# C++ Easy Socket

## Official Documentation

*Author(s)*

**Cryptogenic (Specter)**

# General Information and Purpose

The “C++ Easy Socket” project was developed to make networking projects easier by hiding all of the messy difficult stuff. It provides a higher-level of abstraction without compromising portability in regards to libraries. Though it was built on a Linux system, Windows support has been added. In the recent update 1.1, server-sided functionality was also added.

# Motivation

This class was originally implemented for an IRC client I was creating in C++. After seeing all the code and messing around it took to use sockets, I decided to write my own class. After I implemented the program it occurred to me this might be of use to others. It’s by no means perfect, but improvements will be made over time.

# Latest Version

The latest version for this project is 1.1. Windows support as well as server-sided functionality has been added. For more information on the 1.1 update, check the changelog.txt in /docs.

# Requirements

This project does not require any external libraries, all you need is gcc/g++ or any other compiler to compile the source code.

# License

# Example Usage - Client

To use C++ Easy Socket, it's relatively simple. One just needs to create an instance of the object, set the socket type to either TYPE_CLIENT or TYPE_SERVER, then initialize it with the init() function, using the host and port to use for parameters. Then the socket can be read or written to via the readSocket() and writeSocket() functions. It is recommended to use checkStatus() to ensure the connection is established before attempting to read or write.

```
int main()
{
 Socket sock;

 std::string host = "0.0.0.0";
 std::string port = "13337";
 sock.setSocketType(TYPE_CLIENT);
 sock.init(host, port);

 if(sock.checkStatus())
 {
  std::cout << "The connection was established successfully, terminating...\n";
  return 0;
 }
 else
 {
  std::cout << "The connection could not be established, terminating...\n";
  return -1;
 }
```

# Example Usage - Server

For server-sided usage, there are a few minor differences. Firstly, the socket type must be set to TYPE_SERVER via Socket::setSocketType() and after calling Socket::init(), Socket::listenSocket() must be called. All the binding, client acceptance, and handling occurs in Socket.cpp.

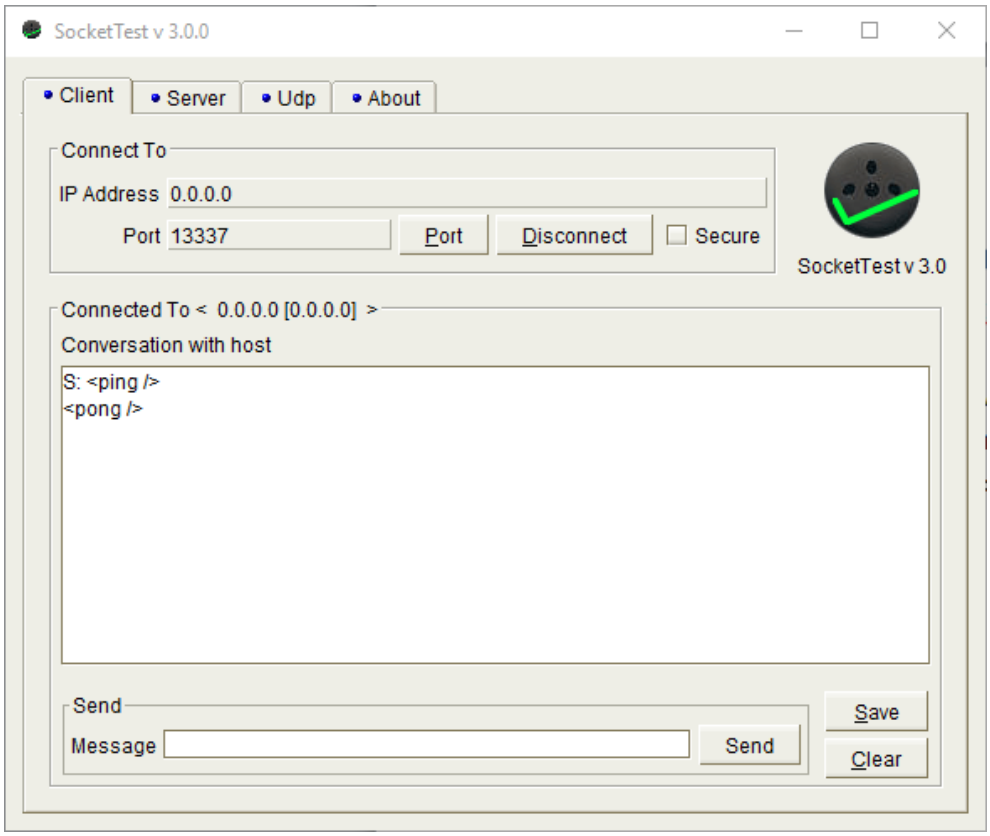
In Socket.hpp, some constants can be found that may be edited to your liking, including maximum clients in a queue, as well as how many bytes that can be received. Handling should be customized to your needs at the top of Socket.cpp, in function Socket::handleClient().

```
int main()
{
  Socket sock;

  std::string host = "0.0.0.0";
  std::string port = "13337";

  sock.setSocketType(TYPE_SERVER);

  sock.init(host, port);

  sock.listen();
}
```

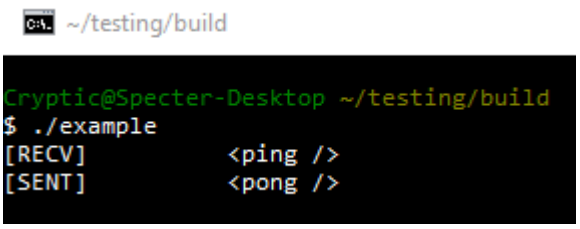
# Example Usage – Server (Demonstration)

For testing, “SocketTest” by Akshathkumar Shetty was used as a client. To show the connection between the server using C++ Easy Socket and the client was established, a “ping-pong” system was setup. The client will send a ping packet, and the server should respond with a pong packet. The ping packet by default is <ping />, and the response packet <pong />. If setup properly, your client should receive this response packet if <ping /> is sent.

Client Screenshot




Server Console Screenshot



# Socket Class Outline

Class: Socket

Description: Allows a basic client socket to be opened and contains method for data to be sent and received.

*Public Methods:*

Socket() – Class constructor, sets status variable to default integer of 1

~Socket() – Class destructor, cleans memory and unsets the socket object

Bool checkStatus() – Returns true if the connection is established

Void setSocketType() – Sets the socket type (TYPE_CLIENT or TYPE_SERVER)

Void init() – Initializes the socket with the given host and port

Void writeSocket – Writes the specified data (argument) to the socket

Void readSocket – Reads and returns data from the socket

Void handleClient() – Handles clients as they're accepted

Void listenSocket() – Binds to the host address, listens for clients and handles them

*Private Methods:*

Void throwError() – Prints given error (argument) and sets false status

Void throwWarning() – Prints given warning, but does not set false status

*Private Properties:*

Int status – Contains the active status code of the socket

Int sockfd – Contains the reference to the socket object as an integer

Int newsockfd – Contains the reference to a new socket for accepting clients

Int portNumber – Contains the port number for the socket as an integer

Int n – Contains the status code of the socket functions (eg. read, write)

Int pid – Contains program PID for forking

Int clientLen – Contains size of cli_addr structure

Bool clientSock – Contains true or false bool if socket is for client connection

Char buffer – Contains the data for reading

Struct sockadd_in – Contains the family, port, host, zero for socket

Struct serv_addr – Also contains family, port, host.

Struct hostent – Contains host name, aliases, address type, and length.

# Additional Information

For additional information on the three private structures, see the TutorialsPoint reference for Unix sockets at http://www.tutorialspoint.com/unix_sockets/socket_structures.htm