Client / Server

Communications

Reference Library

(CSC_REF)

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 library of functions used to create **server** and **client** programs that can communicate with each other across any TCP/IP network such as the Internet or a private network (intranet or LAN [local area net]). Refer to the **CSC User's (**CSC_USR.PDF) for information on the **CSC SDK.** The **Client/Server Communication Library** will work with 32-bit and 64-bit Windows: Windows XP through Windows 10.

1.1 General Remarks

All functions return an integer code. Negative values are always an error. See "CSC Error Codes" in Section 3. Non-negative return codes are never an error.

Note that the **cscErrorText** function is used to get the text message associated with any error code.

Each function argument is marked as:

- `(I) : 4-byte integer (Win32/Win64).`
- `(S) : 2-byte short integer (Win32/Win64).`
- `(P) : 4-byte pointer (Win32/Win64).`

Refer to the declaration files (see Section 1.3 below) for the exact syntax of each **CSC** function. Also note that the example programs show exactly how **CSC** functions are called.

All network functions are TCP unless specifically noted as for UDP.

For the latest version of the **CSC** software, see
http://www.marshallsoft.com/client-server-communication.htm

1.2 Documentation Set

The complete set of documentation is provided in Adobe PDF format. This is the third manual (CSC_REF.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 Declaration Files

The exact syntax for calling **CSC** functions is specific to the host language (C/C++, Delphi, VB, etc.) and is defined for each language in the "**CSC** declaration files". Each Client/Server Communications Library product comes with the appropriate declaration file for the supported language. For example,

```
CSC4C    C/C++, C++ .NET           CSC.H
CSC4VB   Visual Basic              CSC32.BAS
         VB.NET                    CSC32.VB
         VBA (EXCEL,ACCESS,etc.)   CSC32.BAS
CSC4D    Borland Delphi            CSC32.PAS
CSC4FP   Visual FoxPro             CSC32.FOX
CSC4XB   Xbase++                   CSC32.CC
CSC4DB   Visual dBase              CSC32.CH
```

We also have declaration files (and some example programs) for PowerBASIC and Fujitsu COBOL.

1.4 Language Notes

All language versions of **CSC** include the example program CSCVER. Refer to this program and the declaration file as defined in Section 1.3 above to see how **CSC** functions are called. The CSCVER program is also the first program that should be compiled and run.

The best way to see how a function is called is to find it used in one of the example programs. All **CSC** functions are used in one or more examples.

See "Using CSC with Supported Languages" in the CSC User's Manual (CSC_USR.PDF)

1.4.1 C/C++/C#

Project files and/or makefiles supplied for the example programs. **CSC** supports all 32-bit and 64-bit versions of Microsoft Visual C/C++, Visual C++ .NET and Visual C#, and 32-bit Borland C/C++, Borland C++ Builder, Watcom C/C++, Win32-LCC, Digital Mars, and MinGW C++.

1.4.2 Delphi

Functions defined in the Delphi Unit CSCW.PAS begin with "f" rather than "csc".

All versions of 32-bit and 64-bit Delphi through Delphi XE4 are supported.

1.4.3 Visual Basic (and VB.NET)

All versions of Visual Basic are supported through VB.NET (Visual Studio 2012).

1.4.4 Visual FoxPro

All strings passed to CSC functions must be prefixed with the '@' character. All versions of 32-bit Visual FoxPro are supported.

1.4.5 Visual dBase

CSC works with all versions of Visual dBase.

1.4.6 Xbase++

Functions defined for Xbase++ begin with 'X'. All strings passed to **CSC** functions must be prefixed with the '@' character.

2 CSC Functions

2.1 cscAcceptConnect :: Accept Connection from Client.

SYNTAX

```
cscAcceptConnect(vSock)

  vSock  : (I) Virtual (listener) socket number.
```

REMARKS

This function is used by the server to accept a connection from a client. **cscAcceptConnect** returns the (virtual) socket number that must be used in all subsequent calls to **CSC** functions that have a socket argument. Note that **cscAcceptConnect** can only be used in a server application.

EXAMPLE (C/C++ and VB)

```
Code = cscAcceptConnect(vSock)

  vSock  : (I) Virtual (listener) socket number.
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : Virtual data socket.
```

2.2 **cscAttach** :: Initializes the CSC DLL.

SYNTAX

```
cscAttach(DataSocks, ListenSocks, KeyCode)

  DataSocks   : (I) Number of (data) virtual communications sockets.
  ListenSocks : (I) Number of (listen) virtual communications sockets.
  KeyCode     : (I) Key code.
```

REMARKS

The **cscAttach** function must be the first **CSC** function called, and is used to pass the keycode (assigned when the library is purchased) and the number of sockets to allow for both data and for listening for new connections.

For client applications, 'DataSocks' is the number of servers that the clients wants to connect to concurrently. This value is normally one. 'ListenSocks' is always zero for client applications.

For server applications, 'DataSocks' is the number of concurrent data connections (to clients) that are to be supported concurrently, and 'ListenSocks' is the number of different ports that a server wants to be able to accept connections to concurrently. This value is normally one.

EXAMPLE (C/C++ and VB)

```
// allow 1 data port and no listening ports, with keycode 0
Code = cscAttach(1, 0, 0)
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : No error.
```

2.3 **cscAwaitConnect** :: Wait for Connection Attempt from Client.

SYNTAX

```
cscAwaitConnect(vSock, Timeout)

  vSock    : (I) Virtual (listen) socket number.
  Timeout  : (I) Timeout value in milliseconds.
```

REMARKS

This **cscAwaitConnect** function is used by a server program to wait a maximum specified time (in milliseconds) for a connection to be accepted from a client program.

EXAMPLE (C/C++)

```
     if(cscAwaitConnect(ListenSock, 1000))
       {// accept connection from client
        DataSock = cscAcceptConnect(ListenSock);
```

EXAMPLE (VB)

```
     if cscAwaitConnect(ListenSock, 1000) <> 0 Then
       'accept connection from client
       DataSock = cscAcceptConnect(ListenSock)
```

RETURNS

```
Returns TRUE  : Client attempting to connect.
Returns FALSE : No connection attempt detected.
```

2.4 **cscAwaitData** :: Wait for Incoming Data.

SYNTAX

```
cscAwaitData(vSock, Timeout)

   vSock    : (I) Virtual (data) socket number.
   Timeout  : (I) Timeout value in milliseconds.
```

REMARKS

This **cscAwaitData** function is used to wait a specified time (in milliseconds) for incoming data from either a client or a server application.

This function should normally be called before calling **cscGetData**, **cscGetFile, or cscGetPacket.**

EXAMPLE (C/C++)

```
char Buffer[128];
// wait (up to 2 seconds) for incoming data
if(cscAwaitData(DataSock, 2000))
  {// read data
   Code = cscGetData(DataSock, (char *)Buffer, 128);
```

EXAMPLE (VB)

```
Dim Buffer As String * 128
' wait (up to 2 seconds) for incoming data
if cscAwaitData(DataSock, 2000) <> 0 Then
  ' read data
  Code = cscGetData(DataSock, Buffer, 128)
```

RETURNS

```
Returns TRUE  : Data is ready to be read.
Returns FALSE : No data is ready.
```

2.5 **cscByteToShort** :: Converts 8-bit Character Buffer to 16-bit

SYNTAX

```
cscByteToShort(Buffer)

  Buffer : (P) character buffer
```

REMARKS

The **cscByteToShort** function converts the (null terminated) character buffer 'Buffer' from 8-bit ASCII characters to 16-bit Unicode ASCII characters.

The buffer must be null terminated (last character is a hex 00) and the buffer must be at least twice the size (in bytes) of the character string (since 16-bit characters require twice the space as 8-bit characters).

This function is only necessary when working with 16-bit Unicode ASCII characters in Visual C# and Delphi 2005/2007/2009.

RETURNS

None.

EXAMPLE (C#)

Refer to the Visual C# example ClientCS.csproj

```
char[] UnsafeBuffer = new char[128];
// get the registration string
fixed (char* pBuffer = UnsafeBuffer)
Code = cscGetString(-1, CSC_GET_REGISTRATION, pBuffer, 51);
if(Code>0)
  {// convert (null terminated) UnsafeBuffer[] to 16-bit chars (unicode)
   fixed (char* pBuffer = UnsafeBuffer)
   cscByteToShort(pBuffer);
  }
```

ALSO SEE

cscShortToByte

2.6 **cscChallenge** :: Construct Challenge String

SYNTAX

```
cscChallenge(Buffer)

  Buffer  : (P) Buffer into which the challenge string is copied.
```

REMARKS

The **cscChallenge** function is used to construct a random 8-byte challenge string to be used to challenge the client before continuing with the connection.

The purpose of the **cscChallenge** and **cscResponse** functions is to create a challenge / response protocol in order to defeat hackers who may try to connect to your server.

Also see **cscResponse**, which computes the response to the challenge string.

EXAMPLE (C/C++)

```
char Challenge[9];   // challenge string
// create random 8-digit number
Code = cscChallenge((char *)Challenge);
```

EXAMPLE (VB)

```
Dim Challenge As String * 9
' create random 8-digit number
Code = cscChallenge(Challenge)
```

RETURNS

Returns the length of the challenge string, which is always 8.

2.7 **cscClient** :: Starts the Client.

SYNTAX

```
cscClient(ServerName, ServerPort)

  ServerName : (P) Server name.
  ServerPort : (I) Server port.
```

REMARKS

The **cscClient** is function is used to start the client which attempts to connect to the server and port specified. Note that the server must already be running before a connection can be established. **cscClient** returns the (virtual) socket number that must be used in all subsequent calls to **CSC** functions that have a socket argument. Note that **cscClient** can only be used in a TCP client application.

EXAMPLE (C/C++)

```
char *HostName = "10.0.0.6";
short HostPort = 5001;
int DataSock;
// attempt to connect to server
DataSock = cscClient(HostName, HostPort);
```

EXAMPLE (VB)

```
Dim HostName As String
Dim HostPort As Integer
Dim DataSock As Integer
HostName = "10.0.0.2"
HostPort = 5001
' attempt to connect to server
DataSock = cscClient(HostName, HostPort)
```

RETURNS

```
>= 0 : Virtual (data) socket number
else : Error (See error list)
```

2.8 **cscClientExt** :: Starts the Client.

SYNTAX

```
cscClient2(ServerName, ServerPort, LocalName)

  ServerName : (P) Server name.
  ServerPort : (I) Server port.
```

REMARKS

The **cscClient2** is function is used to start the client which attempts to connect to the server and port specified, binding the data socket to the IP address.

Note that the server must already be running before a connection can be established. **cscClientExt** returns the (virtual) socket number that must be used in all subsequent calls to **CSC** functions that have a socket argument. Note that **cscClientExt** can only be used in a TCP client application.

EXAMPLE (C/C++)

```
char *HostName = "10.0.0.6";
char *LocalName = "10.0.0.14";
short HostPort = 5001;
int DataSock;
// attempt to connect to server
DataSock = cscClient(HostName, HostPort, LocalName);
```

EXAMPLE (VB)

```
Dim HostName As String
Dim LocalName As String
Dim HostPort As Integer
Dim DataSock As Integer
HostName = "10.0.0.2"
LocalName = "10.0.0.14";
HostPort = 5001
' attempt to connect to server
DataSock = cscClient2(HostName, HostPort, LocalName)
```

RETURNS

```
>= 0 : Virtual (data) socket number
else : Error (See error list)
```

2.9 **cscClose** :: Closes a Connection.

SYNTAX

```
cscClose(vSock)

  vSock : (I) Virtual socket number.
```

REMARKS

The **cscClose** function closes a connection previously opened. Call this function only to close an open connection.

EXAMPLE (C/C++ and VB)

```
     Code = cscClose(vSock)
```

RETURNS

```
 < 0  : Error. See error list.
>= 0  : No error.
```

2.10 **cscConnectMessage** :: Sends Windows Message (on Connect).

SYNTAX

```
cscConnectMessage(Handle, vSock)

  Handle   : (I) Windows Handle
  vSock    : (I) Virtual listen socket.
  Message  : (I) Message (usually WM_USER)
```

REMARKS

The **cscConnectMessage** function sends a Windows message to the window with handle 'Handle' when a connection from a client is ready to be accepted.

Refer to the windows server example programs.

EXAMPLE (C/C++)

```
// Send WM_USER message when connection is ready to connect
cscConnectMessage(hMainWnd, ListenSock, WM_USER);
```

EXAMPLE (VB)

```
Const WM_LBUTTONDOWN = &H201
' send WM_LBUTTONDOWN message when connection is ready to connect
Code = cscConnectMessage(Client.bReady.hWnd, ListenSock, WM_LBUTTONDOWN)
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : No error.
```

2.11 **cscCreateUDP** :: Creates UDP Socket.

SYNTAX

```
cscCreateUDP(LocalPort, LocalHost)

  LocalPort : (I) Local port
  LocalHost : (I) Local host name.
```

REMARKS

The **cscCreateUDP** function creates a UDP socket that is used by **cscGetUDP** and **cscPutUDP** functions only. The UDP socket is closed with **cscClose**.

Refer to the uEcho_S (UDP server) and uEcho_C (UDP client) example programs.

EXAMPLE (C/C++)

```
    // Create UDP socket
    vSock = cscCreateUDP(ECHO_PORT, (char *)"127.0.0.1");
```

EXAMPLE (VB)

```
    ' Create UDP socket
    LocalHost = "127.0.0.1"
    vSock = cscCreateUDP(ECHO_PORT, LocalHost)
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : No error.
```

2.12 **cscCryptoGetData** :: Receives Encrypted Data.

SYNTAX

```
cscCryptoGetData(vSock, Buffer, BufLen, PadChars, PadLen)

  vSock   : (I) Virtual (data) socket.
  Buffer  : (P) Buffer into which to copy bytes.
  BufLen  : (I) Length of above buffer.
  PadChars: (P) Buffer of "pad" bytes.
  PadLen  : (I) Length of above 'pad' buffer.
```

REMARKS

The **cscCryptoGetData** function is used to receive encrypted data from an established connection. All available data is copied to 'Buffer' up to a maximum of 'BufLen'. **cscCryptoGetData** never waits for data.

The received bytes in 'Buffer' have been XOR'ed with the bytes in 'PadChars' . If PadLen < BufLen, the XOR'ing continues in Buffer at the beginning of the pad buffer (wrap around).

Also read Section 2.11, "Encryption", in the CSC User's Manual (CSC_USR.PDF).

EXAMPLE (C/C++)

```
char Temp[128];
// read data (up to 128 bytes)
Code = cscCryptoGetData(DataSock, (char *)Temp, 128,
                        (char *)PadChars, PadLen);
```

EXAMPLE (VB)

```
Dim Temp As String
Temp = Spaces(128)
' read data (up to 128 bytes)
Code = cscCryptoGetData(DataSock, Temp, 128, PadChars, PadLen)
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : Number of bytes copied to Buffer.
```

2.13 **cscCryptoGetFile** :: Receives an Encrypted File.

SYNTAX

```
cscCryptoGetFile(vSock, FileName, PadChars, PadLen)

  vSock    : (I) Virtual (data) socket number.
  Filename : (P) Filename (not path) for incoming file or NULL.
  PadChars : (P) Buffer of "pad" bytes.
  PadLen   : (I) Length of above 'pad' buffer.
```

REMARKS

The **cscCryptoGetFile** function is used to receive an encrypted file from an established connection transmitted from the **cscCryptoPutFile** function.

The received bytes in 'Buffer' have been XOR'ed with the bytes in 'PadChars' . If PadLen < BufLen, the XOR'ing continues in Buffer at the beginning of the pad buffer (wrap around).

Specify the filename under which the file is to be saved, or specify an asterisk ("*") to save under the filename sent with the file. Then call **cscCryptoGetFile** repeatedly in a loop until 0 is returned.

The incoming data stream will contain the filename and the length of the file, followed by the (binary) file itself. The filename can be recovered by use of the **cscGetString** function and the file length can be recovered by use of the **cscGetInteger** function.

Also see **cscSetString** `(-1, CSC_SET_FILE_PATH,...)` to specify the file directory into which the file is to be saved.

Also read Section 2.11, "Encryption", in the CSC User's Manual (CSC_USR.PDF).

EXAMPLE (C/C++)

```
// prepare to receive file, saving to disk
Code = cscCryptoGetFile(DataSock, (char *)"*",(char *)PadChars, PadLen);
// receive file in 4K blocks
while(Code!=0) Code = cscCryptoGetFile(DataSock, NULL,
                      (char *)PadChars, PadLen);
```

EXAMPLE (VB)

```
' prepare to receive file, saving to disk
X = "*" + Chr(0)
Code = cscCryptoGetFile(DataSock, X, PadChars, PadLen)
// receive file in 4K blocks
EmptyString = Chr(0)
While Code <> 0
  Code = cscCryptoGetFile(DataSock, EmptyString, PadChars, PadLen)
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : No error.
```

2.14 **cscCryptoGetFileExt** :: Receives an Encrypted File (Extended).

SYNTAX

```
cscCryptoGetFileExt(vSock, FileName, PadChars, PadLen)

  vSock    : (I) Virtual (data) socket number.
  FileFlag : (I) T: setup, F: send data packets.
  PadChars : (P) Buffer of "pad" bytes.
  PadLen   : (I) Length of above 'pad' buffer.
```

REMARKS

The **cscCryptoGetFileExt** function is used to receive an encrypted file from an established connection transmitted from the **cscCryptoPutFileExt** function. If the file exists on the receiver and has the same size as on the sender, the file is not sent again.

The received bytes in 'Buffer' have been XOR'ed with the bytes in 'PadChars'. If PadLen < BufLen, the XOR'ing continues in Buffer at the beginning of the pad buffer (wrap around).

Specify the filename under which the file is to be saved, or specify an asterisk ("*") to save under the filename sent with the file. Then call **cscCryptoGetFileExt** repeatedly in a loop until 0 is returned.

The incoming data stream will contain the filename and the length of the file, followed by the (binary) file itself. The filename can be recovered by use of the **cscGetString** function and the file length can be recovered by use of the **cscGetInteger** function.

The **cscCryptoGetFileExt** function transfers only that part of the specified file that has not already been transferred. If only part of a file is transferred (because of a network or computer malfunction), the function will resume the file transfer without having to send the entire file over again.

Also see **cscSetString** `(-1, CSC_SET_FILE_PATH,...)` to specify the file directory into which the file is to be saved.

Also read Section 2.11, "Encryption", in the CSC User's Manual (CSC_USR.PDF).

EXAMPLE (C/C++)

```
// prepare to receive file, saving to disk
Code = cscCryptoGetFileExt(DataSock, 1,(char *)PadChars, PadLen);
// receive file in 4K blocks
while(Code!=0)
   Code = cscCryptoGetFileExt(DataSock, 0,(char *)PadChars, PadLen);
```

EXAMPLE (VB)

```
' prepare to receive file, saving to disk
Code = cscCryptoGetFileExt(DataSock, 1, PadChars, PadLen)
' receive file in 4K blocks
While Code <> 0 Code = cscCryptoGetFileExt(DataSock, 0, PadChars, PadLen)
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : No error.
```

2.15 **cscCryptoGetPacket** :: Receives an Encrypted Packet.

SYNTAX

```
cscCryptoGetPacket(vSock, Buffer, BufLen, PadChars, PadLen)

  vSock   : (I) Virtual (data) socket.
  Buffer  : (P) Buffer into which to copy bytes.
  BufLen  : (I) Length of above buffer.
  PadChars: (P) Buffer of "pad" bytes.
  PadLen  : (I) Length of above "pad" buffer.
```

REMARKS

The **cscCryptoGetPacket** function is used to receive the next encrypted packet from an established connection. The next packet is copied to 'Buffer' up to a maximum of 'BufLen'. **cscCryptoGetPacket** waits for a maximum of 10 seconds (default) for the next packet..

Packets can vary from 1 to 10,000 bytes in length. **cscCryptoGetPacket** will not return until the entire packet has been received.

The received bytes in 'Buffer' have been XOR'ed with the bytes in 'PadChars' . If PadLen < BufLen, the XOR'ing continues in Buffer at the beginning of the pad buffer (wrap around).

Also read Section 11, "Encryption", Section 2.8, "Stream Data I/O" and Section 2.9, "Packet Data I/O" in the CSC User's Manual (CSC_USR.PDF).

EXAMPLE (C/C++)

```
char Temp[128];
// read data (up to 128 bytes)
Code = cscCryptoGetPacket(DataSock, (char *)Temp, 128,
                          (char *)PadChars, PadLen);
```

EXAMPLE (VB)

```
Dim Temp As String * 128
' read data (up to 128 bytes)
Code = cscCryptoGetPacket(DataSock, Temp, 128, PadChars, PadLen)
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : Number of bytes copied to Buffer (packet size).
```

2.16 **cscCryptoPutData** :: Transmits Encrypted Data.

SYNTAX

```
cscCryptoPutData(vSock, Buffer, BufLen, PadChars, PadLen)

  vSock     : (I) Virtual (data) socket number.
  Buffer    : (P) Data to write.
  BufLen    : (I) Length of above buffer (# bytes to write).
  PadChars  : (P) Buffer of "pad" bytes.
  PadLen    : (I) Length of above 'pad' buffer.
```

REMARKS

The **cscCryptoPutData** function is used to write (transmit) an encrypted buffer to an established connection.

The bytes in 'Buffer' are XOR'ed with the bytes in 'PadChars' before sending. If PadLen < BufLen, the XOR'ing continues in Buffer at the beginning of the pad buffer (wrap around).

Also read Section 2.11, "Encryption", in the CSC User's Manual (CSC_USR.PDF).

EXAMPLE (C/C++)

```
char *Temp = "Hello";
// write 5 bytes of data
Code = cscCryptoPutData(DataSock, (char *)Temp, 5,
                        (char *)PadChars, PadLen);
```

EXAMPLE (VB)

```
Dim Temp As String
Temp = "Hello"
' write 5 bytes of data
Code = cscCryptoPutData(DataSock, Temp, 5, PadChars, PadLen)
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : Number of bytes written.
```

2.17 **cscCryptoPutFile** :: Transmits an Encrypted File.

SYNTAX

```
cscPutFile (vSock, FileName, PadChars, PadLen)

  vSock     : (I) Virtual (data) socket number.
  FileName  : (P) Filename (not path) to transmit.
  PadChars  : (P) Buffer of "pad" bytes.
  PadLen    : (I) Length of above 'pad' buffer.
```

REMARKS

The **cscCryptoPutFile** function is used to transmit an encrypted file from an established connection transmitted to the **cscCryptoGetFile** function.

The bytes in 'Buffer' are XOR'ed with the bytes in 'PadChars' before sending. If PadLen < BufLen, the XOR'ing continues in Buffer at the beginning of the pad buffer (wrap around).

Call **cscCryptoPutFile** with a filename to specify the file to be sent, then call **cscCryptoPutFile** repeatedly in a loop until 0 is returned.

The outgoing data steam will contain the filename and the length of the file, followed by the (binary) file itself.

Also see **`cscSetString`**`(-1, CSC_SET_FILE_PATH, ...)` to specify the file directory from which the file will be read.

Also read Section 2.11, "Encryption", in the CSC User's Manual (CSC_USR.PDF).

EXAMPLE (C/C++)

```
// prepare to transmit encrypted file "MyFile.zip" to the remote
Code = cscCryptoPutFile(DataSock,(char *)"MyFile.zip",(char *)PadChars,
                        PadLen);
// transmit file
while(Code!=0) Code = cscCryptoPutFile(DataSock, NULL,
                                       (char *)PadChars, PadLen));
```

EXAMPLE (VB)

```
Dim Filename As String
Filename = "MyFile.zip"
' prepare to transmit file "MyFile.zip" to the remote
Code = cscCryptoPutFile(DataSock, Filename,  PadChars, PadLen)
' transmit file
While Code <> 0
    Code = cscCryptoPutFile(DataSock, Chr(0), PadChars, PadLen))
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : No error.
```

2.18 cscCryptoPutFileExt :: Transmits an Encrypted File (Extended).

SYNTAX

```
cscPutFileExt (vSock, FileName, PadChars, PadLen)

  vSock     : (I) Virtual (data) socket number.
  FileName  : (P) Filename (not path) to transmit.
  PadChars  : (P) Buffer of "pad" bytes.
  PadLen    : (I) Length of above 'pad' buffer.
```

REMARKS

The **cscCryptoPutFileExt** function is used to transmit an encrypted file from an established connection transmitted to the **cscCryptoGetFileExt** function. If the file exists on the receiver and has the same size as on the sender, the file is not sent again.

The bytes in 'Buffer' are XOR'ed with the bytes in 'PadChars' before sending. If PadLen < BufLen, the XOR'ing continues in Buffer at the beginning of the pad buffer (wrap around).

Call **cscCryptoPutFileExt** with a filename to specify the file to be sent, then call **cscCryptoPutFile** repeatedly in a loop until 0 is returned.

The **cscCryptoPutFileExt** function transfers only that part of the specified file that has not already been transferred. If only part of a file is transferred (because of a network or computer malfunction), the function will resume the file transfer without having to send the entire file over again.

The outgoing data steam will contain the filename and the length of the file, followed by the (binary) file itself.

Also see **`cscSetString`**`(-1, CSC_SET_FILE_PATH, ...)` to specify the file directory from which the file will be read.

Also read Section 2.11, "Encryption", in the CSC User's Manual (CSC_USR.PDF).

EXAMPLE (C/C++)

```
// prepare to transmit encrypted file "MyFile.zip" to the remote
Code = cscCryptoPutFileExt(DataSock, (char *)"MyFile.zip",
                           (char *)PadChars, PadLen);
// transmit file
while(Code!=0) Code = cscCryptoPutFileExt(DataSock, NULL,
                            (char *)PadChars, PadLen));
```

EXAMPLE (VB)

```
Dim Filename As String
Filename = "MyFile.zip"
' prepare to transmit file "MyFile.zip" to the remote
Code = cscCryptoPutFileExt(DataSock, Filename, (char *)PadChars, PadLen);
' transmit file
While Code <> 0
    Code = cscCryptoPutFileExt(DataSock, Chr(0), PadChars, PadLen))
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : No error.
```

2.19 **cscCryptoPutPacket** :: Transmits an Encrypted Packet.

SYNTAX

```
cscCryptoPutPacket(vSock, Buffer, BufLen, PadChars, PadLen)

  vSock     : (I) Virtual (data) socket number.
  Buffer    : (P) Data to write.
  BufLen    : (I) Length of above buffer (# bytes to write).
  PadChars  : (P) Buffer of "pad" bytes.
  PadLen    : (I) Length of above 'pad' buffer.
```

REMARKS

The **cscCryptoPutPacket** function is used to write (transmit) an encrypted data packet to an established connection.

Packets can vary from 1 to 10,000 bytes in length. **cscCryptoPutPacket** will not return until the entire packet has been queued for transmission.

The bytes in 'Buffer' are XOR'ed with the bytes in 'PadChars' before sending. If PadLen < BufLen, the XOR'ing continues in Buffer at the beginning of the pad buffer (wrap around).

Also read Section 11, "Encryption", Section 2.8, "Stream Data I/O" and Section 2.9, "Packet Data I/O" in the CSC User's Manual (CSC_USR.PDF).

EXAMPLE (C/C++)

```
char *Temp = "Hello";
// write 5 bytes of data
Code = cscCryptoPutPacket(DataSock, (char *)Temp, 5,
                          (char *)PadChars, PadLen);
```

EXAMPLE (VB)

```
Dim Temp As String
Temp = "Hello"
' write 5 bytes of data
Code = cscCryptoPutPacket(DataSock, Temp, 5, PadChars, PadLen)
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : Number of bytes written.
```

2.20 **cscDataCRC** :: Compute CRC of Data Buffer.

SYNTAX

```
cscDataCRC(InitCRC, Buffer, BufLen)

  InitCRC   : (I) Initial value of CRC.
  Buffer    : (P) Data to write.
  BufLen    : (I) Length of above buffer (# bytes to write).
```

REMARKS

The **cscDataCRC** function is used to compute the 32-bit cyclic redundancy check (CRC) word of the passed buffer.

EXAMPLE (C/C++)

```
UNSIGNED INT CRC;            // CRC
char *Buffer = "ABC"; // data buffer
CRC = cscDataCRC(0L, Buffer, 3);
```

EXAMPLE (VB)

```
Dim CRC As Integer
Dim Buffer As String
Buffer = "ABC"
CRC = cscDataCRC(0L, Buffer, 3)
```

RETURNS

```
= 0  : Error.
Else : CRC
```

2.21 **cscDataMessage** :: Sends Windows Message (on Data Ready).

SYNTAX

```
cscDataMessage(Handle, vSock, Message)

  Handle   : (I) Windows Handle
  vSock    : (I) Virtual data socket.
  Message  : (I) Message (usually WM_USER)
```

REMARKS

The **cscDataMessage** function sends a Windows message to the window with handle 'Handle' when data is ready to be read from the other side (client or server).

Refer to the windows client example programs.

EXAMPLE (C/C++)

```
// Send WM_USER message when data is ready to be read
cscDataMessage(hMainWnd, DataSock, WM_USER);
```

EXAMPLE (VB)

```
Const WM_LBUTTONDOWN = &H201
' send WM_LBUTTONDOWN message when data is ready to be read
Code = cscDataMessage(Client.bReady.hWnd, DataSock, WM_LBUTTONDOWN)
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : No error.
```

2.22 **cscErrorText** :: Gets Text of Error Message.

SYNTAX

```
cscErrorText(ErrCode, Buffer, BufLen)

  ErrCode  : (I) Error code returned from a CSC function.
  Buffer   : (P) Buffer into which to copy text.
  BufLen   : (I) Length of above buffer.
```

REMARKS

The **cscErrorText** function is used to get the text of an error message associated with the error code returned from a CSC function.

Refer to the example programs.

EXAMPLE (C/C++)

```
char Buffer[128];
// get error text associated with error 'ErrCode'
Code = cscError(ErrCode, (char *)Buffer, 128);
```

EXAMPLE (VB)

```
Dim Buffer As String * 128
' get error text associated with error 'ErrCode'
Code = cscError(ErrCode, Buffer, 128)
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : Number of characters copied to Buffer.
```

2.23 **cscFileCRC** :: Compute CRC of File.

SYNTAX

```
cscFileCRC(Pathname)

  Pathname: (I) Pathname of file.
```

REMARKS

The **cscFileCRC** function is used to compute the 32-bit cyclic redundancy check (CRC) word of a file.

EXAMPLE (C/C++)

```
UNSIGNED INT CRC;
CRC = cscFileCRC((char *)"\\csc4c\\apps\\csc.h");
```

EXAMPLE (VB)

```
Dim CRC As Integer
Dim Pathname As String
Pathname = "\csc4c\apps\csc.h"
CRC = cscFileCRC(Pathname);
```

RETURNS

```
= 0  : Error.
Else : CRC
```

2.24 **cscFileLength** :: Returns File length.

SYNTAX

```
cscFileLength(PathName)

  PathName  : (P) Full path to file.
```

REMARKS

The **cscFileLength** function returns the (lower 32 bits) length of the specified file.

This function is provided for use with those languages that do not have a file length function.

EXAMPLE (C/C++)

```
unsigned int Len;
char *File = "\\aes4c\\apps\\examples.txt"
FileLen = cscFileLength((char *)File);
```

EXAMPLE (VB)

```
Dim As String File
File = "\csc4vb\apps\examples.txt"
FileLen = cscFileLength(File)
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : Number of characters copied to the PadChars buffer.
```

2.25 **cscFillRandom** :: Fill Buffer With Random Bytes.

SYNTAX

```
cscFillRandom(PadChars, PadLen, Seed)

  PadChars  : (P) Buffer of "pad" bytes.
  PadLen    : (I) Length of above "pad" buffer.
  Seed      : (I) Random number seed.
```

REMARKS

The **cscFillRandom** function fills the PadChars buffer with 8-bit bytes generated from a pseudo random number generator using the passed Seed value as the seed. Passing a seed of zero will direct **cscFileRandom** to use a randomly generated seed.

This function provides an easy way to populate the PadChars buffer used by the crypto functions. The passed (32-bit) seed is the password.

Also read Section 2.11, "Encryption", in the CSC User's Manual (CSC_USR.PDF).

EXAMPLE (C/C++)

```
char PadChars[1024];
Code = cscFillRandom((char *)PadChar, 1024, 1234567);
```

EXAMPLE (VB)

```
Dim PadChars As String * 1024
Code = cscFillRandom(PadChar, 1024, 1234567)
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : Number of characters copied to the PadChars buffer.
```

2.26 **cscGetData** :: Receives Data.

SYNTAX

```
cscGetData(vSock, Buffer, BufLen)

  vSock   : (I) Virtual (data) socket.
  Buffer  : (P) Buffer into which to copy bytes.
  BufLen  : (I) Length of above buffer.
```

REMARKS

The **cscGetData f**unction is used to receive data from an established connection. All available data is copied to 'Buffer' up to a maximum of 'BufLen'. **cscGetData** never waits for data.

Because TCP is stream oriented, data sent in one network write may not be received in one network read.

EXAMPLE (C/C++)

```
char Temp[128];
// read data (up to 128 bytes)
Code = cscGetData(DataSock, (char *)Temp, 128);
```

EXAMPLE (VB)

```
Dim Temp As String * 128
' read data (up to 128 bytes)
Code = cscGetData(DataSock, Temp, 128)
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : Number of bytes copied to Buffer.
```

2.27 **cscGetFile** :: Receives a File.

SYNTAX

```
cscGetFile(vSock, FileName)

  vSock    : (I) Virtual (data) socket number.
  Filename : (P) Filename (not path) used to save incoming file or NULL.
```

REMARKS

The **cscGetFile** function is used to receive a file from an established connection transmitted from the **cscPutFile** function.

Specify the filename under which the file is to be saved, or specify an asterisk ("*") to save under the filename sent with the file. Then call **cscGetFile** repeatedly in a loop until 0 is returned.

The incoming data stream will contain the filename and the length of the file, followed by the (binary) file itself. The filename can be recovered by use of the **cscGetString** function and the file length can be recovered by use of the **cscGetInteger** function.

Also see **`cscSetString`**`(-1, CSC_SET_FILE_PATH,...)` to specify the file directory into which the file is to be saved.

EXAMPLE (C/C++)

```
// prepare to receive file, saving to disk
Code = cscGetFile(DataSock, (char *)"*");
// receive file in 4K blocks
while(Code!=0) Code = cscGetFile(DataSock, NULL);
```

EXAMPLE (VB)

```
' prepare to receive file, saving to disk
X = "*" + Chr(0)
Code = cscGetFile(DataSock, X)
// receive file in 4K blocks
EmptyString = Chr(0)
While Code <> 0
  Code = cscGetFile(DataSock, EmptyString)
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : No error.
```

2.28 **cscGetFileExt** :: Receives a File (Extended).

SYNTAX

```
cscGetFileExt(vSock, FileName)

  vSock    : (I) Virtual (data) socket number.
  FileName : (I) T: setup, F: send data packets.
```

REMARKS

The **cscGetFileExt** function is used to receive a file from an established connection transmitted from the **cscPutFileExt** function. If the file exists on the receiver and has the same size as on the sender, the file is not sent again.

Specify the filename under which the file is to be saved, or specify an asterisk ("*") to save under the filename sent with the file. Then call **cscGetFileExt** repeatedly in a loop until 0 is returned.

The incoming data stream will contain the filename and the length of the file, followed by the (binary) file itself. The filename can be recovered by use of the **cscGetString** function and the file length can be recovered by use of the **cscGetInteger** function.

The **cscGetFileExt** function transfers only that part of the specified file that has not already been transferred. If only part of a file is transferred (because of a network or computer malfunction), the function will resume the file transfer without having to send the entire file over again.

Also see **`cscSetString`**`(-1, CSC_SET_FILE_PATH,...)` to specify the file directory into which the file is to be saved.

EXAMPLE (C/C++)

```
// prepare to receive file, saving to disk
Code = cscGetFileExt(DataSock, 1);
// receive file in 4K blocks
while(Code!=0) Code = cscGetFileExt(DataSock, 0);
```

EXAMPLE (VB)

```
' prepare to receive file, saving to disk
Code = cscGetFileExt(DataSock, 1)
' receive file in 4K blocks
While Code <> 0
  Code = cscGetFileExt(DataSock, 0)
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : No error.
```

2.29 **cscGetInteger** :: Returns CSC Numeric Parameter with Client/Server processing information.

SYNTAX

```
cscGetInteger(vSock, ParmName)

  vSock    : (I) Virtual socket number or -1.
  ParmName : (I) Parameter name.
```

REMARKS

The **cscGetInteger** function is used to get integer parameters from the DLL.

```
CSC_GET_VERSION     : Get version number (vSock = -1)
CSC_GET_BUILD       : Get build number (vSock = -1)
CSC_GET_BUFFER_SIZE : Get file buffer size used by cscGetFile & cscPutFile.
CSC_GET_DAYS_LEFT   : Get the number days left in the evaluation period.
CSC_GET_SOCK_ERROR  : Get socket error number.
CSC_GET_SOCKET      : Get actual TCP/IP socket number being used.
CSC_GET_FILE_LENGTH : Get file length after receiving file (cscGetFile)
CSC_GET_MAX_PACKET_SIZE : Get maximum packet size.
```

EXAMPLE (C/C++)

```
int Version;
// get CSC version number
Version = cscGetInteger(-1, CSC_GET_VERSION);
```

EXAMPLE (VB)

```
Dim Version As Integer
' get CSC version number
Version = cscGetInteger(-1, CSC_GET_VERSION)
```

RETURNS

< 0 : Error. See error list.
>= 0 : Requested parameter.

2.30 cscGetPacket :: Receives a Packet.

SYNTAX

```
cscGetPacket(vSock, Buffer, BufLen)

  vSock   : (I) Virtual (data) socket.
  Buffer  : (P) Buffer into which to copy bytes.
  BufLen  : (I) Length of above buffer.
```

REMARKS

The **cscGetPacket** function is used to receive the next packet from an established connection. The next packet is copied to 'Buffer' up to a maximum of 'BufLen'. **cscGetPacket** waits for a maximum of 10 seconds (default) for the next packet..

Packets can vary from 1 to 10,000 bytes in length. **cscGetPacket** will not return until the entire packet has been received.

Also read Section 2.8, "Stream Data I/O" and Section 2.9, "Packet Data I/O" in the CSC User's Manual (CSC_USR.PDF).

EXAMPLE (C/C++)

```
char Temp[128];
// read data (up to 128 bytes)
Code = cscGetPacket(DataSock, (char *)Temp, 128);
```

EXAMPLE (VB)

```
Dim Temp As String * 128
' read data (up to 128 bytes)
Temp = Space(128)
Code = cscGetPacket(DataSock, Temp)
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : Number of bytes copied to Buffer (packet size).
```

2.31 **cscGetString** :: Returns CSC String Parameter with Client/Sever Processing Information.

SYNTAX

```
cscGetString(vSock, ParmName, Buffer, BufLen)

  vSock    : (I) Virtual socket number or -1.
  ParmName : (I) Parameter name.
  Buffer   : (P) Buffer into which to copy text.
  BufLen   : (I) Length of above buffer.
```

REMARKS

The **cscGetString** function is used to get string (text) parameters from the DLL.

```
CSC_GET_REGISTRATION     : Get registration string.
CSC_GET_FILE_NAME        : Get filename after receiving file (cscGetFile).
CSC_GET_REMOTE_SERVER_IP : Get IP address in dotted notation of remote server.
CSC_GET_REMOTE_CLIENT_IP : Get IP address in dotted notation of remote client.
CSC_GET_LOCAL_IP         : Get local IP address in dotted notation.
CSC_GET_COMPUTER_NAME    : Get name of local computer.
```

EXAMPLE (C/C++)

```
    // get registration string
    char RegString[128];
    Code = cscGetString(vSock, CSC_GET_REGISTRATION, (char *)RegString, 128)
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : Number of characters copied to Buffer.
```

2.32 **cscGetUDP** :: Reads UDP Datagram.

SYNTAX

```
cscGetUDP(vSock, Buffer, BufLen, Host)

vSock  : (I) Virtual UDP socket.
Buffer : (I) Buffer into which datagram is to be copied.
BufLen : (I) Size of buffer.
Host   : (P) Host name or IP address (dotted format).
```

REMARKS

The **cscGetUDP** function reads an incoming UDP datagram. **cscCreateUDP** must be called first to get a UDP socket.

Refer to the uEcho_S (UDP server) and uEcho_C (UDP client) example programs.

EXAMPLE (C/C++)

```
// Read UDP socket into 'Buffer'
BytesRead = cscGetUDP(vSock, (char *)Buffer, BufSize, (char *)Host);
```

EXAMPLE (VB)

```
' Read UDP socket into 'Buffer'
BytesRead = cscGetUDP(vSock, Buffer, Host)
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : No error.
```

2.33 **cscIsConnected**:: Returns the Current Connection Status.

SYNTAX

```
cscIsConnected(vSock)

  vSock : (I) Virtual socket number.
```

REMARKS

The **cscIsConnected** function is used determine the current connection status.

Awaiting incoming socket data (**cscAwaitData**), reading socket data (**cscGetData**), and writing socket data (**cscPutData**) will all return an error code if the socket connection has been dropped.

EXAMPLE (C/C++)

```
    //test connection
    if(!cscIsConnected(vSock))
      {printf("*** ERROR: Connection has been dropped!\n");
       break;
      }
```

EXAMPLE (VB)

```
     Dim vSock As Integer
     If cscIsConnected(vSock) <> 0 Then
        Result.Text = "*** ERROR: Connection has been dropped!"
     End If
```

RETURNS

```
True  : Connective is OK.
False : Connection has been dropped.
```

2.34 **cscLaunch** :: Starts an External Executable

SYNTAX

```
cscLaunch(PgmPath, CmdLine)

  PgmPath   : (P) The path to the program command (or NULL)
  CmdLine   : (P) Program ommand line
```

REMARKS

The **cscMakeDotted** function is used to create a dotted text version of the specified IP address.

EXAMPLE (C/C++)

```
char Browser[] = "C:\\Program Files\\Internet Explorer\\IEXPLORE.EXE";
// launch the browser
Code = cscLaunch(0, CmdLine);
```

EXAMPLE (VB)

```
Dim Empty As String
Dim Browser As String
Empty = Chr(0)
Browser = "C:\Program Files\Internet Explorer\IEXPLORE.EXE";
// launch the browser
Code = cscLaunch(0, CmdLine)
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : IP address in dotted decimal notation (e.g.: "10.0.0.1")
```

2.35 **cscMakeDotted** :: Create Dotted IP String From IP Address

SYNTAX

```
cscMakeDotted(Addr, Buffer, BufLen)

  Addr     : (L) 32-bit IP address.
  Buffer   : (P) Buffer into which to copy dotted address.
  BufLen   : (I) Length of above buffer.
```

REMARKS

The **cscMakeDotted** function is used to create a dotted text version of the specified IP address.

EXAMPLE (C/C++)

```
UNSIGNED INT Addr;
Addr = cscResolve((char *)"www.marshallsoft.com", 0);
// construct dotted decimal equivalent string
if(Addr) cscMakeDotted(Addr, (char *)Temp, 64);
```

EXAMPLE (VB)

```
Dim Addr As Integer
Dim X As String
Dim Temp As String
X = www.marshallsoft.com
Addr = cscResolve(X, 0)
' construct dotted decimal equivalent string
If Addr <> 0 Then
   Temp = Space(20)
   cscMakeDotted(Addr, Temp, 20)
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : IP address in dotted decimal notation (e.g.: "10.0.0.1")
```

2.36 cscMakeDotted4 :: Create Dotted IP String From IP Components

SYNTAX

```
cscMakeDotted(Byte1, Byte2, Byte3, Byte4, Buffer, BufLen)

  Byte1   : (L) First IP address Byte [0..255]
  Byte2   : (L) First IP address Byte [0..255]
  Byte3   : (L) First IP address Byte [0..255]
  Byte4   : (L) First IP address Byte [0..255]
  Buffer  : (P) Buffer into which to copy dotted address.
  BufLen  : (I) Length of above buffer.
```

REMARKS

The **cscMakeDotted4** function is used to create a dotted text version of the specified IP address as specified by 4 Bytes.

EXAMPLE (C/C++)

```
int Byte1 = 10;
int Byte2 = 0;
int Byte3 = 0,
int Byte4 = 1;
// construct dotted decimal equivalent string "10.0.0.1"
cscMakeDotted2(Byte1, Byte2, Byte3, Byte4, (char *)Temp, 64);
```

EXAMPLE (VB)

```
Dim Byte1 As Integer
Dim Byte2 As Integer
Dim Byte3 As Integer
Dim Byte4 As Integer
Dim X As String
Dim Temp As String
Byte1 = 10
Byte2 = 0
Byte3 = 0
Byte4 = 1
' construct dotted decimal equivalent string "10.0.0.1"
X = Space(20)
Code = cscMakeDotted4(Byte1, Byte2, Byte3, Byte4, Temp, 20)
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : IP address in dotted decimal notation (e.g., "10.0.0.1")
```

2.37 **cscMulticast** :: Sets multicast address

SYNTAX

```
cscMulticast(vSock, MultiIP)

vSock    : (I) Virtual (data) socket number.
MultiIP  : (I) Multicast IP address
```

REMARKS

The **cscMulticast** function sets the multicast IP address for receiving UDP packets.

To send a multicast datagram, specify an IP multicast address using 239.255.x.y as the destination address. The receiving program should call **cscMulticast()** to set up the multicast address. **cscMulticast()** is NOT called to send packets.

EXAMPLE (C/C++)

```
    MulticastIP = cscResolve(MULTICAST_IP, 0);
    Code = cscMulticast(vSock, MulticastIP);
```

EXAMPLE (VB)

```
    MulticastIP = cscResolve(MULTICAST_IP, 0)
    Code = cscMulticast(vSock, MulticastIP)
```

RETURNS

```
 < 0 : Error. See error list.
>= 0 : No error.
```

2.38 cscNetToHost16 :: Converts a 16-bit integer to host byte order

SYNTAX

cscNetToHost16(Integer)

```
Integer : (I) 16-bit integer in network byte order.
```

REMARKS

The **cscNetToHost16** program converts a 16-bit integer from network byte order to host byte order. For example, the Network Time Server protocol returns an integer in network byte order that must be converted to host byte order.

EXAMPLE (C/C++)

```
// convert to host byte order
HostInteger = cscNetToHost(NetInteger);
```

EXAMPLE (VB)

```
' convert to host byte order
HostInteger = cscNetToHost(NetInteger);
```

RETURNS

```
16-bit integer in host byte order.
```

2.39 **cscNetToHost32** :: Converts a 32-bit integer to host byte order

SYNTAX

cscNetToHost32(Integer)

```
Integer : (I) 32-bit integer in network byte order.
```

REMARKS

The **cscNetToHost32** program converts a 32-bit integer from network byte order to host byte order. For example, the Network Time Server protocol returns an integer in network byte order that must be converted to host byte order.

EXAMPLE (C/C++)

```
// convert to host byte order
HostInteger = cscNetToHost(NetInteger);
```

EXAMPLE (VB)

```
' convert to host byte order
HostInteger = cscNetToHost(NetInteger);
```

RETURNS

```
32-bit integer in host byte order.
```

2.40 **cscPutData** :: Transmits Data.

SYNTAX

```
cscPutData(vSock, Buffer, BufLen)

  vSock    : (I) Virtual (data) socket number.
  Buffer   : (P) Data to write.
  BufLen   : (I) Length of above buffer (# bytes to write).
```

REMARKS

The **cscPutData** function is used to write (transmit) a buffer to an established connection.

Because TCP is stream oriented, data sent in one network write may not be received in one network read.

EXAMPLE (C/C++)

```
char *Temp = "Hello";
// write 5 bytes of data
Code = cscPutData(DataSock, (char *)Temp, 5);
```

EXAMPLE (VB)

```
Dim Temp As String
Temp = "Hello"
' write 5 bytes of data
Code = cscPutData(DataSock, Temp, 5)
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : Number of bytes written.
```

2.41 **cscPutFile** :: Transmits a File.

SYNTAX

```
cscPutFile(vSock, FileName)

  vSock    : (I) Virtual (data) socket number.
  FileName : (P) Filename (not path) to transmit.
```

REMARKS

The **cscPutFile** function is used to transmit a file from an established connection transmitted to the **cscGetFile** function.

Call **cscPutFile** with a filename to specify the file to be sent, then call **cscPutFile** repeatedly in a loop until 0 is returned.

The outgoing data steam will contain the filename and the length of the file, followed by the (binary) file itself.

Also see **`cscSetString`**`(-1, CSC_SET_FILE_PATH,  ...)` to specify the file directory from which the file will be read.

EXAMPLE (C/C++)

```
// prepare to transmit file "MyFile.zip" to the remote
Code = cscPutFile(DataSock, (char *)"MyFile.zip");
// transmit file in 4K blocks
while(Code!=0) Code = cscPutFile(DataSock, NULL);
```

EXAMPLE (VB)

```
Dim Filename As String
Filename = "MyFile.zip"
' prepare to transmit file "MyFile.zip" to the remote
Code = cscPutFile(DataSock, Filename)
' transmit file in 4K blocks     While Code <> 0
Code = cscPutFile(DataSock, Chr(0))
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : No error.
```

2.42 cscPutFileExt :: Transmits a File (Extended).

SYNTAX

```
cscPutFileExt(vSock, FileName)

  vSock    : (I) Virtual (data) socket number.
  FileName : (P) Filename (not path) to transmit.
```

REMARKS

The **cscPutFileExt** function is used to transmit a file from an established connection transmitted to the **cscGetFileExt** function. If the file exists on the receiver and has the same size as on the sender, the file is not sent again.

Call **cscPutFileExt** with a filename to specify the file to be sent, then call **cscPutFileExt** repeatedly in a loop until 0 is returned.

The outgoing data steam will contain the filename and the length of the file, followed by the (binary) file itself.

The **cscPutFileExt** function transfers only that part of the specified file that has not already been transferred. If only part of a file is transferred (because of a network or computer malfunction), the function will resume the file transfer without having to send the entire file over again.

Also see **`cscSetString`**`(-1, CSC_SET_FILE_PATH,  ...)` to specify the file directory from which the file will be read.

EXAMPLE (C/C++)

```
// prepare to transmit file "MyFile.zip" to the remote
Code = cscPutFileExt(DataSock, (char *)"MyFile.zip");
// transmit file in 4K blocks
while(Code!=0) Code = cscPutFileExt(DataSock, NULL);
```

EXAMPLE (VB)

```
Dim Filename As String
Filename = "MyFile.zip"
' prepare to transmit file "MyFile.zip" to the remote
Code = cscPutFileExt(DataSock, Filename)
' transmit file in 4K blocks
While Code <> 0
   Code = cscPutFileExt(DataSock, Chr(0))
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : No error.
```

2.43 **cscPutPacket** :: Transmits a Packet.

SYNTAX

```
cscPutPacket(vSock, Buffer, BufLen)

  vSock     : (I) Virtual (data) socket number.
  Buffer    : (P) Data to write.
  BufLen    : (I) Length of above buffer (# bytes to write).
```

REMARKS

The **cscPutPacket** function is used to write (transmit) an encrypted data packet to an established connection.

Packets can vary from 1 to 10,000 bytes in length. **cscPutPacket** will not return until the entire packet has been queued for transmission.

Also read Section 2.8, “Stream Data I/O” and Section 2.9, “Packet Data I/O” in the CSC User’s Manual (CSC_USR.PDF).

EXAMPLE (C/C++)

```
char *Temp = "Hello";
// write 5 bytes of data
Code = cscPutPacket(DataSock, (char *)Temp, 5);
```

EXAMPLE (VB)

```
Dim Temp As String
Temp = "Hello"
' write 5 bytes of data
Code = cscPutPacket(DataSock, Temp, 5)
```

RETURNS

```
< 0  : Error. See error list.
```

2.44 **cscPutUDP** :: Writes UDP Socket.

SYNTAX

```
cscPutUDP(vSock, Buffer, BufLen, RemoteIP, RemotePort)

vSock      : (I) Virtual UDP socket.
Buffer     : (I) Buffer into which datagram is to be copied.
BufLen     : (I) Size of buffer.
RemoteIP   : (I) Remote IP address (32-bit address)
RemotePort : (P) Remote port.
```

REMARKS

The **cscPutUDP** function writes a UDP datagram. **cscCreateUDP** must be called first to get a UDP socket.

Refer to the uEcho_S (UDP server) and uEcho_C (UDP client) example programs.

EXAMPLE (C/C++)

```
    // Write UDP socket from 'Buffer'
    BytesWritten = cscPutUDP(vSock,(char *)Buffer, BufSize, RemoteIP);
```

EXAMPLE (VB)

```
    ' Read UDP socket into 'Buffer'
    BytesRead = cscPutUDP(vSock,Buffer, Host)
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : No error.
```

2.45 **cscReadSize** :: Returns the Number of Bytes Ready to be Read.

SYNTAX

```
cscReadSize(vSock)

  vSock     : (I) Virtual (data) socket number.
```

REMARKS

The **cscReadSize** function is used to get the number of bytes ready to be read from the socket.

EXAMPLE (C/C++)

```
int Bytes;
// get # bytes ready to be read
Bytes = cscReadSize(vSock);
```

EXAMPLE (VB)

```
Dim Bytes As Integer
' get # bytes ready to be read
Bytes = cscReadSize(vSock)
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : Number of bytes ready to be read.
```

2.46 **cscRelease** :: Releases DLL.

SYNTAX

```
cscRelease()
```

REMARKS

The **cscRelease** function is used to release CSC32.DLL or CSC64.DLL and should be the last CSC function called.

EXAMPLE (C/C++)

```
int Code;
// release CSC32.DLL
Code = cscRelease();
```

EXAMPLE (VB)

```
Dim Code As Integer
' release CSC32.DLL
Code = cscRelease()
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : No error.
```

2.47 **cscResolve** :: Resolves Host Name into IP Address.

SYNTAX

```
cscResolve(HostName, HostIndex)

  HostName  : (P) Server name or IP address (in dotted notation).
  HostIndex : (I) Server index (if multi-homed).
```

REMARKS

The **cscResolve** function is used to resolve a host name to an IP address by calling DNS (Domain Name Services).

EXAMPLE (C/C++)

```
UNSIGNED INT Addr;
// resolve host name into an IP address.
Addr = cscResolve((char *)"www.marshallsoft.com", 0);
```

EXAMPLE (VB)

```
Dim Addr As Integer
Dim URL As String
URL = "www.marshallsoft.com"
' resolve host name into an IP address.
Addr = cscResolve(URL, 0)
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : IP address.
```

2.48 cscResponse :: Constructs Response String.

SYNTAX

```
cscResponse(ChallStr,Multi,Mask,Rotate,Response)

  ChallStr : (P) 8 character challenge string (in hex)
  Mult     : (I) 32-bit multiplier
  Mask     : (I) 32-bit mask value
  Rotate   : (I) Left rotate count (0 to 31)
  Response : (P) Buffer for response string
```

REMARKS

The **cscResponse** function constructs the correct response string for the given challenge string.

The purpose of the **cscChallenge** and **cscResponse** functions is to create a challenge / response protocol in order to defeat hackers who may try to connect to your server.

The arguments to the **cscResponse** function define the mapping from the 8-byte challenge string to the 8-byte response string. Each developer should choose unique values for his application.

Also see **cscChallenge**, which constructs the random challenge string.

EXAMPLE (C/C++)

```
char Challenge[9];           // challenge string
char Response[9];            // response string
UNSIGNED INT Multiplier = 321;      // 32-bit multiplier
UNSIGNED INT Mask = 0x1a2b3c4d;     // 32-bit mask value
int RotateCount = 3;         // left rotate count (0 to 31)
// compute correct response for the challenge number
Code = cscResponse((char *)Challenge, Multiplier, Mask,
       RotateCount, (char *)Response);
```

EXAMPLE (VB)

```
Dim Challenge As String
Dim Response As String * 9
Dim Multiplier As Integer
Dim Mask As Integer
Dim RotateCount As Integer
Multiplier = 321
Mask = &H1a2b3c4d
RotateCount = 3
// compute correct response for the challenge number
Code = cscResponse(Challenge, Multiplier, Mask, RotateCount, Response)
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : No error.
```

2.49 cscServer :: Starts the Server.

SYNTAX

```
cscServer(ServerName, ServerPort, MaxConnect)

  ServerName : (P) Server name or IP address, or NULL.
  ServerPort : (I) Server port to listen on.
  MaxConnect : (I) Maximum number of connections to accept.
```

REMARKS

The **cscServer** function is used to accept a connection from a client. Up to 'MaxConnect' connections can be accepted by the server concurrently.

Specify the server name or IP address (in dotted decimal notation) and the port to listen on. Specify NULL or an empty string for ServerName if you want to accept connections on any IP on the local machine.

For server applications, be sure to allocate one listen socket for each port that is to be listened on concurrently.

EXAMPLE (C/C++)

```
HostName = "\0";
short HostPort = 5001;
// start server, accepting a maximum of 1 connection
Code = cscServer((char *)HostName, (int)HostPort, 1);
```

EXAMPLE (VB)

```
Dim HostName As String
Dim HostPort As Integer
HostName = Chr(0)
HostPort = 5001
Code = cscServer(HostName, HostPort, 1)
```

RETURNS

```
>= 0 : Listen socket.
< 0   : Error (See error list)
```

2.50 cscSetInteger :: Sets numeric parameter which contains client/server processing information.

SYNTAX

```
cscSetInteger(vSock, ParamName, ParamValue)

  vSock      : (I) Virtual socket number or -1.
  ParamName  : (I) Parameter name.
  ParamValue : (I) Parameter value.
```

REMARKS

The **cscSetInteger** function is used to set an integer parameter in the DLL. Parameter names are:

```
CSC_SET_BLOCKING_MODE   : Set blocking mode (connect only). Default = TRUE
CSC_SET_BUFFER_SIZE     : Set write file size. Default = 10000.
CSC_SET_SLEEP_TIME      : Set sleep time value when waiting. Default = 100.
CSC_SET_DEBUG_LEVEL     : Set debug level (0=off, 1=low, 2-high). Default = 0.
CSC_SET_LINGER          : Set linger time when socket is closed. Default = 200
CSC_SET_TIMEOUT_VALUE   : Set packet timeout. Default = 10000.
CSC_SET_MAX_PACKET_SIZE : Set maximum packet size. Default = 10000.
CSC_SET_FILE_OVERWRITE  : Sets file overwrite mode. Default = 0.
CSC_SET_CLOSE_TIMEOUT   : Sets maximum time before a socket is forced closed.
CSC_SET_CONNECT_WAIT    : Sets connection wait timeout. Default = 10.
CSC_SET_SOCK_REUSE      : Allows (T/F) listening socket to be reused.
```

BLOCKING_MODE: The blocking mode (1 = block, 0 = don't block) applies only while connecting.

BUFFER_SIZE: The default file buffer size is 10000, the maximum size is 30000.

SLEEP_TIME: The sleep value (mSec) is used in certain functions to introduce a time delay.

DEBUG_LEVEL: Values are off (0), low (1), and high (2). The default debug level is 0.

LINGER: The linger time (mSec) is the delay after closing a socket to allow any ongoing communications to complete on the socket.

TIMEOUT_VALUE: The timeout-value is used by **cscGetFile** and specified how long to wait for individual fields before giving up.

MAX_PACKET_SIZE: The default packet buffer size is 10000, the maximum size is 30000.

FILE_OVERWRITE: When receiving file, will overwrite existing file if TRUE (not 0).

CSC_SET_CLOSE_TIMEOUT : Sets the maximum time (in milliseconds) before a socket is forced closed (when calling cscClose). This function should not be used as a routine matter.

CSC_SET_CONNECT_WAIT: Sets the maximum time (in milliseconds) to wait for a connection to be accepted.

CSC_SET_SOCK_REUSE: [T/F] Enables an application to close the listening socket and immediately reopen without error.

EXAMPLE (C/C++)

```
int Code;
// set 100 ms sleep time
Code = cscSetInteger(vSock, CSC_SET_SLEEP_TIME, 100);
```

EXAMPLE (VB)

```
Dim Code As Integer
' set 100 ms sleep time
Code = cscSetInteger(vSock, CSC_SET_SLEEP_TIME, 100)
```

2.51 **cscSetString** :: Sets parameter string for file processing.

SYNTAX

```
cscSetString(vSock, ParmName, ParmPtr)

  vSock    : (I) Virtual socket number or -1.
  ParmName : (I) Parameter value.
  ParmPtr  : (P) Parameter string to set.
```

REMARKS

The **cscSetString** function is used to pass string (text) parameters in the DLL.

Parameter names are as follows (pass vSock = -1).

```
CSC_SET_LOG_FILE  : name of the log file.
CSC_WRITE_TO_LOG  : write string to log file.
CSC_SET_FILE_PATH : sets path location for cscPutFile and cscGetFile.
```

EXAMPLE (C/C++)

```
int Code;
// set LOG filename
Code = cscSetString (-1, CSC_SET_LOG_FILE, (char *)"MY-PROGRAM.LOG");
```

EXAMPLE (VB)

```
Dim Code As Integer
Dim LogName As String
' set LOG filename
LogName = "MY-PROGRAM.LOG"
Code = cscSetString (-1, CSC_SET_LOG_FILE, LogName)
```

RETURNS

```
< 0  : Error. See error list.
>= 0 : No error.
```

2.52 **cscShortToByte** :: Converts 16-bit ASCII character buffer to 8-bit

SYNTAX

```
cscShortToByte(Buffer)

  Buffer : (P) character buffer
```

REMARKS

The **cscShortToByte** function converts the (null terminated) character buffer 'Buffer' from 16-bit Unicode ASCII characters to 8-bit ASCII characters.

The buffer must be null terminated (last character is a hex 00).

This function is only necessary when working with 16-bit Unicode ASCII characters in C# and Delphi 2005.

RETURNS

None.

EXAMPLE (C#)

See C# example ClientCS.csproj

```
NameString = "MyFile.zip\0"
char[] NameBuffer = NameString.ToCharArray();
// convert (null terminated) 16-unicode buffer to 8-bit
fixed (char* pNameBuffer = NameBuffer)
cscShortToByte(pNameBuffer);
```

ALSO SEE

cscByteToShort

2.53 **cscSleep** :: Sleeps Specified Time.

SYNTAX

```
cscSleep(Milliseconds)

  Milliseconds : (I) Number of milliseconds to sleep.
```

REMARKS

The **cscSleep** function sleeps for the indicated number of milliseconds. 'Milliseconds' must be positive.

This function is included in CSC because it is not available in all computer languages.

EXAMPLE (C/C++)

```
// sleep 1 second
Tics = cscSleep(1000);
```

EXAMPLE (VB)

```
' sleep 1 second
Tics = cscSleep(1000)
```

RETURNS

```
1 is always returned.
```

2.54 **cscSystemTics** :: Returns System Tics Count.

SYNTAX

```
cscSystemTics()
```

REMARKS

The **cscSystemTics** function returns the system tic count, which is the number of milliseconds since the system was booted.

The primary purpose of this function is to time various events.

This function is included in **CSC** because it is not available in all computer languages.

EXAMPLE (C/C++)

```
unsigned long Tics;
// return current system tics
Tics = cscSystemTics();
```

EXAMPLE (VB)

```
Dim Tics As Integer
' return current system tics
Tics = cscSystemTics()
```

RETURNS

```
The number of milliseconds since system bootup.
```

2.55 **cscTestDotted** :: Tests Dotted IP Address

SYNTAX

```
cscTestDotted( DottedIP )

   DottedIP : (P) Dotted IP address
```

REMARKS

The **cscTestDotted** function tests the format of a dotted IP address.

EXAMPLE (C/C++)

```
char *DottedIP = "10.0.0.6";
if( !cscDotted(DottedIP) )
  {// failure…
```

EXAMPLE (VB)

```
Dim DottedIP As String
DottedIP = "10.0.0.6"
if cscDotted(DottedIP) = 0
  ' failure…
```

RETURNS

```
0 if false
```

3 CSC Error Return Code List

The complete list of **CSC** error codes follows.

3.1 WINSOCK Error Codes

```
-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: Prot mismatch.
-10042: Prot option invalid.
-10043: Prot 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.
```

3.2 CSC Error Codes

```
 -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
-16: Too many sockets.
-17: No free sockets.
-18: No such file.
-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: Buffer size too large.
-28: Connect timeout.
-29: Packet size error.
-30: Cannot resolve host.
-74: Bad key code.
```

4.3 VSOCK Error Codes

```
-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.
```