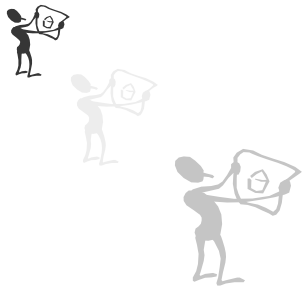
Chapter 8

TCP Communications

Process Control

Outline

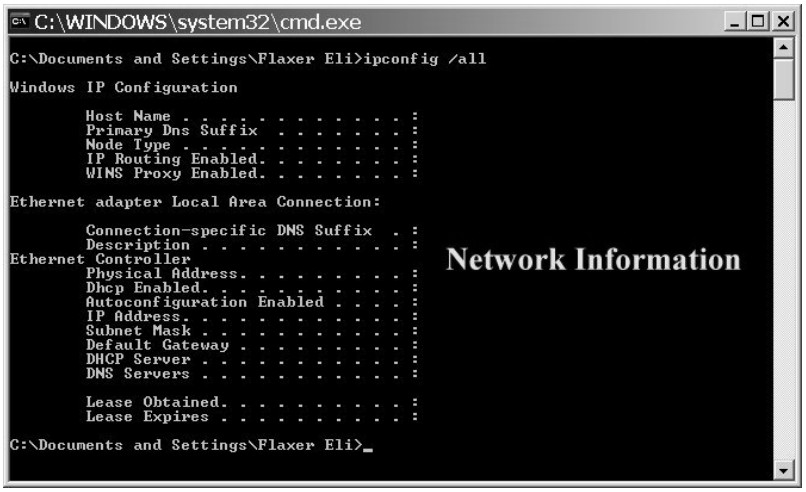
- **Introduction TCP parameters**
- **TCP Library in CVI**
- **Call Back Functions**
- **Functions Pointer**



Introduction

Physical Address (ipconfig /all)
Name & IP
D.N.S. – Domain Name Server
Port Number
Client & Server

Physical & Virtual Address (ipconfig /all)



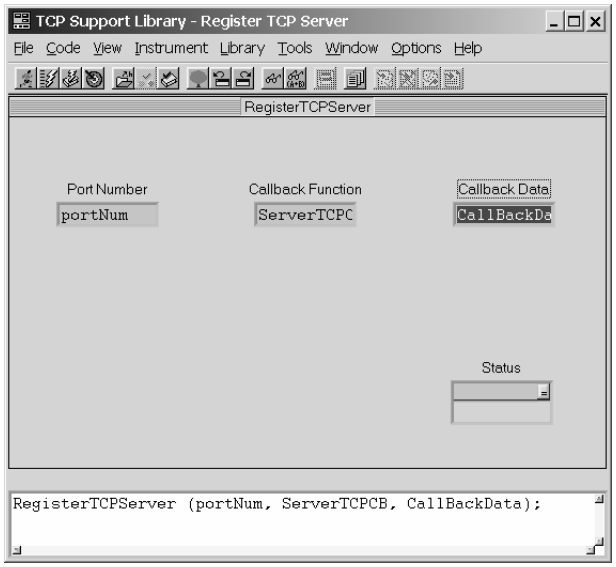
TCP Library in CVI

Server Functions

Client Functions

Support Functions

Register TCP Server



TCP Server Callback Function

The **callback function** must be of the following form:
int (*tcpFuncPtr) (unsigned handle, int xType, int errCode, void *callbackData);

handle:TCP Support Library conversation handle that you obtain from ConnectToTCPServer or receive in a server callback as the handle parameter of a TCP_CONNECT message

xType specifies the type of message the server sends.

The server callback function can receive the following transaction types:

TCP_CONNECT - Received when a client requests a connection.

TCP_DISCONNECT - Received when a client requests a disconnection.

TCP_DATAREADY - Received when there is data to be read by the server. Calls ServerTCPRead to obtain the data.

Use **errCode** only when the transaction type is TCP_DISCONNECT.

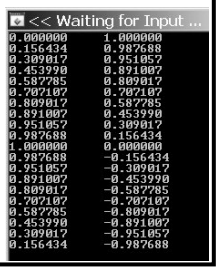
callbackData: void* value that the TCP Support Library passes to the callback function each time the library invokes the callback for the same server.
You can pass zero if you do not want to use the callback data.

Function's Pointer

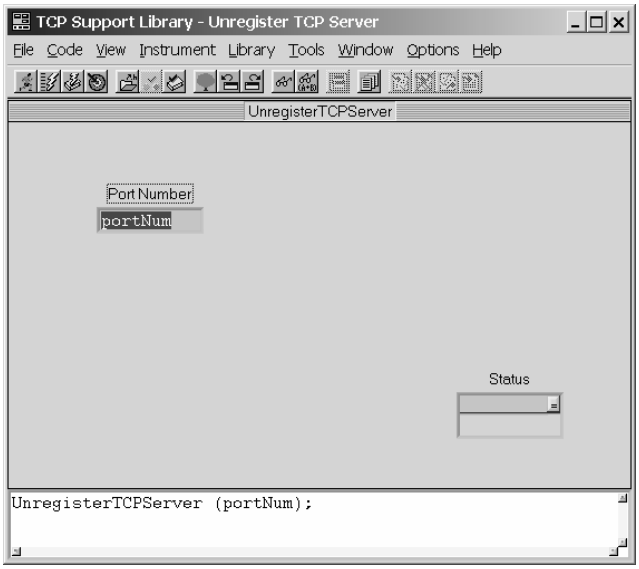
```
##include "toolbox.h"
#include <ansi_c.h>

typedef double (*FPDD)(double); // Pointer to function that return double & accept double

FPDD F1;

void main()
{
double X, Y;
F1 = sin;
X =  Pi()/6;
Y = F1(X);
printf("%lf \n", Y);

F1 = cos;
X =  Pi()/4;
Y = F1(X);
printf("%lf \n", Y);

getchar();
}
```

Function as Argument

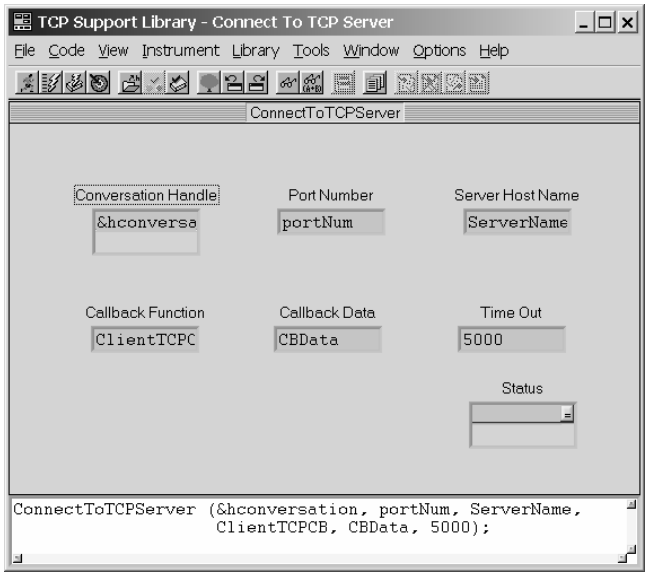
```
#include "toolbox.h"
#include <ansi_c.h>
typedef double (*FPDD)(double); // Pointer to function that returen double

void FillArrayWithFuncValues(FPDD f, double Array[], double Min, double Max, int N)
{
int i;
double T = (Max-Min)/N;
for (i=0; i<N; i++)
        Array[i] = f(Min+T*i);
}

void main()
{
#define LEN 20
int i;
double ArraySin[LEN];
double ArrayCos[LEN];
FillArrayWithFuncValues(sin, ArraySin, 0.0, Pi(), LEN);
FillArrayWithFuncValues(cos, ArrayCos, 0.0, Pi(), LEN);

for (i=0; i<LEN; i++)
        printf("%lf     %lf \n", ArraySin[i], ArrayCos[i]);;
getchar();
}
```



UnRegister TCP Server



Connect To TCP Server

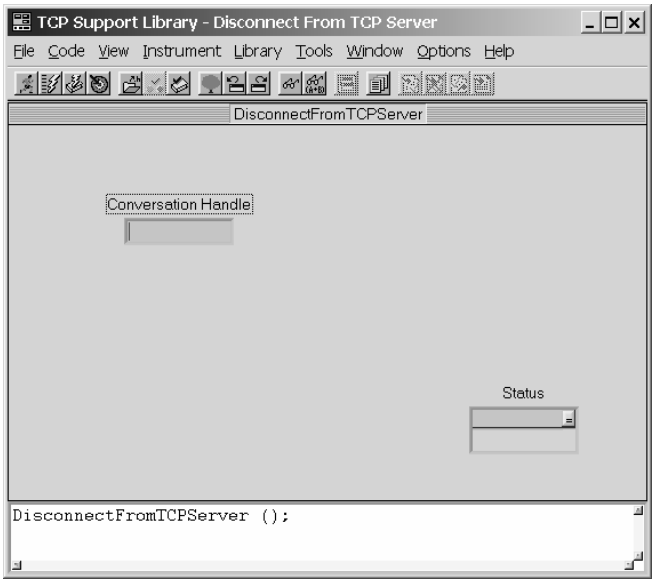


TCP Client Callback Function

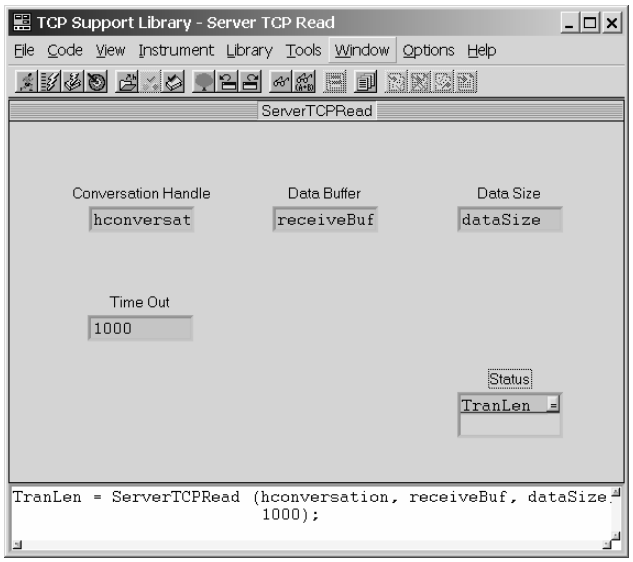
The **callback function** must be of the following form:
int (*tcpFuncPtr) (unsigned handle, int xType, int errCode, void *callbackData);

handle: contains a value that is unique to each client-server connection.
xType specifies the type of message the server sends.
The client callback function can receive the following transaction types:
TCP_DISCONNECT - Received when a server requests the termination of a connection or when a connection terminates because of an error.
TCP_DATAREADY - Received when there is data to be read by the client. Calls ClientTCPRead to obtain the data.
Use **errCode** only when the transaction type is TCP_DISCONNECT.
callbackData: void* value that the TCP Support Library passes to the callback function each time the library invokes the callback for the same server.
You can pass zero if you do not want to use the callback data.

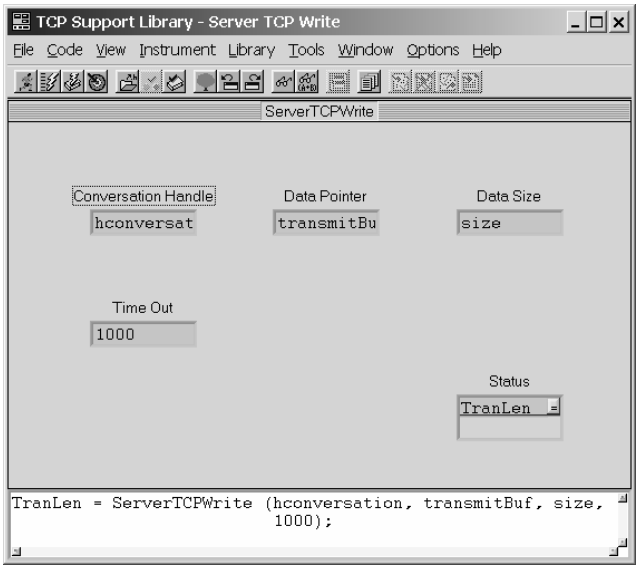
DisConnect From TCP Server



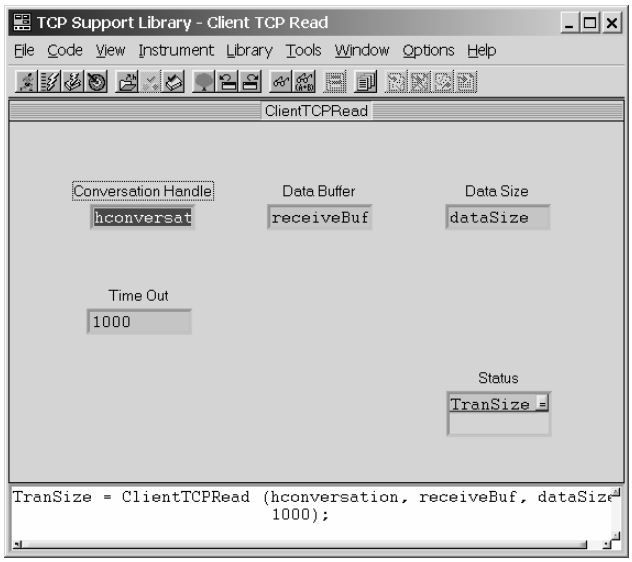
Server TCP Read



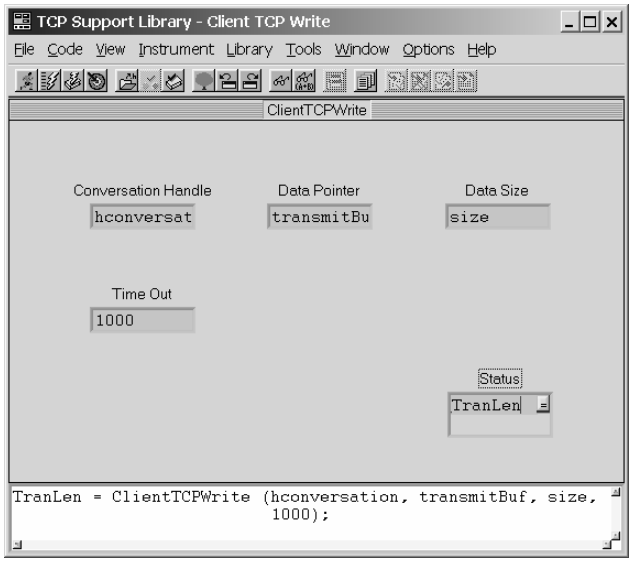
Server TCP Write



Client TCP Read



Client TCP Write



TCP Server CallBack (P1)

```
/*---------------------------------------------------------------------------*/
/* This is the TCP server's TCP callback.  This function will receive event  */
/* notification, similar to a UI callback, whenever a TCP event occurs.      */
/* We'll respond to CONNECT and DISCONNECT messages and indicate to the user */
/* when a client connects to or disconnects from us.  when we have a client  */
/* connected, ee'll respond to the DATAREADY event and read in the avaiable  */
/* data from the client and display it.                                      */
/*---------------------------------------------------------------------------*/
int CVICALLBACK ServerTCPCB (unsigned handle, int event, int error, void *callbackData)
{
   int  dataSize        = 256;
   char receiveBuf[256]  = {0};
   char addrBuf[31];
   switch (event)
      {
      case TCP_CONNECT:
           /* Handle this new client connection */
            hconversation = handle;
            GetTCPPeerAddr (hconversation, addrBuf, 31);
            GetTCPPeerName (hconversation, receiveBuf, 256);
            sprintf (receiveBuf, "-- New connection from %s --\n", addrBuf);
            /* Set the disconect mode so we do not need to terminate */
            /* connections ourselves. */
            SetTCPDisconnectMode (hconversation, TCP_DISCONNECT_AUTO);
            }
         break;
```


TCP Server CallBack (P2)

```
      case TCP_DATAREADY:
            dataSize = ServerTCPRead (hconversation, receiveBuf, dataSize, 1000)
            (*ServerDataRdy)(receiveBuf, &dataSize);
            break;

      case TCP_DISCONNECT:
         if (handle == hconversation)
            {

            /* The client we were talking to has disconnected... */
            SetCtrlVal (serverPanel, SERVERPNL_CONNECTED, 0);
            hconversation = 0;
            SetCtrlVal (serverPanel, SERVERPNL_CLIENT_IP, "");
            SetCtrlVal (serverPanel, SERVERPNL_CLIENT_NAME, "");
            SetCtrlVal (serverPanel, SERVERPNL_MESSAGE,  "-- Client disconnected --\n");

            /* Note that we do not need to do any more because we set the*/
            /* disconnect mode to AUTO. */
            }
         break;
   }
   return 0;
}
```

TCP Client CallBack

```
/*---------------------------------------------------------------------------*/
/* This is the TCP client's TCP callback.  This function will receive event  */
/* notification, similar to a UI callback, whenever a TCP event occurs.      */
/* We'll respond to the DATAREADY event and read in the available data from  */
/* the server and display it.  We'll also respond to DISCONNECT events, and  */
/* tell the user when the server disconnects us.                             */
/*---------------------------------------------------------------------------*/
int CVICALLBACK ClientTCPCB (unsigned handle, int event, int error, void *callbackData)
{
   int  dataSize       = 256;
   char receiveBuf[256] = {0};
   switch (event)
      {
      case TCP_DATAREADY:
            dataSize = ClientTCPRead (hconversation, receiveBuf, dataSize, 1000
            (*ClientDataRdy)(receiveBuf, &dataSize);
         break;
      case TCP_DISCONNECT:
         MessagePopup ("TCP Client", "Server has closed connection!");
         hconversation = 0;
         break;
   }
   return 0;
}
```