EDGETECH SONAR INTERFACE DEMO

DEMONSTRATION OF EDGETECH SONAR INTERFACE

Document No. 990-0000088-1000

Revision: 1.0 / March 2007

Email: sales@edgetech.com
Web: http://www.edgetech.com

4 Little Brook Road
West Wareham, MA 02576
Tel: (508) 291-0057
Fax: (508) 291-2491

141 Holland Drive, Suite 1
Boca Raton, FL 33487
Tel: (561) 995-7767
Fax: (561) 995-7761

TABLE OF CONTENT

TABLES

FIGURES

1 EdgeTech Demonstration Control Interface

EdgeTech Demonstration Control Interface is a sample application that controls the sonar via an Ethernet socket connection to the control port. It does not connect to the data port or handle sonar data in any manner. The purpose is to demonstrate to third party interface developers a minimal functional control system with simplicity and clarity.

The application is written in MFC. The package includes Visual C++ project file, C header files, **.**CPP source files and resource files. They are ready to build in Microsoft Visual Studio 2005 environment. A Microsoft Windows platform and Microsoft Visual Studio 2005 environment are required for running and building this application. However, the header files and **.**CPP files can be edited in other editing environment.

1.1 Functional Description

It is required to be able to control the 3 typical subsystems, being Low frequency Side Scan, High frequency Side Scan, and Sub-Bottom profiler. The controls provided include Enable/Disable Recording, Set the Time, Enable/Disable Pinging per subsystem, Set the Ping Rate based on the range in meters per subsystem, Set the Transmit Gain per subsystem, Set the Trigger mode per subsystem, Set the Coupling Parameters per subsystem, Display Socket connection status information and Display Sonar Status.

1.2 Implementation

The program contains two threads, a socket interface and a user interface.

The socket interface manages the socket connection, sends and receives messages to and from the socket. It receives outgoing messages one at a time from the User Interface thread via outgoing message storage. It stores messages received from the socket in the received message storage buffer. The latest read time and socket connection statistics are also collected here. If the sent message is a set timestamp message, socket's nagle algorithm is disabled. Please see SonarMessages.h and StorageMessages.h for information about these messages, types and formats.

The user interface thread manages the user interface. It displays the socket connection status and Sonar System status. It also sends completely formatted outgoing messages one at a time to the socket interface. When the application starts up, it initiates the graphic command controls, starts the socket thread and timer. When the socket connection is down,

the graphic command controls are disabled. After the connection is made, all the control commands will be send, and the graphic command controls enabled. A timer callback routine is initiated to collect and display socket connection status and sonar status from the socket interface every 200 milliseconds, and to send an outgoing message to the socket interface to collect the Sonar Status information every 2 seconds. The Sonar Status message also serves as a keep-alive message to detect connection failure. When 2 second timer expires, it compares the latest read time set by socket interface to the current time. If the difference is larger than 4 seconds, it closes the socket and the socket reconnects.

1.3 User Interface

The graphic user interface is as shown below:

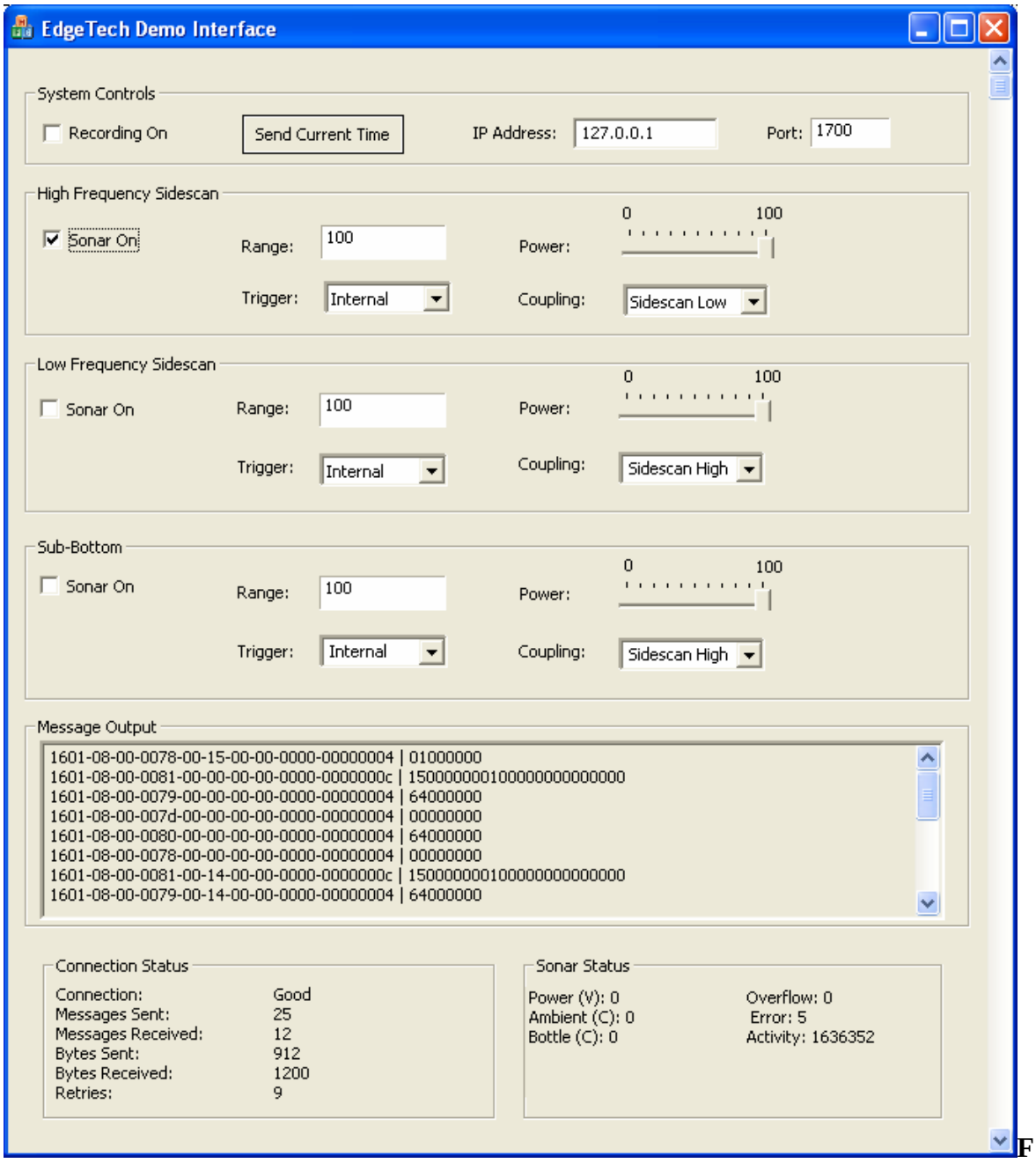

Figure 1: Demo Interface GUI

The system controls consist of configuration settings for the demonstration interface and the two featured system level controls. Unlike most commands that only affect a single subsystem or channel these affect the system as a whole.

There are three sets of sonar controls one for each of the commonly present subsystems. Which system is affected is determined by the subsystem and channel fields of the message header.

In the user interface, changes to the Recording On check box, Send Current Time button, the Sonar On check box, Range edit box, Power sidebar, Trigger combo box, and Coupling combo box under different subsystem group (High Frequency Side Scan, Low Frequency Side Scan and Sub-Bottom) trigger a Sonar message to be sent.

The IP Address edit box, and Port edit box configure the IP address and port number for the control socket. After the value of either box changes with the return key entered, the socket will disconnect and reconnect to the specified IP address and port.

The Message Output edit box displays the message data when the message is sent out.

Connection Status edit box displays the socket connection status information. The information is updated every 200 ms.

Sonar Status edit box displays the Sonar status information. The information is updated every 200 ms.

1.4 Command Messages

The messages sent and received between control applications and sonar systems are Sonar Messages. They always consist of two parts: A header which contains the message type and length, and the actual data. All messages are preceded with a header which indicates the size of the message to follow in bytes and its type. The header is of type SonarMessageHeaderType. The length of the actual data is decided by the data format used for the message type. Please see SonarMessages.h and StorageMessages.h for information about these messages, types and formats

The messages sent and received in the application, corresponding to the controls, are listed in the table below.

CONTROLS	MESSAGE TYPE / (FORMAT)	DESCRIPTION
Recording on check box	`STORAGE_MESSAGE_MASTER_RECORD (SonarMessageLongType)`	Enable / Disable recording when click the check box. The sonar must already be configured with all of the other recording parameters set.
Set Current Time button	`SONAR_MESSAGE_SYSTEM_TIME (TimestampType)`	Set the current time when click the button. Disable the socket nagle algorithm before sending the

		message.
Sonar On check box	SONAR_MESSAGE_PING (SonarMessageLongType)	Enable / Disable pinging when click the check box. Subsystem value is set based on the group the control is in.
Range editbox	SONAR_MESSAGE_PING_RANGE (SonarMessageLongType)	Set the range in millimeters when the range value in meters is updated with return key. Subsystem value is set based on the group the control is in.
Power Sliderbar	SONAR_MESSAGE_PING_GAIN (SonarMessageLongType)	Set the output power level. Value = selected value *10 * 1000 / 100 when value is selected. Subsystem value is set based on the group the control is in. The output power level message is channel specific so both port and starboard messages are sent for Side Scan systems, e.g. two gain message for both channels 0 and channe1 are sent.
Trigger combobox	SONAR_MESSAGE_PING_TRIGGER (SonarMessageLongType)	Set the trigger when value is selected. Subsystem value is set based on the group the control is in.
Coupling combobox	SONAR_MESSAGE_PING_COUPLING_PARAMETERS (SonarMessageCouplingParametersType)	Set the coupling parameters when value is selected. This application always set the Trigger Divisor to 1 and Trigger Delay to 0. Subsystem value is set based on the group the control is in.
Timer	SONAR_MESSAGE_SYSTEM_STATUS (SonarMessageStatusType)	Get the Sonar status. The message is sent every two seconds.
Sonar Status editbox	SONAR_MESSAGE_SYSTEM_STATUS (SonarMessageStatusType)	Display information in received Sonar Status message.

	SONAR_MESSAGE_PING_AUTOSELECTMODE (SonarMessageLongType)	When socket connection is up, three sonar pulse auto select mode messages are sent. One for each subsystem. The auto select mode values are 2 for High Frequency Side Scan and Low Frequency Side Scan, 0 for Sub-Bottom.

Table 1: Demo Interface Control Message Table

1.5 Source Code

Source code files in this application and their descriptions are lists in the table below.

DemoControl.h	Main header file for the DemoControl application
DemoControlDlg.h	Header file for CDemoControlDlg dialog class
DemoSocket.h	Header file for DemoSocket class, socket connection statistic class and other socket parameter structure and class.
Resource.h	Microsoft generated include file.
ReturnEdit.h	Header file for customized editbox control ReturnEdit to handle return key
SonarMessage.h	Header file for SonarMessage structure and SonarMessageHandler class. The sonar message building and parsing interface used in this application are included here.
SonarMessages.h	EdgeTech system header file for sonar messages. This is a common header file for sonar messages in EdgeTech software.
stdafx.h	Microsoft generated include file for standard system include file.
StorageMessages.h	EdgeTech system header file for sonar storage messages. This is a common header file for sonar messages in EdgeTech software.
Subsystem.h	EdgeTech system header file for specification of subsystem for sonar messages. This is a common header file for sonar messages in EdgeTech software.
Timestamp.h	EdgeTech system header file for timestamp utility. This is a common header file for sonar messages in EdgeTech software.
DemoControl.cpp	Source file to define the class behaviors and main entry for the application.
DemoControlDlg.cpp	Source file of CDemoControlDlg dialog class. Gui control behavior and timer are handled here. The socket interface of DemoSocket and SonarMessageHandler interface are called here.
DemoSocket.cpp	Source file of DemoSocket class.
ReturnEdit.cpp	Source file of ReturnEdit class.

SonarMessage.cpp	Source file of SonarMessageHandler class to build and parse SonarMessage.
stdafx.cpp	Microsoft generate source file that includes just the standard includes

Table 2: Source File Table

2 Socket Introduction

Sockets are a mechanism for exchanging data between processes. These processes can either be on the same machine, or on different machines connected via a network. Once a socket connection is established, data can be sent in both directions until one of the endpoints closes the connection.

EdgeTech Side Scans and Sub-Bottoms communicate with control software through client-server socket connections. Side Scan/Sub-Bottom side is the server, and the control software is the client. The client needs to know the address of the server (Side Scan/Sub-Bottom) and the port that the server listens to.

To communicate with the server, the client needs to

1. Create a socket with the socket()function call.
2. Connect the socket to the address of the server using the connect()function call.
3. Send and receive data. There are a number of ways to do this, recv() and send() function calls are used in this application.

Create a Socket

When a socket is created, the program needs to specify the address domain and the socket type. Control software uses Internet domain (AF_INET), stream socket (SOCK_STREAM), and TCP-IP protocol (IPPROTO_TCP). For example, call

m_demoSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

to create the socket. Here, m_demoSocket is the returning socket id (socket file descriptor) from socket() call.

Connect the socket to the server

The connect() function is called by the client to establish a connection to the server. It takes three arguments, the socket file descriptor, the address of the host to which it wants to connect (including the port number), and the size of this address. Example code is like following:

```
  //----------------------
  // The sockaddr_in structure specifies the address family,
  // IP address, and port of the server to be connected to.
  sockaddr_in clientService;
  clientService.sin_family = AF_INET;
  clientService.sin_addr.s_addr = inet_addr( m_socketParam.tcpAddress);
  clientService.sin_port = htons( m_socketParam.socketNumber );
  //----------------------
  // Connect to server.
  result = connect( m_demoSocket, (SOCKADDR*) &clientService,
sizeof(clientService) );
```

Read And Write Data

recv() function is called to receive data and send() function is called to send data. For example:

```
  SonarMessageHeaderType header;
  memset((void*)&header, 0, sizeof(SonarMessageHeaderType));
  bytesRecv = recv( m_demoSocket, (char*)&header,
sizeof(SonarMessageHeaderType), 0 );
  totalBytes = sizeof(m_messageOut->header)+ m_messageOut->
header.byteCount;
  result = send(m_demoSocket, (char *)(m_messageOut), totalBytes, 0);
```

3 About Sample Code on Socket Implementaion

The Demo Interface is a simple application showing how to establish a client socket to communicate with EdgeTech Sonar system. It just provides a tutorial example of DemoSocket class on how to use sockets. A real application may need to have more functionality, better performance, and other requirements. It is the application developers responsibility to decide which mechanism and approach to take, which is beyond the scope of Demo Interface and this document.

If any code in Demo Interface is used in implementing a real application, before using the code, it is recommended that the comments in the program be read carefully.

3.1 Internal Process of DemoSocket Class

DemoSocket class has its own thread running to handle socket creation, connection, reading, writing, closing, and reconnecting (see private member function Run()). When the

thread starts, it creates socket and connects to server (see private member function InitSocket()). After successfully connects, it reads (see private member function ReadMessage()) and writes (see private member function SendMessage()) until the connection is broken or a close request (m_closeRequired) is set by master thread. Then the socket will close (see private member function Close()), and reconnect to the server and start the loop of read and write again. Every time read or write or connect or close socket, the socket connection statistic data is updated.

The message that waits to be written to the socket is stored in private member m_messageOut. The socket thread loop send the message if there is new message needs to be sent out. Since there are two threads are accessing the varible, a flag m_messageWaitingOut is used to avoid the case that m_messageOut is being updated while the socket thread is sending its value to the socket. When sending out timestamp message, nagle algorithm is disabled for the socket. For other messages, nagle algorithm is enabled.

Received message is stored in private member m_receivedMsg. Every time a new message comes in, m_receivedMsg is updated.

Following sections address some issues that need to be considered when implement socket interface with EdgeTech Sonar system.

3.1.1 Control Connection and Data Connection

EdgeTech Sonar System uses two socket connections to communication control messages and data messages. The control connection is used for control and status messages of Sonar system. The data connection is used to for receiving the large sonar scan data from the Sonar system.

In the Demo Interface, only the control socket connection is demonstrated. In a real implementation, a sonar data connection also needs to be implemented.

Note that the data connection will be used to transfer very large amounts of sonar data, and the socket receiving buffer size needs set to be as large as possible, which may vary in different systems. If the buffer is not large enough, the receiving side may block the sending side and cause performance problems or dropping of data. One typical example of the buffer size is 100K.

The function call setsockopt() should be used to set the receiving buffer size. For example:

```
    int bufSize = 100000; // Buffer Size to be set.
      result = setsockopt(socketID, SOL_SOCKET, SO_RCVBUF, (char
*)&bufSize, sizeof(int));
```

3.1.2 Using Blocking mode socket vs. Non-blocking mode socket

The socket interface in Demo Interface is by default using socket in blocking mode for simplicity. In blocking mode, there may be a case that due to a loose cable or other reason during the data communication, the recv() function call will hang if the data can not be received completed, until the connection is restored.

One way to avoid this hanging is to use non-blocking mode socket to collect received data piece by piece. Please see program language manual to check the use of non-blocking mode socket.

In Demo Interface, a different method is applied. A sonar status request message is sent out to server every 2 second to get sonar status information, which also serves the purpose of keep-alive message to detect unreported socket connection failure. If a connection failure is detected, the socket will be closed and attempt to reconnect.

Programming guide on blocking /non-blocking socket can be used as reference.

3.1.3 Performance Consideration

The socket interface in Demo Interface puts read and write in one thread with the use of socket in blocking mode. It reads first, then writes, and waits 10 millisecond before going to next round of read and write. In a real application, a different mechanism may need to be used for enhanced performance. For example, use a non-blocking socket with polling method, or other methods.

3.1.4 Use of Multiple Threads

Multithread is often used with socket implementation for better performance. Whether different thread is used for socket functionality (connect, accept, read, write), or used to handle message queue for incoming message or outgoing message, the thread safety mechanism should be applied in the design and implementation.

Programming guide on multithread can be checked for the implementing multithread.

3.1.5 Message Queues for Read and Written messages

Demo Interface uses single message storage for incoming messages and outgoing messages: m_messageOut for outgoing message, and m_receivedMsg for incoming message.

In a real application, message queues can be used to handle the received message and sent message.

For example, when a message is received, append it to the tail of the received message queue. On the other hand, the message processing thread get new message from the head of the queue, remove it from the queue and process it. The private member function HandleReceiveMessage() and public member function GetReceivedMessage() can be overwritten to handle the received message queue.

When there is new message needs to be sent out, the calling thread calls WriteMessage() function to put the message to the tail of write message queue. The socket thread checks if there is new message in the queue for sending out. It will get the message from the header of the queue to send it, and remove the message from the queue. The public member WriteMessage() function and private member function SendMessage can be overwritten to handle the write message queue.

A message queue is often used by more two different thread, one is put a message in the queue and the other one is get the message and remove it from the queue. Therefore thread safety should be considered in the implementation. Usually, a mutex lock and unlock approach is used to protect the queue from being overwritten by two thread at the same time.

To check if there is a new message in the message queue, event listening/notifying, a loop continually checks for new message, or other methods can be used.

Programming guide on handling message queue can be checked for the use of message queue.

3.1.6 Enable and Disable Nagle Algorithm

One message sent is System Time message, which requires as accurate as possible. So Nagle algorithm is disabled for this message by set No_Delay flag to the socket (private member function SetNodelay()):

```
  int   paramSize;                  /* -> size for system calls   */
  int   trueValue = TRUE;           /* -> true for system calls   */
  char  displayString[128];         /* buffer for ASCII output    */
  if (m_noDelay)
    trueValue = TRUE;
  else
    trueValue = FALSE;
  paramSize  = sizeof(int);
  setsockopt(m_demoSocket, IPPROTO_TCP, TCP_NODELAY, (char *)&trueValue,
paramSize);
```

3.2 External Interface of DemoSocket Class

DemoSocket class provides a set of interface calls for the program to start the socket thread, write message to the socket, get read message, close socket, get socket statistics and status information, and get read time stamp.

3.2.1 Initialize DemoSocket Class and Run the Thread

The socket class DemoSocket has thread built in the socket. When use this class, first initialize the class, then set the IP and port to the socket by calling SetSocketParameter () function, and then start the socket thread by calling StartThread() member function, the socket thread will start and run, to connect, read and write.

Example code to initialize DemoSocket class and run the socket is like this:

```
DemoSocket m_demoSocket;
DemoSocketSonarParameters socketPars;
socketPars.setIPAddress("127.0.0.1");
socketPars.setPortNumber("1700");
m_demoSocket.SetSocketParameter(socketPars);
m_demoSocket.StartThread();
```

3.2.2 Write Message to the Socket

When the program needs to write a message to the socket, after the socket thread is called up, member function WriteMessage() can be called and pass the message wanted to write. Example:

```
SonarMessageHandler sonarMsg;
      sonarMsg.BuildSystemStatusMessage(SONAR_COMMAND_GET);
SonarMessage* msg = sonarMsg.GetMessage();
m_demoSocket.WriteMessage(msg);
```

3.2.3 Get Received Messages from the Socket

When the program needs to get a read message from socket, it can call public member function GetReceivedMessage() to get received message. For example:

```
SonarMessage* receiveMsg = m_demoSocket.GetReceivedMessage();
ProcessSonarMessage(receiveMsg);
```

3.2.4 Close Socket

Sometimes the program needs to close the socket to connect to different server. The class privides two public member functions RequestClose() and Close(). It RequestClose() is called, the socket thread will close the socket and then connect to the server that has the address stored in m_socketParam. If Close() is called, the calling thread closes the socket

immediately, and the socket thread will detect the failure and reconnect to the server. This is used in the application for keep-alive purpose.

Example:

```
m_demoSocket.Close();
m_demoSocket.RequestClose();
```

3.2.5 Get Socket Statistic Data

DemoSocket also provides the interface to get the socket statistic data. Public member function getStatistics() can be called to get a pointer of socket statistic class object which is used by DemoSocket class to collect the data. Another public member function getStatus() can be called to get the socket connection status. Example:

```
char buf[200];

if (m_demoSocket.getStatus() == DS_STATUS_CONNECTED)
{
  m_demoSocket.getStatistics()->BuildSocketStatusText(buf, sizeof(buf));
  m_connectionStatusEdit.SetWindowTextA(buf);
}
```

3.2.6 Get Read Time Stamp

Member function GetReadTime() can be called to get the latest read time in millisecond. This can be used with keep alive message to detected unreport socket connection failed. It is assumed that within a certain time period, if there is no data has been read, the connection is broken by some reason. The program will disconnet the socket and try to reconnect again. Example:

```
struct _timeb currtimeb;
time_t currtimet;

_ftime( &currtimeb );
currtimet = currtimeb.time * 1000 + currtimeb.millitm;
if (currtimet - m_demoSocket.GetReadTime() > READ_TIMEOUT)
{
  CloseSocket();
}
```