

Intrafoundation.TCPClient.3

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

Freeware with C++ source code



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

v3.1, November 5th 2004

<http://www.intrafoundation.com>
products@intrafoundation.com

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

THE TCPCLIENT COM OBJECT ALLOWS YOU TO

talk to any Internet server using TCP/IP such as: NNTP, POP3, SMTP, IRC, FTP, WHOIS, ECHO, RLOGIN, FINGER, etc. You could for instance ask an internic WHOIS server to return information on a domain name. Or you could do a FINGER on JOHNC at IDSOFTWARE.COM to read John Carmack's .plan file. Or query a POP3 email server, or an NNTP news server, or an FTP server, or just about anything else.

Simple, but useful.

More specifically it provides for asynchronous communications to any TCP/IP based server (which is just about everything higher-level on the Internet). You can send text messages to or receive messages from any of these servers. You can send or receive binary data back and forth from them as well. Additionally there is basic encoding/decoding function provided for the MIME Base-64 and UUENCODED formats you may encounter with news (NNTP) or mail (SMTP/POP3/IMAP4) servers.

This COM object provides only the elemental, base functionality you need to talk to these servers. As for what you're supposed to say to them and what do the messages they send back mean, well... Try reading the official RFC text documents to learn their languages.

But, as an example, a good way to start out with the SMTP mail server is to simply hold out a hand and say:

HELO

What you do afterward is up to you....

EXAMPLE SCRIPTS

Active Server Pages

(Note: Some ASP examples require the included global.asa to function. Copy this to your root folder or set up tcpclientcom as a virtual folder.)

internet
[daytime](#)
[time](#)
[whois](#)
[finger idsoftware.com](#)
[http intrafoundation.com](#)
[http microsoft.com](#)
[http 15seconds.com](#)
[http msdn.pe.kr](#)
[http msdn.pe.kr](#) (direct)
[http msdn.pe.kr](#) (direct) (rn)
[nntp forums.macromedia.com](#)
[mud.jedi.betterbox.net](#)

localhost
[hash](#)
[framesize](#)
[frame](#)
[http localhost.csv](#)
[http localhost](#)
[threadsafe](#)
[threadsafe_openwindows](#)
[\(properties localhost\)](#)
[\(properties\)](#)
[\(instance\)](#)

ColdFusion

internet
[daytime/13](#)
[whois](#)
[http macromedia.com](#)
[http allaire.com](#)
[finger idsoftware.com](#)
[finger ravensoft.com](#)
[finger legendent.com](#)
[http houseoffusion.com](#)
[http intrafoundation.com](#)
[http microsoft.com](#)
[http yahoo.com](#)
[http amazon.com](#)
[http cslewis](#)
[http uk.weather.yahoo](#)
[http comunion](#)
[http hercules](#)
[ftp.cdrom.com](#)
[mud.jedi.betterbox.net](#)
[nntp forums.macromedia.com](#)
[nntp](#) (news browser)

localhost
[smtp](#)
[pop3](#)
[http](#)
[http HEAD](#)
[ftp](#)
[nntp](#)
[ftp pasv list](#)

REQUIREMENTS / SERVER PLATFORMS

This COM object was designed for Windows NT 4 or Windows 2000. It has not been tested under Windows 95, 98 or ME. (It was intended only to be used on commercial server machines, not consumer desktops.)

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 dwells in the file called `tcpclientcom.dll`. To install, at the command prompt type: `regsvr32 tcpclientcom.dll`. You may wish to copy it to your project or system folder first.

The precompiled file `tcpclientcom.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.

UNINSTALLATION

At the command prompt type: `regsvr32 /u tcpclientcom.dll`.

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).

For testing purposes most accounts are **tcpclient** with a password of **tcpclient**. Domains, where used, are the the locally used **foundation.local** using the 192.168.0. Class C private subnet ip's for local testing. This mimics the setup of the development server used here. 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

ftp pasv stor
ftp pasv retr
(1mb ftp) setup
ftp 1mb stor
ftp 1mb retr
(30mb ftp) setup
ftp 30mb stor
ftp 30mb retr
imap4
cisco
telnet
echo
(port test)
(recv)
port scan (slow)
(properties localhost)
(properties)
(instance)

custom tags
(none currently)

PHP

(none currently)

JAVA

(none currently)

Send in your examples.

as they talk to other machines out there. If you are not connected to the net and do not have a domain name server installed locally the tag may sit there for several seconds waiting for a name server to answer it before finally giving up.

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

There are two subfolders. One is called "send". It's contents are sent out during some of the ftp and smtp/pop3 tests. The "recv" folder is where these files are downloaded back to.

Send in your own sample scripts.

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/tcpclient.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.

You use this software at your own risk.

THE HARNESS FILES

Q: What are all the "harness" folders about that are sometimes included with this software?

A: A "test harness" is just a simple command-line application that is used to quickly test portions of a program in an easy to debug environment. Without using them, debugging a DLL that runs under a web server would take a very, very, very ... (did I mention very?) long time.

They are not needed to use this software. They are only included for completeness of the project and for use by others who wish to debug an issue.

PROTOCOL RFC'S

Q: Where can you get these RFC's I keep talking about?

A: They can be found at many sites across the internet. Try:

- <http://www.cis.ohio-state.edu/hypertext/information/rfc.html>
- <http://www.rfc.net>
- <http://www.freesoft.org/CIE/RFC/index.htm>

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.

MANUAL

TCP/IP OVERVIEW

METHODS

A few words first....

With TCP/IP there are two modes of communication that may be used: Synchronous and Asynchronous. Synchronous means that the client will send information, then the server will send back information, and then so on in that order. Essentially a question and answer format.

For some services such as **whois** servers this works well. You send a **whois** server the domain name you want information on, and it returns the result to you. It's a little like an SQL query that way.

finger and **http** also work fine with simple synchronous communications.

However, more complicated protocols such as **ftp** may be doing several things at once and in fact on different ports. To simply get an **ftp** directory listing you would open communications to an **ftp** command port and request a directory listing. The listing data then comes in on a *different* port. If it's a large directory then the data being sent may take several seconds or minutes to transfer. In the mean time your command port communications continues on.

This is what is meant by asynchronous communications.

This software uses asynchronous communication mode. You can still talk to any server that operates in the simpler forms such as whois and finger. It's just a little more work to determine when the server is done talking to you is all.

All TCP/IP communications comes in packets. A packet can contain anywhere from 0 to 1536 bytes. You can never tell beforehand how large any packet will be. For example, you could receive a response from a **whois** server in 10 packets of 1 byte each, but generally you'll be sent data in as large of a packet as possible.

This object will automatically continue to receive packets and string it all together for you as long as the remote server continue to send them. It will only stop if the remote server drops it's connection, if it time's out or if you setup a cutoff point.

So, when retrieving data from a remote server via tcp/ip you'll continue to recv(ieve) data until you're done. How do you know when you're done? Some protocols like **ftp** explicitly tell you how many bytes to expect. With others like **whois** you just continue

name	input	output
Open	host (string), port (string)	connected (boolean)
Close		
Send	text (string)	
Recv		text (string)
SendRN	text (string)	
RecvRN		text (string)
FlushRN		
SendFile	filename (string)	
RecvFile	filename (string)	
RecvFileAppend	filename (string)	
SendCSV	csv (string)	
RecvCSV		csv (string)
SendFrame	frame-definition (string), frame	
RecvFrame	frame-definition (string)	frame
FrameSize	frame-definition (string)	framesize (number)
EncodeFrame	frame-definition (string), field-name (string), field-value (string), frame	frame
DecodeFrame	frame-definition (string), frame, field-name (string)	field-value (string)
ClearLog		

PROPERTIES (PUT)	
timeout (alias set_timeout)	seconds
cutoff (alias set_cutoff)	bytes
blocksize (alias set_blocksize)	bytes
packetsize (alias set_packetsize)	bytes
keepalives (alias set_keepalives)	boolean
nagledelay (alias set_nagledelay)	boolean

receiving until they're done and close the connection on you (rude as it may sound). For others you may just have to read their RFC docs and guess.

PEAK PERFORMANCE: TIMEOUTS & CACHE MISSES

One thing that can not be stressed enough about this COM object is that it understands nothing about the higher-level application protocols such as POP3, NNTP, SMTP, NTP, etc. The functions are generic. That is, they have not been optimized to take the cues most protocols send to tell you when they are finished sending a block of data. Recv, for example, will continue to receive and accumulate data forever until 1) the remote server closes the session (normally or abnormally) 2) you use the optional maximum byte transfer limit *cutoff* property or 3) the *timeout* limit is exceeded. (This is true for all the send/receive functions with the exception of RecvRN, which is a special case.)

What this means is that, if you are on good T3 connection, that issuing one single Recv command could cause the COM object to download a block of data for hours, hundreds and hundreds of gigabytes of data. Eventually your storage devices would be flooded and your machine crash -- unless you use either the above mentioned *cutoff* or *timeout* properties to impose limitations.

The concept of a maximum byte cutoff is fairly obvious (see the function docs for more details). As for the timeout, for this COM it specifies the maximum amount of time the COM will wait around for the remote server to send some data, any data, before it gives up trying. On LAN's you might be able to get away with timeouts as low as 10 or 20 milliseconds. On the internet 5 seconds or more is generally required.

Anyway, besides preventing your machine from crashing there are performance issues to consider. Some protocols such as WHOIS and HTTP are simple things; you issue them a command to retrieve

data and they do so and close the connection. Recv and the others work fine here. They grab all the data the remote server sends them until it tells the COM object it's done and closing the session.

Things get trickier with say POP3. POP3 (Post Office Protocol 3) is the protocol you use to receive mail from your internet mail box. It is command-driven. That is, you open a connection, send a command, get a response. While the connection is open you'll probably send several commands and read several responses. If you've subscribed to a mailing list that generates a hundred or so emails a day you could be issuing several hundred commands to the POP3 server to transfer all your email and delete it afterward before closing a connection.

PROPERTIES (GET)	
thread	number
version	string
copyright	string
description	string
instance	number
instances	number
Log	string
is_completed	boolean
is_connected	boolean
local	string "url"
remote	string "url"
bytessent	bytes (number)
bytesreceived	bytes (number)
byteslastsent	bytes (number)
byteslastreceived	bytes (number)
socket	socket (number)
is_cutoff	(boolean)
cutoff	bytes (long integer)
blocksize	bytes (long integer)
packetsize	bytes (long integer)
stack_recv_buffer	bytes (long integer)
stack_send_buffer	bytes (long integer)
recv_packets	(long)
send_packets	(long)
recv_faults	(long)
recvbuffer_faults	(long)
kbpsent	(floating point number)
kbpsreceived	(floating point number)
kbpslastsent	(floating point number)
kbpslastreceived	(floating point number)
ping	milliseconds (integer)
timeout	seconds (floating-point)
last_timeout	milliseconds (integer)
mutex_sleep	(integer)
mutex_locks	(integer)
mutex_sleep	(integer)
is_keepalives	(boolean)
is_nagledelay	(boolean)

That's not good.

For any protocol that is command-driven you should use RecvRN. It specifically caches all data it receives and only retrieves more from the server when its buffer is empty. By using RecvRN you can avoid any needless timeout issues.

SPEC LIMITATIONS

How big a block of data can this transfer? Well, how much memory do you have? Earlier versions sometimes crashed under IIS5/ASP because it was running into its 256kb stack limit, but current versions are, to our knowledge, unlimited. Or that is, limited only by your RAM and drive space.

Is thread-safe? Yes. It should be. The core tcp class has been hammered from time to time by multiple computers. Far as is know, there are no memory leaks and it threads fine.

Accidents could happen between versions however. If you notice any memory or threading issues, just yell.

SSL

There are plans for SSL (Secure Sockets Layer) to be added in the future. Currently however any SSL specific functions in this COM object will silently fail.

FRAMES

New for 3.x is the concept of "frames".

There are some protocols running on top of TCP/IP that exchange data as a set of binary fields of fixed size. Before 3.x these were somewhat difficult to handle as the COM function were primarily aimed at handling text strings, leaving the task of manipulating any binary data received up to the ingenuity of the person writing the scripts using it (i.e., `_you_`.)

The addition of five new functions named: `SendFrame`, `RecvFrame`, `FrameSize`, `EncodeFrame` and `DecodeFrame` change this. All of these functions make use of what is called a "frame-definition". The frame definition of the UDP :) protocol NTP (Network Time Protocol) looks like this: `" +2 LI, +3 VN, +3 Mode, +8 Stratum, +8 Poll, +8 Precision, +32 RootDelay, +32 RootDispersion, +64 ReferenceIdentifier, +64 ReferenceTimeStamp, +32 OriginateTimestamp, +64 ReceiveTimestamp, +64 TransmitTimestamp, s12 Authentication"`

(And before you ask why don't I use a TCP protocol for an example, the answer is just wrote a SNTP time client two days before the release of 3.0 so it's fresh in my mind.)

Frame definition consists of a comma separated set of three field items. These are:

1. Endian indicator
2. Bit length
3. Field-name

The endian indicator is either "+" for Big Endian (used by most network protocols and Apple computers) and "-" for Little Endian, use for most everything else in the world include Intel/AMD processors.

Immediately following the endian flag (no spaces) is the number of bits the data of this field is allowed to occupy. 8 BITS of course is a BYTE. 16 BITS is a WORD, and so on. A "+32", to recap, would mean a 32-bit DWORD in big-endian/network format.

Lastly, seperated by a space, is the name of the field. The field-name is used to allow easy referencing by the EncodeFrame and DecodeFrame functions.

Bit Lengths

Bit lengths can be up 64 bits. HOWEVER: Internally this software operates on all frame fields as binary stream with a 64-bit window. The short of this is that as long as the field is on a byte boundry, it can be up to 64 bits in size. If it is not on a boundry then there may be as little as 56 bits of resolution available due to loss of precise by shift/mask operation.

This technical limitation is however, not generally something to be concerned over as most protocols that use 64-bit data usually have them aligned on `_32-bit_` boundries as this is more efficient on modern processors on network devices. Only if your data is not byte-aligned will it be limited to at most 56 bits of resolution.

On EncodeFrame and Frame reuse

After glancing at the Frame functions it should be noted that the "Frame" itself is binary data contained within a BSTR. This is to be considered only a temporary way to aggregate and hold your data. The raw "frame" data itself is generally non-printable and can only be created or accessed with the EncodeFrame and DecodeFrame functions.

It should especially be noted that EncodeFrame will encode a new field upon `_any_` BSTR you pass it, producing a legal "Frame" out of this data.

METHODS

Open address, port

Parameters address (string), port (string)

Returns boolean

Opens a TCP/IP 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

Parameters

Returns

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*, become unavailable.

Send text

Parameters text (string)

Returns "length", a long integer

Sends a block string of text to the remote server.

SendRN text

Parameters text (string)

Returns "length", a long integer

Sends a string of text and an appended "\r\n" sequence, otherwise known as CRLF, CHR(13) & CHR(10) or a CARRIAGE RETURN and a LINEFEED. Most TCP/IP protocols require an RN sequence to be sent before executing any commands you send them.

The RN or CRLF sequence is, fyi, the character codes sent from your keyboard when you press the RETURN (or ENTER) key. Make sense now?

FlushRN

Parameters

Returns

If you use RecvRN then this software will automatically create an input buffer where all the data it's received is stored until subsequent RecvRN's exhaust the buffer. For instance, if you connect to a HTTP server and send the commands to grab a web page, then the next time you do a Recv or a RecvRN the HTTP server will mostly likely send the results back in one large packet. If you call Recv you'll get back all of this data at once. If you call RecvRN it will return to you only one line at a time, however, the data itself will still remain in memory on your computer until you've called RecvRN enough times to read it all. (Actually, technically, each time you call RecvRN it removes the line it returns from memory.)

In any event, you may have reason to want to force this buffer to be cleared. For instance, you've read the important header information you needed and don't want to bother parsing the rest of it before you issue another command. This function completely clears that internal buffer.

Recv

Parameters

Returns "text", a string

Gets data from the remote server as a string of text. Recv will continue to accumulate data from the remote server until it closes connection or the link times out (see the timeout property).

If the remote server doesn't close connection during a Recv (as HTTP will do) then you will suffer a timeout.

RecvRN

Parameters

Returns "line", a string

The RecvRN function retrieves one CRLF (aka RN) terminated line at a time from a remote server. The trailing CR (carriage return) and LF (line feed) control codes are stripped.

Unlike Recv, RecvRN uses a cache buffer. It only asks the remote server for more data if it can not find a CRLF terminated string in it's cache. Is this important? Oh yes. With Recv you blindly ask server to return data until it closes connection or it errors with a timeout. This is fine for HTTP connections, but for talking with command driven POP3, SMTP, etc where the tcp connection remains open you will encounter a timeout for every Recv issued. Timeouts can be costly in performance.

If there is no remaining data in the buffer and the remote server does not have additional data waiting RecvRN will return a blank string and timeout. You can check TIMEDOUT to see if the last RECVRN failed.

SendFile filename

Parameters filename (a string)

Returns

Sends the entire contents of a file as binary data.

RecvFile filename

Parameters filename (a string)

Returns

Receives all the binary data from the remote server until it closes the socket or timeouts. This data is appended to the specified file. If the file does not exist it is created.

RecvFileAppend filename)

Parameters filename (a string)

Returns

The same as RecvFile, except that data received is appended to the end of the file specified.

SendCSV comma-seperated-values

Parameters comma-seperated-values (a string)

Returns

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

Parameters

Returns comma-seperated-values (a string)

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

ClearLog

Parameters

Returns

Clears the error log. (Function formerly called "Clear" in 2.x.)

SENDFRAME frame-definition, frame

Parameters frame-definition (a string), frame

Returns

Sends the binary data in "frame" after computing the frame-size from the "frame-definition". The function itself operates the same as the Send function with a slight amount of overhead to handle the frame data.

See the [frame](#) section for more details.

RECVFRAME frame-definition (returns frame, a binary string)

Parameters frame-definition

Returns "frame", a binary string

Receives data from the stream the same as the Recv function, except that it always tries to return an amount of data equal to the framesize as specified in the frame-definition. Any additional data that was received at that time will remain in the internal recv buffer and can be retrieve by subsequent calls to any of the "Recv" functions including Recv, RecvRN, RecvCSV and/or RecvFrame.

If it does not recieve any data before timeout occurs it will return a NULL string.

See the [frame](#) section for more details.

FRAMESIZE frame-definition (returns "framesize", a number)

Parameters frame-definition (a string)

Returns "framesize", a long integer

Given a valid frame-definition this function will return the number of bits that will be required to encode it. For example "+8 Mode, +4 Align, +4 Status" would return 16 if given to the FrameSize function.

Primarily, this function can be used to valid your math skills when computing the framesize by hand.

See the [frame](#) section for more details.

ENCODEFRAME frame-definition, field-name, field-value, frame

Parameters frame-definition (a string), field-name (a string), field-value (a string), frame

Returns "frame", a binary string

EncodeFrame creates the binary data structure that composes a frame that you wish to send. It does this by issuing in succession a serious of EncodeFrame commands with the field-name you wish to add/change and it's value.

For example: `enc=obj.EncodeFrame("s20 Name,s20 Genre","Name","Blue Danube","")` creates a binary string of a size equal to 40 bytes and populates the first 20 bytes with the ASCII string "Blue Danube". The last 20 bytes where "Genre" would go would be blank until or unless you populate it as well.

For example: `enc=obj.EncodeFrame("s20 Name,s20 Genre","Genre","Classical",enc)` Takes the variable "enc" which is holding our frame data and adds the string "Classical" to the last 20 bytes.

This function take an existing frame as an input, returning a new frame that has the specified field data changed within it. Generally, you would just pass to it the same string as you return from it.

Note that this function will take any string as an input and will modify only the area as specified by the field-name for the frame-definition. If the input frame contains more data than the frame-definition allows,

the data will be truncated. If less, then it will be zero-padded. In other words, it operates upon a "dirty" data principle -- You can RecvFrame a frame, change one field, and send it directly back without having to reencode the entire thing.

See the [frame](#) section for more details.

DECODEFRAME frame-definition, field-name, frame

Parameters frame-definition (a string), field-name (a string), frame

Returns "field-value", a string

Uses the frame-definition to extract from the binary frame data the value for the field specified by field-name. For example, the following extracts the "Mode" data from an NTP (Network Time Protocol) formatted frame:

```
Mode=obj.DecodeFrame("+2 LI,+3 VN,+3 Mode,+8 Stratum,+8 Poll,+8 Precision,+32
RootDelay,+32 RootDispersion,+64 ReferenceIdentifier,+64
ReferenceTimeStamp,+32 OriginateTimestamp,+64 ReceiveTimestamp,+64
TransmitTimestamp,s12 Authentication","Mode",my_frame_i_just_received)
```

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/set_timeout seconds

Parameter seconds (a floating-point number)

(Note: Has an alternative alias with a "set_" prefix for older versions of ColdFusion which do not properly handle get/put COM names.)

The timeout setting is very important to understand for peak performance. The defines how long any of the Send, Recv, SendRN, RecvRN, etc function 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 simple protocols such as **whois** and **http** timeout's of 15 seconds or higher are fine in most cases. More complex protcols such as ftp are a *completely* different matter however.

ftp, and a few other protocols such as irc maintain a continuous open tcp session. When they complete a command (such as listing the files in the current directory, or changing to a different directory) they **do not** close the session. They perform the command and wait for you to issue another one. Setting the timeout to 15 seconds in this case would be agonizing; everytime you changed to different folder on the remote server it would wait 15 seconds (unless the server remotely terminates you).

Caution: Unless the socket times out or the remote server remotely terminates the session, the RECV* functions will continue to retrieve data without any limits.

The thing about all this is -- unless you're using RecvRN -- you're ALWAYS going to be timing out unless the sessions terminates normally (or abnormally even). This is because it's blindly waiting to receive all the data it can.

cutoff/set_cutoff bytes

Parameter bytes (a long integer)

(Note: Has an alternative alias with a "set_" prefix for older versions of ColdFusion which do not properly handle get/put COM names.)

Normally you'll never have to use this, BUT... it is here if you do. This puts a hard limit on the amount of data for any transfer. If you try to connect to a continuously streaming site (something pumping out a live multimedia stream for example) you'll obviously need to tell the RECV* functions to please-in-the-name-of-god stop! :) otherwise it will continue to download data for ever... eventually overflowing your hard drives and crashing your system.

For example, if you set a cutoff of 65536 (aka 64kb) then a RECVFILE will stop after it downloads 64kb. You can then issue it again in a conditional loop to continue processing.

Cutoff uses an integer number that is in bytes. Setting a cutoff of 0 disables the limit. Any other value engages it. By default it is always disabled.

An important technical note: TCP/IP sends data in packets that can range anywhere from 0 to 65526 bytes in size. Keeping this in mind, when you, for example, specify a cutoff of 6000 bytes you are NOT going to receive back a block of data exactly equal to 6000 bytes in length. The cutoff is only checked after each packet of data is accumulated so you could have up to ~65526 bytes more than your limit. ie, up to 7536 bytes.

To give a precise cutoff would require an integrated buffering system that was deemed too detrimental at this time. (But if I start having to work with streaming video, you can bet I'll take the time to add precision block control and bit-level structure handling.)

Cutoff should be considering a rough "panic limit", not a precision tool.

blocksize/set_blocksize bytes

Parameter bytes (a long integer)

(Note: Has an alternative alias with a "set_" prefix for older versions of ColdFusion which do not properly handle get/put COM names.)

Affects Recv, RecvRN, RecvCSV and RecvFile. By default is 65536 bytes.

A major inhibitor of performance when moving very large streams of data around (30mb files for example) is the overhead associated with managing the memory to hold this data. Specifically, the overhead in creating and destroying continuous memory blocks whenever a block of data "grows".

In our case, as data is streamed from the remote server to this COM object it is tacked on to an internal buffer. Using the default tcp buffer size of 8192 bytes you'd have to "grow" this memory buffer some 3840 times in the course of a transfer of a 30MB file. This overhead is very resource expensive, so we make use of the *blocksize* property to allow you to specify how big of a block of free memory we should allocate everytime we "grow".

All this means is that when you receive the first 8192 bytes of the aforementioned 30mb file, if you had a blocksize of 1mb (1048576) then we'd pre-allocate memory in blocks of 1mb. This reduces the number of times we'd have to grow the buffer down to only 29.

Of course, it also means we're allocating a lot of memory as we go along. You simply have to figure out what the right balance of RAM and processing power is best for what you're doing.

In general this shouldn't be an issue you'd have to really worry about unless you're moving files around that are say half a MB or or more in size. For the most part only ftp client writers will need to worry about this. Although if large mail attachments are involved then SMTP, POP3 and NNTP client code might need to make use of it as well for performance.

The minimum blocksize is that of the tcp buffersize (currently at 8192 bytes).

packetsize/set_packetsize bytes

Parameter bytes (a long integer)

(Note: Has an alternative alias with a "set_" prefix for older versions of ColdFusion which do not properly handle get/put COM names.)

Sets the assumed packet size for outgoing data.

When sending data it is send in "packets" or blocks of a set maximum size. As a general rule of thumb the size of these packets are anywhere from 512 to 1536 bytes.

This software uses the default of 65536 bytes unless you override it. The standard default for TCP/IP stacks however is 512 bytes. The choice of 65536 bytes comes about by performance tests over an Ethernet LAN network when this software was initially designed. 65536 bytes was the "sweet spot" that gave the highest throughput performance. Of course, depending on the hardware you're using, you may wish to change the assumed packet size.

keepalives/set_keepalives flag

Parameter flag (a boolean)

(Note: Has an alternative alias with a "set_" prefix for older versions of ColdFusion which do not properly handle get/put COM names.)

Sets "keepalives" to on or off. As to whether this functions depends upon the TCP/IP stack provider, but if so, helps to keep a connection open during extended peroids of inactivity.

nagledelay/set_nagledelays flag

Parameter flag (a boolean)

(Note: Has an alternative alias with a "set_" prefix for older versions of ColdFusion which do not properly handle get/put COM names.)

Turns on or off the Nagle algorithm.

GET

NOTE: Due to the way this COM is implimented as a shell over a set of "safe" general purpose C++ classes, most properties are not available if you are not connected to a server.

is_completed

Returns a boolean

Boolean return of "false" or "true". Indicates whether the TCP session has completed normally. That is to say, that the remote server has sent the customary 0 byte packet signalling the end-of-file then closed the connection.

For example, if a connection is accidently lost (lightning strikes the power lines, someone unplugs your modem, etc) then connection will be set to "0" and completed will be "0". You will never get a "1" for completed in this case.

is_connected

Returns a boolean

Boolean return of "false" or "true". Indicates whether the connection is still currently active.

is_keepalive

Returns a boolean

Boolean return of "false" or "true". Indicates whether the "keepalives" are on or off.

is_nagledelay

Returns a boolean

Boolean return of "false" or "true". Indicates whether the Nagle delay algorithm is on or off.

local

Returns a string

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

remote

Returns a string

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

bytessent

Returns bytes, a long integer

The total number of bytes sent to the remote server.

bytesreceived

Returns bytes, a long integer

The total number of bytes received from the remote server.

socket

Returns an integer

The current socket (session) number being used.

Log

Returns a string

Gets list of all error messages accumulated since the object was created. Clear out the error cache with the ClearErrorLog method. Note that issuing an "Open" command automatically clears the error log as well.

byteslastreceived

Returns bytes, a long integer

Number of bytes of data received from last RECV* method.

byteslastsent

Returns bytes, a long integer

Number of bytes of data sent with last SEND* method.

timeout

Returns "seconds", a floating-point number

Returns the current timeout settings in milliseconds. This property is set by the *timeout* put property. By default it is usually 200 milliseconds.

See the general [overview](#) on TCP/IP for a discussion on how this affects operations.

last_timeout

Returns "seconds", a floating-point number

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.

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

is_cutoff (returns a boolean)

If "true" then the previous Recv stopped because it reached the prearranged cutoff limit. Otherwise "false".

cutoff (returns "bytes", a long integer)

Returns the cutoff, if set, in bytes. If unset this will be "0".

blocksize (returns "bytes", a long integer)

The current operating blocksize.

recv_packets (returns a long integer)

The number of times data has been requested from the remote server.

send_packets (returns a long integer)

The number of times data has been sent to the remote server.

recv_faults (returns a long integer)

The number of times when doing a recv* that a packet had to be requested from the network.

recvbuffer_faults (returns a long integer)

The number of times while doing a RecvCRLF or RecvFrame that more data had to be requested from the network.

stack_recv_buffer (returns a long integer)

The size of the TCP/IP stack's recv buffer.

stack_recv_buffer (returns a long integer)

The size of the TCP/IP stack's send buffer.

kbpsent (floating point number)

The computed kilobytes per second of data sent.

kbpsreceived (floating point number)

The computed kilobytes per second of data received.

kbpslastsent (floating point number)

The computed kilobytes per second of the last block of data sent.

kbpslastreceived (floating point number)

The computed kilobytes per second of the last block of data received.

ping (integer, milliseconds)

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.

instance (integer)

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 (integer)

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".

mutex_locks (integer)

In a multithreaded environment where you are using the same instance of this software there can be operation collisions resulting in garbled or intermixed data. Tcpclient as of 3.00 supports a thread-safe operation which blocks or stalls operations where a collision might occur. This number is the count of blocking waits that have been encountered. In normal use, this would always be zero. When used on multi-processor systems with the COM as an Application scope object under high stress the number may climb quite high.

mutex_sleep (integer)

See mutex_locks. Returns the total amount of milliseconds the COM object has sleep waiting for a mutex to unlock.

VERSION HISTORY

- 3.1 - 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.



- **3.0** - April 26th 2004.

After letting the project sit there a while, came back to it and decided to completely remove all the new message encoder/decoding classes and give them their own separate COM object. This COM should be lean and mean -- doing TCP only. Anything else belongs elsewhere. See MessageEncoderDecoder COM, now with it's own web page at **MessageEncoderDecoder**, for information on it. In practice though, you'll notice many of the examples for tcpclient require MessageEncoderDecoder to be loaded to operate properly though.

Also finished the Frame functions.

- **3.0** - April 3rd, 4th, 7th, 8th 2004.

After debugging the thread-safe issues made a few last minute changes and decided to go ahead and change the versioning number as well.

1. Mulls a complete rewrite of the documentation so as to be clearer.
2. Added extended winsock error descriptions.
3. Added a Nagle sleep to Recv functions where blocking would occur. Added counters that keep track of ms sleep times for both recv and send buffers. This can allow you to track down where some underperformance issues may be on your network.

From: "Andrew Finkenstadt" andyf@simutronics.com

Subject: RE: using TCPClient

Well, the one thing I didn't take special note of, is whether the .Send() or .SendRN() calls were protected by a critical section, to prevent simultaneous web page access.

...

1. Borrowed a Mutex class from another project, rewrote it and implimented a mutual exclusion mechanism. The original design of this COM essentially assumed each instance of it would be used in isolation. That is, when you created an instance of it, only one "user" would be using that instance at a time. The problem with this is that collision may ocur on multi-processor systems when used in an Application wide scope.

Besides the mutex code, this added the "mutexlocks" property.

2. Added "currentpacketize" property.
 3. Rebuilt interface as "Intrafoundation.Tcpclient.3".
 4. Cleaned up UU, XX, Base64 and hex functions. (Again.) Added new encode and decode functions and changed the way the SendFileEncoded and RecvFileEncoded (previously DecodeAsFile) functions work. As the reworked functions now handle emitting proper headers and multiple files the reserved functions to handle attachments were removed. (SendFileEncoded and RecvFileEncoded now do what they were originally intended to do.)
 5. Added Yenc support.
 6. Added MIME support.
 7. Changed "recv_cachemisses" to "recv_packets". Changed "send_cachemisses" to "send_packets". Changed "buffer_cachemisses" to "recvline_faults".
 8. Among many other things, after exploring some issues that had been bugging me for a while, made the code a **LOT** less apt to long timeouts when communicating with an open stream.
- **2.16** - March 26 - April 3rd 2004.

MAJOR update. What started out as a simple bug fix for two issues reported at the same time (BSTR NULL's and the CSV functions) turned into a complete rewrite of a lot of the base C++ code. So several things ended up being fixed or tweaked this go around. There has been extensive changes to the code in places so, if you've been using this software previously, you may wish to treat this release tentatively as a beta and double check it's operation before releasing on any production servers.

- A new C++ "Log" class was added and the old logging routines removed and replaced with it. The results may not be entirely obvious to the end-users beyond slightly more coherent error logging, but for anyone using the code-base itself as a core class of software that impliments a TCP/IP protocol client, this makes updating errors through the log much easier.
- Changed name of "Clear" function to "ClearErrorLog".
- "binarytext" class was replaced by "Base64", "UU" and "XX" classes. Also added the "Hex" class for conversion between hexadecimal formats.
- Reworked Send/SendRN/Recv/RecvRN to work with binary data (I think.) Specifically, I added a new partial BSTR handling class that now returns the TRUE length of any strings passed to and from the COM object.

Many of you may be aware that when sending or receiving strings from COM objects they use BSTRs (Basic Strings). BSTRs consist internally of the data string itself, an appended NULL (for C/C++ compatibility) and a prepended length counter (for BASIC compatibility). The problem is that most every standard (Microsoft) function for returning the length of these strings returns only the length up to the first NULL it encounters This is proper in C or C++ when dealing with standard text strings but a problem when you're sending around binary data.

To deal with this, code has been added which, as stated, returns the TRUE length of the string, including any embedded NULL's. (Or at least, that was the intent of the changes.)

- Added FlushRN. The buffer for RecvRN previously would be flushed (cleared, emptied, etc) automatically anytime you called Recv but because of internal code changes it made more sense to go ahead and allow the end-user to make this call whenever they themselves saw fit.
- Memory managment has been enhanced (tightened up) in several places. This may boost overall performance when doing intense transfers.
- Added "instance" and "instances" functions which returns the virtual instance number of the COM object. (See docs for explanation.)
- Mull calling this version 3.00 instead of 2.16.
- Added control over TCP/IP send/recv packet size via PacketSize functions. The default is 32768 bytes.
- Changed name of "error" to "errorlog".

TCPCliient Author: taesachi
URL: <http://msdn.pe.kr>

Created: 9/2/2003 10:23:56 PM

I`m Korean.

This component is not supported to another language. (ex. Korean)

So some change the source file.

But change only Send function.

Download

<http://msdn.pe.kr/TCPClient/tcpclientcom.zip>

Fixed. (I hope. Think. As I don't read Korean, you tell me.)

March 26 2004

Dear intrafoundation.com

I am trying to use your tcpclientcom (which seems like a great tag the way!) but for some reason SendCSV is only sending the first byte in the list.

Here's some sample code:

```
<cfset c=obj.Open("my IP","5300")>
<CFIF c IS "1">
<cfset obj.timeout=10.000>
<cfset page=obj.SendCSV("1,2,3,4,5,6,7,8,9")>
<CFINCLUDE TEMPLATE="obj.mod.cfm">
<cfset obj.Close()>
</CFIF>
```

The packet sniffer on the other computer is telling me that only the "1" is being sent. Am I doing something wrong?

Thanks!
Shawn

Fixed. An embarassing typo in the original code would cause this issue.

- **2.15** - July 20th 2003.

Slight cleanup and rebuild. Had to fiddle around a bit to get the Platform SDK to properly handle it.

- **2.14** - September 13rd 2001.

Added PING.

- **2.13** - September 3rd 2001.

Revised copyright text to clarify issues people had been asking me about.

While building UDPClient with TCPClient's project code over the weekend noticed the kbps speed functions were saying a kilo-byte was a 1,000 bytes, not 1,024 bytes as is proper. Oops. Fixed here in tcpclient as well.

- **2.12** - Feb 8th 2001.

Doc, example and source clean-up.

- **2.11** - Feb 3rd 2001.

2.11 consists mainly of bug fixes and performance tweaks. In some situations 2.11 is actually 30 times faster than 2.10. (Seriously.)

- Changed internal buffer size to 32768 bytes which gives about the best LAN performance on w2k according to tests
- Recoded the "connected" functions to be faster
- Added 20 millisecond delay to match the Nagle alg for buffer-full sends

- Fixed RecvRN bug that could cause a crash when remote server drops connection
- Added experimental kilobyte-per-sec stats for send/recv
- **2.10** - Feb 2nd 2001.

(Brought to you by Olivia Tremor Control.) A few minor tweaks, including memory/speed optimizations. In short, when doing large data transfers in the 1mb to 30mb range the tag was scaling very badly. It threatened to crash the machine on more than one occasion. This has been addressed.

Oh... So, it seems the uu encode/decode functions have never worked. Ever. :) I originally coded them completely from memory and never tested them. (Never needed to use them until this week.) Of course it turns out I forgot to code for the line length markers. Anyway... sorry about that. Recoded base64/uu routines. Fixed.

- Added blocksize for Recv.
- Added hascutoff indicator property.
- Changed default tcp buffer size from 512 bytes to 8192 bytes.
- Added several additional status properties relating to network buffer sizes.
- Changed error log to emits CRLF's instead of
 for line ends.
- Some misbehaving servers applications (such as Post.Office 3.5.3) have a habit of emitting spurious NULL's after a CRLF on the mime boundries resulting in Recv/RecvRN thinking there is no further text following. Addressed this.
- The UU and base64 code wasn't working properly. Fixed.
- Send failed with a 30mb file. Reworked to send large blocks of data.

I believe it's safe to say we're probably in a feature-lock now. There are a few undocumented or reserved functions that have yet to be implimented (mainly SSL) but those aside the emphasis from now on is debugging and optimization. The COM interface is locked at *Intrafoundation.TCPClient.1*.

There have been enough significant code changes to 2.10 that I'd almost thought of calling it a beta for a while. Bit of caution here.

- **2.9** - Jan 24th, 27th 2001.

After a few long emails with **Bryan Owen**, the tcp class underwent an intense low-level debugging session to root out the glitches in a new caching implimentation of RecvRN. While I was at it... there were several changes made in the fundamental tcp c++ class that's the heart of this COM object. (For those who haven't looked, this is a C++ project with two classes: The COM object tcpclientcom itself and a class called "tcp". The tcp class is used in several other projects, most commercial or for-hire. This COM is used as a sort of an open-source test-bed for that critical class.)

- Redid the documentation to make things *clearer*.
- Rewrote RecvRN to give it it's own tcp-level class so as to avoid forcing tcp/ip timeouts (ie, cache misses).
- Redid the pop3, smtp and ftp localhost tests to use RecvRN. Much smoother response time.
- The tcp class function send wasn't properly reporting the bytes sent.
- Several misc undocumented polishings to tcp class.
- Fixed sticky error log. Cleared on any open now.
- Fixed misreporting class file names.
- Added a simple, crude little news browser. I keep tripping over the folks at fuseware.com. I'm sure they love me. :)
- **2.8** - Jan 21st 2001.

Misc doc and script cleanup. Changed NNTP example to forums.macromedia.com (our new overlords).

- **2.7** - Jan 3rd 2001.

Misc. Polishing. Been meaning to replace all the place holding graphics with simple (2001: Space Odyssey-ish) white plastic theme. Small spherical indentation grids across top. Washed out by intense white light. Haven't had the time. Oh well.

HACKER WARNING: The DECODEASFILE can be exploited. Then again, this entire tag can be one big hacking exploit when in the right hands. But, just fyi in case you were thinking otherwise. You really shouldn't install this on a shared host. If you have ideas on how to modify it to have a "safe" mode (or, better yet, a conditionally compiled safe release version) feel free to express yourself.

- **2.6** - 12/24/2000.

Fixed cutoff. It was only checking when the remote server paused. Now checks after every packet.

Added RecvRN.

- **2.5** - 12/21/2000.

'ello.

It was intending to **GPL** this software, but as I was preparing a formal GPL notice it occurs to me that since it prohibits inclusion in proprietary software (which parts have already been used in and will be so again) it can not be so.

And thus I mispoke if I said it was to be so.

Sorry.

As to what "license" this software has... Not sure if any fit it. For now I really don't care what you do with it as long as you give proper credit to me (name, url, etc). I would prefer you keep it open source, but if you make significant changes you don't want to share, I understand. The only thing that would really annoy me is if you (like one unnamed creature with 1.x) change the headers and claim you wrote it. :-)

If you develop a nice script (in your native CF or ASP VBSCRIPT, etc) that's a front-end to a protocol (POP3, FINGER, etc), you're perfectly free to distribute the script with the COM object as long as you also include the full archive (the .zip), etc. I also encourage you to send useful completed scripts to me for future inclusion.

You may feel free to place this in any free online component sites. Saves me the hassle of having to do it myself.

Oh, and I changed the ClassID from TCP.Client to Intrafoundation.TCPClient. Meant to do that earlier, before beta. Slipped my mine. Also fixed it so you can use the version independant clsid "Intrafoundation.TCPClient" instead of "Intrafoundation.TCPClient.1".

Removed references to SENDSTRUCT and RECVSTRUCT.

Probably be a few other "settling" issues for a few more revisions. This is brand new code after all.

PAX.

- **2.4** - 12/21/2000. Non-beta. Clean-up.

- **2.3beta** - 12/20/2000. First wide-beta. Final documentation (and replacements for the little placeholder squiggles) remain. Send in your functioning scripts for inclusion in future revisions. BTW, as with 1.x I do essentially consider this software GPL'd. As far as I'm concerned you can pretty much do what you want with it, as long as you don't claim you wrote it yourself. :-)
- **2.2alpha** - 12/19/2000. First documented public release. Finished CSV functions. Only encoding functions, datalimit implementation and final documentation remains. Added to interface possible future additions that would support bit-level structs and SSL (Secure Socket Layer), aka HTTPS.
- **2.1alpha** - 12/18/2000. First (undocumented) public release.
- **2.0alpha** - 12/7/2000, SECOND EDITION. Brand new ATL COM, written completely in C++. This Second Edition of TCPClient completely replaces the old Allaire ColdFusion only version.
- **1.2** - Nov ? 2000. Just a repack with a different version number. Apparently 1.1 was reporting itself as a .14beta in some places, confusing some folks.
- **1.1** - 09/19,25/2000. **Erik Willsey** prompts me to make a few bug fixes. Was already moving that way anyway as I noticed it was twitchy under W2K. Added WAITFORSENDATA. Added TIMEOUT. Fixed a few things.

From: Erik Willsey

Sent: Tuesday, September 19, 2000 6:04 PM

Subject: RE: CFX_TCPClient Bug

I have found 4 bugs within your code regarding the WAITFORRECVDATA tag.

Request.cpp - Line 334:

You have:

```
WaitForRecvData( s, atoi(vWaitForRecvData)*1000);
```

Should Be:

```
WaitForRecvData( s, atoi(vWaitForRecvData));
```

You already specify that the parameter is in seconds, multiplying by 1000 greatly offsets the time. You may have been trying to set milliseconds; which in that case, this line is not wrong, but there is misdocumentation in the guidelines.

--

Request.cpp - Begin @ line 234

You have:

```
tv.tv_sec=0;
```

```
tv.tv_usec=200;
```

Should be:

```
tv.tv_sec=secs;
```

```
tv.tv_usec=0;
```

Again, this depends on whether your original intention was to have seconds or milliseconds. However, either way, the parameter for the time was not being used. Add that into the next bug, and an infinite loop is created.

--

Request.cpp - Begin @ line 240

You have:

```
while( nosocks == 0 )
```

```
{
```

```
FD_ZERO(&readfds);
```

```
FD_SET(sock,&readfds);
```

```
nosocks=select(FD_SETSIZE,&readfds,NULL,NULL,&tv);
```

```
}
```

Should be:

```
FD_ZERO(&readfds);
```

```
FD_SET(sock,&readfds);
```

```
nosocks=select(FD_SETSIZE,&readfds,NULL,NULL,&tv);
```

Reason: select() will return 0 when the time has elapsed (it will also immediately return zero if no time has passed). This means that if no data is ever received from the server, an infinite loop is created (well at least until the socket is closed by host or node). By simply removing the while loop, this bug is taken care of.

--

Request.cpp - Begin @ line 247

The remainder of that method should reside within an if statement checking that "nosocks" is not zero.

"nosocks" will only be zero if the time elapsed. By jumping into the while loop retrieving the data, another infinite loop is created, because recv() will always return 0 if there is no data in the socket's buffer.

I hope this helps you some. You've written a great tag.

Thanks,

Erik Willsey

Blitz Programming

- **1.0gamma** - 09/1/2000. The ancient beast awakes. Fixed a bug....
- **0.15beta** - 02/17/2000. Doc tweak.
- **0.14beta** - 02/11/2000. Bug report by **Jonathan**. No, besides that there have been no changes since the last version. However... a bit of work in a week might require something similiar so some advances might be made. If I know where the problems are. Send in feedback. Feature requests, etc.

Sent: Friday, February 11, 2000 12:51 PM

Subject: cfx_tcpclient bug fix

When I get " RECV Error - no response for %lds %lums "
(i.e. no more data to read) the recv CF variable will have
random crap in it since SocketRecv never appends
a NULL to recvp. So when SetVariable is called, it
copies memory from recvp's address until it does find
a NULL--somewhere else. Obviously this will cause problems
with some recv loops unless you code around it. :)

I fixed this by adding
*recvp=0;
after
recvp=(char *)malloc(1);
in SocketRecv() in Request.cpp.

Thanks for the code! :)

-Jonathan

- **0.13beta** - 01/07/2000. misc. examples & docs changes. contemplation of optional restriction security mode against *FILE functions and option to list ALLOWED ports (those not being listed, being those DENIED).
- **0.12beta** - 01/04/2000. Spawned project to CFX_TCPServer and removed all SERVER/LISTEN code. Went over project. Everything seems to work. More or less. Back to beta status. If you need IDENT for IRC protocol now you'll have to also install CFX_TCPServer when I finish it.
- **0.12alpha** - 01/04/2000. ENCODING functions finished. work now. Only LISTEN/ACCEPT code, and code/example/doc polishing remain. Some socket functions *getsockname*, etc are experiencing long timeouts sporatically if local DNS is overburdened. Will fix.
- **0.11alpha** - 01/04/2000. LISTEN and ENCODING function may not currently work at all and are in an ALPHA state. The CSV functions are slugs that need to be retooled at a lower level. Everything else should be BETA and working as far as I know. This somewhat premature release is so people can do further testing on the new "engine". Have fun. Next release should be full beta. Might get a chance to do that in 3 or 4 days.
- **0.10alpha** - 01/02/2000. not released to public. massive code rewrite. added csv SENDBINARYCSV and RECVBINARYCSV. removed temp internal 256k data limit. misc, etc. This version is possibly very unstable compared to the previous because of the massive rewrite. but very much more useful. caution. It can at least do PASV ftp to get file listings, etc now. the mime/uu/xx code hasn't really been tested. Mostly just typed it in from memory of the theory. It does compile without error though. so.... /-) I need to clean the mime/uu/xx up and then fix it so the tag can listen at a tcp port for full ftp, etc functionality. (ie, OPENSERVER and CLOSESERVER don't work yet.) I'm mulling over what to do with SENDFILE and RECVFILE to make the tag safer.... The tag goes back to *alpha* status for 0.10 because of an aggressive rewriting session last night that leaves much of the tag in an unstable state. However... several interesting new abilities are almost available now. Read the history.
- **0.9beta** - 12/27/1999. misc. If I have time I'll try to debug all the BINARY stuff by next time and add in a working binary ftp example. Someone want to do a vt100 telnet client? /-)
- **0.8beta** - 12/22/1999. minor tweaks.
- **0.7beta** - 12/22/1999. waiting for feedback before much else gets done. yawn.... spawned off a copy of the project for CFX_UDPClient. Back to paying work. If you ever want to see a .8, then send

feedback. :) This tag is still under development, but coming close to being stable. If you have feedback or want to donate* a cfm example for communicating with a certain TCP server type, then feel free.

*Especially those that say "under development" such as telnet, or qotd etc.

- **0.6beta** - 12/21/1999. first mostly stable release. essential feature-lock. donate working examples please so I can test the stability, especially those involving binary transfers. also ip's for public qotd, echo, etc sites would be nice.
- **0.0alpha** to **0.5alpha** - 12/20/1999-12/21/1999. Seriously began work on it again partially to understand TCP/UDP for linux-based network-related project. And because it was fun.
- **0.0** - 5/1999 made the project file. abandoned a few seconds later.

Intrafoundation Software
Making Atomic Warfare Fun Again

