

# Intrafoundation.UDPClient.2

ATL COM Component  
FOR **ASP** and **ColdFusion**

Freeware with C++ source code



by **Lewis Sellers** (aka TommyRaven)  
Intrafoundation Software

**v2.1, November 5th 2004**  
<http://www.intrafoundation.com>  
[products@intrafoundation.com](mailto:products@intrafoundation.com)

Yes, I'm probably for hire. Need something? I may or may not have the time, but you can ask.

**T**HIS UDPCLIENT COM OBJECT ALLOWS YOU TO talk to anything on the Internet that uses the UDP communications protocol.

UDP is part of the TCP/IP suite which powers the internet and is the "little brother" of the more famous TCP. UDP is an "unreliable connection-less" packet protocol. This means you can send small "post-it note" like messages very, very quickly between machines but there is no guarantee that the message will ever arrive. Compare this to TCP, which is slower to startup, but any data you send is automatically error-corrected and resent if lost along the way. TCP also lets you send a stream of essentially unlimited size, where UDP is limited to about 65,507 bytes at the most.

A COM object (for those of you who are using this software who are not web developers but hardcore gamers) is a type of .DLL. This particular type is the kind that makes up most of the code that powers the Windows Operating System itself. They are essentially self-contained ("encapsulated, object-oriented" in tech-speak) programs. Many of the applications you may use (the .EXE files) will make use of COM objects that it brings with it or that are included with the Windows Operating System.

Primarily the only things of real importance these days that use UDP are:

1. Game servers(!) such as: **Quake**, **Half-Life**, **Unreal**, **Wheel of Time**, Baldur's Gate, Diablo, Star Trek : Elite Forces, et al (well, perhaps not ALL, but most the ones that anyone cares about).
2. Hardware-based routers (i.e., CISCOs).
3. Packet messagers such as ICQ.

For that reason most of the example code you will find here deals with the interrogation of on-line game servers. (Did I hear a small cheer go up?)



## ARTICLES

### Getting Live Half-Life Stats From ColdFusion v1.3 April 2003

This is a fairly long, detailed article on everything you need to know about talking to Half-Life from ColdFusion. The bulk of the article also is very useful for those using ASP as well.

## EXAMPLE SCRIPTS

### ASP

daytime (RFC 867 daytime/udp 13)  
time (RFC-868 time/udp 37)

Quake 3  
Quake 2  
Return to Castle Wolfenstein  
Star Trek: Elite Forces  
Half-Life  
Unreal  
Unreal Tournament  
America's Army

(sendpacket)  
(properties)

(Third-party code)  
**MOHAA** (Medal of Honor)

(The following require **Intrafoundation. NetworkDiscovery. 1** to be installed.)  
**Quake 3 LAN Auto-detect**  
**Quake 2 LAN Auto-detect**

### COLDFUSION

Game Scry :->

Starsiege: Tribes  
Return to Castle Wolfenstein  
Quake 3  
Star Trek: Elite Forces  
Quake II  
Unreal Tournament  
Half-Life  
Unreal

Understand that this COM object provides only the elemental, base functionality you need to talk to these UDP servers. As for what you're supposed to say to them and what do the messages they send back mean, well... that's a subject for several heated on-line forum discussions and a half-dozen small project groups to ferret out. I've included example scripts written in **ColdFusion** meta-language (i.e, the files with those .cfm extensions) that show exactly how to talk to all the games I know how to talk to. You'll have to figure the rest out yourselves. Start by reading a games' official (or unofficial) protocol documents, if any... the rest is up to people willing to pester developers for the information or hackers with packet sniffers and a little imagination.

## REQUIREMENTS / SERVER PLATFORMS

This COM object was designed on and for Windows 2000 Professional and Windows XP Professional. It has not been tested under Windows 95, 98, ME or NT 3.51/4. Nevertheless, it may work under some of them.

Additionally you most typically will need either **Macromedia (previously known as Allaire) ColdFusion** or **Microsoft ASP** installed. If you don't know what either of these are... then you're going to need to make friends with a geeky web developer somewhere to help you.

You should be able to use the COM from:

- Macromedia **ColdFusion**
- Microsoft **ASP**
- **PHP4**
- Any other engine or compiler that can use COM objects.

## INSTALLATION

The COM object itself is the file called `UDPClientcom.dll`. To install it, at the command prompt type: `regsvr32 UDPClientcom.dll`. You may first wish to copy it to your project folder or some other folder you make just for it.

The precompiled file `UDPClientcom.dll` is root folder of the archive.

There are `install.bat` and `uninstall.bat` files included that you can simply click from Windows which will do the same.

(Unfinished)  
**Wheel of Time\***  
**Baldurs Gate 1\***

**ICO**

**(instance)**  
**(port test)**  
**(port scan)**  
**(properties)**

(Little FORM script by SteelValor....)

GAMESCRY ;->
Starsiege: Tribes
Half-Life
Quake III
Star Trek: Elite Force
Unreal Tournay

## PHP

NOTE: PHP has built in UDP functions already.

## JAVA

(none currently)

**Send in your examples.**

## UNINSTALLATION

At the command prompt type: `regsvr32 /u UDPClientcom.dll`. Or see above.

## THE EXAMPLE SCRIPTS

There are several example scripts included with this COM object. Some are only partially done. None are optimised for the protocols they use. Additionally, some are written by the author(s) of this software, while others have been developed by it's users.

For most of the examples you'll need to have the appropriate SERVER software installed on your local computer (i.e., either ASP or ColdFusion).

You might want to look at the source of the scripts before you use them.

You do not need to set this up for the COM object -- only to use some of the localhost examples. And you should be connected to the internet for some of them as they talk to other machines out there.

If you are not connected to the Internet then the tag may sit there for several several seconds waiting for a name server to answer it.

*Send in your own sample scripts.*

## FIREWALLS

Firewalls. It's almost impossible these days to safely be connected to your local area network, let alone the Internet, without using a firewall (and a half dozen other anti-viral, anti-trojan, anti-spyware, etc programs). We bring this up because, most likely, you're using this software via a web server such as Microsoft IIS or Apache. This means this COM object will be talking to the network through THEM in some fashion. You won't be getting a simple software message stating that say, Intrafoundation.TCPClient or Intrafoundation.UDPClient, etc are wishing access to the network.

You will in fact most likely get this message:

```
(10061) (Connection refused. No connection could be made because the target machine actively refused it. This usually results from trying to connect to a service that is inactive on the foreign host - i.e. one with no server application running.)
```

So no, you're going to have to specifically set up your firewall to allow network communication for this software. With IIS in the default configuration ("Medium" Application Protection) this would theoretically mean enabling the infamous `dllhost.dll`, known as "COM Surrogate". We say it's infamous because it's been used in a variety of ways by numerous viruses, worms, trojans and the like. So much so that most software firewalls that have application protection will disable it by default.

If you set IIS Application Protection to "Low" it uses a process other than dllhost.dll, but low isn't generally a recommended setting. When set to "High", against it uses a different process to run scripts through, namely Internet Information Services.

For a more detailed explanation of this topic, refer to your particular Web Server and Firewall manuals for exact information on this subject.

## COPYRIGHT / TERMS OF USE

This software is Copyright (c) 2000, 2001, 2003, 2004 by Lewis A. Sellers. It is not public domain, nor is it GPL'ed, but it is very close. As long as you do not modify any files in the archive, nor add to them, nor delete any of them, and do not charge for access to said archive you may redistribute the archive as you like.

You may use this software as-is with any software you wish, so long as said archive is included unmodified with proper credits and link to it's homesite is included also (<http://www.intrafoundation.com/udpclient.asp>).

You may modify and use the source code as you like -- with the understanding that if you do, you still have to include the original, unaltered archive as well as the aforementioned credits and link.

You may use this software in commerical applications, whether closed-source or open-source so long as the aforementioned unaltered archive is included with the application and the aforementioned creditation and hyperlink are included.

If this software is used in a released project or included in a publication you are to make reasonable efforts to contact the author and notify them as to such. The author of this software has a primary email address of: [products@intrafoundation.com](mailto:products@intrafoundation.com).

You use this software at your own risk.

## COMMAND OVERVIEW : MAKING IT WORK

All right then, how the devil do you use it?

Well, assuming you installed/registered the COM object and on the same machine you've got ColdFusion or ASP installed then all you need to do is... open your browser to this web page and click on any of the examples to see it in living action. (On the server... not as a file. That is with "HTTP://" prepended not "FILE://". But I really hope you knew that already.)

NAME	METHODS	
	IN	OUT
Open	string "host", string "port"	boolean "connected"
Close	string "host", string "port"	boolean "connected"
Send	string "text"	
Recv		string
SendCSV	string "csv"	
RecvCSV	string "csv"	
SendPacket		
RecvPacket		
addpacketnumber	type, number	
addpacketchar	character	

web pages -- read on.

Basically all there is to using this COM object is you open a UDP connect and then send and receive whatever UDP packets of data are required to get the information you require. Then you close the connection.

There are several other useful properties you can look at or set that can be used for error handling and tweaking overall performance.

One thing that can not be stressed enough about this COM object though is that it understands nothing about the higher-level application protocols. The functions are generic. They just transfer the data, not interpret it.

How big a block of data can this transfer? 65507 bytes. Maximum. This is inherent to the UDP protocol itself.

**DEALING WITH BINARY DATA**

A lot of things you can talk to with UDP use plain text because that is, as demonstrated by the telnet service, a fairly easy way to communicate. However, some servers like the game Half-Life sends data in a pure C-Struct like binary format. To deal with this several new functions were added as of 1.2. They are:

- SendPacket
- RecvPacket
- packetposition
- packetlength
- packetreposition
- packetEOF
- packetclear
- addpacketnumber
- addpacketchar
- addpacketstring
- packetnumber
- packetchar
- packetstring

addpacketstring	string	
addpacketipport	string "ip", string "port"	
packetnumber	string "type"	number
packetstring		string
packetcharacter		character
packetipport		string "ip:port"
packetEOF	boolean	
is_connected	boolean	
packetclear		
packetreposition	integer (0-65506)	
ClearLog		

PROPERTIES (PUT)	
timeout	floating-point-number "seconds"

PROPERTIES (GET, only when connected)	
timeout	floating-point-number "seconds"
localaddress	string + : + number
remoteaddress	string + : + number
bytessent	number "bytes"
bytesreceived	number "bytes"
kbpsending	floating-point-number
kbpsreceiving	floating-point-number
senttimeout	floating-point-number
receivedtimeout	floating-point-number
packetposition	integer (0-65506)
packetlength	integer (0-65536)
ping	integer "milliseconds"

PROPERTIES (GET, anytime)	
thread	number
version	string
copyright	string
description	string
instance	number
instances	number
Log	string
sendlength	number "bytes"
recvlength	number "bytes"
messagelength	number "bytes"
buffersize	number "bytes"

Simply put, a buffer of 65507 bytes is created and these functions allow you to read off the data from the buffer or write new data to it. If you've had any programming classes at all the usage of these functions should be fairly obvious.

## METHODS

### Open

**returns** boolean "connected"

**parameters** string "address", number "port"

Opens a UDP connection to the server on the specified host and port. If this fails the method returns a "false". If it succeeds it returns a "true".

The address can be a dotted ip address or a registered domain name or subdomain. The port is typically a port number, but it will also accept some well known port names such as "whois".

Opening a connection resets the running error log.

### Close

**returns**

**parameters**

Closes the current socket connection. If the socket is already closed (by the remote server for example) you should still perform a close on your local machine as well. If, you attempt to open a socket, but it does not connect, then you do not have to close it (but you can still issue a Close if you want to.)

Note that once a socket is closed the values of many properties, such as *bytesreceived* may become unavailable.

### Send

**returns** integer "length"

**parameters** string "text"

Sends a block string of data out to the remote server.

### Recv

**returns** string

**parameters**

Gets data from the remote server as a string.

### SendCSV

**returns**

**parameters** string (comma separated values)

For the more advanced among you, this can be very useful. A byte of course is a binary value of 0 to 255 inclusive, and bytes are the foundation of all TCP/IP communications. By specifying a string decimal numbers separated by commas (for example "1,255,123,131,13,10,0"), this function will convert them to bytes codes before sending them on.

Primarily this is of use with TELNET-type servers which use embedded byte-codes that have no keyboard equivalent. You can also use it to build binary file formats, but that's a more complicated example.

### RecvCSV

**returns** string (comma separated values)  
**parameters**

Instead of receiving a purely text string, any data recvd is decoding in a a string of comma separated numbers. The numbers represent bytes as values of 0 to 255 inclusive.

### SendPacket

**returns**  
**parameters**

Sends the binary data puffer.

### RecvPacket

**returns**  
**parameters**

Receives a UDP packet into a binary buffer.

### packetclear

**returns**  
**parameters**

Resets positions of binary packet and blank the packet buffer.

### addpacketnumber

**returns**  
**parameters** string "type" string "value"

Adds a number to the binary packet buffer. type may be: BYTE, WORD, DWORD, int8, int16, int32, float32 or float64.

### addpacketchar

**returns**  
**parameters** character "char"

Adds a single character to the binary packet buffer.

### addpacketstring

**returns**  
**parameters** string

Adds an ascii string to the binary packet buffer.

As of v1.16 treated NULL terminated strings as C/C++ strings (NULLs terminate string).

### addpacketstringN

**returns**  
**parameters** string

Adds an ascii string to the binary packet buffer but without a following c-style NULL character. Basically this is to allow VisualBasic-style strings that have prepended length variables.

As of v1.16 treated NULL terminated strings as VB strings (NULLs are preserved).

### addpacketipport

**returns** string "ip" ("octet.octet.octet.octet"), number "port"  
**parameters** boolean "connected"

Inserts an IPv4 ip:port string into the packet stream.

### packetnumber

**returns** number "num"  
**parameters** string "type"

Reads data from the binary packet buffer. type may be: BYTE, WORD, DWORD, int8, int16, int32, float32 or float64.

### packetchar

**returns** character "char"  
**parameters**

Reads a single character from the binary packet buffer.

### packetstring

**returns** string "text"  
**parameters**

Reads an ascii string from the binary packet buffer.

### packetstringN

**returns** string  
**parameters** integer "length"

Reads in an ascii string from the binary packet buffer of the specified length. This is to facilitate, among other things, the use of VisualBasic-style strings that used prepended length variables.

### packetstringC

**returns** text  
**parameters** character "char"

Reads in an ascii string from the binary packet buffer until it encounters the specified character.

### packetipport

**returns** string "ip" ("octet.octet.octet.octet"), number "port"  
**parameters**

Extracts an IPv4 ip:port string from the packet stream.

### ClearLog

**returns**  
**parameters**

Clears the error log.

# PROPERTIES

All properties are optional. They can be extremely useful for optimizing the performance of your code however, particular where high-performance, highly-responsive operation is required.

## PUT

### timeout

**returns**

**parameters** floating-point-number "seconds"

The timeout setting is very important to understand for peak performance. The defines how long any of the Send, Recv, SendCSV or RecvCSV functions will wait for a response from the remote server before giving up. This is a float point number representing the number of seconds. Millisecond precision can be set using the fractional portion. For example the default setting is 100milliseconds. In ASP you would set this as:

```
obj.timeout=0.100.
```

It is very important to note that because of internet congestion or other situations such as heavy server load that you may not be able to receive the entire text of a transmission at a single time with the default setting of 100ms. Adjust it up to whatever you feel is appropriate.

For gaming purposes this is essentially equal to the milliesecond ping times.

### reposition

**returns**

**parameters** number "position"

Repositions index into binary packet.

## GET (anytime)

### description

**returns** string

**parameters**

Description of this COM object.

### copyright

**returns** string

**parameters**

The copyright notice.

### version

**returns** string

**parameters**

The current version number.

### Log

**returns** string

**parameters**

Returns the error log string.

### instance

<b>returns</b>	number
<b>parameters</b>	

If your code calls several instances of this COM into existence at once you can determine which instance you are talking to by calling this.

This is an integer number, where the default first instance is always numbered "1".

### instances

<b>returns</b>	number
<b>parameters</b>	

See "instance". Calling this returns the number of instances of this COM object that have been called into existence since your server has been rebooted.

Note: When using this COM in desktop software the instance number will reset itself when the software is closed and restarted. When used in sever-side software, as long as the COM stays loaded in memory, then every time the COM is called the instances number will increase by one.

Instances/Instance are mainly of use with debugging/optimizing the COM instantiation process.

This is an integer number, where the default first instance is always numbered "1".

## GET (only when open)

### timeout

<b>returns</b>	floating-point-number "seconds"
<b>parameters</b>	

Returns a floating-point-number indicating the current timeout setting in seconds. For instance "3.50" indicates the functions will give up (timeout) after three and a half seconds.

### localaddress

<b>returns</b>	string "address"
<b>parameters</b>	

Returns a string with the *address:port* url of the local end of the connection.

### remoteaddress

<b>returns</b>	string "address"
<b>parameters</b>	

Returns a string with the *address:port* url of the remote end of the connection.

### bytessent

<b>returns</b>	integer "bytes"
<b>parameters</b>	

The total number of bytes sent to the remote server.

### bytesreceived

<b>returns</b>	[integer:num]
<b>parameters</b>	

The total number of bytes received from the remote server.

**udpsocket****returns** integer**parameters**

The current socket (session) number being used.

**senttimeout****returns****parameters** floating-point-number "seconds"

If a RECV method times out, this will tell you. If it is 0 then the last RECV used did not time out. Otherwise it is the seconds the method waited before giving up.

Timeout is used to dynamically adjust for network latency conditions. How you do this of course is completely up to you.

**receivedtimeout****returns****parameters** floating-point-point "seconds"

If a RECV method times out, this will tell you. If it is 0 then the last RECV used did not time out. Otherwise it is the seconds the method waited before giving up.

**sendlength****returns** integer**parameters**

(Might be the length of the internal send buffer.)

**recvlength****returns** integer**parameters**

(Might be the length of the internal recv buffer.)

**messagelength****returns** integer**parameters**

(Might be the length of the udp buffer.)

**buffersize****returns** integer**parameters**

The size of the internal udp buffer: 65507.

**kbpsending****returns** floating-point-point**parameters**

The computed kilobytes per second of data sent.

**kbpsreceiving****returns** floating-point-number**parameters**

The computed kilobytes per second of data received.

### ping

**returns** integer "milliseconds"

**parameters**

Ping is computed automatically for you. Internally it is computed by simply starting a timer whenever a udp packet is sent and stopping it the next time you receive a packet. Because of this please be aware that if you send a udp then do a lot of computations inbetween that and when you attempt to receive the packet that the ping numbers may be off.

This is in milliseconds.

### packetlength

**returns** integer "bytes"

**parameters**

Length of binary packet.

### packetposition

**returns** integer "bytes"

**parameters**

Position into binary packet.

### packetEOF

**returns** boolean

**parameters**

Returns TRUE if packetposition GTE packetlength. Else FALSE.

### is\_connected

**returns** boolean

**parameters**

Returns TRUE if basic winsock and UDP connection opening sequences occurred without any problems. Generally only returns FALSE if there is a TCP/IP stack or network configuration problem.

## VERSION HISTORY

- **2.1** November 5th 2004.

Updated the C++ classes it shares with several other projects, so recompiled. Cleaned up for the relaunch of the [Intrafoundation](#) web site.

- **2.00** July 6th 2004.

Never released 2.00 but did today after a few tweaks and cleaning up the documentation. The property "error" has become "Log" and a "ClearLog" function has been added.

NOTE: For compatibility with ColdFusion all "boolean" values are passed as "long" (Once again. Now I remember why it was originally doing that -- ColdFusion can't handle boolean's with COM objects.)

- **2.00/1.18** April 23 2004.
  1. Fixed inversion of returns/parameters notation in documents.
  2. Added daytime and time protocol examples.
  3. Added new Log class and extra debugging info.
  4. Misc little tweaks to clean up code a bit.
  5. Oh. Important. Fixed a bug (an oversight actually) that had "open" returning a number instead of a boolean like was stated.

Thought about leaving it that way, but couldn't resist fixing it.

The downside is this breaks some ASP code that's looking for a number to be returned. It now returns "True" or "False" as a boolean should.

- Added instance and instances properties.
- Added an application level thread-safe mutex. (So the COM can be used in application scope of CF or ASP without too much crashing.)
- Changed version to 2.00 when I realized I had to change the COM interface for the additional functions. Before 2.00 all versions used the interface "Intrafoundation.UDPClient.1". This version uses "Intrafoundation.UDPClient.2" which is slightly different (new functions and a change to the open functions' return type.)
- Added high resolution timer code.
- Optimized udp Recv function.
- Added sendpackets and recvpackets stat.
- **1.17** November 20th 2003.

```
UDPClient Author: Eric Shufro URL: http://eric.shufro.com Created: 9/6/
2003 9:44:54 AM Maybe im doing it wrong, but addpacketstringN chr(128-
142) excluding 129 and 141 dont work. Nor does chr(145-159) excluding chr
(157). Very odd. Instead the udpclientcom sends chr(0) when any of those
listed numbers that dont work appear. All other numbers between 0 and 255
work perfectly. There are 27 numbers that will not send and generate 0`s
instead. Any ideas?
```

- **1.16** July 20th 2003.
 

Minor tweak to addpacketstring and addpacketstringN. Cleaned up some docs.
- **1.15 G** May 2003.
 

A few minor fixes to docs and code.
- **1.12** Nov 15th 2001.
  - My ASP is a bit rusty, but... I added ASP examples for Half-life, RTCW, Quake3, STEF, etc after a bit of prodding by Sakke Huhmarniemi.
  - Fixed packetlength and packetposition bug that let "junk" number show up in asp if you weren't connected.
  - Added packetEOF property.
- **1.11a** Oct 8th 2001.

SteelValor sent over some spare gaming gear for the gaming-list help the other day. The attack v-1 gaming mouse pad is **\*nice\***. I mention this because I was trying it out in the new Quake3-based Return to Castle Wolfenstein (yes, I'm addicted to it -- so are a lot of us)... and I kept killing this poor guy named **captureman** after the other folks left the server. Over and over and over. He finally just disconnected. I feel bad about it. Poor guy. These pads are evil. /-)

A few slight script changes. Someone gave me an old copy of Tribes 1.0. Apparently the code doesn't work very well with it or the 1.11 patch. Hrm.

- **1.11** Oct 2nd 2001.

Cleaned up the cfm a bit. Added packetstringN, addpacketstringN and packetstringC functions to COM to handle visualbasic-style strings with prepended length variables. This was done to more easily deal with the format of Starsiege: Tribes (you could have simply used a loops of packetchar's).

Added some experimental Starsiege: Tribes code. Since I don't actually own a copy... it's somewhat of an educated guess as to whether it works.

- **1.10** Sept 21st 2001.

Added wolfenstein test (same as quake3). Muller over adding high-performance functions to batch send/recv for gamescry. Forums at [steelvalor](#).

My game name, BTW, is **TommyRaven** or sometimes **LT.TommyRaven**.

- **1.9** Sept 13th 2001.

Tightened up a lot of the udp class (the low-level class part of the COM). Also straightened up several script bugs or oddities of the protocols in the scripts, especially with unreal. Added some code by [steelvalor](#).

- **1.8** Sept 9th/12th 2001.

Some of the scripts were upgraded, especially gamescry and unreal. Added some experimental IP and Port packet features, but you probably shouldn't try to use them.

- **1.7** Sept 6th 2001.

Duh. Forgot to add a PING function. Fixed. Now every send/recv cycle has ping automatically computed for you.

- **1.6** Sept 5th 2001.

Fixed an oddness in the half-life script that misreported mod byte sizes.

Also added GAME SCRY ;-> page.

- **1.5** Sept 4th 2001.

Munged up the half-life code. Fixed. Misc other.

- **1.4** Sept 2nd 2001.

Cleaned up examples.

- **1.3** Sept 2nd 2001.

Cleaned up packet functions. I haven't as yet used the "packet" write/add functions at all so it's a guess as to whether they actually work or not.

- **1.2** August 31st 2001.

More cleanup.

Binary field structuring/destructuring functions were added to handle half-life. Unlike all the other server query setups which send their data in clear text, half-life sends its data in binary c structs. Efficient, but annoying to deal with.

While redesigning the website I'd completely forgotten that I'd decided not to make tcpclient/udpclient GPL. Why? Because some folks want to use them in commercial apps and then sell the apps. I really don't give a damn what most people do with this code as long as they make good use of it and don't try to claim that THEY wrote the original code (people do that sometimes). If there was a GPL that allowed use in commercial apps without having to reveal the code that others wrap around it then that's what I'd use. And how I'll assume you'll use it.

- **1.1beta** August 30th 2001.

General cleanup to code and docs. Added Unreal derivatives. Looking for Baldur's Gate, etc info.... (btw, before you ask: I don't know what the yellow flower symbol is about. It just wanted to sit there so I let it.)

Still reworking those damn docs.... reading the working scripts would probably be more informative.

- **1.0alpha** August 29th/30th 2001.

Started working on it again in the afternoon. Pretty much had it working later that night after trying a few different strategies. K.I.S.S. /-) The documentation is completely bogus at the moment... leftover from the tcpclient project folder. It's my experience that writing technical docs usually takes longer than writing the project itself.

- **0.1alpha** August 22nd 2001.

Some folks at <http://www.cnyonlinegamers.net> asked if I could list server game stats. What the hell. Was going to have to write something similar for another project soon anyway, sooo... Recreated the projects files.

- **0.0** - 1999 - 2001, shadow thought for tcpclient for years