Client / Server

Communications

Users Manual

(CSC_USR)

Version 7.1

January 11, 2018

This software is provided as-is.
There are no warranties, expressed or implied.

MarshallSoft Computing, Inc.
Post Office Box 4543
Huntsville AL 35815 USA

web: www.marshallsoft.com

TABLE OF CONTENTS

1 Introduction

The **Client / Server Communications Library** (CSC) is a component DLL library used to create server and client programs that can communicate with each other across any TCP/IP or UDP network such as the Internet or a private network (intranet or LAN [local area net]). The **CSC** component library uses the Windows sockets API for all communication.

A simple interface provides the capability to quickly develop Client/Server software applications using TCP/IP or UDP protocols.

CSC can be used to communicate with other **CSC** programs or used to communicate with other TCP programs such as DNS, POP3, SMTP, FTP, HTTP, etc. **CSC** provides the capability to connect to devices such as a relay device, scale device, GPS device or embedded computer device that is controlled by sending commands to its TCP IP address. **CSC** can also be used to encrypt and decrypt data and files and can also be used with the MarshallSoft AES Encryption Library (**AES**) for strong encryption.

Client/Server Communications functions can easily be called from any program written in any language (such as C/C++, C++ .NET, Visual C#, Delphi, Visual Basic, VB.NET, PowerBASIC, Visual FoxPro, dBase, Xbase++, COBOL) that is capable of calling Windows API functions. Both Win32 and Win64 DLLs are provided.

The User's Manual applies to the **Client/Server Communications Library (CSC)** for all supported programming languages. It discusses Client/Server TCP/IP processing, language independent programming issues, as well as purchasing and licensing information.

We have versions of the **Client/Server Communications SDK** for C/C++ (CSC4C), Visual Basic (CSC4VB), Visual FoxPro (CSC4FP), Delphi (CSC4D), and FoxPro (CSC4FP). Purchase a developer license for CSC for one software development language and use it with all others. All versions of CSC use the same DLLs (CSC32.DLL or CSC64.DLL); however, the examples provided for each version are written and tested for the specified computer development language. We also have declaration files and example programs for a few other languages.

The **Client/Server Communication DLLs** will work with 32-bit and 64-bit Windows: Windows XP through Windows 10.

Fully functional evaluation versions of our **Client/Server Communications** software components are provided so that the developer can test the CSC library in their environment. The evaluation version as well as a list of the many **Client/Server Communications** library features provided can be found at:

http://www.marshallsoft.com/client-server-communication.htm

1.1 Application Categories for CSC Programs

CSC program can be used in several classes of applications. Some ideas follow:

(1) **Client and Server Programs -** designed to communicate with each other.

- Chat server & client.
- File transfer server & client (such as an FTP replacement).
- Secure and private messaging.

(2) Client Programs - designed to talk to TCP servers on the net.

TCP Examples

- POP3 (or IMAP) client that gets # emails waiting on server.
- HTTP (web) client that downloads pages (or files) from web site.
- SMTP client that sends a simple text message.
- Control external devices via their TCP/IP ports (relays, scales, etc).

UDP Examples

- DNS Client that gets the MX email record from a DNS server.
- Network Time Client that gets current time from a Network Timer Server.

(3) Proxy Programs.

- SMTP proxy that extracts a copy of all recipient addresses for outgoing email.
- POP3 proxy that filter incoming email for spam.
- HTTP (web) proxy used to filter content.

1.2 Documentation Set

The complete set of documentation is provided in Adobe PDF format. This is the second manual (CSC_USR.PDF) in the set.

- CSC_4x Programmer's Manual (CSC_4x.PDF)
- CSC User's Manual (CSC_USR.PDF)
- CSC Reference Manual (CSC_REF.PDF)

The CSC_4x Programmer's Manual is the programming language specific manual. All language dependent programming issues including installation, compiling and example programs are discussed in this manual. The language specific manuals are as follows:

```
[NAME]        [DESCRIPTION]
CSC_4C     :  CSC Programmer's Manual for C/C++
CSC_4VB    :  CSC Programmer's Manual for Visual Basic
CSC_4D     :  CSC Programmer's Manual for Delphi
CSC_4FP    :  CSC Programmer's Manual for Visual FoxPro
CSC_4DB    :  CSC Programmer's Manual for Visual dBase
CSC_4XB    :  CSC Programmer's Manual for XBase++
```

The CSC User's Manual (CSC_USR.PDF) discusses client/server programming issues. License and purchase information is also provided. Read this manual after reading the CSC Programmer's Manual.

The CSC Reference Manual (CSC_REF.PDF) contains details on each individual CSC function.

All documentation can also be accessed online at
http://www.marshallsoft.com/client-server-communication.htm

1.3 Technical Support

We want you to be successful in developing applications using the Client/Server Communications Library! We are committed to providing the best, most robust library that we can. If you have any suggestions or comments, please let us know.

If you are having a problem using CSC, refer to Section 5.0 “Resolving Problems”. If you still cannot resolve your problem, email us at

`support@marshallsoft.com`

To avoid having your email **deleted** by our Spam scanners, begin the subject of your email with "CSC4C", “CSC4VB”, “CSC4D” or “MSC HELP”. Zip up any attachments and send plain ASCII text email only.

The latest versions of our products are available on our web site at

`http://www.marshallsoft.com`

and on our anonymous FTP site at

`ftp://ftp.marshallsoft.com/pub`

Registered users can update (for a period of one year) to the latest CSC DLLs at

`http://www.marshallsoft.com/update.htm`

1.4 How to Purchase

A developer license for the **Client/Server Communication Library (CSC)** may be purchased for $119 (USD) for electronic (email) delivery, or $199 (USD) with ANSII C source code for the DLLs. This price is good for one year from the release date.

The fastest and easiest way to order is on our web site at

http://www.marshallsoft.com/order.htm

You can also order by completing INVOICE.TXT (pro forma invoice) and emailing (info [at] marshallsoft.com), mailing (see our address at top), or faxing it to us. Our fax number will be provided upon request.

Multiple copy discounts (3 or more) and site licenses are available. Please call for details.

We accept American Express, VISA, MasterCard, Discover, PayPal, checks in US dollars drawn on a US bank, and International Postal Money Orders (such as Western Union).

Print the file INVOICE.TXT if a "Pro Forma" invoice is needed.

The registered package includes:

- CSC32 and CSC64 (for 64-bit programming environment) Library without the “evaluation info” screen.
- Free downloadable updates to the registered DLLs for one (1) year.
- Free technical support by email and telephone for one (1) year.

1.4.1 Academic Discount

We offer an "academic price" of 40% off the normal price for prepaid email orders to faculty and full time students currently enrolled in any accredited high school, college, or university. The software must be used for educational purposes. The academic discount does **not** apply to source code.

To qualify for the discount, your school must have a web site and you must have an email address at your school that is not forwarded. When ordering, ask for the "academic discount", or enter "student at" (or "faculty at") and your schools web site address (URL) in the comments field of the order form on our web site order page. Your order will be sent to your email address at your school.

This offer is not retroactive and cannot be used with any other discount. Products bought with academic pricing cannot be used for any commercial purpose nor can the CSC DLL be distributed.

1.4.2 Source Code

Source code is available for the purpose of re-compiling CSC32.DLL and CSC64.DLL. Source code for the DLL library is standard ANSI C. The source code is copyrighted by MarshallSoft Computing and may not be released in whole or in part.

There are two ways to order Source Code for the **Client/Server Communication Library SDK.**

(1) Source Code can be ordered at the same time as the Developer's License for $199 (for both).

(2) Source Code can be ordered within one year of purchasing a Developer's License for $100. After one year, a Developer's License update ($33, $55 or $77) must be purchased prior to purchasing the source code.

1.5 Updates

When a developer license is purchased for the **Client/Server Communications SDK**, the developer will receive the registered DLLs plus a license file (CSCxxxx.LIC) that can be used to update the registered DLLs (does not include source code) for a period of one year from purchase. Updates can be downloaded from

http://www.marshallsoft.com/update.htm

After one year, the developer license must be updated to be able to download updates. The developer license can be updated for $33 if ordered within one year from the original purchase (or previous update). After one year, licenses can be updated for $55 ($77 after 3 years). A license update includes an additional year of technical support.

Source code previously purchased may be updated for $40 in addition to the cost of the update ($33, $55 or $77).

Note that the registered CSC DLL **does not** expire.

Also see the file UPDATES.TXT.

1.6 Customer ID

The customer ID is the 4 or 5 digits following the product name (CSC) in the license file. For example, customer 12345 would receive license file **CSC12345.LIC**. Provide the Customer ID in the Subject field of an email when contacting us for technical support (CSC4C 12345).

1.7 License File

A license file, CSCxxxxx.LIC, where "xxxxx" is the 5 digit customer ID is provided with each developer license. The license file is an encrypted binary file used for updating CSC as explained in Section 1.5 "Updates". The license file is required in order to create (or update) the registered DLLs. The license file can be found in the /DLLS directory created after SETUP is run.

1.8 Distribution

In order to run an application (that calls CSC functions) on another computer, the file CSC32.DLL or CSC64.DLL must be copied to the Windows directory of the other computer. The Windows directory is normally C:\WINDOWS. Do not attempt to "register" the DLLs.

2 CSC Application Notes

2.1 Keycode

When a developer license is purchased, the developer will receive a new set of DLLs and a keycode for the CSC DLLs. Pass this keycode as the argument to **cscAttach**. The keycode will be found in the file named "KEYCODE". The keycode for the evaluation version is 0. The keycode for the registered version will be a unique 9 or 10 digit number. Note: The keycode is NOT the same as the Customer ID/Registration number.

2.2 Dynamic Link Library

The **Client / Server Communications Library SDK** includes a Win32 [CSC32.DLL] and Win64 [CSC64.DLL] dynamic link library (DLL).

The CSC object files CSC32.OBJ and CSC64.OBJ are also included for use with C/C++ programs.

2.3 GUI and Console Mode

CSC functions can be called from WIN32/WIN64 console mode programs as well as GUI programs. A "console mode" program is a Windows 95/98/Me/NT/2000/2003/2012/XP/VISTA/WIN7/WIN8 Win32/Win64 command line program running in a command window.

2.4 Getting Started Using CSC

The first **Client Server Communications** function that should be called is **cscAttach**, which initializes the CSC library and allocates necessary resources. The function **cscAttach** is typically called in the initialization section of a CSC application and should be called just once. The function **cscRelease** should be called before exiting the CSC application.

The best way to get familiar with CSC is to try out one of the example programs. The example programs are described in the CSC4C (C/C++), the CSC4VB (Visual Basic), the CSC4FP (Visual FoxPro) and the CSC4D (Delphi) Programmer's Manual:

- CSC_4C CSC Programmer's Manual for C/C++
- CSC_4VB CSC Programmer's Manual for Visual Basic
- CSC_4D CSC Programmer's Manual for Delphi
- CSC_4FP CSC Programmer's Manual for Visual FoxPro
- CSC_4XB CSC Programmer's Manual for Xbase++
- CSC_4DB CSC Programmer's Manual for Visual dBASE

After installing CSC, compile and run the CSCVER example program, which should display the CSC version number and registration string. Once CSCVER runs compile and run the SERVER and CLIENT example programs, which communicate with each other.

2.5 Development Languages Supported

We have versions of the **Client / Server Communications** (**CSC)** component library for C/C++ and .NET (CSC4C), Borland (Codegear) Delphi (CSC4D), Visual FoxPro (CSC4FP), Xbase++ (CSC4XB), dBASE (CSC4DB) and Visual Basic and VB.NET (CSC4VB). All versions of CSC use the same DLLs (CSC32.DLL or CSC64.DLL). Evaluation versions for these may be downloaded from our website at

http://www.marshallsoft.com/client-server-communication.htm

The **Client / Server Communications** DLL can also be used with any application written in any language capable of calling the Windows (95/98/Me, NT/2000/2003/2012/XP/VISTA/Win7/Win8/ x64) API.

2.5.1 Using CSC with Supported Languages.

Once a developer license is purchased for one programming language version of the **Client/Server Communication SDK (CSC)**, the same developer can use it with all other supported programming languages. Supported languages are C/C++, Visual Basic, Visual FoxPro, Xbase++, Visual dBASE and Borland/Codegear/Embarcadero Delphi.

For example, assume that the developer has previously downloaded and installed the registered version of CSC4C and now wants to also call CSC functions from Visual Basic.

1. Make a backup copy of CSC32.DLL (and CSC64.DLL) found in the Windows directory (\WINDOWS or \WINNT).
2. Download and install the evaluation version of CSC4VB (http://www.marshallsoft.com/csc4vb.htm) into a separate directory from CSC4C.
3. Compile and run the Visual Basic CSCVER example program found in the APPS directory created in step 2 above. It should display the pop-up evaluation screen.
4. Restore CSC32.DLL saved in step 1 above.
5. Paste the key code value found in (the registered version of CSC4C) KEYCODE.H into KEYCODE.BAS and KEYCODE.VB.
6. Run the Visual Basic CSCVER example program again. It should no longer display the pop-up screen.

A quicker and easier way would be to request multiple programming versions of CSC when a developer license is purchased. There is no additional charge.

2.5.2 Using CSC with Unsupported Languages

The **Client/Server Communications** functions can be called from any program written in any computer language capable of calling 32-bit or 64-bit Windows API functions.

Declaration files are available for the following languages:

[LANGUAGE]	[FILE]
C/C++ (& .NET)	CSC.H
Visual Basic (& VB.NET)	CSC32.BAS, CSC32.VB (VB.Net)
VBA (Excel, Access)	CSC32.BAS
PowerBASIC	CSC32.PBI
Borland Delphi	CSC32.PAS
Fujitsu COBOL	CSC32.CBI
Visual FoxPro	CSC32.FOX
Visual dBase	CSC32.CC
Alaska Xbase++	CSC32.CH

Additional declaration files will be added. Give us a call if a declaration file not listed above is needed. If you have interfaced CSC to an unusual language, email us the declaration file!

2.6 Converting From CSC Version 3 to Version 4

*The following applies **ONLY** to those running CSC version 3.0 or earlier.*

In order to provide the capability of one server program to be able to accept clients on more than one port concurrently, several changes were necessary to version 3 **CSC** functions.

Application programs that use the **Client/Server Communications** (**CSC)** library **Version 3.0 or earlier** will have to be changed as shown below. Notice that most of the changes apply to server applications only.

2.6.1 cscAttach

Server Programs:

```
Change cscAttach(CONNECTS + 1, KeyCode)
    To cscAttach(CONNECTS, 1, KeyCode)
```

Client Programs:

```
Change cscAttach(CONNECTS, KeyCode)
    To cscAttach(CONNECTS, 0, KeyCode)
```

2.6.2 cscServer

Retain the return value (virtual "listen" socket) from **cscServer** that will be used when calling **cscAwaitConnect**, **cscAcceptConnect**, and **cscConnectMessage**.

2.6.3 cscAwaitConnect

Server Programs:

```
Change cscAwaitConnect(Timeout)
    To cscAwaitConnect(Sock, Timeout)
      (where Sock is the value returned from cscServer)
```

2.7.4 cscAcceptConnect

Server Programs:

```
Change cscAcceptConnect()
    To cscAcceptConnect(Sock)
       (where Sock is the value returned from cscServer)
```

2.6.5 cscSendMessage

Server Programs:

```
Change cscSendMessage(Wnd, -1, MsgID)
    To cscConnectMessage(Wnd, Sock, MsgID)
      (where Sock is the value returned from cscServer)
```

Client Programs:

```
Change cscSendMessage(Wnd, Sock, MsgID)
    To cscDataMessage(Wnd, Sock, MsgID)
```

2.7 Example Client/Server Protocol

Several of the **Client/Server Communications Library** demonstration programs use the following example protocol:

(1) The server must be running first at a specified IP address using a specified port number known to both client and server. A host name may be used instead of an IP address. The server waits for a connection attempt by a client.

(2) The client attempts to connect to the server.

(3) The server accepts the connection from the client, and then sends its greeting message, such as:

"CSC Example Server"

(4) The client receives the server's greeting message.

(5) The client sends a request (command) string to the server.

(6) The server receives the client's request.

(7) The server sends back its response string.

(8) Repeat steps (5), (6), and (7) until done.

(9) The client closes its connection to the server.

The server responds with the following response strings when presented with the corresponding requests (REQ) from the client:

REQ	Response String	Request Example	Response Example
WHO	Sends name of the server.	WHO	W_SERVER
VER	Sends server version #.	VER	7.1
BYE	OK (then disconnects)	BYE	OK
ECH	Sends string after "ECHO "	ECH Hello	Hello

The above protocol is just an example. The programmer can create whatever protocol is required. Request strings can be any length, although it is best to keep them as short as possible.

Also see PROTOCOLS.TXT

2.8 Stream Data I/O

Like the Windows winsock I/O functions, the CSC data I/O functions

```
cscGetData
cscPutData
cscCryptoGetData
cscCryptoPutData
```

are stream oriented. This means that all incoming data will not necessarily be available at one time. For example, **cscPutData** may be called to write 10,000 bytes but **cscGetData** may have to be called several times before the entire 10,000 bytes are received.

The data I/O functions return the number of bytes received.

2.9 Packet Data I/O

The CSC packet I/O functions

```
cscGetPacket
cscPutPacket
cscCryptoGetPacket
cscCryptoPutPacket
```

send and receive variable length packets, varying from 1 to 10,000 bytes per packet. Unlike the CSC data I/O functions, there is a one-to-one correspondence between packets send and packets received. There is no such thing as a partial packet.

Because there is always one packet received for one packet transmitted, packet I/O is easier to code than (stream oriented) data I/O.

The packet I/O functions return the number of bytes received.

Note that CSC packets are not the same as TCP packets.

2.10 File Transfer

There are two sets of functions that copy files from one computer to another via a TCP connection: (1) cscPutFile / cscGetFile and (2) cscPutFileExt / cscGetFileExt, along with their crypto equivalents.

2.10.1 cscPutFile / cscGetFile (Standard File Transfer protocol)

Files are transmitted from the program calling **cscPutFile** to the program calling **cscGetFile**.

2.10.2 cscPutFileExt / cscGetFileExt (Extended File Transfer Protocol)

Files are transmitted from the program calling **cscPutFileExt** to the program calling **cscGetFileExt**, but only that part of the specified file that has not already been successfully received. If only part of a file is transferred (because of a network or computer malfunction), the file transfer will (when restarted at a later time) continue at the point that it left off. These functions are particularly useful when transferring large files.

2.11 Encryption

2.11.1 Overview

Encryption is performed by the following CSC functions:

```
cscCryptoPutData - Encrypts data as it is transmitted.
cscCryptoGetData - Decrypts data as it is received.
cscCryptoPutFile - Encrypts a file as it is transmitted.
cscCryptoGetFile - Decrypts a file as it is received.
cscCryptoPutPacket - Encrypts packet as it is transmitted.
cscCryptoGetPacket - Decrypts packet as it is received.
```

See the CSC Reference Manual (CSC_REF.PDF) for a description of each function.

In all cases, encryption is performed by XOR'ing the data or file with the contents of the "pad buffer", named after the cryptographic concept of the "one time pad".

For example, suppose we want to transmit the three characters "ABA". Recalling that the hex code is 41H for 'A' and 42H for 'B', and using (13H, 5CH, 29H) as the first 3 bytes in the pad buffer, the computation is thus

```
41H XOR 13H  = 52H
42H XOR 5CH  = 1EH
41H XOR 29H  = 68H
```

and the three bytes 52H, 1EH, and 68H are transmitted.

The message is recovered by applying the same pad (13H, 5CH, 29H) to the received message

```
52H XOR 13H  = 41H
1EH XOR 5CH  = 42H
68H XOR 29H  = 41H
```

2.11.2 Protocol Issues

The contents of the pad file passed to **cscCryptoPutData** and **cscCryptoGetData** must always be the same. That is, the contents of the pad file cannot be changed between calls to **cscCryptoPutData** and **cscCryptoGetData** unless the receiving side has read all of the data sent by the sending side.

The contents of the pad file passed to **cscPutPacket** and **cscGetPacket** can be changed, if desired, after each packet is sent and received. That is, the content of the pad buffer can be changed if desired, provided that the same pad buffer is used by both the sending side and the receiving side for each individual packet.

2.11.3 Populating the Pad Buffer

There are several ways to populate the pad buffer. Once easy way is to use a pseudo random number generator to generate 8-bit (0 to 255) random variates. The "key" is the random number seed. See the **cscFillRandom** function.

Also see the following section.

2.12 Using the AES (Advanced Encryption Standard) Module

To be the most secure, the CSC encryption pad buffer should be changed after each connection session. This is because if an adversary can get both the encrypted and decrypted copy of a file then the encryption pad buffer can be easily determined which can be used to decode all files using the same pad buffer. The solution is to change the pad buffer after each use.

One practical method for doing this is to create a random pad buffer - also known as the "session key". The session key is then encrypted with strong encryption such as the "Advanced Encryption Standard" before transmitting to the other side that recovers the session by decrypting it. In this way the encryption key is different for each connection session.

Using session keys the CSC logic would be:

CSC CLIENT:

1. Create a random session key.
2. Encrypt the session key.
3. Connect to the server.
4. Transmit the encrypted session key to server.
5. Use the session key as the "CSC encryption pad".

CSC SERVER:

1. Accept connection from CSC client.
2. Receive the encrypted session key from the client.
3. Decrypt the session key.
4. Use the session key as the "CSC encryption pad".

The **aesClient** and **aesServer** example programs requires the **MarshallSoft AES Encryption** module that can be downloaded from http://www.marshallsoft.com/aes.htm

2.13 Direct Device Control

Many external devices can be controlled by writing to their assigned IP address. Examples of such devices include Relay Controllers, Scales, GPS Devices, and embedded computers.

For example, suppose that you want to be able to turn on and off electrical equipment in your home or office by computer control. An internet search will find quite a few "relay controllers". To control them by using your computer, they will have one of several methods of control: RS232 serial, USB, and direct TCP/IP control.

To use RS232 serial or USB control, a serial communications library can be used such as our "Windows Standard Serial Communications Library". See http://www.marshallsoft.com/serial-communication-library.htm

To use CSC for direct device control, you must first know the assigned IP address and port number of the device. If your network uses dynamic assignment of IP addresses (DHCP), the IP address can be found by:

1. Looking in the router's DHCP table.

2. Searching the list of allowed IP addresses assigned by your router by attempting to connect (using cscClient) to each IP on the chosen port.

Once the IP address and port are known, the logic required to send a command (or commands) to the external device is as described in section 3.2 "Clients" with the exception that some external devices do not send greeting messages.

See the CONTROL example program.

3 TCP and UDP Sockets

There are two types of sockets: User datagrams (UDP sockets) and streams (TCP sockets). TCP sockets implement bi-directional data streams, guaranteed to be error free. UDP sockets are rarely used in network application programming. TCP is used for most application level net programming, including most of the example programs.

CSC provides three UDP functions: cscCreateUDP, cscPutUDP, and cscGetUDP. See the uEcho_s (UDP echo server) and uEcho_c (UDP echo client) example programs.

TCP/IP network programs can be classified into two types: servers and clients.

3.1 Servers

The server program must always be running before a client can attempt to connect to it. The server can be programmed to accept one or more connections concurrently.

3.1.1 Server Program Logic

The basic program logic for a server is as follows:

```
1. cscAttach        (attach CSC32.DLL or CSC64.DLL)
2. cscServer        (start server)
3. cscAwaitConnect  (wait for connection attempt from client)
4. cscAcceptConnect (accept connection from client)
5. cscPutData       (send greeting message to client)
LOOP
   6. cscGetData    (read command request from client)
   7. cscPutData    (send response to command request)
   GO TO LOOP until done
8. cscClose         (close connection to client}
9. cscRelease       (release CSC32.DLL or CSC64.DLL)
```

Note that the above logic accepts just one connection, although servers can be written with **CSC** that can handle multiple connections.

The **cscSendMessage** function can also be used to send a Windows message when a connection is ready to be accepted by the server, or when data is ready to be read. Obviously, **cscSendMessage** can be used only in programs written using a Windows message loop. Refer also to the server example programs.

3.1.2 Example Server Programs

Each **CSC** product (CSC4C, CSC4VB, CSC4D, and CSC4FP) contains several server and client programs.

See the “SERVER” example program (SERVER.C, SERVER.BAS, or SRV_PRJ.DPR), which accepts connections from the “CLIENT” example program.

Also refer to Section 4.0 "Challenge and Response Authentication".

3.2 Clients

The server program must always be running before a client can attempt to connect to it.

Although not too common, a single program can be written that acts as both a server and a client. In this case, a minimum of two channels (see **cscAttach, Section** 2.2 in the CSC Reference Manual (CSC_REF) must be allocated for the server part of the program and one channel for the client side.

3.2.1 Client Program Logic:

The basic program logic for a client is as follows:

```
1. cscAttach        (attach CSC32.DLL)
2. cscClient        (start client)
3. cscGetData       (read greeting message from server)
LOOP
   4. cscPutData    (send command to server)
   5. cscGetData    (read response from server)
   GO TO LOOP until done
6. cscClose         (close connection to server}
7. cscRelease       (release CSC32.DLL)
```

The **cscSendMessage** function can also be used to send a Windows message when data is ready to be read. Obviously, **cscSendMessage** can be used only in programs written using a Windows message loop. Also see the client example programs.

3.2.2 Example Client Programs

Each **CSC** product (CSC4C, CSC4VB, CSC4FP and CSC4D) contain several server and client programs.

See the “CLIENT” example program (CLIENT.C, CLIENT.BAS, CLIENT.FOX or CLI_PRJ.DPR), which connect to the “SERVER” example program.

Also refer to Section 4.0 "Challenge and Response Authentication".

4 Challenges and Response Authentication

The challenge / response authentication protocol is designed to prevent unauthorized clients from connecting to your server. Passwords are often used for this purpose, but can be stolen en route if transmitted without being encrypted. With the challenge and response protocol the server presents a question ("challenge") and the client must provide a valid answer ("response") to be authenticated.

The program logic for the challenge / response protocol is as follows:

SERVER

1. Server accepts connection from client.
2. Server generates an 8-byte "challenge" string by calling **cscChallenge**.
3. Server sends challenge string to client.
4. Server calculates the correct response by calling **cscResponse**.
5. Server waits for a response from the client; then reads the response.
6. Server drops client if response string is not equal to the computed response string.

CLIENT

1. Client connects to the server.
2. Client reads the challenge string from the server.
3. Client calculates the correct response by calling **cscResponse**.
4 Client sends response string to server.

The correct response is computed from the following 3 parameters by **cscResponse**.

1. Multiplier (any 32-bit number).
2. Mask (any 32-bit number).
3. Rotate count (from 0 to 31).

The same 3 values above must be used by both the client and the server.

The challenge string is the 8 byte hexadecimal representation of a random 32-bit value.

Each time a client attempts to connect to a server, a different challenge string will be computed (except for once ever 2^32 times). Thus, knowledge of previous challenge / response pairs will not help potential attackers.

See the AUTH_S and AUTH_C programs for an example of the use of the challenge / response protocol.

5 Resolving Problems

The evaluation version example programs must be run on the same machine as on which SETUP was run. If you run into problems running the evaluation version with Windows Server, contact us.

Be sure that the proper keycode is passed to **cscAttach**. Refer to Section 2.1 "Keycode.

Before attempting to write an application using CSC, run several of the example programs. Begin by running CSCVER which should display a pop-up window (for the evaluation version only) and also display the **CSC** version, build number and registration string.

If you cannot get your application to run properly, first compile and run the example programs. If you call us to report a possible bug in the library, the first thing we will ask is if the example programs run correctly. All example programs have been compiled and tested.

Test the code returned from all **CSC** functions. Then, call **cscErrorText** to get the text associated with the error code. All functions return an integer code. Negative values are always errors. Error codes returned by **CSC** functions can be found in the CSC declaration file and are also listed in Section 9.

For example (C Example):

```
Code = cscAttach(2, (long)CSC_KEY_CODE);
if(Code<0)
  {static char Buffer[64];
   cscErrorText(0,Code,Buffer,64);
   printf("Error %d: %s\n", Code, Buffer);
  }
```

Another good idea is to turn on logging by calling

```
cscSetString (-1, CSC_SET_LOG_FILE, (char *)"SERVER.LOG");    {C/C++}
cscSetString (-1, CSC_SET_LOG_FILE, "SERVER.LOG")             {VB}
```

If you encounter a problem that you cannot resolve, email us at info [at] marshallsoft.com. To avoid having your email deleted by our Spam scanners, begin the Subject with HELP CSC. Zip up any attachments and send plain ASCII text email only.

If you still get the evaluation screen (popup window) after purchasing a developer license, the problem is that Windows is finding the evaluation DLL before the registered DLL. The solution is to delete (or zip up) all evaluation versions of the CSC32.DLL or CSC64.DLL and run the SETUP program again.

If you get "error -74" when calling **cscAttach**, the problem is that the keycode passed to **cscAttach** does not match the keycode in the DLL's. This is caused by (1) using the evaluation keycode (value = 0) with the registered DLL, or (2) using the registered keycode with the evaluation DLL.

Review the CSC Programmer's and Reference Manuals. We also suggest reading Section 3 "Servers and Clients".

6 Versions of CSC

The Client / Server Communications (CSC) SDK library is available in three versions. All three versions have identical functionality.

6.1 Evaluation Version

The evaluation version can be differentiated from the other two versions by:

(1) The evaluation reminder screen is displayed at startup.

(2) The evaluation version may not be used for commercial purposes.

(3). The evaluation version can be tested for 30 days.

6.2 Academic Version

The academic version can be differentiated from the other two versions by:

(1) There is no evaluation reminder screen.

(2) The academic version may not be used for commercial purposes.

DLLs purchased with the academic discount may not be distributed, and must be used for educational purposes only.

6.3 Professional Version

The professional version can be differentiated from the other two versions by:

(1) There is no evaluation reminder screen.

(2) Source code may be purchased for CSC.

Compiled DLLs may be distributed royalty free with your compiled applications as specified by the software license. However, neither the License Key nor the source code (if purchased) to the DLLs may be distributed. The Professional version may be used for commercial purposes.

7 Legal Issues

7.1 License

This license agreement (LICENSE) is a legal agreement between you (either an individual or a single entity) and MarshallSoft Computing, Inc. for this software product (SOFTWARE). This agreement also governs any later releases or updates of the SOFTWARE. By installing and using the SOFTWARE, you agree to be bound by the terms of this LICENSE. If you do not agree to the terms of this LICENSE, do not install or use the SOFTWARE.

MarshallSoft Computing, Inc. grants a nonexclusive license to use the SOFTWARE to the original purchaser for the purposes of designing, testing or developing software applications. Copies may be made for back-up or archival purposes only. This product is licensed for use by only one developer at a time. All developers working on a project that includes a MarshallSoft Software SDK, even though not working directly with the MarshallSoft SDK, are required to purchase a license for that MarshallSoft product.

The "academic" registered DLL's may not be distributed under any circumstances, nor may they be used for any commercial purpose.

The "professional" registered DLL's may be distributed (without royalty) in object form only, as part of the user's compiled application provided the value of the Key Code (License Key) is not revealed. The registered DLL's may NOT be distributed as part of any software development system (compiler or interpreter) without our express written permission. The source code (CSC32.C) for the library is copyrighted by MarshallSoft Computing and may not be released or modified in whole or in part.

7.2 Warranty

MARSHALLSOFT COMPUTING, INC. DISCLAIMS ALL WARRANTIES RELATING TO THIS SOFTWARE, WHETHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND ALL SUCH WARRANTIES ARE EXPRESSLY AND SPECIFICALLY DISCLAIMED. NEITHER MARSHALLSOFT COMPUTING, INC. NOR ANYONE ELSE WHO HAS BEEN INVOLVED IN THE CREATION, PRODUCTION, OR DELIVERY OF THIS SOFTWARE SHALL BE LIABLE FOR ANY INDIRECT, CONSEQUENTIAL, OR INCIDENTAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE SUCH SOFTWARE EVEN IF MARSHALLSOFT COMPUTING, INC. HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES OR CLAIMS. IN NO EVENT SHALL MARSHALLSOFT COMPUTING, INC.'S LIABILITY FOR ANY SUCH DAMAGES EVER EXCEED THE PRICE PAID FOR THE LICENSE TO USE THE SOFTWARE, REGARDLESS OF THE FORM OF THE CLAIM. THE PERSON USING THE SOFTWARE BEARS ALL RISK AS TO THE QUALITY AND PERFORMANCE OF THE SOFTWARE.

Some states do not allow the exclusion of the limit of liability for consequential or incidental damages, so the above limitation may not apply to you.

This agreement shall be governed by the laws of the State of Alabama and shall inure to the benefit of MarshallSoft Computing, Inc. and any successors, administrators, heirs and assigns. Any action or proceeding brought by either party against the other arising out of or related to this agreement shall be brought only in a STATE or FEDERAL COURT of competent jurisdiction located in Madison County, Alabama. The parties hereby consent to in personam jurisdiction of said courts.

8 CSC Function Summaries

Refer to the **Client/Server Communication Reference Manual** (CSC_REF.PDF) for detailed information on the **CSC** functions. There are 55 functions in the Client/Server Communication Library.

cscAcceptConnect	Accept a connection from a client
cscAttach	Attaches CSC323.DLL (or CSC64.DLL)
cscAwaitConnect	Waits for a connection attempt by a client.
cscAwaitData	Waits for incoming data.
cscByteToShort	Converts from 8-bit ASCII to 16-bit Unicode ASCII
cscChallenge	Constructs challenge string.
cscClient	Attempts to connect to server.
cscClientExt	Attempts to connect to server on specific local port.
cscClose	Closes an open connection.
cscConnectMessage	Sends message when connection is ready to accept.
cscCreateUDP	Created UDP socket.
cscCryptoGetData	Reads (receives) encrypted data.
cscCryptoGetFile	Reads (receives) encrypted file.
cscCryptoGetFileExt	Reads (receives) encrypted (extended) file.
cscCryptoGetPacket	Reads (receives) encrypted data packet.
cscCryptoPutData	Writes (transmits) encrypted data.
cscCryptoPutFile	Writes (transmits) encrypted file.
cscCryptoPutFileExt	Writes (transmits) and encrypted (extended) file.
cscCryptoPutPacket	Writes (transmits) encrypted data packet.
cscDataCRC	Computes CRC for data buffer.
cscDataMessage	Sends Windows message when data is ready to read.
cscErrorText	Returns text of error code.
cscFileCRC	Computes the CRC of a file.
cscFileLength	Returns the length of a file.
cscFillRandom	Fills buffer with random bytes.
cscGetData	Reads (receives) data.
cscGetFile	Reads (receives) file.
cscGetFileExt	Reads (receives) file (extended).
cscGetInteger	Gets integer parameter.
cscGetPacket	Reads (received) data packet.
cscGetString	Gets string parameter.
cscGetUDP	Reads UDP datagram.
cscIsConnected	Return TRUE (non-zero) if connected.
cscLaunch	Starts an external executable.
cscMakeDotted	Makes dotted decimal string from IP address.
cscMakeDotted4	Makes dotted decimal string from 4 address components.
cscMulticast	Enables UDP multicast packets to be received.
cscNetToHost16	Converts 16-bit integer from network to host byte order.
cscNetToHost32	Converts 32-bit integer from network to host byte order.
cscPutData	Writes (transmits) data.
cscPutFile	Writes (transmits) file.
cscPutFileExt	Writes (transmits) file (extended).
cscPutPacket	Writes (transmits) data packet.
cscPutUDP	Write UDP datagram.
cscReadSize	Returns # bytes ready to be read.
cscRelease	Releases CSC32.DLL
cscResolve	Resolves a host address into a IP address.
cscResponse	Computes the correct response for a challenge string.
cscServer	Waits for a connection attempt by a client.
cscSetInteger	Sets integer parameter.
cscSetString	Sets string parameter.
cscShortToByte	Converts from 16-bit Unicode ASCII to 8-bit ASCII.
cscSleep	Sleeps specified milliseconds.
cscSystemTics	Returns system tics (milliseconds since boot-up).
cscTestDotted	Test format of dotted IP address.

9 CSC Error Return List

-10004: Interrupted system call.
-10009: Bad file number.
-10013: Access denied.
-10014: Bad address.
-10022: Invalid argument.
-10024: Too many open files.
-10035: Would block socket in non-blocking mode.
-10036: Blocking call already in progress.
-10037: Operation already completed.
-10038: Not a valid socket
-10039: Destination address required.
-10040: Message too big for buffer.
-10041: Protocol mismatch.
-10042: Protocol option invalid.
-10043: Protocol not supported.
-10044: Socket type not supported.
-10045: Socket operation not supported.
-10047: Socket address family not supported.
-10048: Socket address already in use.
-10049: Socket address not available.
-10050: Network error.
-10051: Cannot reach network.
-10052: Connection dropped.
-10053: Connection timed-out or aborted.
-10054: Connection reset by remote host.
-10055: Out of buffer space.
-10056: Socket already connected.
-10057: Socket not connected.
-10058: Socket functionality shut down.
-10060: Timed-out attempting to connect.
-10061: Connection refused by remote host.
-10064: Host is down
-10065: No route to host
-10091: Network not yet ready.
-10092: WINSOCK doesn't support requested version.
-10093: Sockets not initialized. Call WSAStartup.
-11001: Host does not exist.
-11002: Host not found. Try again.
-11003: Non-recoverable error has occurred.
-11004: No data is available.

```
   -1: EOF.
   -2: CSC aborted.
   -3: CSC accept error.
   -4: CSC already attached
   -5: Cannot comply.
   -6: No such socket.
   -7: Connect error.
   -8: Listen error.
   -9: No such host.
  -10: CSC not attached.
  -11: NULL argument.
  -12: NULL pointer.
  -13: Cannot allocate memory
  -14: Buffer size error
  -15: Packet CRC error
  -18: Too many sockets.
  -19: File format protocol error.
  -20: File name only.
  -21: Packet timeout.
  -22: Packet error.
  -23: Transfer cancelled.
  -24: File too large
  -25: No listen socket
  -26: Argument out of range
  -27: Data size too large
  -28: Timed out during connect.
  -29: Maximum packet size exceeded.
  -30: Cannot resolve name.
  -74: Bad key code.
  -75: Bad file offset.

-1001: No socket address.
-1002: No free sockets.
-1003: No such vsock.
-1004: Bad status flag.
-1005: Invalid socket.
-1006: No such parameter.
-1007: Cannot comply.
-1008: String size error.
-1009: No such server.
-1010: Buffer length error.
-1011: Connect error.
```