



Tutorial

Instant Messaging

Mauricio B. Longo
February, 2001

Copyright 2001 by ASTA Technology Group. All rights reserved.

Microsoft and Windows are trademarks of Microsoft Corporation. All other trademarks and tradenames are the property of their respective owners and are used here for identification purposes only.

This document was created using LetterArt Word Processor. The PDF (Portable Document Format) version of this document was also created with LetterArt Word Processor.

Summary

Introduction

Part I - Basic Concepts

What is Instant Messaging?

How does it work?

Architecture

Client/Server Development Simplified

What is a Client/Server Application?

How do the Client and the Server Communicate

Part II - Asta Instant Messenger

Components, Methods & Events

Communication Components

Communication Methods

Communication Events

Building the Applications

The Instant Messaging Server

Handling Connections, Disconnections and Logins

Authenticating Users

Relaying and Responding to Client Messages

Receiving a Message

Relaying a Message

The Instant Messaging Client

Basic Usage

Establishing a Connection and Sending Personal Information

Receiving the User List

Sending a Message

Requesting Personal Information

Receiving a Message From the Server

Wrapping it All Up

About the Author

I n t r o d u c t i o n

Why Messaging?

First things, first. Let's establish why we are here. This document covers the basics of using Asta Messaging which offers you a very easy way of creating powerful client/server applications.

But you thought you were already doing that! Well, after reading this paper you might think differently. We are going to explore the true power of client/server computing.

To explain how to create sophisticated applications using Asta Messaging I wanted to have an example that was at the same time simple and powerful. The first thing that came to my mind was an Instant Messaging application - like ICQ or MSN Messenger.

This kind of application fits my description exactly. It is quite simple and at the same time very powerful in how it can help bring people together from around the world.

P a r t I

Basic Concepts

What is Instant Messaging?

Chances are, if you haven't been hiding in a cave for the past five or six years, that you know what an Instant Messaging application is. The most well known is ICQ which has over eighty five million users world wide.

An instant messaging application allows you to be notified when your friends are online and then offers the ability to send messages that pop-up right in front of their eyes. For being a simple application (when you are not trying to do what the professional applications do) I chose to use it to exemplify Asta's great messaging features.

The very name of the application Instant Messaging makes it a prime candidate for a demonstration.

How Does it Work?

Let's start with a look at how a basic Instant Messaging application should work. What are the features we must have in order to get our application to work adequately?

First, we must let know the users know who else is online. For this example I made the decision not worry about a list of contacts for each user, since this would not substantially alter the way the application is built.

In order to know who is online we must have some sort of "login" procedure. This will allow us to get the names of each user.

Knowing that someone is online, we must be able to send that person a pop-up message. We must, also, be able to get some information about the person, so that we don't end up sending messages to the wrong person.

Architecture

There are essentially two main groups applications that communicate through networks or the Internet - client server and the and peer-to-peer. Instant Messaging applications usually employ a mix of both architectures for better performance and lower server resource consumption.

Our little application - the Asta Instant Messenger - will be built in a client/server architecture to simplify our task, but I will give you some tips as to what could be added or changed in order for it to use the peer-to-peer approach, also.

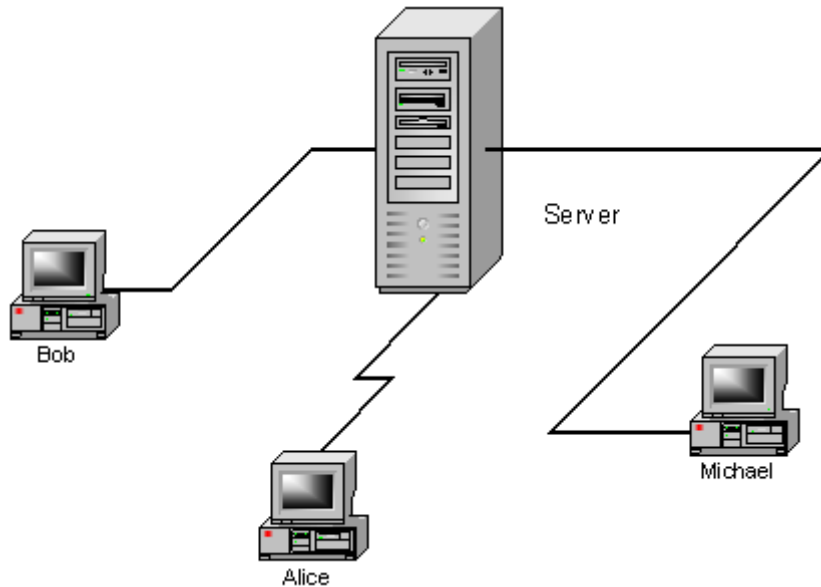


Diagram 1 - Several clients connected to a central server

Client/Server Development Simplified

Most of the time, client/server computing is mistaken for some kind of alien technology belonging to realm of the big software houses. If you are one of those who think this way, I have news for you - you are wrong. Client/Server development is not so difficult as you may have thought and Asta makes it much easier that you could have possibly imagined.

What is a client/Server Application?

What we, generally, call a client/server application is actually a pair of applications divided as the name implies into an application that makes requests for information - the client and an application that services these requests - the server.

These applications, basically, exchange a pre-determined set of messages that relays the client's request and the server's answers to these requests. Mostly these are fairly simple conversations where the client sends a command to the server, which in turn returns data. This is the classic case of the database server (such as Oracle, DB2 and MS SQL Server).

How do the Client and the Server Communicate?

The client and server communicate by exchanging messages. These messages must follow a pre-determined pattern and be part of a group previously defined. In our Instant Messaging example we will be using a special set of methods and events that the Asta components make available for sending and receiving user defined messages.

One interesting thing to know is that the pre-defined set of messages and the pattern in which they are exchanged make up what is commonly known as a protocol. In our case it will be an application protocol, which is a protocol defined for data exchange between applications (as the name suggests).

P a r t II

The IM Applications

Components, Methods & Events

Before going into the construction of the actual IM client and server I believe we should take look at what components, methods and events we will need to build such applications. We will start with which components and classes are used in each application and then proceed to see how they are used.

Communication Components

It should be quite obvious that to implement an application set that is devoted to communication, we will need components to specifically handle communication operations.

In this case everything is easily achieved by using a component of the *TAstaClientSocket* class in the client and a *TAstaServerSocket* in the server application. This might sound obvious to you but, actually, that is not always the case, since there are many situations where a client might be required to use the Server socket component and vice-versa.



These icons identify the *TAstaClientSocket* and *TAstaServerSocket* components, respectively. They can be found on your Asta tab of the component palette.

We'll need to make use of an Asta support class called *TAstaParamList*. This is a very powerful tool for exchanging parameters between applications, apart being the sort of tool that might just come in handy in several kinds of jobs around the house. It's a bit like a screwdriver. :-)

The *TAstaParamList* allows you to create a list of named parameters with data type and value. This list can then be sent from one application to another in an easy and convenient manner. Though this is not, technically a communication component it is an important piece of our communications mechanism and I thought it deserved to be mentioned.

Communication Methods

What I'm considering communication methods, are those that are directly related to the communication process. In this application I have used very few methods related to communication, as strange as that might seem when we are talking about and instant messaging application.

Almost all logic in these applications is built around the versions of the *SendCodedParamList* method of the *TAstaServerSocket* and *TAstaClientSocket* components. These methods function in exactly the same way. The only difference between the client and server versions is that the server version takes an extra parameter indicating to which client the server should send the message. The client component does not need this information since a *TAstaClientSocket* component can only be connected to **one** server at a time.

Tip | Even though a *TAstaClientSocket* component can only be connected to one server at a time, nothing keeps you from having more than one such component in your application. This allows you create client applications that can be connected to multiple servers simultaneously.

There are several other methods you might use to implement Asta messaging applications, all of these function in a very similar way to *SendCodedParamList*. This method allows you send *TAstaParamList* objects from one application to another. The remaining methods allow you to send other types of data such as Strings (*SendCodedMessage*) and Streams (*SendCodedStream*).

I could have built this very same application based on either of these methods and have reached the same result. I chose to use the *TAstaParamList* based methods because they offer more ease of use when handling the transfer of multiple parameters at a time.

Communication Events

Most of the code for both the server and the client IM applications is written as response to events. Why? Because on each side of the wire an event is triggered when a message is received. When this event is triggered it is up to your code to identify the message, its contents and take appropriate action in response.

The main event in both the client and server IM applications is the *OnCodedParamList* event. In the client application the following communication-related events are also used: *OnDisconnect* and *OnLoginAttempt*.

On the server application the *OnClientLogin* and *OnUserListChange* events are also used.

Building the Applications

We are now going to take a look at how both the Instant Messaging Client and Server are built. We will start for the server, since its existence is a pre-condition for the client to be able to do anything.

The Instant Messaging Server

The Instant Messaging Server application is actually, quite simple, using few controls and having just the basic capabilities needed for such a demonstration. I did not attempt to make the server into a service application, since that would be beyond the scope of this document, so it can be used in any flavor of Windows, not being limited to Windows NT.

Figure 1 shows the Main Window of the server, running with two connected users. As you can see it is a very simple interface, made up of very few components. By monitoring the this window you can know exactly who is online at the moment. Actually, if all you wanted was to see the list, the same effect could be achieved by simply using the client application. However, in the initial states of creating a server application it is sometimes useful to have it display status information so that you can monitor the process of communication.

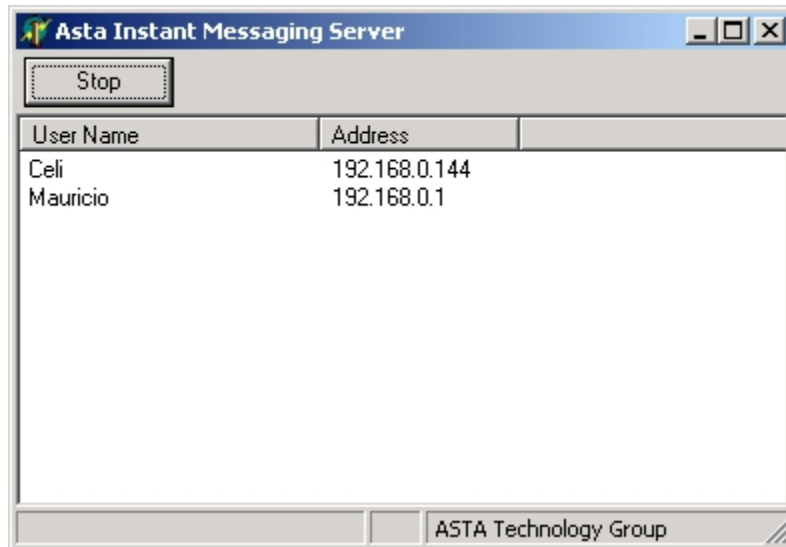


Figure 1 - The Instant Messenger Server with two connected users.

In creating this server, I tried to keep it as simple as possible, adding only the basic functionality needed for our example. If you are interested, there is ample space for increasing the features of the server.

All work on the server is centered around a *TAstaServerSocket* component. In figure 2, you can see the same form that is shown in Figure 1, but this time it is shown in design state in the Delphi IDE.

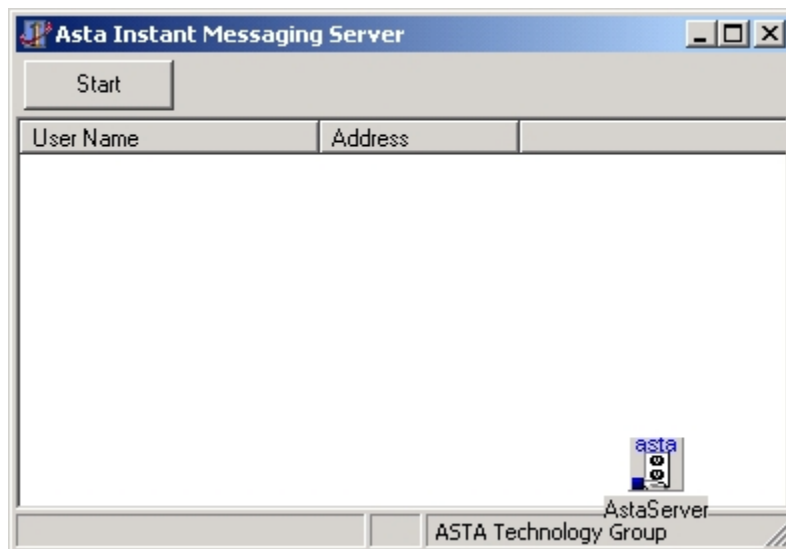


Figure 2 - The Main Form of the IM Server at design-time.

On the form is only one non-visual component, its name is *AstaServer* and it is a *TAstaServerSocket* component. Almost all work in the server is done in the *OnCodedParamList* and *OnUserListChange* events of the *AstaServer* component. There is additional code in other events, but it is auxiliary code.

In order to understand how these applications work we will now examine each routine of the server and see what it all adds up to.

On startup, the server application it is not yet operating as a server. In order for this to happen you must click the Start button. This button has the follow code associated

with it.

```
procedure TAstaIMServerMainForm.btnToggleClick(Sender: TObject);  
begin  
    AstaServer.Active := not AstaServer.Active;  
    if AstaServer.Active then  
        btnToggle.Caption := 'Stop'  
    else  
        btnToggle.Caption := 'Start'  
end;
```

This simple code either starts or stops the server, depending on its current status.

Handling Connections, Disconnections and Logins

Once the server is started, it is ready to start receiving client connections. As a client attempts to connect to the server the *OnUserListChange* event occurs for the first time. Its code is shown below.

```
procedure TAstaIMServerMainForm.AstaServerUserListchange(  
    Sender: TObject; Socket: TCustomWinSocket;  
    Astate: TUserRecordState);  
var  
    ParamList: TAstaParamList;  
    I: Integer;  
    UserItem: TListItem;  
  
    function FullUserListParamList: TAstaParamList;  
    var  
        Index: integer;  
    begin  
        Result := TAstaParamList.Create;  
        for Index := 0 to AstaServer.UserList.Count-1 do  
            result.FastAdd(AstaServer.UserList[Index].UserName,  
                AstaServer.RemoteAddressAndPort(  
                    AstaServer.UserList[Index].FClientSocket));  
    end;  
  
    function UserListLessDisconnectedSocket: TAstaParamList;  
    var  
        Index: integer;  
    begin  
        Result := TAstaParamList.Create;  
        for Index := 0 to AstaServer.UserList.Count-1 do  
            if Socket <> AstaServer.UserList[Index].FClientSocket then  
                Result.FastAdd(AstaServer.UserList[Index].UserName,  
                    AstaServer.RemoteAddressAndPort(  
                        AstaServer.UserList[Index].FClientSocket));  
    end;  
  
    function FindSocket: TListItem;  
    var  
        Index: Integer;  
    begin  
        Result := nil;  
        for Index :=0 to lvUsers.Items.count-1 do  
            if LvUsers.items[Index].Data=Socket then  
                begin  
                    Result:=LvUsers.Items[Index];  
                    exit;  
                end;  
    end;  
end;
```

```

begin
  case Astate of
    tuConnect:
      begin
        UserItem := lvUsers.Items.Add;
        UserItem.Caption := 'Connected No UserName Yet';
        UserItem.SubItems.Add(Socket.RemoteAddress);
        UserItem.Data:=Socket;
      end;

    tuLogin:
      begin
        userItem:=FindSocket;
        UserItem.Caption:=AstaServer.UserList.GetUserName(Socket);
        ParamList:=FullUserListParamList;
        try
          for I := 0 to AstaServer.UserList.Count-1 do
            AstaServer.SendCodedParamList(
              AstaServer.UserList[I].FClientsocket, 300, ParamList);
          finally
            ParamList.free;
          end;
        end;
      end;

    tuDisConnect:
      begin
        userItem:=FindSocket;
        UserItem.Delete;
        ParamList:=UserListLessDisconnectedSocket;
        try
          for I := 0 to AstaServer.UserList.Count-1 do
            if Socket<> AstaServer.Userlist[I].FClientSocket then
              AstaServer.SendCodedParamList(
                AstaServer.UserList[I].FClientsocket, 300, ParamList);
            if assigned(Socket.Data) then
              TAstaParamList(Socket.Data).Free;
            finally
              ParamList.free;
            end;
          end;
        end;
      end;
  end;
end;

```

This event is called in three different moments: when the user first connects to the server, when the user is authenticated and finally when the user disconnects from the server. These three situations are covered by the case statement in the procedure's main body.

When the user first connects to the server it a placeholder item in the server's main form listview is created with a caption of "Connected no user name yet". This happens when the event is called and the *Astate* parameter has a value of *tuConnect*.

The next time this event is called for this same user the *Astate* parameter will have a value of *tuLogin* which means that the user has been validated and accepted. When this happen the sub-function *FindSocket* will be used to retrieve the listitem object that corresponds to that user and it will be updated to reflect the user name of the user.

After the user's entry in the listview is updated a list of all currently logged on users is prepared, though a call to *FullUserListParamList* and sent back through a call to the

SendCodedParamList of the *AstaServer* object. This will be used by the client to display the list of users currently online.

When the *OnUserListChange* event is called and the *Astate* parameter has a value of *tuDisconnect* the entry for that particular user is deleted from the server's listview and a new list of online users (minus the user who is disconnecting) is sent out to all remaining connected clients. This is done through the use of a loop calling *SendCodedParamList* to send the list to all individual clients.

Note | This event made the first use of the *SendCodedParamList* method. This method will be used repeatedly in both the server and client applications and will be discussed in detail ahead.

Authenticating Users

User authentication is handled through an event called *OnClientLogin*. This event is called whenever a user needs to be authenticated and has a *var* (reference) parameter called *Verified* that should be set to true or false depending on the result of the authentication process.

In order to properly authenticate a user the event receives three parameters: *UserName*, *Password* and *AppName* (which is the name of the application the user is using to connect to the server). Based on this info you either grant or deny access to the server by setting the *Verified* parameter.

In our example server, I decided not to enforce any rules for login control. Any combination of user name and password is accepted. The *OnClientLogin* event for this application actually does nothing more than always return true in the reference parameter *Verified*, as you can see in the code below.

```
procedure TAstaIMServerMainForm.AstaServerClientLogin(Sender: TObject;  
    UserName, Password, AppName: String; var Verified: Boolean);  
begin  
    Verified := true;  
end;
```

Relaying and Responding to Client Messages

The main functions of the our IM Server are: providing a list of available users and relaying messages from one client application to another. We are now going to examine how the messages get relayed through the server.

Receiving a Message

The first step in relaying a message is, of course, receiving the message. The IM server receives all messages from the client applications through the *OnCodedParamList* event.

When the *OnCodedParamList* event is called it receives several parameters. Amongst these are: *ClientSocket*, *MsgID* and *Params*. The *ClientSocket* object represents the connection through which the message was received and can be used to send a return message back to the client who originally sent the message. *MsgID* is a numeric identifier for the message. This parameter is used in a manner much like Windows' message IDs. The *Params* object is an instance of the *TAstaParamList* class and should contain a set of parameters. The exact number and the names of

the parameters should be determined based on the value of *MsgID*. Each kind of message will have its corresponding set of parameters. You can see, below, the code for the *AstaServer* object's *OnCodedParamList*

```
procedure TAstaIMServerMainForm.AstaServerCodedParamList(  
    Sender: TObject;  
    ClientSocket: TCustomWinSocket;  
    MsgID: Integer;  
    Params: TAstaParamList);  
  
var  
    User: UserRecord;  
begin  
    Case MsgID of  
        100: begin  
            AstaServer.SendCodedParamList(  
                AstaServer.UserList.SocketFromUserName(  
                    PParams.ParamByName('To').AsString),  
                200, Params);  
  
            end;  
        500: begin  
            User := AstaServer.UserList.GetUserRecordSocket(  
                AstaServer.UserList.SocketFromUserName(  
                    PParams.ParamByName('UserName').AsString));  
            AstaServer.SendCodedParamList(ClientSocket, 600,  
                User.FParamList);  
  
            end;  
        end;  
    end;  
end;
```

As you can see by examining the code just shown for the *OnCodedParamList* event, the server only knows about the existence of two kinds of messages. These messages are identified by the numbers 100 and 500.

Tip | When you are building your own server it is a good idea to return a message to the client indicating an error condition, in case you receive an unexpected message ID.

In our IM server the message identified by the number 100 indicates that a client wishes to send a message to a specific user, also connected to the server.

Note | The use of the Asta message format provides you with a consistent interface for the exchange of data between applications in all supported platforms.

Relaying a Message

When the message identified by the ID 100 is received, the server uses its "TO" parameter, contained in the *Params* object, (that is a *TAstaParamList* instance) to identify the intended recipient for the message and then relays the message by calling the *SendCodedParamList* method of the *AstaServer* object. To call this method of a *TAstaServerSocket* object you need to pass it the *ClientSocket* parameter. In

order to pass the correct argument to this method you must determine the socket which corresponds to the desired user. This is accomplished by the following line of code:

```
User := AstaServer.UserList.GetUserRecordSocket(  
    AstaServer.UserList.SocketFromUserName(  
        PArms.ParamByName('UserName').AsString));
```

The *TAstaServerSocket* component has a property called *UserList* that holds a list of records with information about the users connected to the server. This object has method called *SocketFromUserName* which returns the *Socket* object that is needed for the call to *SendCodedParamList*. All we have to do is supply this method with the desired user name and it will return us the corresponding socket.

Once we have the appropriate socket object we can use it to relay the message to the desired recipient through a call to the *SendCodedParamList* method of the *AstaServer* object as shown in the code snippet that follows.

```
AstaServer.SendCodedParamList(  
AstaServer.UserList.SocketFromUserName(  
PArms.ParamByName('To').AsString), 200, Params);
```

Actually, the code above does all relaying in a single statement. This just goes to show you the power of the Asta components.

Observe that the server is sending a message, to the client application, identified by the ID 200. This, as we shall see further ahead, will be treated by the client in much the same way we just saw the server treat messages.

The Instant Messaging Client

The Instant Messaging client application is almost as simple as the server. Actually, it implements just the basic functions necessary for our example. Due its nature it must contain more code and more than one form but, wherever possible, I tried to keep from adding unnecessary complexity.

I tried to give the client application a familiar look n' feel by designing it as a thin and tall form, much in the style of the generally available Instant Messaging applications. Figure 3 shows the client application running and connected to a server that has two users connected, one of which is the user of the application captured in the picture.

Basic Usage

The main form, as shown in Figure 3, has a list of online users and three buttons. These three buttons have the following functionality, from left to right: Connect/Disconnect, retrieve user info and configure application parameters.

On startup our client does not connect immediately to the server. This must be done manually by clicking the "Connect" button. When clicked the button's caption will immediately change to show "Disconnect" and the list of currently online users will be displayed. By clicking the same button, now showing the caption "Disconnect", you will disconnect from the server and the list of users will be cleared.

In order to configure your client IM application you should click on the "Options" button. This will bring up a dialog where you can input your configuration parameters. These parameters include connection information such as your user name and password and the host and port to which you wish to connect. In this dialog you can also enter some personal information as your full name, the department where you

work, your phone extension and email address.

The options dialog can be seen in Figure 4.

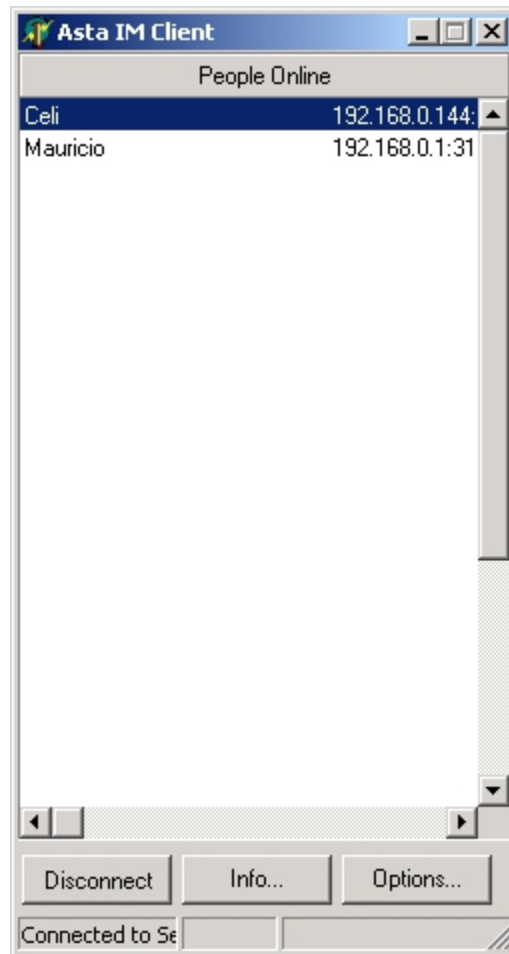


Figure 3 - The Instant Messaging Client showing 2 online users.

Through the use of the Options dialog you can configure your client application to connect to any Instant Messaging Server that you may have setup on your network, or on the Internet.

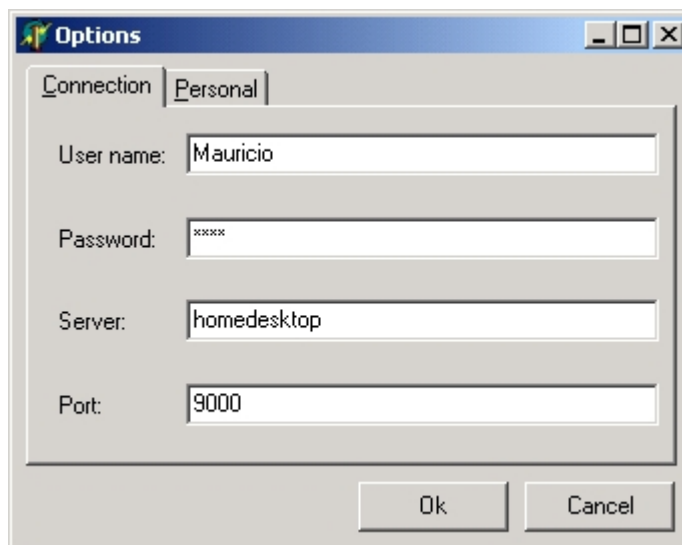


Figure 4 - The Options dialog of the Instant Messaging Client.

Note | It is important to keep in mind that these client and applications were designed to work together and with no other applications. You should not expect this client application to connect to any other server application, but the one I have created and explained in this document.

By selecting a user in the client's list of online users and clicking the "Info..." button you will bring up a small dialog with the personal information that was entered by that user in his Options dialog.

Establishing a Connection and Sending Personal Information

Establishing a connection to the server is quite simple, as we will see. In our example the code associated with the click event of the "Connect" button is responsible for establishing the connection. Most of the code in this example is related to setting up the user's personal information to be sent to the server, The event's code is shown bellow.

```
procedure TfrmIMClientMain.btnConnectClick(Sender: TObject);  
begin  
  if not AstaClient.Active then  
    begin  
      AstaClient.ClientSocketParams.Clear;  
      AstaClient.ClientSocketParams.FastAdd('FullName',  
        ConfigFile.ReadString('Personal', 'FullName', ''));  
      AstaClient.ClientSocketParams.FastAdd('Department',  
        ConfigFile.ReadString('Personal', 'Department', ''));  
      AstaClient.ClientSocketParams.FastAdd('Extention',  
        ConfigFile.ReadString('Personal', 'Extention', ''));  
      AstaClient.ClientSocketParams.FastAdd('email',  
        ConfigFile.ReadString('Personal', 'email', ''));  
      AstaClient.UserName := ConfigFile.ReadString('UserData',  
        'UserName', '');  
      AstaClient.Password := ConfigFile.ReadString('UserData',  
        'Password', '');  
      AstaClient.Host := ConfigFile.ReadString('ServerData',  
        'Host', '');  
      AstaClient.Port := StrToInt(ConfigFile.ReadString('ServerData',  
        'Port', ''));  
  
      AstaClient.Active := true;  
    end  
  else  
    begin  
      btnConnect.Caption := 'Connect';  
      AstaClient.Active := false;  
    end;  
  
  //at this point the Socket Will NOT be active!!!  
  //event driven sockets! use the events!  
end;
```

You should observe that most of the code in this event is related to reading user

configuration options from an INI file and assigning this information to the appropriate properties of the *AstaClient* component. *AstaClient* is a component of the class *TAstaClientSocket*.

Tip | Notice that in the example code for the connect button, several parameters are inserted into a *TAstaParamList* that is a property of the *AstaClient* object. When a component of the *TAstaClientSocket* connects to a server it sends this list as additional information. This is how we send our personal information to the server.

Once again, the procedure seems too simple. Actually, it's because a lot is done for you in the background by the Asta components.

Receiving the User List

Once you are connected the server will send you the list of currently online users. This list will arrive in the form of a *TAstaParamList* component in the *OnCodedParamList* event.

The code that deals with this message in the *OnCodedParamList* event is shown below.

```
300: begin
    UserDataSet.DisableControls;
    try
        if not UserDataSet.Active then
            UserDataSet.Open;
            UserDataSet.Empty;
        for Index:=0 to Params.Count-1 do
            UserDataSet.AppendRecord(
                [Params[Index].Name,Params[Index].AsString]);
        finally
            UserDataSet.EnableControls;
        end;
    end;
```

This message will arrive with an Id off 300. The message ID is used in a case statement to determine what course of action the application should take.

This code uses a *TAstaDataset* component as the repository of the information and since the *DBGrid* component references a *DataSource* which in turn references the *AstaDataset*, the list is refreshed.

Sending a Message

Once you have established a connection to the server, you can choose one of the users from the list to send a message. You do this by double-clicking on the user's name in the list. A pop-up dialog will appear so that you can type your message. This dialog is shown in Figure 5.

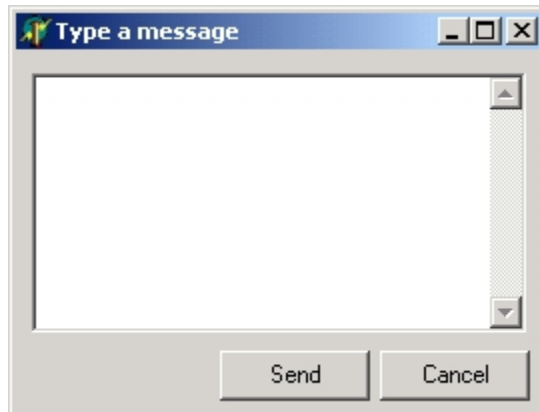


Figure 5 - The Instant Message dialog where you type your message.

The code for the double-click event of the DBGrid component used to list the users is the following:

```
procedure TfrmIMClientMain.DBGrid1DblClick(Sender: TObject);
var
    MsgForm: TfrmTypeMessage;
    ParamList: TAstaParamList;
begin
    MsgForm := TfrmTypeMessage.Create(self);
    ParamList := TAstaParamList.Create;
    try
        MsgForm.ShowModal;
        if MsgForm.ModalResult = mrOk then
            begin
                ParamList.FastAdd('To', UserDataset.Fields[0].Value);
                ParamList.FastAdd('From', AstaClient.UserName);
                ParamList.FastAdd('MsgText', MsgForm.MsgMemo.Lines.Text);
                AstaClient.SendCodedParamList(100, ParamList);
            end;
        finally
            FreeAndNil(MsgForm);
            FreeAndNil(ParamList);
        end;
    end;
```

This code is quite straight forward. First, the message acquisition dialog (fancy name for the dialog where you type your message) is displayed. Second, assuming the user has clicked the ok button, a *TAstaParamList* object is configured with three parameters which are: destination user, sender user and the message text itself.

This data is sent to the server through a call to the *SendCodedParamList* method of the *AstaClient* component, there to be relayed - as we have already seen - to the destination user.

Requesting Personal Information

The process used to request personal information about a user is very similar to the processe of sending a message to another user. Basically, we are just sending a pre-determined message that will be replied automatically by the server, returning whatever personal information the user has entered. The code that initiates a request for personal information is shown in the example that follows.

```

procedure TfrmIMClientMain.btnInfoClick(Sender: TObject);
var
    ParamList: TAstaParamList;
begin
    ParamList := TAstaParamList.Create;
    try
        ParamList.FastAdd('UserName', UserDataset.Fields[0].Value);
        AstaClient.SendCodedParamList(500, ParamList);
    finally
        FreeAndNil(ParamList);
    end;
end;

```

Receiving a Message From the Server

Since the client application should be all that is necessary for two people to communicate it needs to handle the messages it receives from the server. So far we have sent a message to another user and requested a user's personal information. Both of these things were done by sending messages to the server.

When another user sends you a message or the server replies to your request for the personal information of a user, you receive a message. This is treated in an event of the the *AstaClient* object - *OnCodedParamList*. This totally analogous to what was implemented in the server.

The code that treats the *OnCodedParamList* event is shown bellow.

```

procedure TfrmIMClientMain.AstaClientCodedParamList(Sender: TObject;
    MsgID: Integer; Params: TAstaParamList);
var
    Index: integer;
begin
    Case MsgId of
        300: begin
            UserDataSet.DisableControls;
            try
                if not UserDataSet.Active then
                    UserDataSet.Open;
                UserDataSet.Empty;
                for Index := 0 to Params.Count-1 do
                    UserDataSet.AppendRecord([Params[Index].Name,
                                            Params[Index].AsString]);
            finally
                UserDataSet.EnableControls;
            end;
        end;
        200: begin
            ShowMessage('From: '+ Params.ParamByName('From').AsString
                +#10#13+Params.ParamByName('MsgText').AsString);
        end;
        600: begin
            ShowMessage('Full name: '+
                Params.ParamByName('FullName').AsString+
                #10#13+'Department: '+
                Params.ParamByName('Department').AsString+
                #10#13+'Extention: '+

```

```
Params.ParamByName('Extention').AsString+#10#13+
'email: '+ Params.ParamByName('email').AsString);
end;
end;
end;
```

This event treats three different kinds of messages, identified by the numbers 300, 200 and 600. The message identified by the number 300 brings, from the server, the list of connected users and is received just after the client's logon information is validated by the server.

The messages identified by the numbers 200 and 600 bring, respectively, a user sent message and the response, from the server, to a request for a user's personal information.

Wrapping it All Up

What we have seen in this document is a basic example of how to put together applications that use Asta Messaging as the basic communication infrastructure in a Client/Server system.

Asta is a very powerful tool for the creation of applications in a Client/Server architecture. This is true in more than one way and the Asta Messaging features are, generally, under used, since most such applications are database-related and Asta offers an incredible quantity of features specially designed for this kind of development.

It is important to note that the availability of Asta Messaging support for several different platforms, such as Windows, Linux, Windows CE and Palm, offers you a tremendous opportunity for making applications exchange data. As new platforms and new technology, such as XML, are integrated into the Asta family, your options for application inter-operability grow.

It is important that you, as an Asta user, know that there is a lot more that you can do and other kinds of applications that you can create through the use of this technology that is already available to you.

About the Author

Mauricio Longo is an IT Consultant acting in the area of solutions architecture and the author of several successful computer-related books published in Brazil, including six titles on Delphi development. He, also, enjoys developing applications using Delphi, such as his current project - a word processor.

Mauricio can be reached through the following address: mlongo@ctupdate.com.