

Forum:[Hack Forums](#)

Thread: [Winsock Wrapper](#)

NullSystem - December 1, 2019, 2:04 pm

Socket.c

```
Code #include <WinSock2.h>
```

```
#include <ws2tcpip.h>
```

```
#include <stdio.h>
```

```
#include "Sockets.h"
```

```
// Add thread for accept()ing connections for CreateSockServer
```

```
// Add IPv6 shit
```

```
// Add UDP functionality
```

```
// Make example implementation by making functional HTTP GET and POST requests
```

```
BOOL InitializeWinsock()
```

```
{
    WSADATA WSA;

    if (WSAStartup(MAKEWORD(2, 2), &WSA) != 0)
    {
#ifdef _DEBUG
        printf("[ERROR] WSAStartup: %d\n", WSAGetLastError());
#endif
        return FALSE;
    }

    return TRUE;
}
```

```
BOOL CreateSockClient(SOCK_INFO *Sock, char *Server, char *Port)
```

```
{
    struct addrinfo Hints, *Result, *Pointer;
    int Status;
    int Mode = 1;

    // zero out addrinfo structure
    RtlSecureZeroMemory(&Hints, sizeof(Hints));

    // fill addrinfo structure
    Hints.ai_family = AF_INET;
    Hints.ai_protocol = Sock->Protocol;

    if (Sock->Protocol == IPPROTO_TCP)
        Hints.ai_socktype = SOCK_STREAM;
    else if (Sock->Protocol == IPPROTO_UDP)
        Hints.ai_socktype = SOCK_DGRAM;
```

```

// resolve DNS to address
Status = getaddrinfo(Server, Port, &Hints, &Result);
if (Status != 0)
{
#ifdef _DEBUG
    printf("[ERROR] getaddrinfo: %d\n", WSAGetLastError());
#endif
    return FALSE;
}

for (Pointer = Result; Pointer != NULL; Pointer = Pointer->ai_next)
{
    Sock->Socket = socket(Pointer->ai_family, Pointer->ai_socktype, Pointer->ai_protocol);
    if (Sock->Socket == INVALID_SOCKET)
    {
#ifdef _DEBUG
        printf("[ERROR] socket: %d\n", WSAGetLastError());
#endif
        WSACleanup();
        return FALSE;
    }

    Status = connect(Sock->Socket, Pointer->ai_addr, Pointer->ai_addrlen);
    if (Status == SOCKET_ERROR)
    {
#ifdef _DEBUG
        printf("[ERROR] connect: %d\n", WSAGetLastError());
#endif
        closesocket(Sock->Socket);
        Sock->Socket = INVALID_SOCKET;
        continue;
    }

    break;
}

freeaddrinfo(Result);

if (Sock->Socket == INVALID_SOCKET)
{
#ifdef _DEBUG
    printf("[ERROR] CreateSock: connection to %s:%s failed\n", Server, Port);
#endif
    WSACleanup();
    return FALSE;
}

// set socket non-blocking
if (SetSockBlocking(Sock->Socket, TRUE) == FALSE)
{
#ifdef _DEBUG
    printf("[ERROR] Failed to set socket to non-blocking\n");
#endif
}

```

```

        closesocket(Sock->Socket);
        return FALSE;
    }

    Sock->Connected = TRUE;

#ifdef _DEBUG
    printf("Initiated connection to %s:%s\n", Server, Port);
#endif

    return TRUE;
}

BOOL CreateSockServer(SOCK_INFO *Sock, char *Port)
{
    struct addrinfo Hints, *Result;
    int Status;
    int Mode = 1;

    // zero out addrinfo structure
    RtlSecureZeroMemory(&Hints, sizeof(Hints));

    // fill addrinfo structure
    Hints.ai_family = AF_INET;
    Hints.ai_flags = AI_PASSIVE;
    Hints.ai_protocol = Sock->Protocol;

    if (Sock->Protocol == IPPROTO_TCP)
        Hints.ai_socktype = SOCK_STREAM;
    else if (Sock->Protocol == IPPROTO_UDP)
        Hints.ai_socktype = SOCK_DGRAM;

    // resolve localhost to address
    Status = getaddrinfo(NULL, Port, &Hints, &Result);
    if (Status != 0)
    {
#ifdef _DEBUG
        printf("[ERROR] getaddrinfo: %d\n", WSAGetLastError());
#endif
        return FALSE;
    }

    Sock->Server = socket(Result->ai_family, Result->ai_socktype, Result->ai_protocol);
    if (Sock->Server == INVALID_SOCKET)
    {
#ifdef _DEBUG
        printf("[ERROR] socket: %d\n", WSAGetLastError());
#endif
        freeaddrinfo(Result);
        WSACleanup();
        return FALSE;
    }
}

```

```

// bind to localhost
Status = bind(Sock->Server, Result->ai_addr, Result->ai_addrlen);
if (Status == SOCKET_ERROR)
{
#ifdef _DEBUG
    printf("[ERROR] bind: %d\n", WSAGetLastError());
#endif
    freeaddrinfo(Result);
    DestroySock(Sock);
    return FALSE;
}

freeaddrinfo(Result);

// start listening
Status = listen(Sock->Server, SOMAXCONN);
if (Status == SOCKET_ERROR)
{
#ifdef _DEBUG
    printf("[ERROR] listen: %d\n", WSAGetLastError());
#endif
    DestroySock(Sock);
    return FALSE;
}

Sock->Listening = TRUE;

// set socket non-blocking
if (SetSockBlocking(Sock->Socket, TRUE) == FALSE)
{
#ifdef _DEBUG
    printf("[ERROR] Failed to set socket to non-blocking\n");
#endif
    closesocket(Sock->Socket);
    return FALSE;
}

#ifdef _DEBUG
    printf("Initiated listening server on port %s\n", Port);
#endif

return TRUE;
}

void DestroySock(SOCK_INFO *Sock)
{
    if (Sock->Socket != INVALID_SOCKET)
    {
        Sock->Connected = FALSE;
        closesocket(Sock->Socket);
        WSACleanup();
    }
}

```

```

if (Sock->Server != INVALID_SOCKET)
{
    Sock->Listening = FALSE;
    closesocket(Sock->Server);
    closesocket(Sock->Client);
    WSACleanup();
}

RtlSecureZeroMemory(Sock->Data.Buffer, sizeof(Sock->Data.Buffer));
}

```

```

BOOL SendSockData(SOCK_INFO *Sock, char *Format, ...)

```

```

{
    char Buffer[512];
    int Written;
    int Status;
    va_list va;

    va_start(va, Format);

    // create argument list from function parameters
    Written = vsprintf(Buffer, Format, va);
    if (Written <= 0)
        return FALSE;

    Buffer[Written] = '\0';

    va_end(va);

    // send constructed data
    Status = send(Sock->Socket, Buffer, strlen(Buffer), 0);
    if (Status == SOCKET_ERROR)
    {
#ifdef _DEBUG
        printf("[ERROR] send: %d\n", WSAGetLastError());
#endif
        DestroySock(Sock);
        return FALSE;
    }

#ifdef _DEBUG
    printf("\nSent %d bytes of data: \n%s\n", Status, Buffer);
#endif

    return TRUE;
}

```

```

BOOL ReceiveSockData(SOCK_INFO *Sock)

```

```

{
    fd_set fds;
    int Status;

    FD_ZERO(&fds);

```

```

    FD_SET(Sock->Socket, &fds);

    // check if theres any waiting data on socket
    if (select(0, &fds, NULL, NULL, Sock->Timeval) <= 0)
    {
#ifdef _DEBUG
        printf("[ERROR] select: %d\n", WSAGetLastError());
#endif
        DestroySock(Sock);
        return FALSE;
    }

    Status = recv(Sock->Socket, Sock->Data.Buffer, sizeof(Sock->Data.Buffer), 0);
    if (Status == 0)
    {
#ifdef _DEBUG
        printf("ReceiveSockData: connection closed\n");
#endif
        DestroySock(Sock);
        return FALSE;
    }
    else if (Status < 0)
    {
#ifdef _DEBUG
        printf("[ERROR] ReceiveSockData: %d\n", WSAGetLastError());
#endif
        DestroySock(Sock);
        return FALSE;
    }

    Sock->Data.Buffer[Status] = '\0';

#ifdef _DEBUG
    printf("\nReceived %d bytes of data: \n%s\n", Status, Sock->Data.Buffer);
#endif

    return TRUE;
}

BOOL SetSockBlocking(SOCKET Socket, BOOL Blocking)
{
    unsigned long mode = Blocking ? 0 : 1;

    if (Socket < 0)
        return FALSE;

    return (ioctlsocket(Socket, FIONBIO, &mode) == 0) ? TRUE : FALSE;
}

Socket.h
Code #ifndef __Socket_H__
#define __Socket_H__

#pragma comment(lib, "ws2_32.lib")

```

```
typedef struct _SOCK_DATA_  
{  
    char Buffer[1024];  
} SOCK_DATA;
```

```
typedef struct  
{  
    SOCK_DATA Data;  
    SOCKET Server;  
    SOCKET Client;  
    SOCKET Socket;  
    int Protocol;  
    BOOL Connected;  
    BOOL Listening;  
    TIMEVAL *Timeval;  
} SOCK_INFO;
```

```
BOOL InitializeWinsock();  
BOOL CreateSockClient(SOCK_INFO *Sock, char *Server, char *Port);  
BOOL CreateSockServer(SOCK_INFO *Sock, char *Port);  
void DestroySock(SOCK_INFO *Sock);  
BOOL SendSockData(SOCK_INFO *Sock, char *Format, ...);  
BOOL ReceiveSockData(SOCK_INFO *Sock);  
BOOL SetSockBlocking(SOCKET Socket, BOOL blocking);
```

```
#endif __Socket_H__
```

HFDone - December 2, 2019, 7:43 am

you may get more ideas [GitHub - AHXR/ahxrwinsocket: Light C++ Winsock Wrapper Library](#)