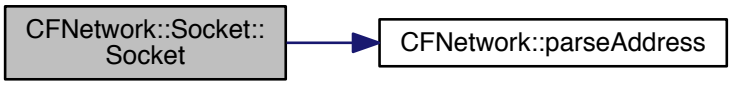
`Socket` Constructor.

Constructs a `Socket` object given a listening address/port and begins listening for clients.

Parameters

addr	`std::string` object containing the listen address
port	`int` containing the port number to listen on

Here is the call graph for this function:



7.3.2.2 CFNetwork::Socket::~Socket ( )

`Socket` Destructor.

Upon destruction of a `Socket` object, close its associated file descriptor.

Here is the call graph for this function:

7.3.3 Member Function Documentation

7.3.3.1 std::shared_ptr< Connection > CFNetwork::Socket::accept ( ) const

Accepts an incoming client and creates a `Connection` object for it.

This method blocks execution until a client is accepted.

Returns

`Connection` object representing the accepted client.

Here is the call graph for this function:

7.3.3.2 int CFNetwork::Socket::getDescriptor ( ) const

Fetches the file descriptor of the `Socket` instance.

The internal file descriptor can be used to perform more advanced actions that this class doesn't accommodate for.

Returns

`int` representing a file descriptor.

7.3.3.3 SocketFamily CFNetwork::Socket::getFamily ( ) const

Fetches the address family of the `Socket` instance.

See also

`SocketFamily` for more information on socket families.

Returns

`SocketFamily` value describing the address family.

7.3.3.4 const std::string & CFNetwork::Socket::getHost ( ) const

Fetches the listening address of the associated `Socket`.

This method can produce a `std::string` of either an IPv4 address or an IPv6 address. This method will not produce hostnames.

Returns

`std::string` of the listening address.

7.3.3.5 int CFNetwork::Socket::getPort ( ) const

Fetches the port of the `Socket` instance.

The port should represent the value that the `Socket` was constructed with.

Returns

`int` representing the port.

7.3.3.6 bool CFNetwork::Socket::valid ( ) const

Determines if the file descriptor is considered valid for read, write, or any other operations.

A file descriptor is considered invalid if a call requesting its flags fails with the return value of `-1` or `errno` is set to `EBADF` (the provided argument is not an open file descriptor). If neither case is satisfied, the file descriptor is considered valid.

See also

`fcntl()` for more information regarding this procedure's test.

Returns

`true` if the file descriptor is valid, `false` otherwise.

The documentation for this class was generated from the following files:

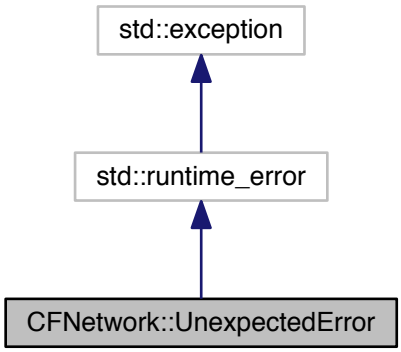
- Socket.hpp
- Socket.cpp

7.4 CFNetwork::UnexpectedError Class Reference

The `UnexpectedError` exception can be thrown by methods in the `CFNetwork` namespace when an unexpected error is encountered.

`#include <CFNetwork.hpp>`

Inheritance diagram for CFNetwork::UnexpectedError:



7.4.1 Detailed Description

The `UnexpectedError` exception can be thrown by methods in the `CFNetwork` namespace when an unexpected error is encountered.

This is a non-critical exception, and can safely be caught.

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

- CFNetwork.hpp

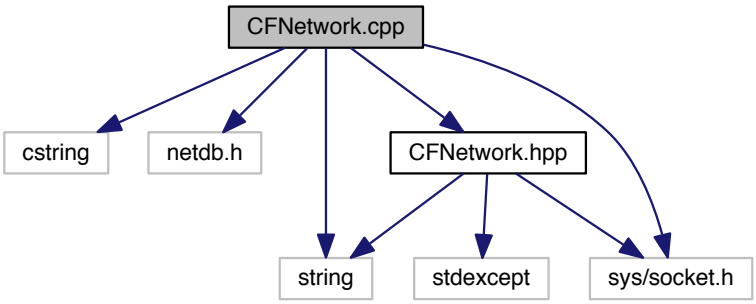
Chapter 8

File Documentation

8.1 CFNetwork.cpp File Reference

```
#include <cstring>
#include <netdb.h>
#include <string>
#include <sys/socket.h>
#include "CFNetwork.hpp"
```

Include dependency graph for CFNetwork.cpp:



Namespaces

- CFNetwork

  `CFNetwork` is a collection of utilities that simplifies the process of developing an application that will make use of the network.

Functions

- struct sockaddr_storage CFNetwork::parseAddress (const std::string &addr)

  Dynamically parse a `std::string` into a `sockaddr_storage` structure that is capable of being used in socket operations.

8.1.1 Detailed Description

Copyright

Copyright 2016 Clay Freeman. All rights reserved

License:

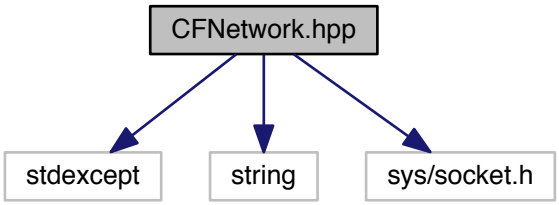
GNU Lesser General Public License v3 (LGPL-3.0)

Implementation source for the `CFNetwork` helper functions.

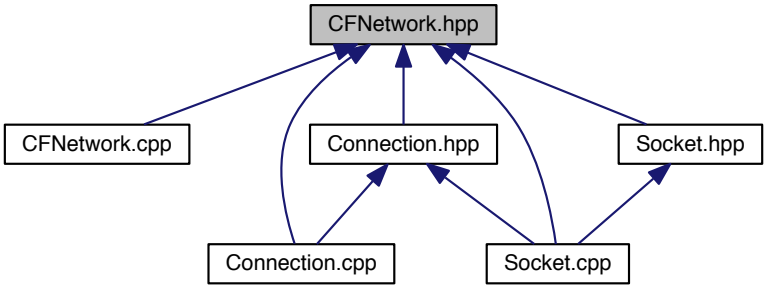
8.2 CFNetwork.hpp File Reference

```
#include <stdexcept>
#include <string>
#include <sys/socket.h>
```

Include dependency graph for CFNetwork.hpp:



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



Classes

- class CFNetwork::InvalidArgument

  The `InvalidArgument` exception can be thrown by methods in the `CFNetwork` namespace when an invalid argument is provided.
- class CFNetwork::UnexpectedError

  The `UnexpectedError` exception can be thrown by methods in the `CFNetwork` namespace when an unexpected error is encountered.

Namespaces

- CFNetwork

  `CFNetwork` is a collection of utilities that simplifies the process of developing an application that will make use of the network.

Enumerations

- enum CFNetwork::ConnectionFlow { CFNetwork::ConnectionFlow::Inbound, **Inbound**, CFNetwork::↩ ConnectionFlow::Outbound, **Outbound** }

  The `ConnectionFlow` enum is responsible for communicating whether or not a given `Connection` is setup for outbound connectivity or was received inbound from a `Socket` object.
- enum CFNetwork::SocketFamily { CFNetwork::SocketFamily::IPv4, **IPv4** = AF_INET, CFNetwork::Socket↩ Family::IPv6, **IPv6** = AF_INET6 }

  The `SocketFamily` enum is responsible for communicating which address family that a `Socket` object is using.

Functions

- struct sockaddr_storage CFNetwork::parseAddress (const std::string &addr)

  Dynamically parse a `std::string` into a `sockaddr_storage` structure that is capable of being used in socket operations.

Variables

- const int CFNetwork::MAX_BYTES = 8192

  The maximum number of bytes that should be contained within all buffers in this namespace's classes.

8.2.1 Detailed Description

Copyright

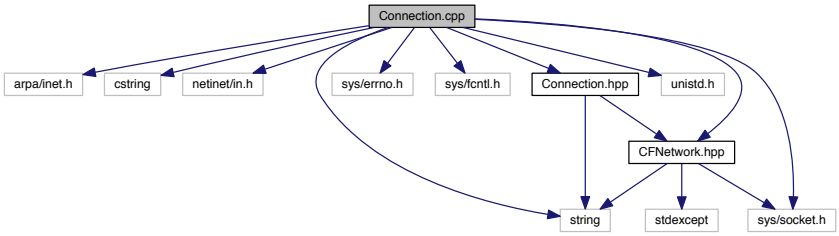
License:

Forward declaration of the `CFNetwork` namespace and related items.

8.3 Connection.cpp File Reference

```
#include <arpa/inet.h>
#include <cstring>
#include <netinet/in.h>
#include <string>
#include <sys/errno.h>
#include <sys/fcntl.h>
#include <sys/socket.h>
#include <unistd.h>
#include "CFNetwork.hpp"
#include "Connection.hpp"
```

Include dependency graph for Connection.cpp:



Namespaces

- CFNetwork

  CFNetwork is a collection of utilities that simplifies the process of developing an application that will make use of the network.

8.3.1 Detailed Description

Copyright

Copyright 2016 Clay Freeman. All rights reserved

License:

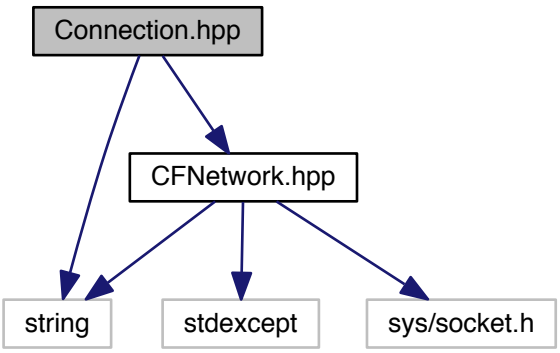
GNU Lesser General Public License v3 (LGPL-3.0)

Implementation source for the `Connection` object.

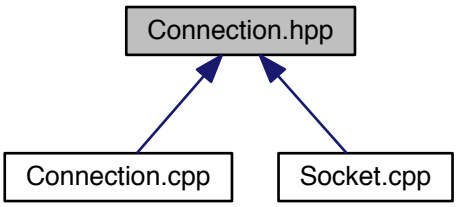
8.4 Connection.hpp File Reference

```
#include <string>
#include "CFNetwork.hpp"
```

Include dependency graph for Connection.hpp:



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



Classes

- class CFNetwork::Connection

  An object-oriented encapsulation for network connections.

Namespaces

- CFNetwork

  CFNetwork is a collection of utilities that simplifies the process of developing an application that will make use of the network.

8.4.1 Detailed Description

Copyright

Copyright 2016 Clay Freeman. All rights reserved

License:

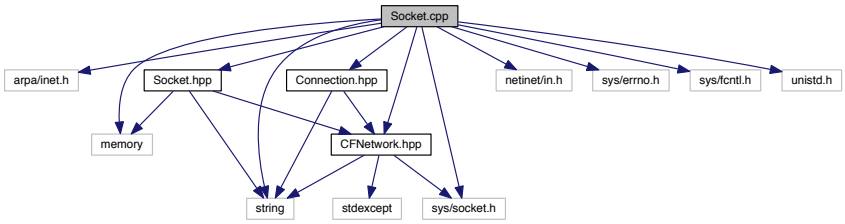
GNU Lesser General Public License v3 (LGPL-3.0)

Implementation reference for the `Connection` object.

8.5 Socket.cpp File Reference

```
#include <arpa/inet.h>
#include <memory>
#include <netinet/in.h>
#include <string>
#include <sys/errno.h>
#include <sys/fcntl.h>
#include <sys/socket.h>
#include <unistd.h>
#include "CFNetwork.hpp"
#include "Connection.hpp"
#include "Socket.hpp"
```

Include dependency graph for Socket.cpp:



Namespaces

- CFNetwork

  `CFNetwork` is a collection of utilities that simplifies the process of developing an application that will make use of the network.

8.5.1 Detailed Description

Copyright

Copyright 2016 Clay Freeman. All rights reserved

License:

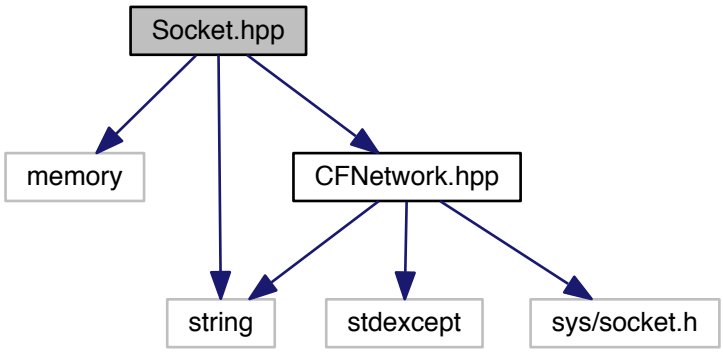
GNU Lesser General Public License v3 (LGPL-3.0)

Implementation source for the `Socket` object.

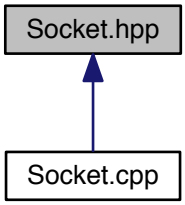
8.6 Socket.hpp File Reference

```
#include <memory>
#include <string>
#include "CFNetwork.hpp"
```

Include dependency graph for Socket.hpp:



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



Classes

- class CFNetwork::Socket

  An object-oriented encapsulation for sockets.

Namespaces

- CFNetwork

  `CFNetwork` is a collection of utilities that simplifies the process of developing an application that will make use of the network.

8.6.1 Detailed Description

Implementation reference for the `Socket` object.

Index