Server socket io research

I grabbed this info from the internet, this guy walks through the service easily

http://www.codeproject.com/Articles/4016/Server-Client-Sockets

Server Client Sockets



[**Emiliano**](http://www.codeproject.com/script/Membership/View.aspx?mid=258532), 25 Apr 2003 http://www.codeproject.com/App_Themes/CodeProject/Img/read32.png 349.3Khttp://www.codeproject.com/App_Themes/CodeProject/Img/download32.png 11.7Khttp://www.codeproject.com/App_Themes/CodeProject/Img/bookmark32.png 181

|  |  |
| --- | --- |
| http://dj9okeyxktdvd.cloudfront.net/script/Ratings/Images/stars-fill-lg.png  http://dj9okeyxktdvd.cloudfront.net/script/Ratings/Images/stars-empty-lg.png | 4.72 (78 votes) |
|  | |  |  |  | | --- | --- | --- | | Rate this: | [vote 1vote 2vote 3vote 4vote 5](http://www.codeproject.com/Articles/4016/Server-Client-Sockets) |  | |

Sample application Uses Server and client socket to establish exchange of data

* [**Download source files - 5 Kb**](http://www.codeproject.com/KB/IP/Server_Client_Sockets/Server_Client_Sockets_src.zip)

Introduction

The reason why I decided to write this article is because, I was learning myself how to use sockets with under windows, I could not find one place that had a tutorial which included every thing i was looking for, or if it did it was way to complicated for me to understand. In the end after doing my research I put together my resources and came up with this tutorial. I hope that it is clear and informative so that other people can benefit from it

This article be divided into 3 section:

* part 1 - Create a server socket that listen for a client to connect
* part 2 - send / receive data from client to server
* part 3 - Read unknow size of data from client

Part 1 - Creating a listening socket

To use WinSock before we start writing any code we must include the wsock32.lib and include the

Hide   Copy Code

#pragma comment(lib, "wsock32.lib")

Normally a socket that waits for a connection is a server, Once the connection has been made it can spawn off a new Thread to deal with that connection.

Before going into the acual code lets have a look some struct we will need to set up a socket:

Hide   Copy Code

WSDATA:

Any code that is compiled using a winsock accesses the ws2\_32.dll and this struct is used during the process of doing so. The program MUST call WSAstartup to initialise the DLL for later use

Hide   Copy Code

SOCKADDR\_IN:

This is used to specify how the socket is used and contains the field for the IP and port

Hide   Copy Code

SOCKET:

This is an object that stores a handle to the socket

Key functions to create the listening server socket

Hide   Copy Code

WSAStartup(WORD wVersionRequested, LPWSADATA lpWSAData )

This function *must* be the first Windows Sockets function called by an application or DLL. It allows an application or DLL to specify the version of Windows Sockets API required and to retrieve details of the specific Windows Sockets implementation. The application or DLL may only issue further Windows Sockets API functions after a successful WSAStartup invocation.

Hide   Copy Code

socket(int af, int type, int protocol)

This method creates the socket

Hide   Copy Code

bind(SOCKET s, const struct sockaddr FAR \* name, int namelen)

Associates a local address with a socket This routine is used on an unconnected datagram or stream socket, before subsequent connects or listens. When a socket is created with socket, it exists in a name space (address family), but it has no name assigned. bind establishes the local association (host address/port number) of the socket by assigning a local name to an unnamed socket. In the Internet address family, a name consists of several components. For SOCK\_DGRAM and SOCK\_STREAM, the name consists of three parts: a host address, the protocol number (set implicitly to UDP or TCP, respectively), and a port number which identifies the application. If an application does not care what address is assigned to it, it may specify an Internet address equal toINADDR\_ANY, a port equal to 0, or both. If the Internet address is equal to INADDR\_ANY, any appropriate network interface will be used; this simplifies application programming in the presence of multi- homed hosts. If the port is specified as 0, the Windows Sockets implementation will assign a unique port to the application with a value between 1024 and 5000. The application may use getsockname after bind to learn the address that has been assigned to it, but note that getsockname will not necessarily fill in the Internet address until the socket is connected, since several Internet addresses may be valid if the host is multi-homed. If no error occurs, bindreturns 0. Otherwise, it returns SOCKET\_ERROR, and a specific error code may be retrieved by callingWSAGetLastError.

Hide   Copy Code

listen(SOCKET s, int backlog )

Establishes a socket to listen to a incoming connection To accept connections, a socket is first created withsocket, a backlog for incoming connections is specified with listen, and then the connections are accepted with accept. listen applies only to sockets that support connections, i.e. those of type SOCK\_STREAM. The socket s is put into "passive'' mode where incoming connections are acknowledged and queued pending acceptance by the process. This function is typically used by servers that could have more than one connection request at a time: if a connection request arrives with the queue full, the client will receive an error with an indication of WSAECONNREFUSED. listen attempts to continue to function rationally when there are no available descriptors. It will accept connections until the queue is emptied. If descriptors become available, a later call tolisten or accept will re-fill the queue to the current or most recent "backlog'', if possible, and resume listening for incoming connections.

Hide   Copy Code

accept(SOCKET s, struct sockaddr FAR \* addr, int FAR \* addrlen)

This routine extracts the first connection on the queue of pending connections on s, creates a new socket with the same properties as s and returns a handle to the new socket. If no pending connections are present on the queue, and the socket is not marked as non- blocking, accept blocks the caller until a connection is present. If the socket is marked non-blocking and no pending connections are present on the queue, accept returns an error as described below. The accepted socket may not be used to accept more connections. The original socket remains open. The argument addr is a result parameter that is filled in with the address of the connecting entity, as known to the communications layer. The exact format of the addr parameter is determined by the address family in which the communication is occurring. The addrlen is a value-result parameter; it should initially contain the amount of space pointed to by addr; on return it will contain the actual length (in bytes) of the address returned. This call is used with connection-based socket types such as SOCK\_STREAM. If addr and/oraddrlen are equal to NULL, then no information about the remote address of the accepted socket is returned.

Hide   Copy Code

closesocket(SOCKET s)

closes a socket

Hide   Copy Code

WSACleanup()

Ends the use of the Windows Sockets DLL.

Example program

Hide   Shrink http://www.codeproject.com/images/arrow-up-16.png   Copy Code

#include <stdio.h>

#include <winsock.h>

#include <windows.h>

#define SERVER\_SOCKET\_ERROR 1

#define SOCKET\_OK 0

#pragma comment(lib, "wsock32.lib")

void socketError(char\*);

int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInstance,

LPSTR lpCmdLine, int nShow)

{

WORD sockVersion;

WSADATA wsaData;

int rVal;

sockVersion = MAKEWORD(1,1);

*//start dll*

WSAStartup(sockVersion, &wsaData);

*//create socket*

SOCKET s = socket(PF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

if(s == INVALID\_SOCKET)

{

socketError("Failed socket()");

WSACleanup();

return SERVER\_SOCKET\_ERROR;

}

*//fill in sockaddr\_in struct*

SOCKADDR\_IN sin;

sin.sin\_family = PF\_INET;

sin.sin\_port = htons(8888);

sin.sin\_addr.s\_addr = INADDR\_ANY;

*//bind the socket*

rVal = bind(s, (LPSOCKADDR)&sin, sizeof(sin));

if(rVal == SOCKET\_ERROR)

{

socketError("Failed bind()");

WSACleanup();

return SERVER\_SOCKET\_ERROR;

}

*//get socket to listen*

rVal = listen(s, 2);

if(rVal == SOCKET\_ERROR)

{

socketError("Failed listen()");

WSACleanup();

return SERVER\_SOCKET\_ERROR;

}

*//wait for a client*

SOCKET client;

client = accept(s, NULL, NULL);

if(client == INVALID\_SOCKET)

{

socketError("Failed accept()");

WSACleanup();

return SERVER\_SOCKET\_ERROR;

}

*//close process*

closesocket(client);

closesocket(s);

WSACleanup();

return SOCKET\_OK;

};

void socketError(char\* str)

{

MessageBox(NULL, str, "SOCKET ERROR", MB\_OK);

};

Making client connection with server

In order to create a socket that connects to an other socket uses most of the functions from the previous code with the exception of a struct called HOSTENT

Hide   Copy Code

HOSTENT:

This struct is used to tell the socket to which computer and port to connect to. These struct can appear asLPHOSTENT, but it actually means that they are pointer to HOSTENT.

Client key function

Most of the functions that have been used for the client to connect to the server are the same as the server with the exception of a few. I will just go through the different functions that have been used for the client.

Hide   Copy Code

gethostbyname(const char\* FAR name)

gethostbyname returns a pointer to a hostent structure as described under gethostbyaddr. The contents of this structure correspond to the hostname name. The pointer which is returned points to a structure which is allocated by the Windows Sockets implementation. The application must never attempt to modify this structure or to free any of its components. Furthermore, only one copy of this structure is allocated per thread, and so the application should copy any information which it needs before issuing any other Windows Sockets API calls. Agethostbyname implementation must not resolve IP address strings passed to it. Such a request should be treated exactly as if an unknown host name were passed. An application with an IP address string to resolve should use inet\_addr to convert the string to an IP address, then gethostbyaddr to obtain the hostentstructure.

example code

Hide   Shrink http://www.codeproject.com/images/arrow-up-16.png   Copy Code

#include <windows.h>

#include <winsock.h>

#pragma comment(lib, "wsock32.lib")

#define CS\_ERROR 1

#define CS\_OK 0

void sError(char\*);

int WINAPI WinMain(HINSTANCE hHinst, HINSTANCE hPrevHinst, LPSTR lpCmdLine,

int nShow)

{

WORD version;

WSADATA wsaData;

int rVal=0;

version = MAKEWORD(1,1);

WSAStartup(version,(LPWSADATA)&wsaData);

LPHOSTENT hostEntry;

*//store information about the server*

hostEntry = gethostbyname("hibbert");

if(!hostEntry)

{

sError("Failed gethostbyname()");

*//WSACleanup();*

return CS\_ERROR;

}

*//create the socket*

SOCKET theSocket = socket(PF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

if(theSocket == SOCKET\_ERROR)

{

sError("Failed socket()");

return CS\_ERROR;

}

*//Fill in the sockaddr\_in struct*

SOCKADDR\_IN serverInfo;

serverInfo.sin\_family = PF\_INET;

serverInfo.sin\_addr = \*((LPIN\_ADDR)\*hostEntry->h\_addr\_list);

serverInfo.sin\_port = htons(8888);

rVal=connect(theSocket,(LPSOCKADDR)&serverInfo, sizeof(serverInfo));

if(rVal==SOCKET\_ERROR)

{

sError("Failed connect()");

return CS\_ERROR;

}

closesocket(theSocket);

WSACleanup();

MessageBox(NULL, "Connection was made", "SOCKET", MB\_OK);

return CS\_OK;

}

void sError(char \*str)

{

MessageBox(NULL, str, "SOCKET ERROR", MB\_OK);

WSACleanup();

}

Part 2 - Send / recieve

Up to this point we have managed to connect with our client to the server. Clearly this is not going to be enough in a real-life application. In this section we are going to look into more details how to use the send/recvfunctions in order to get some communication going between the two applications.

Factually this is not going to be difficult because most of the hard work has been done setting up the server and the client app. before going into the code we are going to look into more details the two functions

Hide   Copy Code

send(SOCKET s, const char FAR \* buf, int len, int flags)

send is used on connected datagram or stream sockets and is used to write outgoing data on a socket. For datagram sockets, care must be taken not to exceed the maximum IP packet size of the underlying subnets, which is given by the iMaxUdpDg element in the WSAData structure returned by WSAStartup. If the data is too long to pass atomically through the underlying protocol the error WSAEMSGSIZE is returned, and no data is transmitted.

Hide   Copy Code

recv(SOCKET s, const char FAR \* buf, int len, int flags)

For sockets of type SOCK\_STREAM, as much information as is currently available up to the size of the buffer supplied is returned. If the socket has been configured for in- line reception of out-of-band data (socket optionSO\_OOBINLINE) and out-of-band data is unread, only out-of-band data will be returned. The application may use the ioctlsocket SIOCATMARK to determine whether any more out-of-band data remains to be read.

Code Example

The following code example demonstrates how to make use of the recv function. The recv function is used after the accept function, and the socket must be connected in order to receive the data.

Hide   Shrink http://www.codeproject.com/images/arrow-up-16.png   Copy Code

client = accept(s, NULL, NULL);

cout << "newclient found" << endl;

if(client == INVALID\_SOCKET)

{

socketError("Failed accept()");

WSACleanup();

return SERVER\_SOCKET\_ERROR;

}

char buf[4];

rVal = recv(client, buf, 4, 0);

if(rVal == SOCKET\_ERROR)

{

int val = WSAGetLastError();

if(val == WSAENOTCONN)

{

cout << "socket not connected" << endl;

}

else if(val == WSAESHUTDOWN )

{

cout << "socket has been shut down!" << endl;

}

socketError("Failed recv()");

return SERVER\_SOCKET\_ERROR;

}

cout << buf << endl;

This code example below works fine when you know exactly how much data you are about to receive. The problem comes when you do not know how much data will arrive. For now we will ignore this problem because the aim here is actually prove that data has been received. In the next section we will evolve the way we receive data.

Now we are going to look at how to implement the send function. In Actual fact it is the reverse of receiving data!

Hide   Copy Code

rVal=connect(theSocket,(LPSOCKADDR)&serverInfo, sizeof(serverInfo));

if(rVal==SOCKET\_ERROR)

{

sError("Failed connect()");

return CS\_ERROR;

}

char \*buf = "data";

rVal = send(theSocket, buf, strlen(buf), 0);

if(rVal == SOCKET\_ERROR)

{

sError("Failed send()");

return CS\_ERROR;

}

part 3 - Read unknow size of data from client

Us mentioned earlier in part 2, we are noe going to expand on the way that we receive data. The problem we had before is that if we did not know the size of data that we where expecting, then the would end up with problems.

In order to fix this here we create a new function that receive a pointer to the client socket, and then read a char at the time, placing each char into a vector until we find the '\n' character that signifies the end of the message.

This solution is clearly not a robust or industrial way the read data from one socket to an other, because but its a way to start reading unknown length strings. the function will be called after the accept method

Hide   Shrink http://www.codeproject.com/images/arrow-up-16.png   Copy Code

char \* readline(SOCKET \*client)

{

vector<char> theVector;

char buffer;

int rVal;

while(true)

{

rVal = recv(\*(client), &buffer, 1, 0);

if(rVal == SOCKET\_ERROR)

{

int errorVal = WSAGetLastError();

if(errorVal == WSAENOTCONN)

{

socketError("Socket not connected!");

}

socketError("Failed recv()");

WSACleanup();

}

if(buffer == '\n')

{

char \*data = new char[theVector.size() + 1];

memset(data, 0, theVector.size()+1);

for(int i=0; i<theVector.size(); i+=1)

{

data[i] = theVector[i];

}

cout << data << endl;

return data;

}

else

{

theVector.push\_back(buffer);

}

}

}

As we can see this is simple and rudimentary way to read a line, we can increase its functionality adding support for \b, \r, \0, and others depending on what the need is.

Conclusion

I hope this tutorial has been of some use, even if the code implementation is fairly simple it might be of help to further develop programs that need to implement the socket API.

There are many more things to consider when developing with sockets, i.e. NON-Blocking and Asynchronous Socket. I intend to write some more regarding the obove topics mentioned as they are very important for more robust and industrial programs

License

This article has no explicit license attached to it but may contain usage terms in the article text or the download files themselves. If in doubt please contact the author via the discussion board below.

A list of licenses authors might use can be found [here](http://www.codeproject.com/info/Licenses.aspx)

Here is the oracle page explaining it

# Lesson 1: Socket Communications

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | |  |  | | --- | --- | |  |  | |

**Lesson 1: Socket Communications**

[ [<<BACK](http://www.oracle.com/technetwork/java/index-139917.html)] [ [CONTENTS](http://www.oracle.com/technetwork/java/index-139917.html#contents)] [ [NEXT>>](http://www.oracle.com/technetwork/java/ui-140349.html)]

[Java Programming Language Basics, Part 1](http://www.oracle.com/technetwork/java/index-139917.html), finished with a simple network communications example using the Remote Method Invocation (RMI) application programming interface (API). The RMI example allows multiple client programs to communicate with the same server program without any explicit code to do this because the RMI API is built on sockets and threads.

This lesson presents a simple sockets-based program to introduce the concepts of sockets and multi-threaded programming. A multi-threaded program performs multiple tasks at one time such as fielding simultaneous requests from many client programs.

* [What are Sockets and Threads?](http://www.oracle.com/technetwork/java/socket-140484.html#sockets)
* [About the Examples](http://www.oracle.com/technetwork/java/socket-140484.html#about)
* [Example 1: Server-Side Program](http://www.oracle.com/technetwork/java/socket-140484.html#server)
* [Example 1: Client-Side Program](http://www.oracle.com/technetwork/java/socket-140484.html#client)
* [Example 2: Multithreaded Server Example](http://www.oracle.com/technetwork/java/socket-140484.html#multi)
* [More Information](http://www.oracle.com/technetwork/java/socket-140484.html#more)

**Note:** See [Creating a Threaded Slide Show Applet](http://www.oracle.com/technetwork/java/socket-140484.html) for another example of how multiple threads can be used in a program.

|  |
| --- |
|  |

### What are Sockets and Threads?

A socket is a software endpoint that establishes bidirectional communication between a server program and one or more client programs. The socket associates the server program with a specific hardware port on the machine where it runs so any client program anywhere in the network with a socket associated with that same port can communicate with the server program.

A server program typically provides resources to a network of client programs. Client programs send requests to the server program, and the server program responds to the request.

One way to handle requests from more than one client is to make the server program multi-threaded. A multi-threaded server creates a thread for each communication it accepts from a client. A thread is a sequence of instructions that run independently of the program and of any other threads.

Using threads, a multi-threaded server program can accept a connection from a client, start a thread for that communication, and continue listening for requests from other clients.

### About the Examples

The examples for this lesson consist of two versions of the client and server program pair adapted from the [FileIO.java](http://www.oracle.com/technetwork/java/socket-140484.html) application presented in [Part 1, Lesson 6: File Access and Permissions](http://www.oracle.com/technetwork/java/data-142896.html).

Example 1 sets up a client and server communication between one server program and one client program. The server program is not multi-threaded and cannot handle requests from more than one client.

Example 2 converts the server program to a multi-threaded version so it can handle requests from more than one client.

#### Example 1: Client-Side Behavior

The [client program](http://www.oracle.com/technetwork/java/socket-140484.html) presents a simple user interface and prompts for text input. When you click theClick Me button, the text is sent to the server program. The client program expects an echo from the server and prints the echo it receives on its standard output.



#### Example 1: Server-Side Behavior

The [server program](http://www.oracle.com/technetwork/java/socket-140484.html) presents a simple user interface, and when you click the Click Me button, the text received from the client is displayed. The server echoes the text it receives whether or not you click the Click Me button.



#### Example 1: Compile and Run

To run the example programs, start the server program first. If you do not, the client program cannot establish the socket connection. Here are the compiler and interpreter commands to compile and run the example.

|  |
| --- |
| javac SocketServer.java  javac SocketClient.java  java SocketServer  java SocketClient |

### Example 1: Server-Side Program

The [server program](http://www.oracle.com/technetwork/java/socket-140484.html) establishes a socket connection on Port 4321 in its listenSocket method. It reads data sent to it and sends that same data back to the server in its actionPerformed method.

#### listenSocket Method

The listenSocket method creates a ServerSocket object with the port number on which the server program is going to listen for client communications. The port number must be an available port, which means the number cannot be reserved or already in use. For example, Unix systems reserve ports 1 through 1023 for administrative functions leaving port numbers greater than 1024 available for use.

|  |
| --- |
| public void listenSocket(){  try{  server = new ServerSocket(4321);  } catch (IOException e) {  System.out.println("Could not listen on port 4321");  System.exit(-1);  } |

listenSocketSocketserver.acceptSocket

|  |
| --- |
| try{  client = server.accept();  } catch (IOException e) {  System.out.println("Accept failed: 4321");  System.exit(-1);  } |

listenSocketBufferedReaderclientPrintWriter

|  |
| --- |
| try{  in = new BufferedReader(new InputStreamReader(  client.getInputStream()));  out = new PrintWriter(client.getOutputStream(),  true);  } catch (IOException e) {  System.out.println("Read failed");  System.exit(-1);  }  } |

listenSocket

|  |
| --- |
| while(true){  try{  line = in.readLine();  //Send data back to client  out.println(line);  } catch (IOException e) {  System.out.println("Read failed");  System.exit(-1);  }  } |

#### actionPerformed Method

The actionPerformed method is called by the Java platform for action events such as button clicks. This actionPerformed method uses the text stored in the line object to initialize thetextArea object so the retrieved text can be displayed to the end user.

|  |
| --- |
| public void actionPerformed(ActionEvent event) {  Object source = event.getSource();  if(source == button){  textArea.setText(line);  }  } |

### Example 1: Client-Side Program

The [client program](http://www.oracle.com/technetwork/java/socket-140484.html) establishes a connection to the server program on a particular host and port number in its listenSocket method, and sends the data entered by the end user to the server program in its actionPerformed method. The actionPerformed method also receives the data back from the server and prints it to the command line.

#### listenSocket Method

The listenSocket method first creates a Socket object with the computer name ( kq6py) and port number (4321) where the server program is listening for client connection requests. Next, it creates a PrintWriter object to send data over the socket connection to the server program. It also creates a BufferedReader object to read the text sent by the server back to the client.

|  |
| --- |
| public void listenSocket(){  //Create socket connection  try{  socket = new Socket("kq6py", 4321);  out = new PrintWriter(socket.getOutputStream(),  true);  in = new BufferedReader(new InputStreamReader(  socket.getInputStream()));  } catch (UnknownHostException e) {  System.out.println("Unknown host: kq6py");  System.exit(1);  } catch (IOException e) {  System.out.println("No I/O");  System.exit(1);  }  } |

#### actionPerformed Method

The actionPerformed method is called by the Java platform for action events such as button clicks. This actionPerformed method code gets the text in the Textfield object and passes it to the PrintWriter object, which then sends it over the socket connection to the server program.

The actionPerformed method then makes the Textfield object blank so it is ready for more end user input. Lastly, it receives the text sent back to it by the server and prints the text out.

|  |
| --- |
| public void actionPerformed(ActionEvent event){  Object source = event.getSource();  if(source == button){  //Send data over socket  String text = textField.getText();  out.println(text);  textField.setText(new String(""));  out.println(text);  }  //Receive text from server  try{  String line = in.readLine();  System.out.println("Text received: " + line);  } catch (IOException e){  System.out.println("Read failed");  System.exit(1);  }  } |

### Example 2: Multithreaded Server Example

The example in its current state works between the server program and one client program only. To allow multiple client connections, the server program has to be converted to a [multithreaded server](http://www.oracle.com/technetwork/java/socket-140484.html)program.

|  |  |
| --- | --- |
| http://www.oracle.com/ocom/groups/public/@otn/documents/digitalasset/145391.gif  First Client  http://www.oracle.com/ocom/groups/public/@otn/documents/digitalasset/148446.gif  Second Client  http://www.oracle.com/ocom/groups/public/@otn/documents/digitalasset/145418.gif  Third Client | http://www.oracle.com/ocom/groups/public/@otn/documents/digitalasset/146376.gif |

In this example the listenSocket method loops on the server.accept call waiting for client connections and creates an instance of the ClientWorker class for each client connection it accepts. The textArea component that displays the text received from the client connection is passed to the ClientWorker instance with the accepted client connection.

|  |
| --- |
| public void listenSocket(){  try{  server = new ServerSocket(4444);  } catch (IOException e) {  System.out.println("Could not listen on port 4444");  System.exit(-1);  }  while(true){  ClientWorker w;  try{  //server.accept returns a client connection  w = new ClientWorker(server.accept(), textArea);  Thread t = new Thread(w);  t.start();  } catch (IOException e) {  System.out.println("Accept failed: 4444");  System.exit(-1);  }  }  } |

The important changes in this version of the server program over the non-threaded server program are the line and client variables are no longer instance variables of the server class, but are handled inside the ClientWorker class.

The ClientWorker class implements the Runnable interface, which has one method, run. Therun method executes independently in each thread. If three clients request connections, threeClientWorker instances are created, a thread is started for each ClientWorker instance, and therun method executes for each thread.

In this example, the run method creates the input buffer and output writer, loops on the input stream waiting for input from the client, sends the data it receives back to the client, and sets the text in the text area.

|  |
| --- |
| class ClientWorker implements Runnable {  private Socket client;  private JTextArea textArea;  //Constructor  ClientWorker(Socket client, JTextArea textArea) {  this.client = client;  this.textArea = textArea;  }  public void run(){  String line;  BufferedReader in = null;  PrintWriter out = null;  try{  in = new BufferedReader(new  InputStreamReader(client.getInputStream()));  out = new  PrintWriter(client.getOutputStream(), true);  } catch (IOException e) {  System.out.println("in or out failed");  System.exit(-1);  }  while(true){  try{  line = in.readLine();  //Send data back to client  out.println(line);  //Append data to text area  textArea.append(line);  }catch (IOException e) {  System.out.println("Read failed");  System.exit(-1);  }  }  }  } |

JTextArea.appendJTextArea.appendtextArea.append(line)synchronizedruntextArea.append(line)appendText(line)

public synchronized void appendText(line){

textArea.append(line);

}

synchronizedtextAreatextArea

The finalize() method is called by the Java virtual machine (JVM)\* before the program exits to give the program a chance to clean up and release resources. Multi-threaded programs should close all Files and Sockets they use before exiting so they do not face resource starvation. The call to server.close() in the finalize() method closes the Socket connection used by each thread in this program.

|  |
| --- |
| protected void finalize(){  //Objects created in run method are finalized when  //program terminates and thread exits  try{  server.close();  } catch (IOException e) {  System.out.println("Could not close socket");  System.exit(-1);  }  } |

### More Information

You can find more information on sockets in the [All About Sockets](http://docs.oracle.com/javase/tutorial/networking/sockets/index.html) section in [The Java Tutorial](http://docs.oracle.com/javase/tutorial/index.html).

Here is a really cool very expanded explanation and client script

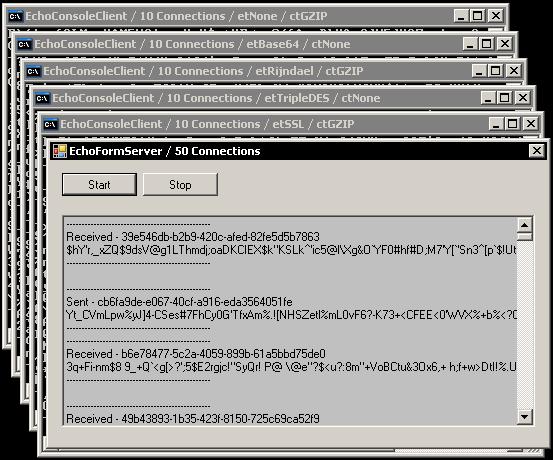
# An Asynchronous Socket Server and Client

[**Andre Azevedo**](http://www.codeproject.com/script/Membership/View.aspx?mid=912732), 29 Apr 2009 [CPOL](http://www.codeproject.com/info/cpol10.aspx)http://www.codeproject.com/App_Themes/CodeProject/Img/read32.png 1.6Mhttp://www.codeproject.com/App_Themes/CodeProject/Img/download32.png 18.7Khttp://www.codeproject.com/App_Themes/CodeProject/Img/bookmark32.png 1K

|  |  |
| --- | --- |
| http://dj9okeyxktdvd.cloudfront.net/script/Ratings/Images/stars-fill-lg.png  http://dj9okeyxktdvd.cloudfront.net/script/Ratings/Images/stars-empty-lg.png | 4.89 (257 votes) |
|  | |  |  |  | | --- | --- | --- | | Rate this: | [vote 1vote 2vote 3vote 4vote 5](http://www.codeproject.com/Articles/14155/An-Asynchronous-Socket-Server-and-Client) |  | |

An asynchronous socket server and client with encryption and compression.

* [**Download source and demo project - 195.1 KB**](http://www.codeproject.com/KB/IP/AsyncSocketServerandClien/AsyncSocketServerandClient.zip)



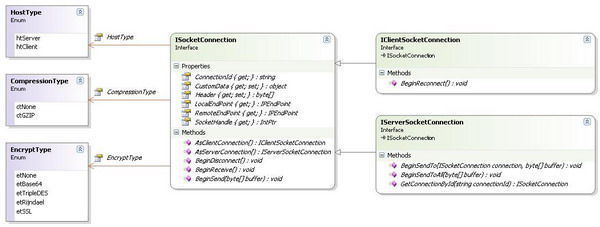
## Introduction

I've been working with sockets since 2000, using Delphi 5.0 and some third-party libraries [(Synapse)](http://synapse.ararat.cz/). My very first socket application just copied files between many clients and one server. The client app checks a folder to see if files exist, asks the server where to copy the files in the network and, after copying the files, flags the database record indicating that a file has been moved. The server listens to the client connections, and both exchange XML messages indicating the state of each file copy. Synapse is a blocking socket implementation, and I needed a thread pooling mechanism that works like an HTTP server, because I couldn't keep the connection open (one thread per connection). My solution was to use some IOCP functions to pool the client requests [(code)](http://synapse.ararat.cz/files/contrib/IocpPool.zip) and close the connection after the message exchange was terminated.

Now, using C#, I decided to write a socket server and client library that helps me to only have to think about the message exchange (the process) and let .NET do the hard job. So, I needed the following features:

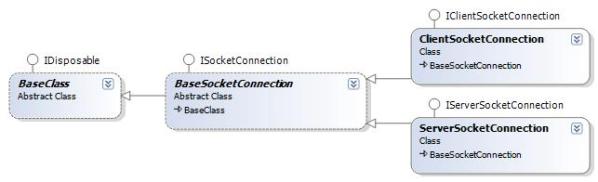
* Asynchronous processing
* Some encryption and compression capabilities
* Encapsulate the socket, and encrypt the services in the interfaces and separate them from the host implementation

## Socket Connection

[](http://www.codeproject.com/KB/IP/AsyncSocketServerandClien/socketconnection.JPG)

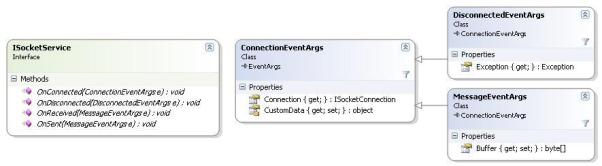
The ISocketConnection is the base interface for the socket connections, and describes all the connection properties and the methods. The ConnectionID property defines a unique connection ID using a GUID string. The CustomData property defines a custom object that can be associated with the connection. The Headerproperty is the socket service header used in each message that is encapsulated in a packet message. Only messages with a defined header will be accepted. The LocalEndPoint and RemoteEndPoint are the socket IP end points used in the connection. SocketHandle is the socket handle given by the OS.

The IClientSocketConnection and IServerSocketConnection inherit the ISocketConnection, and each one has special functions. The IClientSocketConnection can reconnect to the server using theBeginReconnect method, and the IServerSocketConnection can communicate with other connections in the server host using the BeginSendTo and BeginSendToAll methods, and can get the ConnectionId using the GetConnectionById method. Every connection knows the host, the encryption, the compression type, and can send, receive, and disconnect itself from the other part. This interface is used in the ISocketServiceinterface to allow the user to interact with socket connections.



Internally, in the library implementation, all the connection interfaces are created using the base connection implementations: BaseSocketConnection, ClientSocketConnection, and ServerSocketConnection.

## Socket Service



The ISocketService describes the connection events. These events are fired by the host, and have aConnectionEventArgs argument which has an ISocketConnection that identifies the connection. In theOnReceived and OnSent events, a MessageEventArgs is passed, which has the sent or received array of bytes. In the OnDisconnected event, a DisconnectedEventArgs is passed; the Exception property indicates if the disconnection has been caused by an exception.

Here is an example of a ISocketService implementation:

Hide   Shrink http://www.codeproject.com/images/arrow-up-16.png   Copy Code

public class SimpleEchoService : ISocketService

{

public void OnConnected(ConnectionEventArgs e)

{

*//----- Check the host!*

if (e.Connection.HostType == HostType.htServer)

{

*//----- Enqueue receive!*

e.Connection.BeginReceive();

}

else

{

*//----- Enqueue send a custom message!*

byte[] b =

GetMessage(e.Connection.SocketHandle.ToInt32());

e.Connection.BeginSend(b);

}

}

public void OnSent(MessageEventArgs e)

{

*//----- Check the host. In this case both start a receive!*

if (e.Connection.HostType == HostType.htServer)

{

*//----- Enqueue receive!*

e.Connection.BeginReceive();

}

else

{

*//----- Enqueue receive!*

e.Connection.BeginReceive();

}

}

public override void OnReceived(MessageEventArgs e)

{

*//----- Check the host!*

if (e.Connection.HostType == HostType.htServer)

{

*//----- If server, send the data buffer received!*

byte[] b = e.Buffer;

e.Connection.BeginSend(b);

}

else

{

*//----- If client, generate another*

*//----- custom message and send it!*

byte[] b = GetMessage(e.Connection.SocketHandle.ToInt32());

e.Connection.BeginSend(b);

}

}

public override void OnDisconnected(DisconnectedEventArgs e)

{

*//----- Check the host!*

if (e.Connection.HostType == HostType.htServer)

{

*//----- Nothing!*

}

else

{

*//----- Reconnect with server!*

e.Connection.AsClientConnection().BeginReconnect();

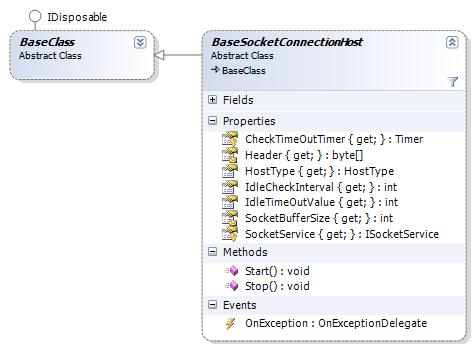
}

}

}

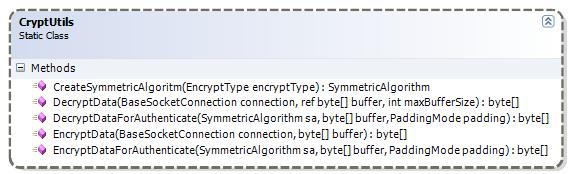
The ISocketService implementation can be done in the same host assembly, or another assembly referenced by the host. This allows the user to separate the host implementation from the socket service, helping the administration in a server or a domain.

## Connection Host

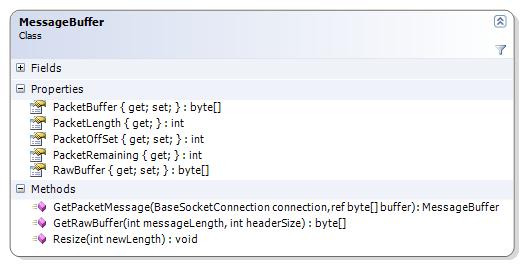


With the ISocketService created, you need to host the service and the service connections. Both the server and the client host have the same parent class, BaseSocketConnectionHost, which keeps a list of connections, encrypts and compresses the data buffers, enqueues the service requests and ensures that all data buffer has been sent or received, checks messages headers, and checks for idle connections. The CheckTimeoutTimer, periodically, at IdleCheckInterval, checks if the connections become idle, using the IdleTimeOutValue as the idle timeout. Header is the socket service header used by the host. HostType indicates if a host is a server or a client host. SocketBufferSize defines the size of the socket send and receive buffer. SocketService is the instance of ISocketService that drives the message exchange between the connections.

### Encrypt and Compress



Every time you send and receive messages, the host checks if the data must be encrypted and/or compressed, and this work is made by the CryptUtils static class. The CreateSymmetricAlgoritm creates anISymmetricAlgoritm based on the encryptType parameter. The DecryptData andDecryptDataForAuthenticate are used, respectively, to decrypt the received message and check the hash sign on the authenticate procedure. The EncryptData and EncryptDataForAuthenticate, respectively, encrypt the data to be sent and sign the authenticated message.



The encrypted data buffer is labelled with the service header and the data buffer length, becoming a packet buffer. This packet buffer is controlled by the MessageBuffer class that keeps information about the packet buffer offset, length, the remaining bytes, and the raw buffer.

### Enqueuing requests

Every time you call BeginReceive or BeginSend in ISocketService, the host checks if some request has been initiated. If a request is in process, the host enqueues the request. If not, it fires the request.

#### Send request

In the BeginSend method, the following enqueuing is used:

Hide   Copy Code

internal void BeginSend(BaseSocketConnection connection, byte[] buffer)

{

...

*//----- Check Queue!*

lock (connection.WriteQueue)

{

if (connection.WriteQueueHasItems)

{

*//----- If the connection is sending, enqueue the message!*

connection.WriteQueue.Enqueue(writeMessage);

}

else

{

*//----- If the connection is not sending, send the message!*

connection.WriteQueueHasItems = true;

...

When the message is sent, in the send callback, the host checks the queue again and initiates another send process, if needed:

Hide   Copy Code

private void BeginSendCallback(IAsyncResult ar)

{

...

*//----- Check Queue!*

lock (connection.WriteQueue)

{

if (connection.WriteQueue.Count > 0)

{

*//----- If has items, send it!*

MessageBuffer dequeueWriteMessage =

connection.WriteQueue.Dequeue();

...

}

else

{

connection.WriteQueueHasItems = false;

}

}

...

#### Receive request

The same technique applies to the receive method: all the calls to BeginReceive are enqueued if the receive method is in action. If no receive process was initiated, the host starts to receive:

Hide   Copy Code

internal void BeginReceive(BaseSocketConnection connection)

{

...

*//----- Check Queue!*

lock (connection.SyncReadCount)

{

if (connection.ReadCanEnqueue)

{

if (connection.ReadCount == 0)

{

*//----- if the connection is not receiving, start the receive!*

MessageBuffer readMessage = new MessageBuffer

(FSocketBufferSize);

...

}

*//----- Increase the read count!*

connection.ReadCount++;

}

}

...

After that, when the message is received and parsed in the receive callback, the host checks the read queue again, and initiates another receive process, if needed:

Hide   Copy Code

private void BeginReadCallback(IAsyncResult ar)

{

...

*//----- Check Queue!*

lock (connection.SyncReadCount)

{

connection.ReadCount--;

if (connection.ReadCount > 0)

{

*//----- if the read queue has items, start to receive!*

...

}

}

...

### Ensure send and receive

To ensure that all data buffer is sent, the BaseSocketConnectionHost checks the bytes sent, and compares it to the MessageBuffer class. It continues to send the remaining bytes till all the data buffer is sent:

Hide   Copy Code

private void BeginSendCallback(IAsyncResult ar)

{

...

byte[] sent = null;

int writeBytes = .EndSend(ar);

if (writeBytes < writeMessage.PacketBuffer.Length)

{

*//----- Continue to send until all bytes are sent!*

writeMessage.PacketOffSet += writeBytes;

.BeginSend(writeMessage.PacketBuffer, writeMessage.PacketOffSet,

writeMessage.PacketRemaining, SocketFlags.None ...);

}

else

{

sent = new byte[writeMessage.RawBuffer.Length];

Array.Copy(writeMessage.RawBuffer, 0, sent, 0,

writeMessage.RawBuffer.Length);

FireOnSent(connection, sent);

}

}

The same approach is used in the receive data buffers because, to read data, a MessageBuffer is used as the read buffer. When the receive callback is called, it continues to read till all the bytes in the message are read:

Hide   Shrink http://www.codeproject.com/images/arrow-up-16.png   Copy Code

private void BeginReadCallback(IAsyncResult ar)

{

...

CallbackData callbackData = (CallbackData)ar.AsyncState;

connection = callbackData.Connection;

readMessage = callbackData.Buffer;

int readBytes = 0;

...

readBytes = .EndReceive(ar);

...

if (readBytes > 0)

{

...

*//----- Has bytes!*

...

*//----- Process received data!*

readMessage.PacketOffSet += readBytes;

...

if (readSocket)

{

*//----- Read More!*

.BeginReceive(readMessage.PacketBuffer,

readMessage.PacketOffSet,

readMessage.PacketRemaining,

SocketFlags.None, ...);

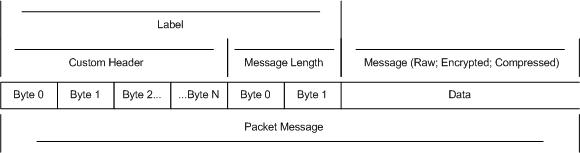
}

}

...

### Check message header

If the socket service uses some header, all the send and receive processes need to create a packet message indicating the header and the message length. This packet label is created using the following structure:



The first label's part is the socket service header. The header is an array of bytes of any length, and you need some advice here: if you choose a very small header, maybe you can have a message with the same array of bytes somewhere, and the host will lose the sequence. If you choose a very long array of bytes, the host can spend the processor's time to verify if the message header is equal to the socket service. The second part is the packet message length. This length is calculated adding the raw message data buffer length, encrypted and/or compressed, plus the header length.

#### Sending packets

As said before, every time you send messages, the host checks if the data must be encrypted and/or compressed, and, if you choose to use some header, the raw buffer is controlled by the MessageBuffer class. This class is created using the GetPacketMessage static method:

Hide   Shrink http://www.codeproject.com/images/arrow-up-16.png   Copy Code

public static MessageBuffer GetPacketMessage(

BaseSocketConnection connection, ref byte[] buffer)

{

byte[] workBuffer = null;

workBuffer = CryptUtils.EncryptData(connection, buffer);

if (connection.Header != null && connection.Header.Length >= 0)

{

*//----- Need header!*

int headerSize = connection.Header.Length + 2;

byte[] result = new byte[workBuffer.Length + headerSize];

int messageLength = result.Length;

*//----- Header!*

for (int i = 0; i < connection.Header.Length; i++)

{

result[i] = connection.Header[i];

}

*//----- Length!*

result[connection.Header.Length] =

Convert.ToByte((messageLength & 0xFF00) >> 8);

result[connection.Header.Length + 1] =

Convert.ToByte(messageLength & 0xFF);

Array.Copy(workBuffer, 0, result,

headerSize, workBuffer.Length);

return new MessageBuffer(ref buffer, ref result);

}

else

{

*//----- No header!*

return new MessageBuffer(ref buffer, ref workBuffer);

}

}

#### Receiving packets

The receive process, if you're using some socket service header, needs to check the header, and continues to read bytes till all the packet message is received. This process is executed in the read callback:

Hide   Shrink http://www.codeproject.com/images/arrow-up-16.png   Copy Code

private void BeginReadCallback(IAsyncResult ar)

{

...

byte[] received = null

byte[] rawBuffer = null;

byte[] connectionHeader = connection.Header;

readMessage.PacketOffSet += readBytes;

if ((connectionHeader != null) && (connectionHeader.Length > 0))

{

*//----- Message with header!*

int headerSize = connectionHeader.Length + 2;

bool readPacket = false;

bool readSocket = false;

do

{

connection.LastAction = DateTime.Now;

if (readMessage.PacketOffSet > headerSize)

{

*//----- Has Header!*

for (int i = 0; i < connectionHeader.Length; i++)

{

if (connectionHeader[i] != readMessage.PacketBuffer[i])

{

*//----- Bad Header!*

throw new BadHeaderException(

"Message header is different from Host header.");

}

}

*//----- Get Length!*

int messageLength =

(readMessage.PacketBuffer[connectionHeader.Length] << 8) +

readMessage.PacketBuffer[connectionHeader.Length + 1];

if (messageLength > FMessageBufferSize)

{

throw new MessageLengthException("Message " +

"length is greater than Host maximum message length.");

}

*//----- Check Length!*

if (messageLength == readMessage.PacketOffSet)

{

*//----- Equal -> Get rawBuffer!*

rawBuffer =

readMessage.GetRawBuffer(messageLength, headerSize);

readPacket = false;

readSocket = false;

}

else

{

if (messageLength < readMessage.PacketOffSet)

{

*//----- Less -> Get rawBuffer and fire event!*

rawBuffer =

readMessage.GetRawBuffer(messageLength, headerSize);

*//----- Decrypt!*

rawBuffer = CryptUtils.DecryptData(connection,

ref rawBuffer, FMessageBufferSize);

readPacket = true;

readSocket = false;

received = new byte[rawBuffer.Length];

Array.Copy(rawBuffer, 0, received, 0,

rawBuffer.Length);

FireOnReceived(connection, received, false);

}

else

{

if (messageLength > readMessage.PacketOffSet)

{

*//----- Greater -> Read Socket!*

if (messageLength > readMessage.PacketLength)

{

readMessage.Resize(messageLength);

}

readPacket = false;

readSocket = true;

}

}

}

}

else

{

if (readMessage.PacketRemaining < headerSize)

{

*//----- Adjust room for more!*

readMessage.Resize(readMessage.PacketLength + headerSize);

}

readPacket = false;

readSocket = true;

}

} while (readPacket);

if (readSocket)

{

*//----- Read More!*

...

.BeginReceive(readMessage.PacketBuffer, readMessage.PacketOffSet,

readMessage.PacketRemaining, SocketFlags.None, ...);

...

}

}

else

{

*//----- Message with no header!*

rawBuffer = readMessage.GetRawBuffer(readBytes, 0);

}

if (rawBuffer != null)

{

*//----- Decrypt!*

rawBuffer = CryptUtils.DecryptData(connection,

ref rawBuffer, FMessageBufferSize);

received = new byte[rawBuffer.Length];

Array.Copy(rawBuffer, 0, received, 0, rawBuffer.Length);

FireOnReceived(connection, received, true);

readMessage.Resize(FSocketBufferSize);

...

The read callback method first checks if the connection has some header and, if not, just gets the raw buffer and continues. If the connection has some header, the method needs to check the message header against the socket service header. Before doing that, it checks if the packet message length is greater than the connection header length, to ensure that it can parse the total message length. If not, it reads some bytes. After checking the header, the method parses the message length, and checks with the packet length. If the length is equal, it gets the raw buffer and terminates the loop. If the message length is less than that of the packet message, we have the message plus some data. So, the method gets the raw buffer and continues to read using the sameMessageBuffer class. If the length of the message is greater than that of the packet message, before reading some data, it just resizes the packet buffer to the message size, ensuring enough room for more read bytes.

### Checking idle connections

Using the BeginSend and BeginReceive methods of ISocketConnection doesn't return someIAsyncResult to know if the method was completed or not allowing disconnection after some timeout value. To prevent this, the BaseSocketConnectionHost has a System.Threading.Timer that periodically checks the LastAction property of BaseSocketConnection. If LastAction is greater than the idle timeout, the connection is closed.

## Crypto Service



The ICryptoService describes the authentication methods fired when the connection is made to the other part. The OnSymmetricAuthenticate method is fired when EncryptType.etRijndael orEncryptType.etTripleDES is used, and OnSSLXXXXAuthentication is fired when EncryptType.etSSL is used. Like ISocketService, the ICryptService can be done in the same host assembly, or another assembly referenced by the host, so you can have one ICryptoService implementation used in many ISocketServiceimplementations.

### SSL authentication

There's a new stream class called SslStream in .NET 2.0 which can authenticate SSL streams. The SslStream's constructor accepts a NetworkStream class, and this stream is created using the Socket class. So, usingSslStream, you can send and receive data buffers using socket connections.

#### Server authentication

The SslStream authentication is done in both the client and the server, but each one has different parameters. In the server side, you need to pass a certificate using the X509Certificate2 class, either finding in the certificate store using X509Store, or by creating it from a certification file (.cer). Also, you can request a client authentication and check the certificate's revocation. The following code is an example of an SSL server authentication using ICryptService:

Hide   Copy Code

public void OnSSLServerAuthenticate(out X509Certificate2 certificate,

out bool clientAuthenticate, ref bool checkRevocation)

{

*//----- Set server certificate, client*

*//----- authentication and certificate revocation!*

X509Store store = new X509Store(StoreName.My,

StoreLocation.LocalMachine);

store.Open(OpenFlags.ReadOnly);

X509Certificate2Collection certs =

store.Certificates.Find(X509FindType.FindBySubjectName,

"ALAZ Library", false);

certificate = certs[0];

clientAuthenticate = false;

checkRevocation = false;

store.Close();

}

#### Client authentication

On the client side of the SSL authentication, you need to pass the host name of the server certificate, and if this name doesn't match, the authentication fails. You can pass a client certificate collection usingX509Certificate2Collection. If the server doesn't request a client authentication, you don't need to pass the collection but, if the server requests it, you can find the certificates using X509Store. You can also request a client certificate's revocation. This is an example of SSL client authentication in ICryptoService:

Hide   Copy Code

public void OnSSLClientAuthenticate(out string serverName,

ref X509Certificate2Collection certs, ref bool checkRevocation)

{

serverName = "ALAZ Library";

*/\**

*//----- Using client certificate!*

*X509Store store = new X509Store(StoreName.My,*

*StoreLocation.LocalMachine);*

*store.Open(OpenFlags.ReadOnly);*

*certs = store.Certificates.Find(*

*X509FindType.FindBySubjectName,*

*serverName, true);*

*checkRevocation = false;*

*store.Close();*

*\*/*

}

#### Certificates

To create certificates, you can use the MakeCert.exe tool found in .NET, and there's a lot of information available about it. You can take a look at [John Howard](http://blogs.technet.com/jhoward/archive/2005/02/02/365323.aspx)'s page, this [MS post](http://www.mcse.ms/archive113-2005-5-1638636.html), and [this website](http://www.leastprivilege.com/).

### Symmetric authentication

To implement some symmetric encryption and authentication in this library, I decided to put a [post](http://www.eggheadcafe.com/aspnet_answers/NETsecurity/Mar2006/post26035603.asp) in Microsoft newsgroups. Unfortunately, for the post, but luckily for the knowledge sharing (many thanks to Joe Kaplan, Dominick Baier, and Valery Pryamikov), I decided to use [William Stacey's](http://spaces.msn.com/staceyw/PersonalSpace.aspx?_c11_BlogPart_p=1&_c11_BlogPart_handle=cns!F4A38E96E598161E!348&_c11_BlogPart_FullView=1&_c=BlogPart) implementation example **"A generic method to send secure messages using an exchanged session key"**. In this code, the symmetric key used in the session is encrypted and signed using RSA key pairs, and the client part needs to know the encrypted server's public key, meaning that this key isn't received from the server in the authentication process. Both the client and the server need to know this key through a manual process. To ensure this, the OnSymmetricAuthenticateneeds a RSACryptoServiceProvider class providing the key pair for encryption. You can fill theRSACryptoServiceProvider from an XML string, a file, a CspParameters class, or a certificate. Here is an example of symmetric authentication:

Hide   Shrink http://www.codeproject.com/images/arrow-up-16.png   Copy Code

public void OnSymmetricAuthenticate(HostType hostType,

out RSACryptoServiceProvider serverKey)

{

*/\**

*\* A RSACryptoServiceProvider is needed to encrypt and send session key.*

*\* In server side you need public and private key to decrypt session key.*

*\* In client side you need only public key to encrypt session key.*

*\**

*\* You can create a RSACryptoServiceProvider from a string*

*\* (file, registry), a CspParameters or a certificate.*

*\*/*

*//----- Using string!*

*/\**

*serverKey = new RSACryptoServiceProvider();*

*serverKey.FromXMLString("XML key string");*

*\*/*

*//----- Using CspParameters!*

CspParameters param = new CspParameters();

param.KeyContainerName = "ALAZ\_ECHO\_SERVICE";

serverKey = new RSACryptoServiceProvider(param);

*/\**

*//----- Using Certificate Store!*

*X509Store store = new X509Store(StoreName.My,*

*StoreLocation.LocalMachine);*

*store.Open(OpenFlags.ReadOnly);*

*X509Certificate2 certificate = store.Certificates.Find(*

*X509FindType.FindBySubjectName,*

*"ALAZ Library", true)[0];*

*serverKey = new RSACryptoServiceProvider();*

*if (hostType == HostType.htClient)*

*{*

*//----- In client only public key is needed!*

*serverKey = (RSACryptoServiceProvider)certificate.PublicKey.Key;*

*}*

*else*

*{*

*//----- In server, both public and private key is needed!*

*serverKey.FromXmlString(certificate.PrivateKey.ToXmlString(true));*

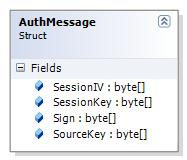
*}*

*store.Close();*

*\*/*

}

#### The authentication message



The symmetric authentication uses the AuthMessage structure to exchange session keys between the client and the server. The SessionKey and SessionIV properties are, respectively, the symmetric key and the initialization vector of the algorithm. The Sign property is the hash code generated by the client using the signRSACryptoServiceProvider class created internally, and its public key is exchanged using the SourceKeyproperty. This internal sign key pair is necessary to sign the AuthMessage, and the server can ensure that theAuthMessage is accurate. This process is done using the following code:

#### Client side

Hide   Shrink http://www.codeproject.com/images/arrow-up-16.png   Copy Code

...

*//----- Sign Message!*

private byte[] signMessage = new byte[]

{ <sign message array of bytes for authentication> };

...

protected virtual void InitializeConnection(BaseSocketConnection connection)

{

...

*//----- Symmetric!*

if (connection.EncryptType == EncryptType.etRijndael ||

connection.EncryptType == EncryptType.etTripleDES)

{

if (FHost.HostType == HostType.htClient)

{

*//----- Get RSA provider!*

RSACryptoServiceProvider serverPublicKey;

RSACryptoServiceProvider clientPrivateKey =

new RSACryptoServiceProvider();

FCryptoService.OnSymmetricAuthenticate(FHost.HostType,

out serverPublicKey);

*//----- Generates symmetric algorithm!*

SymmetricAlgorithm sa =

CryptUtils.CreateSymmetricAlgoritm(connection.EncryptType);

sa.GenerateIV();

sa.GenerateKey();

*//----- Adjust connection cryptors!*

connection.Encryptor = sa.CreateEncryptor();

connection.Decryptor = sa.CreateDecryptor();

*//----- Create authenticate structure!*

AuthMessage am = new AuthMessage();

am.SessionIV = serverPublicKey.Encrypt(sa.IV, false);

am.SessionKey = serverPublicKey.Encrypt(sa.Key, false);

am.SourceKey =

CryptUtils.EncryptDataForAuthenticate(sa,

Encoding.UTF8.GetBytes(clientPrivateKey.ToXmlString(false)),

PaddingMode.ISO10126);

*//----- Sign message with am.SourceKey,*

*//----- am.SessionKey and signMessage!*

*//----- Need to use PaddingMode.PKCS7 in sign!*

MemoryStream m = new MemoryStream();

m.Write(am.SourceKey, 0, am.SourceKey.Length);

m.Write(am.SessionKey, 0, am.SessionKey.Length);

m.Write(signMessage, 0, signMessage.Length);

am.Sign = clientPrivateKey.SignData(

CryptUtils.EncryptDataForAuthenticate(sa,

m.ToArray(), PaddingMode.PKCS7),

new SHA1CryptoServiceProvider());

*//----- Serialize authentication message!*

XmlSerializer xml = new XmlSerializer(typeof(AuthMessage));

m.SetLength(0);

xml.Serialize(m, am);

*//----- Send structure!*

MessageBuffer mb = new MessageBuffer(0);

mb.PacketBuffer =

Encoding.Default.GetBytes(Convert.ToBase64String(m.ToArray()));

connection.Socket.BeginSend(

mb.PacketBuffer, mb.PacketOffSet,

mb.PacketRemaining, SocketFlags.None,

new AsyncCallback(InitializeConnectionSendCallback),

new CallbackData(connection, mb));

m.Dispose();

am.SessionIV.Initialize();

am.SessionKey.Initialize();

serverPublicKey.Clear();

clientPrivateKey.Clear();

}

...

}

On the client side of the symmetric authentication, the OnSymmetricAuthenticate is called, getting theRSACryptoServiceProvider to encrypt the session key generated by theCryptUtils.CreateSymmetricAlgoritm method. The AuthMessage is filled with the encrypted session key, session IV, and the sign public key. To sign the message, the SourceKey, SessionKey, and signMessage are used, and the resulting hash is assigned to the Sign property.

#### Server side

Hide   Shrink http://www.codeproject.com/images/arrow-up-16.png   Copy Code

protected virtual void InitializeConnection(BaseSocketConnection connection)

{

...

if (FHost.HostType == HostType.htClient)

{

...

}

else

{

*//----- Create empty authenticate structure!*

MessageBuffer mb = new MessageBuffer(8192);

*//----- Start receive structure!*

connection.Socket.BeginReceive(mb.PacketBuffer, mb.PacketOffSet,

mb.PacketRemaining, SocketFlags.None,

new AsyncCallback(InitializeConnectionReceiveCallback), ...);

}

}

private void InitializeConnectionReceiveCallback(IAsyncResult ar)

{

...

bool readSocket = true;

int readBytes = ....EndReceive(ar);

if (readBytes > 0)

{

readMessage.PacketOffSet += readBytes;

byte[] message = null;

try

{

message = Convert.FromBase64String(

Encoding.Default.GetString(readMessage.PacketBuffer,

0, readMessage.PacketOffSet));

}

catch (FormatException)

{

*//----- Base64 transformation error!*

}

if ((message != null) &&

(Encoding.Default.GetString(message).Contains("</AuthMessage>")))

{

*//----- Get RSA provider!*

RSACryptoServiceProvider serverPrivateKey;

RSACryptoServiceProvider clientPublicKey =

new RSACryptoServiceProvider();

FCryptoService.OnSymmetricAuthenticate(FHost.HostType,

out serverPrivateKey);

*//----- Deserialize authentication message!*

MemoryStream m = new MemoryStream();

m.Write(message, 0, message.Length);

m.Position = 0;

XmlSerializer xml = new XmlSerializer(typeof(AuthMessage));

AuthMessage am = (AuthMessage)xml.Deserialize(m);

*//----- Generates symmetric algorithm!*

SymmetricAlgorithm sa =

CryptUtils.CreateSymmetricAlgoritm(connection.EncryptType);

sa.Key = serverPrivateKey.Decrypt(am.SessionKey, false);

sa.IV = serverPrivateKey.Decrypt(am.SessionIV, false);

*//----- Adjust connection cryptors!*

connection.Encryptor = sa.CreateEncryptor();

connection.Decryptor = sa.CreateDecryptor();

*//----- Verify sign!*

clientPublicKey.FromXmlString(Encoding.UTF8.GetString(

CryptUtils.DecryptDataForAuthenticate(sa,

am.SourceKey, PaddingMode.ISO10126)));

m.SetLength(0);

m.Write(am.SourceKey, 0, am.SourceKey.Length);

m.Write(am.SessionKey, 0, am.SessionKey.Length);

m.Write(signMessage, 0, signMessage.Length);

if (!clientPublicKey.VerifyData(

CryptUtils.EncryptDataForAuthenticate(sa, m.ToArray(),

PaddingMode.PKCS7),

new SHA1CryptoServiceProvider(), am.Sign))

{

throw new

SymmetricAuthenticationException("Symmetric sign error.");

}

readSocket = false;

m.Dispose();

am.SessionIV.Initialize();

am.SessionKey.Initialize();

serverPrivateKey.Clear();

clientPublicKey.Clear();

FHost.FireOnConnected(connection);

}

if (readSocket)

{

....BeginReceive(readMessage.PacketBuffer,

readMessage.PacketOffSet,

readMessage.PacketRemaining,

SocketFlags.None,

new AsyncCallback(

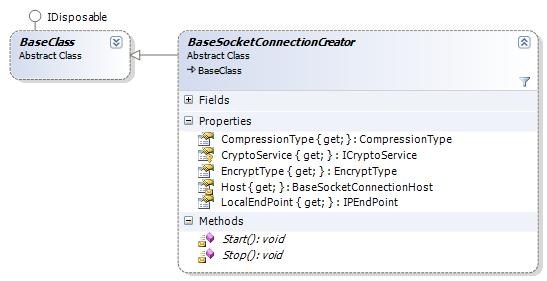
InitializeConnectionReceiveCallback), ...);

}

}

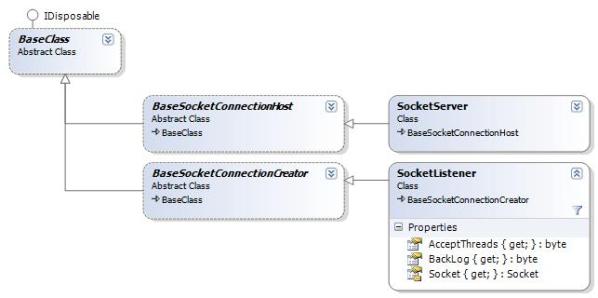
On the server side of the symmetric authentication, a MessageBuffer is used to receive the socket buffer. The read callback method continues to read till a completed AuthMessage is received. With this message, the method calls the OnSymmetricAuthenticate to get the RSACryptoServiceProvider to decrypt the session key, session IV, and the sign public key. With all the keys decrypted, the method verifies the Sign property to ensure that the AuthMessage is accurate, using the SourceKey, SessionKey, and signMessage.

## Connection Creator



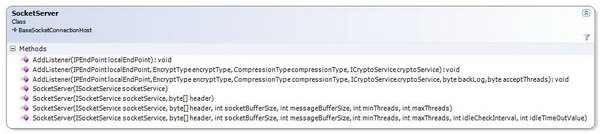
Although BaseSocketConnectionHost can manage ISocketConnection connections, it cannot create them. This job is made by BaseSocketConnectionCreator which creates and initializes ISocketConnections. TheCompressionType and EncryptType properties define, respectively, the compression and the encryption types that will be used in the connection. The CryptoService defines the ICrytoService instance used to initialize the connection, if needed. The Host property defines the host of the BaseSocketConnectionCreator; it can be a server or a client host. The LocalEndPoint defines the socket IP end point used in the connection, and it can have different behavior depending on the type of the creator.

## SocketServer and SocketListener



The SocketServer and SocketListener are the classes needed to create a socket server. SocketServer is derived from BaseSocketConnectionHost, and manages ISocketConnections. The SocketListener is derived from BaseSocketConnectionCreator, and listens for incoming connections, accepts a connection, and creates a new ISocketConnection to be used. A SocketServer can have as many SocketListeners attached as required, each one assigned to a local port to listen.

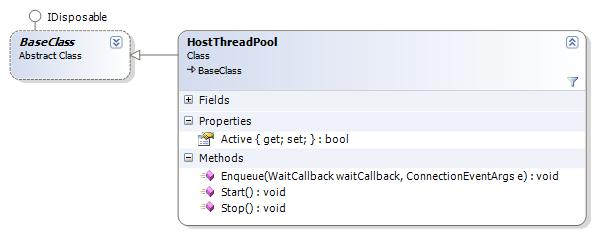
### SocketServer constructor and methods

[](http://www.codeproject.com/KB/IP/AsyncSocketServerandClien/socketserver.JPG)

In the SocketServer constructor, the socketService parameter defines the ISocketService instance used by the server. The header parameters define the array of bytes used in the message header exchange. ThesocketBufferSize adjusts the socket buffer size. The messageBufferSize defines the maximum message size of the service. The idleCheckInterval indicates the interval for idle connections checking, in milliseconds. The idleTimeoutValue defines the timeout, in milliseconds, to be compared to each connection LastActionproperty.

To add SocketListener items in SocketServer, the method AddListener must be used. ThelocalEndPoint parameter defines the local socket IP endpoint used to listen to connections. The encryptTypeand compressionType defines, respectively, the encryption and compression methods used in the new accepted connection. The cryptoService defines the ICryptoService used to authenticate the encryption method chosen. The backLog limits the listen queue of the OS socket to the defined number, andacceptThreads sets the calling number of the socket's BeginAccept to increase the accepted performance.

#### HostThreadPool



This library uses asynchronous socket communication which, in turn, uses the .NET ThreadPool. In the .NET 2.0ThreadPool, the thread number can be controlled using the SetMaxThreads and SetMinThreads methods, and I think there are a lot of improvements in this class. But, if you don't want to use the .NET class, you can use a managed thread pool called HostThreadPool, very similar to [Stephen Toub's ManagedThreadPool](http://www.gotdotnet.com/Community/UserSamples/Details.aspx?SampleGuid=bf59c98e-d708-4f8e-9795-8bae1825c3b6).HostThreadPool uses a list of managed threads that keeps increasing as more enqueueing tasks are provided. To use this class instead of the .NET ThreadPool in SocketServer, just set the minThreads and maxThreadsconstructor parameters to non-zero numbers.

Here are some examples of using SocketServer and SocketListener:

Hide   Copy Code

*//----- Simple server!*

SocketServer server = new SocketServer(new SimpleEchoService());

*//----- Simple listener!*

server.AddListener(new IPEndPoint(IPAddress.Any, 8087));

server.Start();

*//----- Server with header!*

SocketServer server = new SocketServer(new SimpleEchoService(),

new byte[] { 0xFF, 0xFE, 0xFD });

*//----- Listener with simple encryption!*

server.AddListener(new IPEndPoint(IPAddress.Any, 8087),

EncryptType.etBase64, CompressionType.ctNone, null);

server.Start();

*//----- Server with header and buffer*

*//----- sizes, no hostthreadpool and idle check setting!*

SocketServer server = new SocketServer(new SimpleEchoService(),

new byte[] { 0xFF, 0xFE, 0xFD },

2048, 8192, 0, 0, 60000, 30000);

*//----- More than one listener each one with different listen port number!*

server.AddListener(new IPEndPoint(IPAddress.Any, 8087));

server.AddListener(new IPEndPoint(IPAddress.Any, 8088),

EncryptType.etBase64, CompressionType.ctNone, null);

server.AddListener(new IPEndPoint(IPAddress.Any, 8089),

EncryptType.etRijndael, CompressionType.ctGZIP,

new SimpleEchoCryptService(), 50, 10);

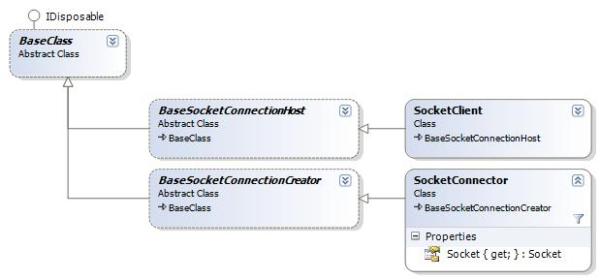
server.AddListener(new IPEndPoint(IPAddress.Any, 8090),

EncryptType.etSSL, CompressionType.ctNone,

new SimpleEchoCryptService());

server.Start();

## SocketClient and SocketConnector



The SocketClient and SocketConnector are the classes needed to create a socket client. SocketClient is derived from BaseSocketConnectionHost and, like SocketServer, manages ISocketConnections. TheSocketConnector is derived from BaseSocketConnectionCreator, and it connects with the socket server and creates a new ISocketConnection to be used. A SocketClient can have as many SocketConnectors attached as required, each one connecting to a socket server, and they can be assigned to a local address and a local port to start the connection.

### SocketClient constructor and methods

[](http://www.codeproject.com/KB/IP/AsyncSocketServerandClien/socketclient.JPG)

The SocketClient constructor has the same parameter signature as the SocketServer class. To addSocketConnector items in SocketClient, the method AddConnector must be used. The remoteEndPointparameter defines the remote socket IP endpoint used for the connection. The encryptType andcompressionType define, respectively, the encryption and compression methods used in the new connection. The cryptoService defines the ICryptoService used to authenticate the encrypted method chosen. ThereconnectAttempts and reconnectAttemptInterval define, respectively, the number of reconnect attempts when using BeginReconnect method and the time interval to reconnect. The localEndPoint defines the local socket IP endpoint used to start the connection process to the remote endpoint.

Here are some examples of using SocketClient and SocketConnector:

Hide   Shrink http://www.codeproject.com/images/arrow-up-16.png   Copy Code

*//----- Simple client!*

SocketClient client = new SocketClient(new SimpleEchoService());

*//----- Simple connector!*

client.AddConnector(new IPEndPoint(IPAddress.Parse("10.10.1.1"), 8087));

client.Start();

*//----- Client with header!*

SocketClient client = new SocketClient(new SimpleEchoService(),

new byte[] { 0xFF, 0xFE, 0xFD });

*//----- Connector with simple encryption!*

client.AddConnector(new IPEndPoint(IPAddress.Parse("10.10.1.1"), 8087),

EncryptType.etBase64, CompressionType.ctNone, null);

client.Start();

*//----- Client with header and buffer sizes,*

*//----- no hostthreadpool and idle check setting!*

SocketClient client = new SocketClient(new SimpleEchoService(),

new byte[] { 0xFF, 0xFE, 0xFD },

2048, 8192, 0, 0, 60000, 30000);

*//----- Connector with encryption and reconnect!*

client.AddConnector(new IPEndPoint(IPAddress.Parse("10.10.1.1"), 8087),

EncryptType.etSSL, CompressionType.ctGZIP,

new SimpleEchoCryptService(),

5, 30000);

client.Start();

*//----- Client with header and buffer sizes,*

*//----- using hostthreadpool and idle check setting!*

SocketClient client = new SocketClient(new SimpleEchoService(),

new byte[] { 0xFF, 0xFE, 0xFD },

4096, 8192, 5, 50, 60000, 30000);

*//----- Connector with encryption, reconnect and local endpoint!*

client.AddConnector(new IPEndPoint(IPAddress.Parse("10.10.1.1"), 8087),

EncryptType.etSSL, CompressionType.ctGZIP,

new SimpleEchoCryptService(),

5, 30000,

new IPEndPoint(IPAddress.Parse("10.10.3.1"), 2000));

client.Start();

*//----- Simple client!*

SocketClient client = new SocketClient(new SimpleEchoService());

*//----- More than one connector each one with different remote socket servers!*

client.AddConnector(new IPEndPoint(IPAddress.Parse("10.10.1.1"), 8087));

client.AddConnector(new IPEndPoint(IPAddress.Parse("10.10.1.2"), 8088),

EncryptType.etBase64, CompressionType.ctNone, null);

client.AddConnector(new IPEndPoint(IPAddress.Parse("10.10.1.3"), 8089),

EncryptType.etRijndael, CompressionType.ctGZIP,

new SimpleEchoCryptService());

client.AddConnector(new IPEndPoint(IPAddress.Parse("10.10.1.4"), 8090),

EncryptType.etSSL, CompressionType.ctNone,

new SimpleEchoCryptService(),

5, 30000,

new IPEndPoint(IPAddress.Parse("10.10.3.1"), 2000));

client.Start();

## Echo demo project

There's an Echo demo project available in the article download file, using **Console**, **Windows Forms**, and**Windows Service** hosts and clients, and all them use the same EchoSocketService and EchoCryptService. The demos are divided by their type as follows:

### Hosts

* Console
  1. EchoConsoleClient
  2. EchoConsoleServer
* Windows Forms
  1. EchoFormClient
  2. EchoFormServer
  3. Echo<code>Form (Forms template)
* Windows Service
  1. EchoWindowsServiceServer

### Services

* EchoSocketService
* EchoCryptService

## Conclusion

There's a lot going on here, and I think this library can help anyone who wants to write asynchronous sockets with encryption and compression. Any comments will be appreciated.