

TCP4W API User Manual

License

TCP4W was written by Philippe Jounin and is Copyrighted 1994-1996 by him. The author disclaims all liability for its use or for problems, data corruption, data loss, or other loss that may result from its use.

Permission is given without restriction to use and distribute the program provided that it is distributed without charge, that it is not modified in any way, and that this file accompanies the DLL file.

This program may be included on CD-ROMs or other distribution methods freely, provided any charge for such is for recovering the cost of distribution and reasonable profit and not for the purpose of "selling" the program. In this case the distribution must contain the complete program including this file.

Send any comments to ark@ifh.sncf.fr.

The pascal samples provided are written and copyrighted by Andreas Tikart. Please, send your comments or questions about them at Andreas.Tikart@uni-konstanz.de.

Overview

TCP4W.DLL provides APIs which allow an easy use of the TCP protocol in connected mode.

Some features are related to the Telnet protocol .

It is a Windows Dynamic Library (DLL), which can be used by any language (and any compiler). It requires a Windows Sockets DLL (Winsock.DLL).

All functions provided return only when the associated operation is finished or if a timeout has occurred.

While a process is waiting for a blocking call, it continues to receive messages (Timer, user's actions), and it can call again a network related function, this second call will failed in order to avoid reentrance. To cancel a blocking call, the application should call `TcpAbort` (or `WSACancelBlockingCall` which marks the first call as cancelled (but do not cancel it immediatly), then return control to Windows before calling any other network related function (see example in the description of the `TcpAbort` call).

Programming with the TCP4W API

To use the TCP4W functions, 9 files are provided:

- This reference
- The 16 bits DLL TCP4W.DLL and its export library TCP4W.LIB
- The 32 bits version TCP4W32.DLL and its export library TCP4W32.LIB
- A C-header file TCP4W.H
- A Pascal-header file USETCP4W.PAS written by Andreas Tikart.
- The 16 bits static library TCP4WLIB.LIB
- The 32 bits static library TCP4W32L.LIB

The first function that an application should call is **Tcp4uInit**. It allocates buffers and get some information about the task which has called it.

The task is ready to open connections. It must either call TcpConnect (client) or wait for a connection with both TcpGetListenSocket and TcpAccept.

Now it can send or receive data with TcpRecv, TcpSend. If the remote peer uses a telnet based protocol, the function should uses TnSend, TnReadLine or TnGetAnswerCode.

Before quitting, the application must close opened sockets with TcpClose, and call Tcp4uCleanup.

The TCP4W functions

This table lists alphabetically all the functions implemented in this version. The remainder of this chapter describes them one by one.

Tcp4uCleanup	Last function to be called, frees local resources
Tcp4uInit	First function to be called
Tcp4uVer	Gives the 2-part version of the DLL (packed into an int).
Tcp4uErrorString	Returns error string about an error code
TcpAbort	Aborts the current blocking call
TcpAccept	Wait for a connection
TcpClose	Close a socket
TcpConnect	Establishes a connection to a peer
TcpFlush	Flushes the buffer associated with a socket
TcpGetListenSocket	Create a socket and listen for incoming connection
TcpGetLocalID	Returns name and address of the local host
TcpGetRemoteID	Returns name and address of the connected host
TcpIsDataAvail	Returns TRUE if unread data are available
TcpIsOOBDataAvail	Returns TRUE if unread Out of Band data are available
TcpPPRecv	Receives data with length of frame in the two first bytes
TcpPPSend	Sends data, the two first bytes of the frame are its length
TcpRecv	Receives data
TcpRecvUntilStr	Receives data until a given character/string
TcpSend	Sends data to connected host
TnGetAnswerCode	Receives data until end of a Telnet frame
TnReadLine	Receives data until a end of line
TnSend	Sends a 0-terminated string

Note: The old functions Tcp4wCleanup, Tcp4wInit, Tcp4wVer are still supported but called now the functions Tcp4uCleanup, Tcp4uInit, Tcp4uVer.□

Tcp4uCleanup

This function closes opened sockets, frees internal buffers. This function fails only if a blocking call is in progress. In this case, it tries to cancel the blocking call, thus the next Tcp4uCleanup succeeds.

If Tcp4uInit has not been called, Tcp4uCleanup returns TCP4U_SUCCESS.

The sample below shows a "simple" way to close a application.

Syntax: Tcp4uCleanup ()

Return: TCP4U_ERROR A Blocking call is in progress
 TCP4U_SUCCESS Cleanup done

Example :

```
case WM_TIMER :
    KillTimer (hWnd, wParam);
    PostMessage (hWnd, wParam, 0, 0);
    break;

case WM_CLOSE :
    if (Tcp4uCleanup ()==TCP4U_ERROR)
    { SetTimer (hWnd, WM_CLOSE, 5001, 0);
      return FALSE; }
    else
    { DestroyWindow (hWnd);
      PostQuitMessage (); }
    break;
```

Tcp4uErrorString

This function returns the error string associated with the given error code.

This function uses the following association:

TCP4U_SUCCESS	Successful call
TCP4U_ERROR	Network error
TCP4U_TIMEOUT	Timeout in recv or accept
TCP4U_HOSTUNKNOWN	Reference to Unknown host
TCP4U_NOMORESOCKET	All sockets has been used up
TCP4U_NOMORERESOURCE	No more free resource
TCP4U_CONNECTFAILED	Connect function has failed
TCP4U_UNMATCHEDLENGTH	TcpPPRecv : Error in length
TCP4U_BINDERROR	Bind failed (Task already started?)
TCP4U_OVERFLOW	Overflow during TcpPPRecv/TcpRecvUntilStr
TCP4U_EMPTYBUFFER	TcpPPRecv receives 0 byte
TCP4U_CANCELLED	Call cancelled by signal or TcpAbort
TCP4U_INSMEMORY	Not enough memory to perform function
TCP4U_SOCKETCLOSED	Host has closed connection
OTHER...	Not an Tcp4 return code

Syntax: Tcp4uErrorString (int Rc)

Argument: Rc An error code

Return: A long pointer on the error string associated with Rc.

Example :

```
Rc = TcpRecv (Socket, sz, sizeof sz, HFILE_ERROR);
if (Rc < TCP4U_SUCCESS)
{
    MessageBox (hWnd, Tcp4uErrorString(Rc), "Sample App", MB_OK);
    PostMessage (hWnd, WM_CLOSE, 0, 0);
}
```

Tcp4uInit

This function initializes the internal structures of Tcp4w and calls WSASStartup which initializes the module Winsock.Dll.

It must be called by each task which want to use Tcp4w services.

Syntax: Tcp4uInit ()

Return:	TCP4U_ERROR	Tcp4w has not found a correct Winsock.Dll.
	TCP4U_SUCCESS	Task is registered.

Example :

```
case WM_CREATE :
    if (Tcp4uInit ( ) != TCP4U_SUCCESS)
    {
        MessageBox (hWnd, "Init Error", "Sample App", MB_OK);
        PostMessage (hWnd, WM_CLOSE, 0, 0);
    }
```

Tcp4uVer

Tcp4w returns the version number of the DLL, as an integer. The low order byte is the release number, the high order byte is the major version number.

The function copies in the user's buffer a string which contains information on the DLL (name, version, author, copyright). This string is guaranteed not to exceed 100 characters.

Syntax: `Tcp4uVer(LPSTR szVerStr, int nStrSize)`
(LPSTR is a 32-bits pointer)

Arguments: `szVerStr`: a buffer which is to receive the version information
`nStrSize`: its size

Return: an integer which contains the version.

Example:

```
char szStr[100];  
Tcp4uVer (szStr, sizeof szStr);  
MessageBox (NULL, szStr, "Tcp4w Test", MB_OK);
```

TcpAbort

This function cancels a Winsocket blocking operation for this task.

As said above, while a process is waiting for a blocking call to be completed, it continues to receive messages (Timer, user's actions). This blocking operation must be cancelled before any other network related functions can be called.

When TcpAbort returns, the blocking call is cancelled, but has not returned the control to the application. Thus the following code does not free the socket:

```
case WM_CLOSE :
    TcpAbort ();
    TcpClose (& Socket);
    PostQuitMessage(0);
```

The TcpClose call will fail if a blocking call is in progress. TcpAbort has marked it as cancelled, but a pause is necessary.

Syntax: TcpAbort ()

Return: TCP4U_SUCCESS OK

Example:

This example shows one of the safest ways to close an application.

```
case WM_TIMER :
    KillTimer (hWnd, wParam);
    PostMessage (hWnd, wParam, 0, 0);
    break;

case WM_CLOSE :
    TcpAbort ();
    SetTimer (hWnd, WM_USER, 100, NULL);
    break;

case WM_USER :
    TcpClose (& Socket);
    Tcp4uCleanup ();
    DestroyWindow (hWnd);
    PostQuitMessage(0);
    break;
```


TcpAccept

This function waits for a connection and establishes it. The function blocks until a client tries to establish a connection with the host.

This function must be called after a successful TcpGetListenSocket.

Syntax: TcpAccept (SOCKET FAR *pConnSock, SOCKET ListenSock, UINT TimeOut)
(SOCKET is an unsigned 16-bit integer)

Arguments:	pConnSock	A 32-bit pointer on a socket descriptor. TcpAccept fills it with a value which identifies the connection.										
	ListenSock	A socket descriptor given by the FtpGetListenSock										
	TimeOut	the value of the timeout in second. 0 means that the function will block until a connection is requested.										
Returns	<table><tr><td>TCP4U_BUFFERFREED</td><td>The function can not write into pConnSock</td></tr><tr><td>TCP4U_CANCELLED</td><td>The call has been cancelled by TcpAbort or Tcp4uCleanup</td></tr><tr><td>TCP4U_ERROR</td><td>An unexpected error has occurred or a blocking call is in progress</td></tr><tr><td>TCP4U_TIMEOUT</td><td>A timeout occurred</td></tr><tr><td>TCP4U_SUCCESS</td><td>A connection has been requested and its descriptor is available in the pConnSocket descriptor</td></tr></table>		TCP4U_BUFFERFREED	The function can not write into pConnSock	TCP4U_CANCELLED	The call has been cancelled by TcpAbort or Tcp4uCleanup	TCP4U_ERROR	An unexpected error has occurred or a blocking call is in progress	TCP4U_TIMEOUT	A timeout occurred	TCP4U_SUCCESS	A connection has been requested and its descriptor is available in the pConnSocket descriptor
TCP4U_BUFFERFREED	The function can not write into pConnSock											
TCP4U_CANCELLED	The call has been cancelled by TcpAbort or Tcp4uCleanup											
TCP4U_ERROR	An unexpected error has occurred or a blocking call is in progress											
TCP4U_TIMEOUT	A timeout occurred											
TCP4U_SUCCESS	A connection has been requested and its descriptor is available in the pConnSocket descriptor											

TcpClose

This function closes a socket created with TcpAccept, TcpGetListenSocket or TcpConnect.

Syntax: TcpClose (SOCKET FAR *pSock)

Arguments: pSock A 32-bit pointer on a socket descriptor. TcpClose fills it to INVALID_SOCKET.

Returns	TCP4U_BUFFERFREED	The function can not write into pSock
	TCP4U_ERROR	An unexpected error has occurred or a blocking call is in progress
	TCP4U_SUCCESS	The Socket has been successfully closed

TcpConnect

This function tries to establish a connection with a remote peer.

Syntax: TcpConnect (SOCKET far *pS, LPSTR szHost, LPSTR szService, short far *lpPort)

Arguments:	pS	A 32-bit pointer on a socket descriptor. It is filled by TcpConnect
	szHost	The name or the IP-Address of the server to be reached
	szService	The name of the TCP service which is to be used (NULL if any)
	lpPort	A 32-bit pointer on a 16-bit integer. This integer contains the value of the Port which is to be used. If the argument szService is not NULL, TcpConnect tries to locate the port in the file <i>service</i> , if it fails the value contained into lpPort is used, if it succeeds TcpConnect fills the value to the port returned.
Return:	TCP4U_BUFFERFREED	The function can not write into pS
	TCP4U_CONNECTFAILED	The server can not be reached or no process are listening on the port.
	TCP4U_ERROR	An unexpected error has occurred or a blocking call is in progress
	TCP4U_HOSTUNKNOWN	The host has not been found in the file <i>hosts</i> , or the dotted address is not valid (ie: "10.10.10.543")
	TCP4U_NOMORESOCKET	All sockets have been used.
	TCP4U_SUCCESS	The connection has been established and its descriptor is available in the pS descriptor

Example:

This sample app will try to establish a connection with the FTP server *ftp.ciril.fr*. It waits for its answer, then closes the connection and terminates.

```
#include <windows.h>
#include <tcp4w.h>

static char szHost[] = "ftp.ciril.fr";

int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
                    LPSTR lpszCmdLine, int nCmdShow)
{
    char    szStr [256];
    char    szReply [100];
    SOCKET  CSock;
    int     Rc;
    unsigned short  nPort=0;

    if (Tcp4uInit () != TCP4U_SUCCESS)    return 0;
```

```
Rc = TcpConnect (& CSock, szHost, "ftp", & nPort);
if (Rc==TCP4U_SUCCESS)
{
    TcpRecv (CSock, szReply, sizeof szReply-1, 60, HFILE_ERROR);
    TcpClose (& CSock);
    wsprintf ( szStr,
        "Connection successful on port %d\nserver said:\n\n%s",
        nPort, (LPSTR) szReply);
}
else
    wsprintf (szStr, "Connection failed\nerror %d", Rc);
MessageBox (NULL, szStr, "Test Tcp4w", MB_OK);
Tcp4uCleanup ();
return 0;
} /* WinMain */
```

TcpFlush

This function empties the reception buffer of a given socket.

Syntax: TcpFlush (SOCKET S)

Arguments: S A Socket returned by TcpConnect or TcpAccept.

Return:

TCP4U_CANCELLED	Operation has been cancelled with TcpAbort or Tcp4uCleanup
TCP4U_ERROR	An unexpected error has occurred or a blocking call is in progress
TCP4U_SOCKETCLOSED	The remote peer has closed its connection. No more data are to be received.
TCP4U_SUCCESS	Reception buffer is now empty.

TcpGetListenSocket

This function prepares the application for accepting connection on a given port.

Syntax: `TcpGetListenSocket (SOCKET far *pS, LPSTR szService, short far *lpPort, int nPendingConnection)`

Arguments: `pS` A 32-bit pointer on a socket descriptor. It is filled with a value which is to be passed to `TcpAccept` in order to establish connection.
`szService` A string pointer on the service used by the application. if the pointer is NULL or the service has not been found, the value given into `lpPort` is used.
`lpPort` A 32-bit pointer on a value which should contain the port on which the application should await connections. If `szService` is a valid service name, `lpPort` will contain the port corresponding to this service.
`nPendingConnection` The maximum length to which the queue of pending connection may grow. Illegal values (less than 1 or greater than 5) are replace by the nearest legal value.

Return: `TCP4U_BUFFERFREED` The `pS` pointer is not usable.
 `TCP4U_CANCELLED` Operation has been cancelled with `TcpAbort` or `Tcp4uCleanup`
 `TCP4U_ERROR` An unexpected error has occurred or a blocking call is in progress
 `TCP4U_NOMORESOCKET` All sockets have been taken.
 `TCP4U_SUCCESS` The application is now listening for incoming connections `TcpAccept` must be called to establish them.

Example:

This sample application is an FTP server which accepts any incoming connection then rejects the client by sending a reject string and dies. Since this sample does not have a window, it must be tested by an FTP client as shown above.

```
#include <windows.h>
#include <tcp4w.h>

static char szRejectStr[] = "421 No command implemented\r\n";

int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
                    LPSTR lpszCmdLine, int nCmdShow)
{
    SOCKET        ListenSock, ConnSock;
    int           Rc;
    short        nPort=0;
    if (Tcp4uInit () != TCP4U_SUCCESS)    return 0;
    Rc = TcpGetListenSocket (& ListenSock, "ftp", & nPort, 1);
    if (Rc==TCP4U_SUCCESS)
    {
        Rc = TcpAccept (& ConnSock, ListenSock, 600);
        if (Rc==TCP4U_SUCCESS)
        {
            TcpSend (ConnSock, szRejectStr, sizeof szRejectStr-1,
```

```
                FALSE, HFILE_ERROR);  
        TcpClose (& ConnSock);  
    }  
    TcpClose (& ListenSock);  
}  
    Tcp4uCleanup ();  
return 0;  
} /* WinMain */
```

TcpGetLocalID

This function identifies the local station.

Syntax: **TcpGetLocalID** (LPSTR szStrName, int uNameSize, DWORD far *IpAddress)

Arguments: szStrName A 32-bit pointer on a string which is to be filled with the name of the station. If the pointer is NULL, the name of the station is not returned.

 uNameSize The size of the buffer szStrName.

 IpAddress A 32-bit pointer on a double word which is to be filled with the IP address of the station. If this pointer is NULL, the IP-Address is not returned.

Returns: **TCP4U_ERROR** The IP-stack was unable to get the name of the local station.

TCP4U_OVERFLOW The name is longer than the given buffer.

TCP4U_SUCCESS The requested data are available.

Example:

```
char StationName[64];
DWORD IPAddress;
unsigned char *cAdd = (char *) &IPAddress;

TcpGetLocalID( StationName, sizeof StationName, & IPAddress);
wsprintf (szBuf, "Local Station is %s, Address %u.%u.%u.%u",
          StationName, cAdd[0], cAdd[1], cAdd[2], cAdd[3]);
```


TcpGetRemoteID

This function identifies the peer station.

Syntax: TcpGetRemoteID (SOCKET s, LPSTR szStrName, int uNameSize,
 DWORD far *IpAddress)

Arguments: S The socket by which the peer station is connected.
 szStrName A 32-bit pointer on a string which is to be filled with the name of the
 station. If the pointer is NULL, the name of the station is not
 returned.
 uNameSize The size of the buffer szStrName.
 IpAddress A 32-bit pointer on a double word which is to be filled with the IP
 address of the station. If this pointer is NULL, the IP-Address is
 not returned.

Returns: TCP4U_ERROR The IP-stack was unable to get the name of the remote
 station. The socket descriptor does not identify a
 connection.
 TCP4U_OVERFLOW The name is longer than the given buffer.
 TCP4U_SUCCESS The requested data are available.

Example:

```
char StationName[64];
DWORD IPAddress;
unsigned char *cAdd = (char *) &IPAddress;

TcpGetRemoteID (Skt, StationName, sizeof StationName, &IPAddress);
wsprintf (szBuf, "Local Station is %s, Address %u.%u.%u.%u",
          StationName, cAdd[0], cAdd[1], cAdd[2], cAdd[3]);
```

TcpPPRecv

This function receives a frame with a timeout. The first 16-bit of this frame contains its length in network order. They are NOT passed into the user's buffer.

Syntax: TcpPPRecv (SOCKET s, LPSTR szBuf, unsigned uBufSize, unsigned uTimeout, BOOL bExact, HFILE hLogFile)

Arguments:	s	The descriptor of the connection returned by TcpConnect or TcpAccept
	szBuf	The user's buffer
	uBufSize	Its size in bytes
	uTimeOut	The timeout in second
	bExact	Specify if the TcpPPRecv function should receive exactly uBufSize bytes.
	hLogFile	A file (returned by _lopen, _lcreat) in which the data are written. To disable this feature, this argument should be HFILE_ERROR

Returns:

- TCP4U_BUFFERFREED The user's buffer has been released.
- TCP4U_CANCELLED The request has been cancelled by TcpAbort or Tcp4uCleanup
- TCP4U_EMPTYBUFFER The frame is empty. (Its length is 0 byte)
- TCP4U_ERROR An unexpected error has occurred or a blocking call is already in progress
- TCP4U_INSMEMORY Tcp4w can not allocate its temporary buffers
- TCP4U_OVERFLOW The frame to be received is greater than the user's buffer
- TCP4U_SOCKETCLOSED The remote host has closed its connection.
- TCP4U_UNMATCHEDLENGTH The parameter bExact is set to TRUE and the expected length differs from the received one.
- >0 The length in bytes of the received frame. The frame is copied into the user's buffer except the first two bytes.

Example:

```
char szBuf[64];

Rc = TcpPPRecv (Skt, szBuf, sizeof szBuf, 60, FALSE, HFILE_ERROR);
wsprintf (szStr, "%d bytes have been received\nBuffer %s", Rc, szBuf);
MessageBox (hWnd, szStr, "Test Tcp4w", MB_OK);
```

TcpPPSend

This function sends a two-bytes frame which are the size of the data to be sent, then the data are sent.

Syntax: TcpPPSend (SOCKET s, LPSTR szBuf, unsigned uBufSize, HFILE hLogFile)

Arguments: s The descriptor of the connection returned by TcpConnect or
 TcpAccept
 szBuf The user's buffer
 uBufSize Its size in bytes
 hLogFile A file (returned by _lopen, _lcreat) in which the data are written.
 To disable this feature, this argument should be HFILE_ERROR

Returns: TCP4U_CANCELLED The request has been
 cancelled by TcpAbort or Tcp4uCleanup
 TCP4U_ERROR An unexpected error
 has occurred or a blocking call is already in progress
 TCP4U_INSMEMORY Tcp4w can not allocate its
 temporary buffers
 TCP4U_SUCCESS The data have been
 sent.

Example:

```
char szBuf[] = "Bonjour";  
Rc = TcpPPSend (Skt, szBuf, strlen(szBuf), HFILE_ERROR);
```

TcpRecv

This function receives data sent by the remote station.

Syntax: TcpRecv (SOCKET s, LPSTR szBuf, unsigned uBufSize, unsigned uTimeout, HFILE hLogFile)

Arguments:	s	The descriptor of the connection returned by TcpConnect or TcpAccept
	szBuf	The user's buffer
	uBufSize	Its size in bytes
	uTimeOut	The timeout in second
	hLogFile	A file (returned by _lopen, _lcreat) in which the data are written. To disable this feature, this argument should be HFILE_ERROR

Returns:

- TCP4U_BUFFERFREED The user's buffer has been released.
- TCP4U_CANCELLED The request has been cancelled by TcpAbort or Tcp4uCleanup
- TCP4U_EMPTYBUFFER The frame is empty. (Its length is 0 byte)
- TCP4U_ERROR An unexpected error has occurred or a blocking call is already in progress
- TCP4U_INSMEMORY Tcp4w can not allocate its temporary buffers
- TCP4U_OVERFLOW The frame to be received is greater than the user's buffer
- TCP4U_SOCKETCLOSED The remote host has closed its connection.

>0 The length in bytes of the received frame. The frame is copied into the user's buffer except the first two bytes.

Example:

```
char szBuf[64];

Rc = TcpRecv (Skt, szBuf, sizeof szBuf, 60, HFILE_ERROR);
wsprintf (szStr,"%d bytes have been received\nBuffer %s",Rc,szBuf);
MessageBox (hWnd, szStr, "Test Tcp4w", MB_OK);
```

TcpRecvUntilStr

This function receives data sent by the remote station until a particular string has been received.

Note: This function should not be used with the Novell stack LAN Workplace.

Syntax: TcpRecvUntilStr (SOCKET s, LPSTR szBuf, unsigned far *lpBufSize,
 LPSTR szStop, unsigned uStopSize, BOOL bCaseSensitive,
 unsigned uTimeout, HFILE hLogFile);

Arguments: s The descriptor of the connection returned by TcpConnect or
 TcpAccept
 szBuf The user's buffer
 lpBufSize A 32-bit pointer on a word which must be initialized with the size
 of the user's buffer. In return, this word will contain the length of the
 received frame.
 szStop The string which ends the reception
 uStopSize The length of this string
 bCaseSensitive set to TRUE if the end-string is case sensitive.
 uTimeout The timeout in second
 hLogFile A file (returned by _lopen, _lcreat) in which the data are written.
 To disable this feature, this argument should be HFILE_ERROR

Returns: TCP4U_BUFFERFREED The user's buffer has been
 released.
 TCP4U_CANCELLED The request has been
 cancelled by TcpAbort or Tcp4uCleanup
 TCP4U_ERROR An unexpected error
 has occurred or a blocking call is already in progress
 TCP4U_INSMEMORY Tcp4w can not allocate its
 temporary buffers
 TCP4U_OVERFLOW The user's buffer has been
 filled and the searched string has not been received.
 TCP4U_SOCKETCLOSED The remote host has
 closed its connection.
 TCP4U_SUCCESS The string has been
 received (but has not been copied into the user's
 buffer).

Example:

```
char szBuf[64];
char szEndStr[] = "#END#";
UINT nLength = sizeof szBuf;

Rc = TcpRecvUntilStr (Skt, szBuf, & nLength,
                     szEndStr, sizeof szEndStr,
                     60, HFILE_ERROR);
wsprintf (szStr,"%d bytes have been received\nBuffer %s",
          nLength, szBuf);
MessageBox (hWnd, szStr, "Test Tcp4w", MB_OK);
```

TcpSend

This function sends the given data.

Syntax: `TcpSend (SOCKET s, LPSTR szBuf, unsigned uBufSize, BOOL bHighPriority, HFILE hLogFile)`

Arguments: `s` The descriptor of the connection returned by `TcpConnect` or `TcpAccept`
 `szBuf` The user's buffer
 `uBufSize` Its size in bytes
 `bHighPriority` set to TRUE if the data are to be sent in Out Of Band mode.
 `hLogFile` A file (returned by `_lopen`, `_lcreat`) in which the data are written.
 To disable this feature, this argument should be `HFILE_ERROR`

Returns: `TCP4U_CANCELLED` The request has been cancelled by `TcpAbort` or `Tcp4uCleanup`
 `TCP4U_ERROR` An unexpected error has occurred or a blocking call is already in progress
 `TCP4U_INSMEMORY` `Tcp4w` can not allocate its temporary buffers
 `TCP4U_SUCCESS` The data have been sent.

Example:

```
char szBuf[] = "Bonjour";  
Rc = TcpSend (Skt, szBuf, strlen(szBuf)+1, FALSE, HFILE_ERROR);
```

TnGetAnswerCode

This function receives a complete telnet string. Valid strings are described by the RFC 764. For instance, here are two valid strings:

```
220 Hello
or
220-
220-      Welcome to our FTP server
220-
220
```

The function returns either a telnet code (the 3 digits at the beginning of the string) or an error code.

Syntax: TnGetAnswerCode (SOCKET skt, LPSTR szBuf, UINT uBufSize, UINT uTimeOut, HFILE hf);

Arguments:	skt	The descriptor of the connection returned by TcpConnect or TcpAccept
	szBuf	The user's buffer into which the string will be copied
	uBufSize	Its size
	uTimeOut	The timeout
	hf	A file handler into which the data will be copied

Returns:	TN_ERROR	An error or a timeout has occurred
	100< Rc < 999	Rc is a telnet answer code

TnReadLine

This function receives a string ended by an EOL character (ASCII 10).
A NUL character is added to the returned string.

Syntax: TnReadLine (SOCKET skt, LPSTR szBuf, UINT uBufSize,
 UINT uTimeOut, HFILE hf);

Arguments:	skt	The descriptor of the connection returned by TcpConnect or TcpAccept
	szBuf	The user's buffer into which the string will be copied
	uBufSize	Its size
	uTimeOut	The timeout
	hf	A file handler into which the data will be copied

Returns:	TN_BUFFERFREED	szBuf is not a writeable buffer
	TN_ERROR	An error has occurred
	TN_OVERFLOW	uBufSize character have been transferred into the user's buffer, but the EOL character has not been found
	TN_TIMEOUT	A timeout has occurred
	TN_SOCKETCLOSED	Remote host has closed the connection
	TN_SUCCESS	The string is available into szBuf

TnSend

This function sends a 0-terminated string, then sends the telnet termination string (<CR><LF>).

Syntax: TnSend (SOCKET skt, LPSTR szString, BOOL bHighPriority, HFILE hf);

Arguments: skt The descriptor of the connection returned by TcpConnect or
 TcpAccept
 szString The string to be sent
 bHighPriority TRUE if the string must be sent in Out Of Band mode
 hf A file handler into which the data will be written

Returns: TN_ERROR An error has occurred
 TN_SUCCESS The string has been successfully sent