Create Tomorrow’s Peer-to-Peer Applications Today
In this unique guide, a crack team of software engineers delivers the programming solutions and source code you need to jump-start a state-of-the-art peer-to-peer application project. Using flow charts and line-by-line analysis of a full-scale P2P file-sharing application, they show you how to solve for typical P2P programming challenges—and create your own state-of-the-art P2P application using Java or C#, XML, sockets, threading, ASP, and SQL Server 2000.

Take an In-Depth Look at a Professional-Grade P2P Application
• A complete “discovery and look-up server” model P2P application
• Listener component that handles requests of incoming peer connections
• Server component that maintains the database required for the application
• Browser component with a user-friendly interactive interface
• Plus modules for streaming, Palm device P2P searching, and a chat client

Application tested at Dreamtech Software Research Laboratory
Visit www.dreamtechsoftware.com for updates

CD-ROM Includes:
A professional-quality P2P file-sharing application, plus add-ons,
Java 2 Software Development Kit Standard Edition version 1.3 for Windows
Forte for Java, release 2.0, Community Edition for all Platforms
Microsoft DirectX application programming interface SDK, version 8.1a

Reader Level
Intermediate to Advanced
Shelving Category
Programming

$49.99 US
$74.99 CANADA
£39.99 UK incl. VAT
www.hungryminds.com
Peer-to-Peer Application Development
Peer-to-Peer Application Development

Cracking the Code

Dreamtech Software Team
Dreamtech Software India, Inc., is a leading provider of corporate software solutions. Based in New Delhi, India, the company is a successful pioneer of innovative solutions in e-learning technologies. Dreamtech’s developers have over 50 years of combined software-engineering experience in areas such as Java, wireless applications, XML, voice-based solutions, .NET, COM/COM+ technologies, distributed computing, DirectX, Windows Media technologies, and security solutions.
About the Authors

Lead Author Team

Vikas Gupta, Co-founder and President. Vikas holds a B.E. in electronics, with a postgraduate diploma in sales and marketing and in publishing and printing studies. Actively engaged in developing and designing new technologies in wireless applications, e-learning, and other cutting-edge areas, he is also the Managing Director of IDG Books India (P) Ltd.

Avnish Dass, Co-founder and CEO. Avnish is a talented and seasoned programmer who has 15 years of experience in systems and application/database programming. Avnish has developed security systems, antivirus programs, wireless and communication technologies, and ERP systems.


Ankur Verma, Sr. Software Developer. Ankur has expertise in technologies such as Windows Media, Direct X (Direct Show), and Windows Shell Programming Services for Windows NT series. Ankur also provides the commercial COM/DCOM solutions to Dreamtech Software India, Inc.

Yashraj Chauhan, Sr. Software Developer. Yashraj has an advanced diploma in software development from IBM, with over three years of experience in Java, XML, and C++, and is pursuing his Masters in computer science.

Other Contributors

Pooja Aggarwal and Manish Aggarwal, Technical writers. They have contributed to developing the contents of this book.

Gaurav Malhotra, Pankaj Kumar, Manish N. Srivastav, A team of programmers of Dreamtech Software India, Inc. They contributed to the development of software in this book.

Acknowledgments

We acknowledge the contributions of the following people for their support in making this book possible:

John Kilcullen for sharing his dream and providing the vision for making this project a reality.

Mike Violano and Joe Wikert for believing in us.

M.V. Shastrri, Asim Chowdhury, V.K. Rajan, Sanjeev Chatterjee, and Priti for their immense help in coordinating various activities throughout this project.

To our parents and family and beloved country, India,
for providing an excellent environment
for nurturing and creating world-class IT talent.
Preface

The Internet evolved out of the need to access information from one computer through another or over a network of computers. In the beginning, the Internet was a modest network of a few computers allowing access to data through any of its terminals. Gradually, the concept opened endless possibilities, and the Internet emerged as a super network — a network of networks — spanning the globe and involving millions of machines over which information could be shared, accessed, or communicated freely by the user.

Today, with the Internet’s having graduated to the status of a religion that promises to fulfill the needs of everyone, regardless of caste, creed, or color, and having become potent enough to grant anything beseeched, you can find yourself quite lost in its world. Consequently, the need for personal attention and smaller subnetworks has risen once again. A handful of people got wise to this idea and developed software that allowed users to share data and to communicate with a set of selective people only: friends, relatives, or colleagues.

This has become among the most sought-after software on the IT market. A big user group is benefiting from such software, and many companies are venturing into the concept and capitalizing on it.

In this scenario, an application made on the concept of Peer-to-Peer (P2P), written in popular languages such as Java and C#, with its foundations for database interaction and communication laid in a platform as ubiquitous as XML and, above all, with open-source code and elaborate code explanations, is what developers of such software ultimately dream of. This may sound idealistic, but it is exactly such an application that this book is all about.

What This Book Covers

This book covers the technology of P2P programming. This is the first book to cover the entire code behind a Napster-style, file-sharing model. It contains unique coverage of Windows Media Technology development for making your P2P application multimedia aware. The book is loaded with code, keeping theory to a minimum. The applications, for which the source code is given in the CD accompanying this book, are 100 percent tested and working at Dreamtech Software Research Lab. The source code provided in the book is based on commercial applications, which have been developed by the software company. Each program of the application is explained in detail so that you gain insight into the implementation of the technology in a real-world situation. At the end of the book, some add-ons to this application are provided so that you can further explore new developments.

This book deals with the design, implementation, and coding of the latest form of the client/server model, the P2P model. The book serves to equip you with enough know-how on the subject so as to enable you to design a P2P model of your own.

The book begins with the history of the P2P model and goes on to explain the various types of P2P models, with detailed diagrams to elucidate the subject. After equipping you with basic concepts, it goes on to develop, step by step, a full-fledged application, which has the scope of being extended with add-ons.

This book is not meant for beginners. It teaches you the basics of specific technologies only. The Cracking the Code series is meant for software developers/programmers who wish to upgrade their skills and understand the secrets behind professional-quality applications. This book starts where other tutorial books end. It enhances your skills and takes them to the next level as you learn a particular technology. A thorough knowledge of the Java or C# programming languages is the prerequisite for benefiting the most from this book. Experience in network programming is an added advantage. For developing streaming
applications, knowledge of Visual C++ is a must. At least a nodding acquaintance with the XML markup language is desirable, although the book includes a section on XML. Instructions for embedding existing chat-client and audio/video components have been included. You can craft this application in such a way that you are able to send your files to be printed to any part of the globe. Besides Globalized Printing, you can make the application run on wireless models, too. The opportunity is open for you to assess your networking skills and to improve them.

The pivotal feature of the book is that it offers a complete, ready-to-deploy application with source code. The purpose of this book is to acquaint programmers with the subject thoroughly so that they are in a position to write their own codes to build P2P applications. Detailed explanations of the steps involved in writing your own code to build a P2P application in Java as well as in C# have been furnished.

Although the topic is highly technical, every effort has been made to make the presentation lucid, interesting, and reader friendly.

How This Book Is Organized

Chapter 1 begins with a discussion of various P2P models, goes on to consider design parameters (including the target audience, the environment, and the possible range this application may be required to serve), and finally covers the implementation aspects of one of the aforementioned models. The last chapter is devoted to the add-ons incorporated in the application.

Chapter 2 explains designing the application you intend to develop. Having provided a basic overall view and the technical background of the application, the process of designing the application is taken up in this chapter. The chapter begins with a reiteration of design considerations and parameters. The sort of communication sought and the means to achieve it by way of this P2P application are detailed. The XML design specifications are given in detail, as the interoperability of the P2P application has been achieved by using them. The server side is discussed with a description of using the database, the table design, the XML implementation in SQL Server 2000, and so on. The client side is described with details of the two modules the client is made up of — the browser and the listener — along with the techniques for handling XML parsing, managing peers, connecting to the server and other peers, and finally searching.

It is imperative to mention here that the core of this application is discussed in Chapter 2, and it is exceptionally beneficial and mandatory for product designers and programmers aspiring to develop similar applications to peruse this chapter thoroughly.

Chapter 3 contains every detail of the server in reference to the P2P application. Various aspects of using SQL Server 2000, such as table creation with reference to this application and writing queries for retrieving data from tables, have been discussed in this chapter. This chapter also walks the reader through XML-related capabilities of SQL Server 2000.

Apart from giving you a comprehensive picture of the entire development cycle of the server for this application, this chapter presents a few facts about SQL Server 2000. It elaborates on the complete cycle of database structure tailoring, mounting it, and writing ASPs over it for connected users to be able to interact with it easily.

A sound command of RDBMS concepts helps you get the maximum benefit from this chapter.

Chapter 4 introduces you to the task of real cracking of code and illustrates all aspects of high-level programming. In the Java version of the P2P application, all codes carry a detailed explanation wherever needed so that you can fully appreciate the code.

The code explanation starts with a discussion of classes pertaining to operations on XML involved in the application.
After discussing the building blocks of our application, we discuss the application itself. This comprises the listener module and the browser module (client). These two interact with each other to give the final shape to this P2P application.

Apart from covering the entire application development in Java, this chapter describes the use of some of the XML parsers offered by IBM through Java. A firm command of Java and a nodding acquaintance with IBM Xerces XML parser is desirable, as this chapter deals primarily with implementation without giving any tutorial of either the language or the parser.

If you intend to make your application in C#, skip this chapter and proceed to the next chapter.

Chapter 5 covers the C# version of this P2P application. All codes and documentation processes carry a detailed commentary whenever needed so as to enable you to gain a sound understanding of the concept in question. The code in this chapter has been written using Microsoft Visual Studio .NET (Beta 2).

This application constitutes a listener module and a browser module (client), which interact with each other to effect the completion of this P2P application. While taking you through the development of the essential modules of this application, this chapter provides you with the opportunity to evaluate your skills in the C# language and to improve them. It lets you know how COM components are used in C#, how windows APIs (such as ShellExec()) can be used with interoperable namespaces, and how delegates can be used judiciously in programs.

A working knowledge of C# and MS XML parser is expected, as this chapter deals primarily with implementation without giving a tutorial of either the language or the parser. If you envisage your application in Java, skip this chapter.

Chapter 6 begins with an elaboration of streaming and goes on to discuss the tools, libraries, and resources required for extending this application to be able to stream audio/video data. The chapter concludes by explaining, in steps, the development of a streaming add-on for this P2P application.

The understanding of Windows Media technologies you gain through this chapter can be directed toward developing or extending applications. This knowledge can be used as a bridge to gain access to even better applications of Windows Media technologies and DirectX.

Because the SDKs involved for developing this add-on are available for VC++ only as of now, you need to know VC++ to understand or develop this add-on. This chapter makes use of Windows Media Encoder SDK. Familiarity with Windows Media technologies is expected of you.

Chapter 7 offers you an opportunity to be innovative in extending this so-far-elementary P2P application. It provides a deeper insight into the flexibility and the prudence of the design of this application that makes it possible for you to incorporate add-ons to the application even at an advanced stage in its development cycle. The implementations of the following add-ons have been described and serve as practical guidelines if you want to explore the possibilities offered by this application:

- Chat client
- Globalized Printing
- P2P wireless search (Java version only)

The fully functional chat client incorporated in this application provides you with an understanding of fundamental techniques such as socket programming and introduces you to .NET’s powerful resource designer. Globalized Printing convinces you how easily a new idea can be incorporated into an application if some care is taken to keep the foundation of the application ubiquitous. This notion is further substantiated by the incorporation of the P2P wireless search, which is written in Java and therefore calls for a working knowledge of CLDC (Connected Limited Device Configuration) for a complete appreciation of this extension.
Preface

The other two add-ons discussed are developed in C#, and you are expected to possess a working knowledge of this language.

Who Should Read This Book

This book is intended for those aspiring to learn specific technologies. It is meant for developers who wish to join the evolutionary pathway of innovative software that gives new dimensions to existing technology.

The book primarily targets programmers and project designers who wish to learn the concept of P2P thoroughly so as to be able to develop their own applications.

This book offers code-intensive coverage of Windows Media technologies. It presents a revolutionary combination of two nascent technologies: P2P and Windows Media. The two have been interlaced so that you can extend your applications in a number of ways by using the code for streaming audio/video over the network furnished herein.

This book provides programmers of Java and C# the opportunity to assess their skills and to improve them.

This book also throws light on aspects of CLDC required by a CLDC programmer to appreciate a case study, which demonstrates the technique of enabling a device to communicate with a desktop computer by means of simple-socket programming using XML as the language for communication. The pertinent tool kits and the CLDC VM have been detailed.

The overall objective of this book is to acquaint you with developing cutting-edge P2P applications and creating and extending other software that keeps you in the vanguard of the technical race.
Contents

Preface .................................................................................................................................................. viii

Chapter 1: An Introduction to the Peer-to-Peer Programming Model ...... 1
  A Brief History of P2P .................................................................................................................. 1
  From the Client/Server Model to P2P ......................................................................................... 2
  Various P2P Models.................................................................................................................... 2
    Pure P2P Model ......................................................................................................................... 3
    P2P with Simple Discovery Server .......................................................................................... 3
    P2P with a Discovery and Lookup Server ................................................................................ 4
    P2P with a Discovery, Lookup, and Content Server ............................................................... 4
  Existing P2P Applications .......................................................................................................... 5
    Napster .................................................................................................................................... 5
    Gnutella .................................................................................................................................. 6
  Working of Various P2P Models ................................................................................................. 8
    Tracing Out Other Peers .......................................................................................................... 8
    Querying Peers for Required Content .................................................................................... 9
    Sharing Contents with Other Peers ....................................................................................... 10
  Searching for Shared Resources ............................................................................................... 10
    Server-Side Searching ............................................................................................................ 11
    Client-Side Searching ............................................................................................................. 11
    Searching Techniques in P2P Models ..................................................................................... 11
  A Brief Description of Our P2P Application .............................................................................. 12
    Role of Each Component ......................................................................................................... 13
    The Relationship of Three Components ............................................................................... 14
    How to Achieve the Preceding Design .................................................................................. 15
    The Purpose of Using the Three Components ........................................................................ 17
    Summary ................................................................................................................................ 17

Chapter 2: Designing the P2P Application ................................................................. 18
  Design Considerations for the P2P Application ........................................................................ 18
  Constraints ................................................................................................................................ 18
    Proxy IP Resolution ................................................................................................................. 19
    Problems in Transfer of Data ............................................................................................... 19
    Security ................................................................................................................................. 19
  Structure of the P2P Application ............................................................................................. 19
  Achieving Proper Communication among Modules .................................................................... 21
    Listener to Server Communication Using ASP ....................................................................... 22
    Browser to Server Using ASP ................................................................................................ 22
    Listener to Browser Using XML ............................................................................................. 23
    XML Request and Response Design Specifications ............................................................... 24
  Server Design ............................................................................................................................ 25
    SQL Server 2000 .................................................................................................................... 25
    Complete Database Design .................................................................................................... 26
Chapter 1

An Introduction to the Peer-To-Peer Programming Model

This book discusses the design, implementation, and coding of the peer-to-peer programming model (P2P). It equips you with enough know-how on the subject to enable you to design your own unique P2P model by offering you a complete, ready-to-deploy application with source code. This application, the Dreamtech P2P application, incorporates some interesting features. For instance, it enables you to send your files to be printed to any part of the globe. In addition to globalizing printing, you can make the application run on wireless models.

A thorough knowledge of the XML, Java, and C# programming languages is the prerequisite for getting the maximum benefit from this book. At least rudimentary knowledge of the XML mark-up language is desirable. The opportunity is open for you to assess your networking skills and to improve them.

Although the subject is highly technical, we have made every effort to make our presentation lucid, friendly, and interesting. Since the purpose of this book is to help you to write your own code to build P2P applications, it introduces you to the vital concepts of P2P and briefly goes over networking and networking protocols besides explaining, in detail, the various components of P2P and their designs. Further, it offers an exhaustive description of the steps involved in building a P2P application in Java as well as in C#.

This book deals with the concept of P2P programming that holds immense utility, scope for innovation, and enough potential to change the entire complexion of networking procedures as they stand today.

A Brief History of P2P

P2P is not an altogether novel concept. It has existed since the Internet was taking form in the 1970s. Recent changes in technology and the improved computing capability of desktops have brought about P2P's revival on a larger scale.

Until the recent past, similar systems were used primarily for sharing files within a closed circle of known users over LANs, BBSs, and FTPs. As such, the reach of the P2P model-based applications is confined to sharing files within a known group of computer users. If one wants to transfer files with unknown users, one has to use IRC (Internet Relay Chat) or other BBSs (Bulletin Board Services). In the '80s, the restriction of mainframes to nongraphical user interfaces and the increase of online users created a rush to wire PCs to one another and, more important, to a new class of server computers.

Individual computers were assigned permanent IP addresses until 1994. Newer browser software, such as Mosaic and Netscape, were introduced to access the Web from PCs. These PCs, unlike older mainframe computers, were not always on and entered and exited the Web unpredictably. The sudden gush of Web users caused a scarcity of IP addresses. ISPs (Internet Service Providers) began assigning new IP addresses to users for each session. The temporary IP addresses, in turn, prevented users from hosting any data or Web-enabled application for other peers.
Chapter 1: An Introduction to the Peer-To-Peer Programming Model

As a result of the increased demand for Web-based services, users felt the need to control, exchange, and share resources directly. In the meantime, in the late 1990s, PCs became increasingly powerful in terms of speed and processing power. Hence, software developers realized that they could put server software on individual PCs and initiate a direct two-way flow of information between peers. This led to the revival of P2P.

Interestingly, the IP routing infrastructure is still P2P. Internet routers act as peers in finding the best route from one point on the net to another. Yet overlaid on this structure are several layers of hierarchy. Users get their Internet connectivity from ISPs who, in turn, connect to each other in hierarchies hidden from the end user. These ISPs depend on the same P2P architecture. Similarly, a network of peered-mail servers routes e-mail. Between 1999 and 2000, when Napster was revolutionizing the way users shared music on the Internet, people began to realize the true potential of P2P. Napster is the first large-scale commercial success of P2P in resource sharing.

From the Client/Server Model to P2P

As against the client/server architecture, the greatest strengths of P2P-based models are their decreased dependency on the server and their decentralization of control from servers, which used to be workstations, to peers. Some P2P models do not require servers. End users can directly establish connections with other users without involving servers. Users have more command in P2P-based models than in the typical client/server architecture, in which conventional rules must be followed. Unlike in the C/S system, there is no single point of failure in P2P; in some models, in which P2P puts the server in place, the role of the server is restricted to a bare minimum. To share files, users do not have to seek the help of the server, as they can do this directly among themselves.

In view of these advantages, many corporate houses and computing firms consider the P2P model to be as important as the C/S model. Both models have advantages as well as disadvantages. For example, in the C/S model, the server becomes a bottleneck when too many users log in to download information. In contrast to the P2P model, too many requests sent across the network among users keeps network administrators busy and puts a load on the network itself. Besides, in terms of financial management and administration, the P2P model definitely has an edge over the C/S model. However, practical realization of the pure P2P model still remains to be achieved. Companies such as Intel and IBM are spending millions of dollars and expending significant labor on P2P applications.

In the 1990s, client/server computing architecture was at the peak of its popularity. It attained popularity because it broke down the regime of monopoly of a few data providers across the world; also, it encouraged resource sharing and provided various firewalls to its users. However, in 1999 Napster challenged the C/S architecture. Napster, a P2P-based application, stretched the meaning of sharing beyond the imagination of the C/S creators. With its file-sharing system of MP3 files, Napster gave a new boost and dimension to the network and optimized its role toward greater scalability.

With regard to P2P's popularity and utility, many business organizations are seeking to incorporate it in their purview. Another advantage of P2P is that companies can build collective computing powers and thereby forget servers and expensive storage devices. P2P has shaken the boundaries of networking in terms of sharing resources and costs incurred on servers. Compared with the C/S model, P2P is the better alternative, being more flexible and versatile.

Various P2P Models

P2P models can be divided into the following categories:

- Pure peer-to-peer
- Peer-to-peer with a simple discovery server
Chapter 1: An Introduction to the Peer-To-Peer Programming Model

- Peer-to-peer with discovery and lookup servers
- Peer-to-peer with discovery, lookup, and content servers

Pure P2P Model

The pure P2P model entirely depends on computers (clients in the C/S model). This may seem contradictory because each networking model involves both the computer and the server, like the typical client/server architecture. But the pure P2P model works without relying on any central server. Once the P2P application is downloaded in the memory of the machine, peers find other connected peers on the network dynamically. The entire communication occurs among connected peers without any assistance from a server (see Figure 1-1). By communication, we mean transferring data in the form of uploading and downloading files, carrying out on-line activities, sending requests, receiving responses, and so on.

Figure 1-1: The pure peer-to-peer model

This feature of the pure P2P-based model breaks the conventional method of communication in client/server-based models in which the entire communication process between the client and server takes place based on rules the server sets. The pure P2P-based model allows users to set their own rules and to set up their own networking environments. The P2P model completely eliminates the headache of becoming part of any server or ISP to utilize the Internet.

Pure P2P models provide almost plug-and-play features for working with the Internet, in the sense that you just connect to the Internet and you can use the P2P feature. Another advantage of the pure P2P model is that it not only works efficiently for the Internet but also is quite beneficial for LAN or an intranet.

The only problem with the pure P2P model is finding peers on the network. Because no central administration registers the entry of peers that log in to the network, the users themselves have to locate other peers.

P2P with Simple Discovery Server

The very name of this model suggests its constitution. Such P2P models do not actually involve a server. To affect some administration, server boundaries have been laid down in this model. But the role of the server in this model is restricted to providing the names of already connected peers to the incoming peer, which notifies the server about its presence by logging in. It must be noted that the server only assists peers by providing a list of connected peers and that establishing connection and communication still
remains the job of the peers (see Figure 1-2). Such P2P models surpass the pure P2P model by providing peers the list of already connected peers, which increases the chances of finding a larger number of peers on the network. To download a resource, a peer has to approach each connected peer individually and post its request, which makes the process time consuming.

In contrast, in the client/server-based models, any peer looking for resources needs not go around other connected peers, as the server itself maintains all the required content.

**P2P with a Discovery and Lookup Server**

In this model, the server is used to provide the list of connected peers along with the resources available with each of them (refer to Figure 1-2). Hence, this model integrates the features of the pure P2P and the P2P with simple discovery server models for enhanced functionality of the server.

This model reduces the burden on peers, as there is no longer a need to visit each peer personally for the required information. The server in such a model initiates communication between two peers; once again, the two connected peers establish communication, keep it alive, and perform various activities, like logging into the database the information about the connecting peers, entering an index of resources shared by them, and so on.

**P2P with a Discovery, Lookup, and Content Server**

In this model, the server dominates as in a typical client/server architecture. All the facets of catering to the requests of peers are removed from the purview of peers and reside with server (see Figure 1-3).
Chapter 1: An Introduction to the Peer-To-Peer Programming Model

Also, peers are not permitted to connect with each other directly, as all resources are stored in the database of the centrally located server. If a peer requires information, instead of communicating with another peer, it approaches the server. The server processes requests and displays sources of information.

The major disadvantage of this model is that the server slows down if too many requests come up simultaneously. Another disadvantage of such models is high cost because the server has to manage and store data, and cater to all requests, by itself.

![Figure 1-3: P2P with a discovery, lookup, and content server](image)

Because such models are entirely dependent on the central server, chances of failure through a single point increase, affecting the entire system. This is not the case with the previously discussed P2P models.

Existing P2P Applications

Now that we have described the various P2P models, let’s proceed to examine two well known P2P applications: Napster and Gnutella.

**Napster**

Napster, which is based on the third variety of the P2P models we discussed, is the first commercial success of P2P on a big scale. Shawn Fanning, Napster’s developer, was 19 when he invented it in 1999. Napster’s style model focuses on sharing a particular file extension. Executable files or simple document files can be shared. This is the concept we have used in our application so that files can easily be downloaded or uploaded among computers.

Because Napster allows only files with MP3 extensions, whenever you decide to look for a song, you open the Napster utility. Napster logs on to the central server via the user’s Internet connection. This central server keeps an index of all the registered users that are online. It also maintains directories of the MP3 music files stored on the machines of users. These directories are updated every time you log onto or off of the Napster server.
Chapter 1: An Introduction to the Peer-To-Peer Programming Model

When you send a request or search for a particular song, the central Napster server looks for the song in the indexes of the users currently online. The Napster server then displays the list of the currently connected users whose collections contain the song you request.

You can then click any name from the list and open a direct link with that user. The central server connects you to the user and gets out of the way. The file is downloaded from one user to the other. The actual file is never stored on the Napster server.

Napster does not sell, store, or transfer music files. It is only a platform to enable you to find a file that is requested by the peer. The Napster server enables you to search music available on other PCs and allows direct traffic among users. The Napster central server records its registered users and the music files available on their hard disks. This record is updated each time you log in. When you download and install the Napster program on your PC, it asks you which folders you want to share with other users.

Gnutella

In 1999, Napster forced serious thinkers to reconsider what network is and how Napster had redefined it by stretching the rules of the information technology industry. In spite of criticism from various copyright lawyers and music industries, Napster’s popularity never diminished but kept increasing. Inspired by the way Napster changed the rules, another P2P-based mode, Gnutella, entered the market. It went one step beyond Napster. Napster was the catch phrase of the late ‘90s, but this century belongs to Gnutella.

Gnutella works more or less like a pure P2P model. Gnutella is downloaded and installed on the computer. Once the computer on which you have installed Gnutella, is connected with the network, a message is sent to a computer, which passes it to the other computers, which are also connected to the Gnutella network via the Internet, to let them know about your presence. Message forwarding works this way: You get connected on the network and inform a computer, which, in turn, informs 10 others. These 10 computers inform 9 more computers, and so on.

Gnutella is a client-based software that has a file-sharing service and a mini search engine. Once Gnutella is installed on your local machine, you can serve the requests of other users, acting somewhat like a server and, on the other hand, can find the desired content by sending requests on the Gnutella network for locating the user who has the requested content. You can directly download the content from its machine. Searching on Gnutella is almost like working with any other search engine. In the same way, Gnutella works on the network processes and shows the client what has been requested. Searching on Gnutella is more convenient and promising, as the search occurs directly on Gnutella users’ computers without involving any centralized server search. One limitation is that if the load on the server increases or the server slows down due to some technical problem, all services slow down or come to a halt. Though the searching service in Gnutella is free, this also has some limitations. From the data seeker’s point of view, there is no certainty of the direction in which the request is proceeding, as many data providers are on the network, making it impossible for the client to discover who fulfills the request. This information may be needed for future references. From the data provider’s point of view, there is no guarantee that you hear all the queries.

But in terms of its file-sharing system, Gnutella certainly maintains an edge over other P2P models. Napster encourages its users to share music files via the centralized server solely. Gnutella facilitates not only your ability to share music files but also to share all kinds of files, from MP3 to executable files, without a server.

Some key terms often used in Gnutella networking are the following:

- **Servent**: Because in Gnutella architecture users can be either clients or service providers, depending on the situation, they are collectively referred to as *servents*.
- **Horizon**: A horizon refers to a group of servents. A horizon does not represent the entire Gnutella network but is itself divided into various horizons.
Firewall: A firewall is a shield that protects users inside the Gnutella network. Users outside the firewall cannot send requests to servents inside the firewall, and servents cannot send requests/responses to users or servents outside the firewall.

Broadcasting: Broadcasting is just like the messaging service servents carry out within themselves across Gnutella network. Broadcasting can be done in two ways: send messages over the entire network, or send messages to a particular location.

Dropped Packets: Dropped packets are small packages of data that have been lost or dropped across the Gnutella network. This happens because the client connected to the other end does not keep pace with the data flow. Dropped packets can be minimized by revisiting servents and asking them for dropped packets.

Push Request: Whenever a client behind the firewall is unable to download a file from a servent, the firewall sends a request to the server for uploading the file. This is called a push request.

Port: An application on a computer is assigned a specific port number. For example, the default port number for Gnutella servents is 6346. This means a servent running Gnutella software is available on port 6346.

Although Gnutella is gaining popularity, it is also encountering several bugs in its functioning across the network. Gnutella is still in its infancy and has not matured enough to match its popularity. Some of the major drawbacks of Gnutella are the following:

- The Gnutella protocol, being very simple, knows only how data is transferred from one location to another. It does not guarantee that each site interprets data. The number of lost packets in Gnutella is quite high. In addition, no provision stops requests that keep crowding a servent. In such situations, the servent can easily become a bottleneck.

- The fixed time to remain an eligible user of the Gnutella network imposes a limit on its users.

The biggest drawback of Gnutella is that it is very hard for the user to verify the source of information, that is, the peer on which the client’s information is processed, as security is not addressed adequately. In other words, it is very difficult to find the server where the client’s request is being processed.

Yet, in spite of all its limitations and the criticism it faces from the IT gurus, Gnutella and Napster are in the race neck and neck. It is very hard to judge which one is better, because in one way or another, each is maintaining an edge over the other. A brief comparison is presented for you to decide whether Gnutella or Napster deserves the higher pedestal (see Table 1-1).

### Table 1-1: Pros and Cons of Napster and Gnutella

<table>
<thead>
<tr>
<th>Napster</th>
<th>Gnutella</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Positive</strong></td>
<td></td>
</tr>
<tr>
<td>Gives material to its customer in the form of MP3 music files all across the continents. This is the key reason for its popularity.</td>
<td>Truly decentralized network due to the absence of the server. Chances of failure at single point to affect services is completely ruled out.</td>
</tr>
<tr>
<td>Provides a protective layer to its user due to the involvement of a server. Fully authenticated data travels on Napster network.</td>
<td>Shares not only MP3 files but also other files such as .exe, jpg, and so on.</td>
</tr>
<tr>
<td>User friendly and easy to download on your computer.</td>
<td>Has a provision for redirecting the request, plus supports HTTP protocol.</td>
</tr>
<tr>
<td><strong>Negative</strong></td>
<td></td>
</tr>
<tr>
<td>Server presence slows down or brings services to a halt in case of a technical problem. Server can</td>
<td>No provision for stopping the flood of requests.</td>
</tr>
</tbody>
</table>
Chapter 1: An Introduction to the Peer-To-Peer Programming Model

- Easily become the bottleneck once requests flood it, as there is no provision for redirecting the requests.
- Works for only MP3 files. It does not share any other files on its network.
- Facing the opposition of copyright lawyers and the music industry.
- Absence of the server makes user feel unsafe, as it is tough to find out on Gnutella network where data is sent from.
- Encounters many bugs in its application, such as lost packets.

<table>
<thead>
<tr>
<th>Working of Various P2P Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Whether it is a P2P-based application or a typical client/server-based application, the way an application works matters a lot. A typical P2P-based application can be based on any of the models discussed previously, but some prerequisites exist for an application to be an ideal P2P application. Some of the key features every model P2P application should have are the following:</td>
</tr>
<tr>
<td>♦ <strong>Tracing out other peers:</strong> Finding other peers connected to the server</td>
</tr>
<tr>
<td>♦ <strong>Querying peers for required content:</strong> Getting lists of shared resources from other peers</td>
</tr>
<tr>
<td>♦ <strong>Sharing content/resources with other peers:</strong> Learning how contents are shared among the peer network</td>
</tr>
</tbody>
</table>

In the upcoming discussion, a comprehensive explanation makes you more familiar with how various P2P models work (that is, how P2P-based applications incorporate these essential features).

**Tracing Out Other Peers**

Tracing out other peers is an essential feature of every P2P-based application. This feature is discussed with reference to different P2P models in the following sections.

- **Discovering other peers in pure P2P-based model:** Because this model is without a server, peers find others dynamically and communicate between themselves directly. This communication, therefore, is not restrained by the terms and regulations that the conventional methods impose. However, though local configuration schemes and network messaging services are available, a user logging on to a pure P2P model may not always obtain a substantial number of peers to cater to the request posted. Besides, the direct communication between peers affects security.

- **Discovering peers in P2P with simple discovery server-based model:** This model incorporates a centrally located server. This server, in its database, stores the information related to all registered users. Any query for searching a particular peer is processed by the server, which returns a list of other peers from its database. The main advantages with this method are enhanced security and the availability of a large number of peers to the requesting peer. However, if the server slows down or crashes, locating peers becomes difficult, and other peers are also affected.

- **Discovering peers in P2P with a discovery and lookup server-based model:** In this model, the server provides a list of services as well as a list of peers. All users are required to notify their presence to the server while they log in. The server not only discovers other peers but also returns vital information regarding all logged in peers. Therefore, this model reduces time consumption considerably as compared to the other models.

- **Discovering peers in P2P with a discovery, lookup, and content server-based model:** This model traces out peers in response to a request much like the others. The centrally located server in this case maintains all the vital information and also furnishes the content to requesting peers. Each peer is registered with the server, and the server handles all requests on its own and serves responses. The dependency on the server is therefore very high.
Chapter 1: An Introduction to the Peer-To-Peer Programming Model

Querying Peers for Required Content

The user initiates the request for content. Any application can make a request for content to a peer once a peer has been traced and located. Only a peer can provide the content required. A peer might not entertain the query the user presents or might not have the information the user seeks. In such cases, the better option is to utilize the server to send the request over the network, as in the pure P2P model, than to approach a single peer.

Yet the strong point of the P2P model is that against conventional models, which invariably require a centrally located server for communication among peers, it allows direct communication among peers. If the approached peer has the requested content and it is willing to furnish it, the positive points of this architecture become evident. The relegation of the server to the background pre-empts the situation in which overload on the server renders it a bottleneck. Because the load on the server is reduced, the overheads to be incurred on account of the server are reduced. Also, if the server breaks down, services are not halted. Apart from tracing out potent peers and initiated connections, the role of the server is not significant. Once connection among peers has been established, communication takes place among peers, and the server is not in the picture.

Tracing out other peers and initiating connections are the server's tasks. Peers query directly with each other once the preceding two tasks are complete. The process by which peers query one another for content also varies slightly depending on the P2P model being employed:

- **Querying peers for content in the pure P2P model:** A peer passes its request for content to another peer, the latter being referred to as the host peer. If the host peer contains the required information and it is within the scope of its criteria for shared resources, the latter satisfies the former’s query. Thus, a peer can procure an executable file from another peer, which is found dynamically. This model provides an interchangeability of the role of peers. Thus, when a peer requests information, it acts as client; and when a peer is providing information, it assumes the role of a server. In effect, this model permits the exchange of the client/server functions.

- **Querying peers for content in the P2P model with a simple discovery server:** This model uses the functionality of a server (that is, of providing the list of logged-in peers to a requesting peer); however, another peer provides this service. The most patent advantage of this model over the pure P2P model is that because the presence of a centrally located peer acts as a server, a requesting peer always gets a large number of peers of interest so it can communicate with them. The main disadvantage is that because the requesting peer has to contact each peer in the list by itself and individually for processing the information, processing time is abundant. Overheads are the regular maintenance required for the infrastructure, such as data-storage capability and related peripherals.

- **Querying peer for content in P2P with a discovery and lookup server:** This model provides additional advantages over the previous models, attributed to the more pronounced role of the server. The role of the server is not limited here to just maintaining a list of registered users but extends to processing queries for the content the requesting peer seeks. The requesting peer in this case passes its query to the server, not to individual peer. The server processes the query to locate the peer that has matching contents, and this information is returned to the requesting peer. The searching path of a peer seeking content is thereby shortened. But with this model, the server is taxed, and its increased demands are liable to affect its speed and general performance. This shortcoming is more than offset by the drastic reduction in network traffic.

- **Querying for content in P2P with a discovery, lookup, and content server:** In this model, the role of the server is the most significant. The requesting peers approach the server, and the server not only processes the request but also procures the result and returns it to the requesting peer. The server not only maintains a list of registered users but also undertakes the entire connection management to provide the content by itself. Because the server has to manage the whole session for providing the content, it is prone to become a gridlock. Also, costs on account of the server are high. However, this method protects information from invasion. This advantage is substantial. The server ensures reliable, uniform process handling coupled with caution that makes for high security.
Sharing Contents with Other Peers

Sharing refers to how contents are asked for, how resources are shown and shared, and how connected peers share resources over the network on locating each other. As mentioned previously, the role of the server is not mandatory with some P2P models. Files can be passed between peers without resorting to the server, except for some cases in which the server initiates connection. Thus, information and resources are shared among peers over the network. The peer provides the resource that parses the query of a requesting peer to return the result. This must not be confused with the result(s) the server provides. The server provides information about connected peers only. Note that in the P2P model, with a discovery, lookup, and content, the server controls all the operations and that this model is an exception.

♦ Sharing resourcea with peers in the pure P2P model: When a peer sends a request for content to another peer, if the desired content is available with the latter, it is downloaded across the network. In the case of the pure P2P model, peer is the service provider which has shared resources. The requesting peer just sends its request to the peer it approaches. It is the peer that has the requested information that opens the connection.

♦ Sharing resources in the P2P model with a simple discovery server: Here, a peer that requires content sends its query to the server. The server, in response, returns the list of all connected peers. The requesting peer approaches peers individually from this list for required content. Once the requesting peer locates the content required, the server connects them. The requesting peer downloads the required content from the network. Though the process is lengthy, the requesting peer obviously has a substantial number of peers it may approach. Uploading content cannot occur in this model. We discuss this limitation in the following section.

♦ Sharing resources in the P2P model with a discovery and lookup server: As with the previous model, in this model the requesting peer approaches the server but differently. The server doesn't just furnish the list of peers; it carries out the search on the basis of the search criteria and traces out peers of relevance. The requesting peer downloads the required content from the network stream. Uploading is also possible with this method. A peer can upload contents to the server as well as to other peers. The availability of both uploading and downloading makes this model highly flexible as compared with the other P2P models.

♦ Sharing resources in P2P with a discovery, lookup, and content model: This P2P model is almost like the C/S computing architecture. In this model, all information is housed in the centrally located server. Here, the peer that requires content passes its request to this centrally located server. The server processes the request, procures the result, and returns it to the requesting peer. The server itself carries out both uploading and downloading. For retrieving content over the network easily, this is the best-suited option. This model offers high security to users. Uniformity in accessing the contents and information reliability are other advantages, whereas high costs due to the server might be daunting.

Searching for Shared Resources

Searching is a term every computer user is familiar with, but it has many subtle connotations that are often overlooked. Generally, it means searching for a file or a folder by using a user-entered phrase. Search engines such as Google or even the local machine itself may be used for this purpose. Searching is used mainly to save time and to reduce effort. Thus, a search technique may win a required file from a large database of files in no time. But searching techniques are best appreciated if you consider how to sort out a situation over a network when you know what you want but do not know where to find it in the vast diversity of randomly distributed files. Here, you may type the phrase that you want to search, and the server will do the searching and return the results. All Web sites and service providers incorporate searching facilities as imperative features because they accumulate an otherwise unfathomable ocean of information.

Searching may be divided into two distinct categories:
Chapter 1: An Introduction to the Peer-To-Peer Programming Model

- Server-side searching
- Client-side searching

Server-Side Searching

Networks such as LAN, MAN, WAN, and so on, employ server-side searching. In general, a server, wherever present in computing architecture, acts as the reservoir of all information. Often, data integrity, consistency, and security a computing architecture can boast of can be ascribed to a server acting as a centralized data-management unit. Multiple-user requests are processed on the common platform the server offers. Server-side searching always results in the most updated information. Search criteria have to be passed to the server, and the server must refer to its database or file system to cater to the request. The search result is returned to the user as the response. The greatest advantage of server-side searching is that the user need not worry about the location of a file. Connection may be made only to peers that possess required content. This is particularly relevant in a situation in which many users are connected and are liable to modify a file. Although safe and descriptive, the requesting clients may send an avalanche of requests and may slow down server’s services.

Client-Side Searching

Client-side searching comes into play when peers are connected with each other and are communicating without the involvement of any other machine or server. But even with client-side searching, things can look a bit cluttered if too much of the information is shared by any of the peers involved in communication. Thus, the search facility finds its relevance at this level, too. Usually, client-side searching is performed on the basis of prior information about the contents residing on one peer. Other peers connected to such peers may search at this level to make their work easier. Once they are connected to the peer that has the required information, they don’t need to look for that information on that computer; search facility, at this level, expedites the process of finding information.

Such surfing occurs on a local machine (that is, a search is confined to a particular computer only). It is not concerned with network searching performed on a server. Here you search for a particular file or folder within your machine. Client-side searching reflects the changes and modifications you have made before saving the document. In addition, client-side searching is limited to the user’s machine but is faster than server-side searching because it occurs on a single machine, and a single user performs it.

Searching Techniques in P2P Models

In the P2P model, the search for content keeps changing according to the applicability of the model. Some models of the P2P architecture follow client-side searching, others follow server-side searching, and others follow both types of searching. Searching facilities always have to strike a balance between the merits of speed and reliability on the one hand and the flaws of nonpertinent information and unreliability on the other. Advantages are speed, direct communication, and reliability; disadvantages are unreliable and nonapplicable information. The following are the types of searching techniques various models follow:

- **Searching for content in the pure P2P model:** Because this model is without a server, it employs the client-side searching model. If a requesting peer has prior information regarding the location of another peer, which has the required content, the process provides unsurpassed speed. There is no queuing of requests on a server to rank priority, as with the conventional client/server models, for procuring the content is a matter of direct communication among peers. However, the absence of a server imposes a dearth of peers that may be approached for content.

- **Searching contents in the P2P with a simple discovery server model:** This model does have a server, yet it employs the client-side searching technique. The server presents the requesting peer a list of connected users. The requesting peer finds which peer has the required information, approaches it, and procures content. Obviously, the process is lengthy as far as the requesting peer
An Introduction to the Peer-To-Peer Programming Model

is considered. The advantage of this method is that it provides a large number of peers that may be approached for the required content.

♦ Searching contents in P2P with a discovery and lookup server model: This model follows both client-side searching and server-side searching. Among all P2P models, this model is the most efficient and flexible. Whenever a peer wishes to find or search some particular content, it can search directly on the already connected peers on the server, or it can simply query the server to reply with the results of the required information. In this model, apart from connected peers over the network, the server remains available to serve the requesting peer. Such a searching facility is possible on the server because whenever a registered peer logs on to it, apart from authentication, it sends the list of shared resources on the server along with its login name. Hence, it becomes easier for the requesting peer to search the content instead of approaching each peer individually. In case the requesting peer does not want to take the assistance of the server, it can directly approach the peer and can pass search criteria to it. The advantage of this search technique is a flexible approach that facilitates the requesting peer with both searching options. The time consumed to search information decreases rapidly in this model. But in spite of the heavy cost the server incurs, this model of P2P computing architecture is still the favorite among developers and users.

♦ Searching contents in P2P with a discovery, lookup, and content server model: This model restricts all client-side searching techniques, as the centrally located server takes command of the entire network. Here, the requesting peer is not required to contact other peers for information, as the server caters to all search processes. This model is completely based on the server-side searching technique. Whenever a peer requires information, it simply contacts the server, as the server keeps the entire information in database, ranging from information about the registered peers to all possible contents. Because the entire network in such models remains in the hands of the server, it is very easy for the server to become a bottleneck. If too many peers approach the server for content, its processing speed decreases and so do other related services. The positive side is that peers perform in a uniform way and receive results in the same way. Besides, there is no risk that any malicious peer is sitting at the other end, thereby shielding users.

A Brief Description of Our P2P Application

The application developed in this book involves various high-level programming techniques: C# and Java programming; socket programming; Web requests and responses; threading; and XML documents. You must have a working knowledge of the preceding concepts to understand the application developed in this book. Because XML communication is the backbone of our application, allowing an interoperable design, you must know XML thoroughly. The application also involves wide usage of threading and sockets.

We have discussed developing this application in Java as well as in C#. The software requirements for the Java version are JDK1.3, Apache Xerces XML Parser, CLDC 1.0.2, and KXML Parser. For the C# version, the software requirements are Microsoft Visual Studio .NET (Beta2) and MS XML Parser3.0. The hardware requirements for both the versions are Pentium1 or above, 128MB RAM or more, 1.4G or more of free hard disk space, and an Internet connection. The Operating System is Windows 2000 (Professional Edition) for both versions.

Our P2P application is based on the third model (P2P with discovery and lookup server model) of the P2P hierarchy. In our application, file sharing takes place between two connected peers regardless of their file extensions. The application consists of three components:

♦ Listener: Handles requests of incoming peer connections
♦ Server: Maintains the database required for the application
♦ Browser: Shows results to the peer through a user-friendly interface and acts as an interactive layer for the client
Chapter 1: An Introduction to the Peer-To-Peer Programming Model

Though applications in the P2P model can be built without involving the server, in the long run it may cause problems for end users and at the network-traffic-management level. Moreover, with the presence of a server, users automatically feel secure, as it protects them and applications from any invasion such as unauthorized access and downloading files. This application is the first step toward resource sharing in a true sense; from server technology to client technology, every bit of information is provided for the user in an interactive and user-friendly manner. Application components handle and perform the following tasks described in the following section.

Role of Each Component
Each of the preceding components performs a specific role to enable the application to run smoothly. The boundaries and roles of each component are described sequentially.

The listener
The first jobs the listener performs are logging on to the server, notifying the server about its own presence, and listing its shared resources. After logging on, the listener can handle the requests of clients the server passes to it. The listener downloads files whenever a client approaches it. Along with the downloading process, a listener can easily handle multiple clients with their requests. In essence, while handling multiple clients, the listener acts like a server. While accepting requests and responding to clients, the listener, like a server, enables you to use the search option in an expanded way. As mentioned previously, the server performs only root-level searches (searching for the shared resources at the basic level). To search beyond the root level, the listener helps you.

If the listener declares any folder directly as a shared resource, you can easily open the folder for details and view all files listed under it. Such a search is much faster than a root search performed at the server level. At the listener level, the search occurs on a single machine with prior information about the location of content. One important feature of the listener is that it can run in unattended mode; that is, in the absence of the user.

The server
The server holds the list of all registered listeners along with the detailed description of their resources, which listeners offer to share with other peers. Notice that a file on the client or the listener machine never passes through the server. After providing the list of listeners, the server initiates a connection between the listener and the client. The rest remains in the hands of the listener and the client. If the client wants to download a file from the listener machine, it does so directly without involving the server. A client can query the server to show the list of listeners, which can fulfill its request.

In the P2P environment, such searching is called a global request. Another term associated with searching while using the server is root search, which occurs while the listener declares its shared resources. In simple terms, after establishing a connection between the listener and the client, the server points out the folder or directory that contains the required content for the client. It does not facilitate an in-depth search facility of every folder or directory. To do this, you have to understand the listener's role detailed in the previous section.

The browser
The browser acts as the interface between the user and the computer. Through the browser, the client sends its requests and receives the listener's response. Unlike the listener, the browser always runs in attended mode. The browser shows the list of listeners logged in and sends all requests to the listener, which the listener itself processes. Once the listener processes the request, the browser shows the results at the user end. Note that it is not the browser that does the processing; the listener does this. The user sees only the result of the query. Processing content is kept encapsulated from users.

You can search for shared contents based on various criteria with the help of the browser. For instance, a client can request that the listener show only document files or request that only executable files be
Some of the highlights of this application are as follows:

♦ Downloading and uploading files is possible.
♦ Intense search-option facilities are available to easily locate files ranging from document files to executable files and Dynamic Link Libraries.
♦ The interface is user friendly with online help to guide you.
♦ High-speed connectivity with the server is available.
♦ Easy chatting over the network is possible.

This is our first step toward redefining the network, and we expect a positive response from our users.

The Relationship of Three Components

Apart from performing their individual tasks, the server, listener, and browser bear some relationship to each other. Based on this application’s P2P computing architecture, we categorize the relationship of the three components in the following way:

♦ Listener to server relationship
♦ Listener to browser relationship
♦ Browser to server relationship

Listener to server relationship

In this relationship, the server maintains the database of all listeners along with their names, IP addresses, and, most important, their shared resources. After being logged on to the server and declaring its shared resources, the listener sends only the names of files and folders along with their destined locations, not the contents of files and folders. By disposing of all files and folders and their contents, the server becomes a bottleneck; plus, the entire load of file sharing and downloading is the server's burden, degrading its performance. Another important point is that the server gives you the list of listeners currently online only, not the list of logged-out listeners.

Listener to browser relationship

The relationship between the browser and listener, to a great extent, is a request/response relationship in which the browser sends all requests to the listener and receives a response in the form of a processed query or message. In this relationship, the listener processes all requests at the listener level, and the browser receives the result at the client end. The client can send requests in various forms:

♦ The client can show all the shared files and folders.
♦ The client can search or ask the listener to show contents in files and folders, which can't be done at the server level.
♦ The client can download the contents by passing a downloading request to the listener.
♦ The client can upload its files into the listener's account.
♦ When it comes to downloading and uploading files from listener by client, the listener has an upper edge since while sharing contents with the client, the listener has the right to impose its content or information as readable only or readable as well as writable. These rights can be put along with the folder as its property. With these rights, the listener can restrict malicious clients from uploading unnecessary files into its account as well as restrict clients from downloading important information.
Chapter 1: An Introduction to the Peer-To-Peer Programming Model

**Browser to server relationship**

In this relationship, the browser brings the list of the listeners at the user end from the server, and it checks with the list with which the listener is sharing resources. Along with retrieving the list of listeners from the server, the browser can pass the root-level search requests. In other words, the browser can search all listeners for the content you require and can inform the client about all eligible listeners as the client demands or requests.

**How to Achieve the Preceding Design**

To achieve the preceding design, you need three components: the server, listener, and browser. But before proceeding, be aware that, unlike in the pure P2P model, this model incorporates the role of a server. Though the role of the server in this model is not as active as in the C/S architecture, to some extent peers do need to rely on the server for first-hand connection. Sharing of files occurs between the listener (peer) and the client (peer) without involving the server. Contents cannot pass through the server while they are going through the exchange between one peer and another. This model extends the role of the server to include content lookup services.

In this case, the peer-to-peer application not only registers with a server but also uploads the lists of its contents at regular intervals. When an application (client) is looking for some particular content, it queries the central server rather than sending the query to each client. The central server responds with a list of clients (listeners) that contain the requested content, and the peer application (listener) can contact these clients directly to retrieve the content. This model proves to be more efficient than the pure P2P model because it reduces the number of requests going over the network. But, of course, the reduction in load over the network is offset by the higher cost of the server.

Figure 1-4 shows a server (listener) and four separate peers that run the browser software. In fact, the listener can be a browser for other listeners also in cases where one listener is drawing services from the other listener (services can include any of those that a browser can ask for). In this case, after getting the list of the currently active listeners from the server, the browser establishes a direct connection with the listener. More than one browser can be connected to one listener, as shown in Figure 1-4. The browser computer (the client) sends a request to the listener computer in the XML format, and the listener computer responds with the result to the request. If any error occurs in the middle, the listener sends an error-XML response to the browser, and the browser informs the user. This communication is open both ways, from the listener to browser and vice-versa. Exchanging information takes place by using the standard XML document format. Every connection is socket oriented. The listener listens on a particular socket for the browser to connect.

**The Purpose of Using the Three Components**

In our application, we are using three modules acting like three pillars, which hold the application. These modules are the server, the browser, and the listener, each having a specific role. While preparing the application, we made thorough considerations to enable the application to move among distributed users and various operation systems. Languages such as XML, C#, and Java have been incorporated for easy debugging and increased modularity. Code written for this application is not confined to it, but users can implement code into their own applications. For instance, an audio/video module written as an extension to this application can be used in its existing form — no change made — in any new application to support audio/video streaming. Almost all functions used in this application are implemented and written in such a way that with minor changes in naming conventions, anyone can use them in an application.

Whenever the Java appears, interoperability follows. Java makes this application more flexible, as it is not restricted to any operating system.
Figure 1-4: Once the list of active listeners is obtained, browsers establish direct connections with them.

Various features are in this application to make it more scalable for the near future. Our application has been designed to survive the changes foreseen in the near future. A server component in our application reduces the traffic over the network and provides protection and performance speed to its users. On
behalf of the end user, the server handles the searching process quickly and efficiently, thereby reducing the need for front-end processing. Because our application is distributive, the server can handle multiple clients at any particular moment, thereby increasing the reach of the application.

Summary

In this chapter, we have gone through the history of the evolution of the P2P model. We have discussed the types of P2P models to enable you to understand every aspect of P2P technology. Moreover, we have had a short discussion of the existing P2P models so that you may understand the application better. In addition, we have mentioned the basic technologies you must acquire before exploring application development by using the P2P model. In the following chapters, you find the details of the design of our P2P application, complete coding, and an exercise to help you explore your own skills.
Chapter 2

Designing the P2P Application

This chapter explains the process of designing the P2P application described in the previous chapter. Having provided a basic overview and the technical background of the application and considerations for designing the application in the previous chapter, here the actual process of designing the application is taken up. The chapter begins with a reiteration of design considerations and design parameters. The communication requirements and the means to achieve them by way of this P2P application are detailed. XML design specifications are provided in detail, as the interoperability of the P2P application has been achieved through XML. The server side has been discussed with descriptions of the database, table design, XML implementation in SQL Server 2000, and so on. The client side has been described with details of the browser and the listener, such as handling XML parsing, managing peers, connecting to the server and other peers, and searching.

Design Considerations for the P2P Application

While designing this application, the following key points have been borne in mind.

Applications of this kind are likely to be used on global scale. Thus, it is quite probable that the application will encounter networking problems. Its global nature also leads to a situation in which a good number of users are communicating with one another simultaneously. Therefore, network traffic may exceed the limits estimated to deploy the application to handle data without facing memory problems.

Eventually, users of the application interact with it. Thus, the design should be equipped to be of use in a variety of environments, whether it is used by users at the individual level or by corporate organizations at the enterprise level. Users might propose to avail the design for maintaining their own universities, schools, or community-level networks. Thus, the design has to be flexible enough to easily scale down to work under smaller networks such as LANs.

Without altering its foundation, the application should have the flexibility to be molded as required by the ever-changing technology. This application may serve the basic purpose of resource sharing, but the possibility of the user extending its scope and usability by incorporating additional functionality cannot be overlooked. Because this application is chiefly devoted to educating readers about the idea and the practical implementation of P2P, the design must be such that it easily offers itself to any programmer seeking to add modules written in any language in the existing model of this application. This can be made possible by using ubiquitous techniques legible to all programming languages and platforms. The application should be able to support sharing of all file extensions, not just a few specific ones, and the design must tide over the obstacles imposed by the previously mentioned design considerations.

Constraints

In view of the preceding considerations, it is quite obvious that very little latitude was available as far as the designing process is concerned. The application had to regard and honor all the aspects of design considerations and make its way through the restrictions imposed by them.
Chapter 2: Designing the P2P Application

Proxy IP Resolution
Because the proxy server serves many nodes through a single Internet connection, individual IP addresses no longer have meaning while working on the Internet. The proxy server recognizes all IP addresses behind it by using its own IP address. On behalf of the user, the proxy server sends and receives data. In essence, the proxy server acts as a layer between the end user and the Internet. In the absence of a proxy server, all machines are free to establish connections directly on the Internet. So this is a scenario in which dynamic IP addresses assigned to the machine are being used for communication through sockets. How do you make this application work behind a proxy? This has been reserved as an interesting topic for you to explore.

Problems in Transfer of Data
Because users of this application are likely to be scattered around the globe, communication might be the main issue as the probability of a large number of users flocking to the server all the time is high. Download and upload processes are the most sensitive operations, being highly susceptible to communication problems. This problem is tackled to an extent in this application by keeping the connection open long enough only to serve the request. Keeping only the active connections (connections on which some operation is being carried out) open reduces the number of connections at a given time to be monitored, thus reducing load on the server.

Security
While discussing the possible users of this application under design considerations, we have considered whether this application will be used in corporate environments or by individuals. The main difference between the two is that under security-intensive corporate environments, in which corporate firewalls generally allow communication on a particular port only, this application may fail at times to communicate. If this application were aiming for a corporate audience, we would have to talk to the administrator of the concerned corporate network; then the application would have to communicate through corporate firewalls. But because this application is designed for users who are not likely to go through a firewall to find other peers, just about any port that is not already reserved for another protocol can work.

This kind of communication demands that you use registered ports ranging from 1024 to 49151 or dynamic or private ports ranging from 49152 to 65535.

For a list of the well-known and registered ports, an IANA port-assignments list is available at http://www.iana.org/assignments/port-numbers. For this application, port number 7070 has been declared.

Structure of the P2P Application
As mentioned in Chapter 1, our P2P application is based on three components: listener, browser, and server. In Figure 2-1 we depict two peers, the listener and the browser running our application from remote locations intending to share resources with each other. While sharing resources with each other, both peers perform two basic operations:

♦ Making requests: A process for establishing communication for further interactions
♦ Responding: A process for returning an appropriate response either through the messaging scheme or by simply returning the result corresponding to the request
In our application, the peer depicted as listener has been assigned the job of catering to the requests made by other peers, whereas the browser is the peer from which all requests are made.

The server maintains the database that holds records of information of all logged-in peers, such as their IP addresses and login names, along with summaries of their shared resources (resources peers have declared as shared for other peers). It is important to mention that all components involved in our application use XML. Our application heavily depends on XML, as whatever communication takes place between the components in the background is through XML documents. We have chosen XML as the communication medium for our application because XML is legible to all contemporary programming languages.

Two prewritten components are used for handling XML in our application:
- **The XML parser**: Used for parsing the responses and requests generated
- **The XML creator**: Used for creating appropriate requests and responses

At the start of the application, the listener logs on to the server by sending an HTTP request containing information such as the login name, the IP address, and a description of shared resources. The listener is required to make entries to the server, which authenticates the information it receives from the listener. Authentication by the server is successful if the information passed is correct; on the other hand, authentication fails if the server finds something wrong, such as duplicity, with the information sent. In both cases, the server returns the appropriate response in XML format, which is then forwarded to the XML parser for parsing. Finally, the appropriate message is displayed to the listener.

Now we assume that the peer acting as the listener has successfully logged on to the server. The listener is ready to cater to the requirements of other peers. Meanwhile, the browser looking for content approaches the server by sending an HTTP request.

Once again, the server responds to the peer in XML format but with a slight difference. In the former case, the XML response is returned subject to authentication. In case of a browser request, the server
returns the XML response as the list of all connected peers. The server equips the browser with the list of
the various listeners along with their shared resources and respective IP addresses.

Now, based on its requirement, the browser can choose the listener. From here on, the role of the server is
absent, as it has completed the task of initiating communication between peers. It must be emphasized
that the server initiates communication, whereas the browser establishes the connection between itself
and the peer for further communication. The listener never initiates the connection with the browser;
instead, it searches for the requests on the connection opened by browsers.

Thus, the browser performs two steps: It approaches the server and then establishes communication with
the listener. Subsequently, the communication exists directly between the two peers (the listener and the
browser). The browser opens the network streams for the listener for reading and writing data in XML
format. Generally, uploading and downloading files/folders takes place between peers.

To retrieve content from the listener, the browser downloads specific information from the remotely
located listener. For this, the browser creates the request XML by using the XML creator component and
writes it on the network stream of the listener. The listener then parses the incoming request and finally
judges which file needs to be uploaded for browser calls waiting on the XML parser. It can either be a
response written for the request made by using XML creator or the file requested for download, which is
then uploaded by writing it on the network stream of the browser. Finally, the browser reads the file
uploaded at its end by the listener, checks whether the file is a response for the request the browser has
made or a file the browser has requested for downloading, and the appropriate message is displayed.

Just as the browser downloads files from the listener, it can upload files to the listener. To achieve this,
the browser selects a memory area that is shared by the other peer through its listener, selects a file that it
intends to upload, and generates a request XML to the listener. Upon receiving such a request, the listener
checks the shared memory area for the credentials for the file to be written. Upon finding sufficient
credentials, it begins to read the file being written by the browser running on the other peer and writes the
data coming its way. If adequate credentials are not found, a denial response is made and returned to the
browser.

These two processes are described in more detail later in this chapter in the section titled “Downloading
and Uploading Files.”

Achieving Proper Communication among Modules

This application is designed to make room for third-party add-ons to be incorporated effectively. Achieving
communication with add-ons written by third parties has been the major obstacle to
accomplishing this design, as new extensions written for this application should be able to converse with
it in its existing form. Also, the incorporation of new modules must not tax the performance and must be
compatible with the application’s existing design.

To construct such a flexible structure, there are only a few technical options to choose from. Foremost
among such techniques is COM, which has been popular with programmers lately for such designs. COM
is a popular model because of its capabilities place special emphasis on communication among different
and distant modules. COM's competence in effective communication and callbacks, even across networks
(through DCOM), makes it a good choice for the foundation of software.

But the biggest shortcoming of COM is that in its present form COM/DCOM is not well supported by
languages other than those Microsoft has hybridized to support it, one such language being Java. Neither
on technical grounds nor on commercial ones can it be assumed that Java programmers would never turn
toward writing add-ons for this application. So COM has had to be ruled.

The option left is to custom craft an agile structure that perfectly suits the work at hand. This is not as
difficult as it sounds. After a little thought and exploration, we have encountered SOAP (Simple Object
Access Protocol), in which every software essentially stands on a basic exchange of data blocks called
events. If this basic data exchange can be standardized, any kind of module is able to communicate with any other. This can be done by using XML as the mediator for communication. When we write communication, we actually mean firing events, handling them, and responding back. This idea has been chosen to be the foundation of our software. In the following sections, we explain how communication among different modules has been achieved.

**Listener to Server Communication Using ASP**

When the listener is initiated, a login window is displayed. In this login window, you are prompted to enter the name with which you want to be identified. Internally, the IP address of the listener, not the login name, is used for communication. After entering the login name, click the login button. The listener then calls an ASP page named Login.ASP. To register you on the server, the listener has to pass the shared resources list, your login name, and your IP address (see Figure 2-2). The login page and parameters are as follows:

\[
\text{http://abc/login.asp?USERID=username&IP=ip\ addresses\ SHARE=file1*file2*...*filen}
\]

Make all the necessary entries after validating each and every argument and then throw an XML as the response to the listener who has made the request. After this, the server is bypassed, and the listener never communicates with the server until it logs out. When the listener logs out, a request to unregister the listener from the server by removing all the relevant entries is sent via ASP. The parameter for log out is:

\[
\text{http://abc/logout.asp? IP=ip\ address\ of\ the\ user}
\]

**Browser to Server Using ASP**

After the listener has successfully connected with the server, the browser (client) communicates with the server (Figure 2-3). The browser obtains the list of all the currently running listeners and displays the list in its window. Then the browser sends a request to the server by loading an ASP page from the server. The address of the server's ASP page is as follows:

\[
\text{http://abc/userlist.asp}
\]
Figure 2-3: Server’s ASPs entertaining the user running Browser

The server, in turn, retrieves the list of all the currently running listeners available in its database and passes the response to the browser in the form of an XML. The browser shows the list from this XML to the user after parsing it.

The server again enters the picture when the browser needs to search some file globally. In such a case, the browser sends a request to the server for the particular search by passing parameters to an ASP page. The address is as follows:

http://abc/search.asp?US="computername criteria"&FS="file search criteria"

The parameters passed to the ASP are the search criteria for the user names and the shared files. The server conducts the search by applying the search criteria passed by the browser in the database and returns the result in the form of an XML. When the user selects a name from the user list and connects to it, the browser establishes a connection with the selected listener. A direct communication is established between the listener and the browser by using the IP address from the user list. The server is bypassed for all further communication between the browser and the listener.

Listener to Browser Using XML

This communication in this application works on sockets, which uses the network stream of the system for communicating with each other. The listener and the browser software write to their respective streams and read from there. In this case, a request/response mechanism is involved. The browser sends requests in the form of XMLs to the listener by writing the request to the listener's network stream, and the listener reads from its network stream and processes the request (shown in Figure 2-4).
The response is written by the listener on the browser’s network stream over a socket connection, and the browser starts reading from its network stream to get the data the listener sends. The data can come in XML format or in standard-byte format. When XML data is received on any end, the XML parser parses that XML and sends it to its respective owner. The XML is created through the XML creator component of the system. If the browser wants to download or upload any file to the listener, it sends an XML request through the XML creator and transfers the file directly over the socket connection. The listener interprets this data and processes it accordingly.

**XML Request and Response Design Specifications**

As the first step, the software logs on to the server and is processed for authentication by the server. Authentication information is sent back to the requestor in XML format. From here after, modules communicate with each other in the form of making requests and getting the responses to them. XML mediations for communication designed for this purpose can be divided broadly into two categories: Request XMLs and Response XMLs.

*Request XMLs* are further elaborated to make different kind of requests; *Response XMLs* respond to different kinds of requests. Malfunctioning of the application at either end is signaled at the other end as an Error Response XML. Using this design and XML for communication make this application readable for modules and add-ons written in any programming language, as all the contemporary programming languages support XML.

Each request carries information in the form of attributes. This information is needed at the end where these requests are processed for responding accordingly.
Here is a search request XML, with search criteria specified as attributes of a scope child node:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<p2p_lng>
  <request type="SEARCH">
    <scope type="C:\TempDownload\*.exe" mask="1" />
  </request>
</p2p_lng>
```

Response XMLs are created according to the requests made; information here also is passed as child nodes and their attributes. Here is an example of a SHOWFILES response:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<p2p_lng>
  <response type="SHOWFILES">
    <fileinfo filename="C:\TempDownload\define1a.exe" mask="1" filesize="3072" />
    <fileinfo filename="C:\TempDownload\Define3.exe" mask="1" filesize="3584" />
    <fileinfo filename="C:\TempDownload\Ganesh.exe" mask="1" filesize="553788" />
  </response>
</p2p_lng>
```

The design of XMLs is flexible enough to incorporate any other kind of functionality to this application; you just have to add new request/response types and add-ons to process them.

### Server Design

The server for this software is used for keeping track of online and offline users and to maintain the index of the resources they share. This temporary indexing mechanism has been used to expedite the overall working of this software and to provide useful facilities to the users. Instead of making use of Web services that would have been a good option to implement server side functionality, ASPs are written considering the fact that ASPs are a more common form of server side programming.

Because virtually nothing happens on the server side during a given session of this software, ASPs are better, as they are legible to all the contemporary programming techniques. This can be said about Web services, too, as SOAP services can always be applied to Web services to make them accessible to any platform; ASPs have a shorter learning cycle than Web services, so we have adhered to ASPs for this software. By temporarily indexing the shared resources of the connected users, this server has been enabled to provide a powerful global search facility to users.

### SQL Server 2000

A few features of SQL Server 2000 are worth discussing here for you to appreciate why it has been preferred over other databases.

SQL Server 2000 offers the following features:

- **Internet Integration**: The SQL Server 2000 database engine provides integrated XML support.
- **Scalability and Availability**: The same database engine can be used across platforms, ranging from laptop computers running Microsoft Windows 98 to large, multiprocessor servers running Microsoft Windows 2000 Data Center Edition.
- **Enterprise-Level Database Features**: The SQL Server 2000 relational database engine supports the features demanded by enterprise data-processing environments. The database engine protects
Chapter 2: Designing the P2P Application

Data integrity while minimizing the overhead of managing thousands of users concurrently modifying the database.

- **Ease of Installation, Deployment, and Use:** SQL Server 2000 offers a set of administrative and development tools that improve installing, deploying, managing, and using SQL Server across several sites.
- **Data Warehousing:** SQL Server 2000 offers tools for extracting and analyzing summary data for online analytical processing. SQL Server 2000 also features tools for visually designing databases and analyzing data using English language-based questions.

Complete Database Design

In this section, the design of the database for our application is presented. The database consists of two tables: the Peer Table and the Share Table.

The structure of the Peer Table is shown in Figure 2-5 and Table 2-1.

![Figure 2-5: Peer Table](image)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ip_address</td>
<td>Stores the remote IP address of the user</td>
</tr>
<tr>
<td>user_name</td>
<td>Stores the name of the user</td>
</tr>
<tr>
<td>status</td>
<td>Stores the status of the connection (0 or 1)</td>
</tr>
<tr>
<td>connected_time</td>
<td>Stores the time of connection when the user has logged in.</td>
</tr>
</tbody>
</table>

This Peer Table in our P2P application is used to store information related to the user who is logging through the application. The Peer Table stores the remote IP address, the user name, the status of the connection, whether it is active or not. For successful connection, it stores 1. The connected time field is used to store the time of connection when the user has logged in.

When a user logs out of our P2P application, entries of that user are deleted from this Peer Table.
The Share Table structure is depicted in Figure 2-6 and Table 2-2.

![Image of SQL Server Enterprise Manager](image)

**Table 2-2: Share Table**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ip_address</td>
<td>Stores the remote IP address for the user</td>
</tr>
<tr>
<td>file_name</td>
<td>Stores the file names shared by the user</td>
</tr>
</tbody>
</table>

This Share Table in our P2P application is used to store information of the files shared by users. Field `ip_address` is used to store the remote IP address of the user who has just logged in through our P2P application. This IP address is the same as the one we have entered in the Peer Table. The list of files shared by the user is stored using the field `file_name`. Any user availing the service of our application can search these files, which are shared by the other users who are availing the services of our application.

**XML Implementation in SQL Server 2000**

SQL Server 2000 introduces features that support XML functionality. The combination of these features makes SQL Server 2000 an XML-enabled database server. These new features are the following:

- The ability to access SQL Server using HTTP
- Support for XDR (XML-Data Reduced) schemas and the ability to specify XPath queries against these schemas
- The ability to retrieve and write XML data:
  - Retrieve XML data using the SELECT statement and the FOR XML clause.
  - Write XML data using OPENXML rowset provider and UpdateGrams.
  - Retrieve XML data using the XPath query language.
- Enhancements to the Microsoft SQL Server 2000 OLE DB provider (SQLOLEDB) that allow XML documents to be set as command text and to return the result sets as a stream
You can execute SQL queries to return results in XML format rather than standard rowsets. These queries can be executed directly or from within stored procedures. To retrieve results directly, you use the FOR XML clause of the SELECT statement, and within the FOR XML clause you specify an XML mode: RAW, AUTO, or EXPLICIT. For instance, FOR XML clause is used in the code given below to obtain the list of connected users at a particular time. This code can be used to replace userlist.asp on the server which is performing the same task.

The code for userlist.xml is the following:

```xml
<?xml version='1.0' encoding='UTF-8'?>
<p2p_lng xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <response type="USERLIST">
    <sql:query>
      SELECT 1 as Tag,
      null as Parent,
      user_name as [userinfo!1!username],
      ip_address as [userinfo!1!ip]
    FROM peer
    FOR XML EXPLICIT
  </sql:query>
  </response>
</p2p_lng>
```

The XML generated by this method is directly thrown by SQL server 2000 (that is, ASP is not used to generate this XML).

This userlist.xml file runs on a virtual path named template (for better understanding on the concept of virtual path, the reader can refer to the explanation for Listing 3-4 in Chapter 3) for XML support of SQL server 2000. When this file runs on this virtual path, it communicates with the Peer Table of the P2P database and shows the user information present in the Peer Table, such as username, IP address, and so on.

Client Design

Client side is made up of two modules: the listener and the browser. The design and the role of each has already been discussed in the sections “Design Considerations for the P2P Application” and “Structure of the P2P Application,” respectively, toward the beginning of this chapter. We will now go through the process of implementation of the design laid in previous sections.

Both the listener and the browser have two versions: one written in Java and the other in C#.

Java was chosen as a language to implement these modules due to the fact that it is comparatively easy to comprehend (easier than C or C++), thus providing the programmers some respite by keeping the intricacies of networking and other concepts hidden from them. Also, being a pure OOPS based language, Java helps a lot in modularizing the tasks. Besides, Java is not only gaining popularity among programmers but is also emerging as the language of choice among hardware vendors. Almost all the new devices (handheld devices, mobile phones, tabloid computers) that hit the market come up with some degree of support for Java.

C# seems to be Microsoft’s answer to Java. With the same programming style and similar set of libraries as Java combined with native Microsoft techniques like COM and APIs, C# has a lot of potential and offers immense scope for exploration. In view of these, we decided to develop the two modules of Client in these two languages.

The foremost functionality of the two modules is to be able to communicate with each other, which they do by exchanging XML request/response documents between each other. Exchanging XML documents involves two processes: creating XML and parsing XML.
Chapter 2: Designing the P2P Application

Creating XML
This module is the generic XML creator for our application, and it handles all requests. The listener and the browser share this module for creating XML requests and responses. XML creator creates XML as indicated by the parameters sent by the browser or listener; then the XML creator writes the XML to the temporary file, which is processed accordingly by either module.

Parsing XML
This module helps the browser or listener in parsing the XML for values. This module can be called by the browser as well as by the listener. All XMLs received are passed to the XML parser for parsing, after which values are returned to the caller module. The values in these structures are used by the caller module for displaying data to the user.

Using IBM XERCES XML Parser (Java)
Modules written in Java make use of Xerces Java Parser 1.2.2, which supports XML 1.0 recommendations and contains advanced parser functionality, such as XML Schema, DOM Level 2 version 1.0, and SAX Version 2, in addition to supporting the industry-standard DOM Level 1 and APIs.

Using MS XML 3.0 component (C#)
Modules written in C# make use of the MS XML 3.0 component, which supports the World Wide Web Consortium (W3C) XML Recommendation specification. Namespace support is integrated into all aspects of the MS XML parser, including the Document Object Model (DOM) and Extensible Stylesheet Language (XSL) support.

Listener Design
Figure 2-7 shows how the listener prepares to listen.

First, the listener application logs on to the server. While logging on to the server, the listener opens share.ini (the file that keeps a record of all the files and folders shared by the listener), reads all shared files/folders, and makes a list of them. This list is then sent to the server, along with username and IP address as parameters, while calling login.asp (the ASP that handles the login process), residing on the server.

If the application is unable to make the call to login.asp, the application terminates (as it indicates some problem while establishing connection with the server and as there is no point in going further).

On the other hand, if login.asp is called successfully, it receives the response from the server and parses it to check whether the login has been authenticated. Authentication is done by imposing checks on the information passed by the listener, for duplication, like there may be a case where user is trying to log in with the name that already exists in the list of the names of connected users, in which case the application is rolled back, and the user is prompted to log on again. However, in the case of correct authentication, the listener is logged-on to the server successfully.

If desired, the listener can go for sharing resources before logging on to the server. The option for sharing resources has been given on the login screen itself, so that readers have the choice of logging in or to share the resources first.
Figure 2-7: Listener start-up flow

Start

Login

Open shared files and read all the shared files and folders and make a list of all of them and send this list along with IP and USERID as parameters while calling login asp on server

Share Resources

Yes

No

login.asp called successfully

Yes

Receive server response and parse the response, see if login was authenticated

Login Successful and no duplication is found

No

Yes

Logout

Stop

Listener wants to close

Yes

No
This option is available to the listener throughout the life cycle of this application. With this option, the listener can select any folder or file residing on the computer and can share it if it has not been shared. Attempts to share anything twice will be discarded. Figure 2-8 shows how the sharing process is carried out.

Once the listener has successfully logged on to the server, its information is entered in the list of active listeners, and this entry remains there till the listener logs out calling logout.asp.

The listener that logs in has to prepare itself to answer peers who try to connect to it and post requests. Listener has to begin listening on a common port, which all other peers use for communication under a mutual agreement. Listener does this by using the various wrapper classes that Java and C# provide.

**ServerSocket class (Java)**

This class implements server sockets. A server socket waits for requests to come in over the network. It performs an operation based upon that request and possibly returns a result to the requester.

The actual work of the server socket is performed by an instance of the `SocketImpl` class. An application can change the socket factory that creates the socket implementation to configure itself to create sockets appropriate to the local firewall.
For the purpose of our application, we make an object of `ServerSocket` class, passing the port number to listen on and call its `accept()` function to initiate the process of monitoring for incoming connection requests as the listening process starts as soon as you instantiate this class, and we are through.

**TCPListener class (C#)**

The `TCPListener` class provides TCP services at a higher level of abstraction than the `Socket` class. `TCPListener` is used to create a host process that listens for connections from TCP clients. Application-level protocols such as FTP and HTTP are built on the `TCPListener` class.

`TCPListener` class is very easy to use and provides programmers respite from almost all the intricacies of socket programming, helping them concentrate on logic.

This class has a lot in common with Java's `ServerSocket` class. Both share almost the same set of functionality. We call its `Start()` functions to initiate the process of listening, and then we call the same `Accept()` functions to monitor for incoming connection requests. As soon as a connection is accepted, the `Accept()` function returns a `Socket` class object for that connection. Communication thereafter is done on the `Socket` class object returned.

**Handling multiple connections**

As in every ideally designed application, the listener component is capable of handling multiple connections, thereby widening the scope of its application and increasing the capability of the software as a whole.

For every connection received, a thread is created to cater to the requests. Under this thread, the connection is monitored for requests, and requests received are sent for parsing to determine the type of request. Once this is done, the appropriate response is made and written on the network stream for the browser to read it from there.

Because this software aims to cater to a good number of peers simultaneously, managing peers is an important feature of this application and is a vital factor that determines the efficiency of the software as a whole. Considering this, intense care has been taken to address each parallel communication effectively.

Some techniques for managing peers are discussed in the following sections.

**Threading** is a special technique for accomplishing multitasking. The main advantage of threading is that two or more tasks that originate from the same process can run simultaneously.

In simple words, a single program can handle multiple activities simultaneously. Multiple activities in our application pertain to the listener handling multiple connections. In other applications such as MS-Word, multiple activities could be printing one document while checking it for spelling mistakes.

As soon as a connection is detected, it is forwarded to a function called in a new thread, which takes on and serves all communication on that thread.

Requests obtained from browsers are sent for parsing and for determining the type of request. Once the request type is determined, the appropriate response is written and forwarded to the browser.

The listener runs its application to accept connections to fulfill requests coming its way. Notice that regardless of the request made by the browser, the listener always processes requests to generate appropriate responses and answers the browser's requests on its network stream. The listener waits for the connection to be made. Once the connection has been made and accepted by the listener to handle the requests from the browser, the listener reads the requests on that connection and parses the request to determine the type of request made by the browser. Requests made to listener can be any of the following types:
Chapter 2: Designing the P2P Application

- Show files request for shared resources
- Search request along with search criteria
- Request for uploading files/folder
- Request for downloading files/folder

After reading and determining the request type, the listener reacts and responds accordingly.

If the request type generated by the browser is of Show File type, the listener simply responds correspondingly and returns the list of all shared resources, after which it closes the connection and starts waiting or serving the next request coming its way.

Instead of asking for the entire list of shared resources, the browser can also ask for a specific file/folder. This is usually done by sending the Search request to the listener. The browser sends the search request with criteria; in response, the listener returns the corresponding results, matching the search criteria passed to it by the browser.

Apart from a search request, an upload request can be placed to the listener if the browser is interested in giving rather than taking and is not interested in searching for any content. In such a situation, the listener first verifies whether the folder chosen by the browser for uploading has read/write permission or not. If the file/folder does not have enough credentials, the connection between the listener and the browser comes to a halt. On the other hand, if the selected file/folder has valid permission, the listener reads the file from its network stream, which is uploaded by the browser. Finally, the listener writes the file at the specified location.

Just as the browser can send requests for uploading, it can send the listener requests for downloading. In such a situation, the listener writes the requested file on the network stream for the browser to read and stores the downloaded file in the specific location. As usual, upon completing the downloading process, the connection between the listener and the browser is closed.

When a file/folder is uploaded, the browser writes and the listener reads from the stream. While downloading, the listener writes and the browser reads from the stream.

In an exceptional case in which the request type is not among those previously discussed, an error message is displayed to the user, indicating that the request type could not be resolved and that the connection has been terminated (Figure 2-9).
Figure 2-9: The Request Response flow
Searching for files and folders

Searching for the required information is one of the most prominent services this software offers its users. This application can be used on LANs such as an organization network, a university network, or a community network as successfully as it can be used on Wide Area Networks.

In these networking environments, with all the probabilities of a plethora of resources being shared by connected peers, it becomes pretty hard to search for the information you need without searching facilities. Hence, search processes receive a lot of emphasis in this software.

Once a peer is connected to another, searching for the required information from shared resources is very powerful and highly customizable in this application (see Figure 2-10). Upon connecting to a peer, users have all the search criteria at their disposal that can be used with the DIR command of MS DOS, making it a handy utility.

![Figure 2-10: The search process](image)

**Browser Design**

Figure 2-11 depicts how the client starts up gets to know the list of connected listeners and how the connection with any one of the connected listeners is established.

The browser part of this software provides the actual interface for all the activities and operations performed during the life cycle of any given session. This is a control panel for the user to leverage all the facilities that this software offers. It facilitates the user's effort to request the server for the list of listeners. Once the list is obtained, the user can select any one of them and establish connection with it.

Before you connect to any listener, you might want to see which listeners have the required information. For instance, if you are in search of the latest song of a particular group, it is possible that the listener you have approached has the songs of that group, but not the ones you want, or is not in a position to share it although it is there; worse, the listener might not have any songs of that group. To avoid such situations, you can fire a global search for the song. The list of all the listeners who have that song is at your disposal the very next moment. How this can be done is explained in the following paragraphs.
Connecting to the server and retrieving lists of connected users

As the browser initiates, the list of all the users connected at that instant is picked up from the server calling userlist.asp. If the call is successful, it provides an XML response that constitutes the list of all the users connected at that moment. If no user is connected or, in other words, if the list of users is empty, an
error message is displayed to the user, informing that at the particular instant, no user (listener) is available to listen to the request of the user.

If the user list is not empty, the list that will be in XML form will first be parsed to extracted values from it. The data or values extracted from the XML form of the list are sorted out alphabetically in accordance with the computer names of the users appearing in the list and are stored into the computer memory of the browser.

From the list, the user may select a computer to connect to or a file or folder to browse through. If a connection to a selected remote computer is made successfully, the browser picks up the list of the resources that are shared between the two ends. This list is parsed, and the values are extracted, sorted, and displayed to the user at the client end.

Calling ASP pages on the server has been made easy by a few classes C# and Java offer; such classes veil all the intricacies of making calls on the server. Some of these classes are explained in the next section.

For making calls on the server, .Net has provided a System.Net assembly that carries various useful classes and interfaces. One such class is WebRequestFactory. It is a static class that returns an instance of an object derived from WebRequest. The object returned is based on the URI scheme passed to its Create method.

The object returned this way can be type cast to the HttpWebRequest class, which contains support for the properties and methods defined in WebRequest along with additional properties and methods that enable the user to interact directly with the HTTP protocol.

By using its GetResponse() function, you can get a response for an Internet resource. GetResponse() function returns a WebResponse object that can be type cast to the HttpWebRequest class, which again is a descendent of the WebResponse class for better efficiency on HTTP protocol.

By calling the GetResponseStream() function, you can get the stream used for reading the body of the response from the server, which, in this case, is of XML format written in a temporary file for handing it over to the parsing mechanism for retrieving the list of connected users.

The java.net package in Java is the counterpart of the System.Net assembly of .Net. It also has classes with capabilities similar to those defined in System.Net, which are as suitable for our purpose as the C# classes are.

The URL class defined in this package can be used to establish connection with the concerned Internet resource. Its openConnection() function returns an object of type URLConnection whose connect() function can be used to make a call on the server and get the response. The response can then be read through its input stream obtained by calling the URLConnection's getInputStream(), which makes placing calls on the server really hassle free.

Once the connection to the server has been established and userlist.asp has been called successfully, the XML returned as the response is parsed, and the list of connected users is displayed to the user, who may choose one to connect to.

**Connecting to other peers**

Once the list of listeners is obtained, the user is likely to connect to one of them. The list provides the user with the names and IP addresses of listeners. When the user double clicks or hits the Connect button after selecting a listener from the list, a socket connection is established with the concerned listener using its IP address.

Once the connection is successfully established, the first request sent to the connected listener is to show all the shared resources. When this list is displayed on the user’s end, the user confirms that connection has been successfully made.
For communicating with different peers online, the wrapper classes provided by C# and Java are used.

This class is given as an implementation for clients’ sockets. It has functions such as `getOutputStream()`, to get the output stream to write data for the connected person to read from there, and functions such as `getInputStream()`, to get the input network stream to read the data written by the person the user connects to from there.

This class is similar to Socket but provides TCP services at a higher level of abstraction. After initializing its object, you just have to call its `Connect()` function, give the IP address of the computer to connect to and the port number to communicate and the job is done. After getting connected, you can retrieve the network stream, calling its `GetStream()` function for writing to or reading from the stream accessible by the connected machine.

.Net’s System.Net provides a static class called DNS that provides access to information from the Internet Domain Name System (DNS).

The information returned includes multiple IP addresses and aliases if the host specified has more than one entry in the DNS database.

The host information from the DNS queried by calling `GetHostByName()` is returned in an instance of the `IPHostEntry` class. This class enables you to provide a list of addresses with host names and aliases.

Java's way of resolving DNS is very similar to that of C#. Java's java.net.InetAddress package implements a very simple class called `InetAddress` for this purpose. This class represents an IP address. Applications should use the method `getLocalHost` to create a new `InetAddress` instance and to get the local host.

```java
InetAddress localHostAddress = InetAddress.getLocalHost();
```

**Downloading and uploading files**

The download procedure is initiated by the browser with the listener for downloading any file or folder (see Figure 2-12). This is the pivotal aspect of the P2P application: that files and folders get exchanged without going over to the server. Once the connection has been established, the server is bypassed and there is direct communication between the browser and the listener. The following diagram shows schematically how a user (client) downloads the file selected from the other user (listener) via the browser.

The end user (client) selects a file from the list provided by the server. The application first checks whether the selected file is available for downloading. If the file is available, details of the destination to which it is to be downloaded are determined. The file name, size, and permission are extracted from the relevant array that stores these details regarding the file.

Till this point, the user collects only the necessary information regarding the file that is to be downloaded. With these, the user determines whether or not downloading is valid; if it is, an XML request is sent to the user at the other end for downloading.
Figure 2-12: Download procedure flow

1. **START**
2. **If the file is selected**
   - **YES:** Get the selected file from the list
   - **NO:** Displays an error message
3. **NO**
4. Ask where to download that file on the local machine
5. Extract the File name and its mask value from the server
6. Make an XML request for downloading that file
7. Send the request to the Listener
8. Listener will write the file to its own stream
9. Read a block of data from that stream at a time
10. **NO:** How many bytes left for reading?
11. **YES:** Write the block to the file
12. **STOP**
Once the request is sent, the user (listener) opens the networking stream to establish communication with the user at the client's end. The user (listener) starts writing the file into the stream. Meanwhile, the browser starts reading from the same network stream.

Thus, two-way communication occurs over this network stream. The listener uploads the file and writes it into the stream, whereas the browser downloads the file and reads it from the same stream. The browser reads blocks of data and writes to the file at the specified location. This process of reading and writing continues until the entire file has been downloaded. Once no more data is to be read, the downloading process stops.

The upload procedure is also initiated by the user, with the help of the browser, for uploading the file to the other user (listener) (see Figure 2-13). As in downloading, there is no need of the server for uploading files. When the user initiates to upload a file, it is determined whether or not the selected folder or file has rights for uploading. If the file or folder selected does not possess rights for uploading, an error message is generated, and the uploading process comes to a halt.

If, on the other hand, the file/folder satisfies the rights criteria, the file/folder is processed for uploading. Then the filename is converted into a string, which represents the remote filename. The browser then generates the XML request of uploading type and sends it to the listener. Upon receiving the request, the listener opens the stream, thereby preparing for communication with the user.

The browser, after the stream has been opened, starts writing data on the stream, and the listener starts reading data from the same stream. Again, two-way communication exists between the listener and the user. But this time the browser writes and the listener reads from the same stream.

Once data starts to flow through the network stream, the listener starts reading the block of data and writing the blocks on its local machine until the entire uploading process has been executed. After the process is complete, uploading stops. Whether for uploading or for downloading, the network stream is always opened by the listener.

Downloading and uploading are the two key operations likely to be carried out more often than any other processes this application is capable of performing. How these two services are incorporated in this application is veiled in a few wrapper classes, which are explained in this section.

♦ **NetworkStream (C#):** This class provides the underlying stream of data for network access. Each time the listener receives a connection, it obtains an object of this class by calling the method `GetStream()`. Through this class, you can very conveniently write to or read from the network stream by using its `Read()` and `WriteI()` functions.

♦ **InputStream (Java):** An object of this class is obtained by calling the socket's `getInputStream()` method to read the data written by the connected user on this stream.

♦ **OutputStream (Java):** An object of this class is obtained by calling the socket's `getOutputStream()` method to write data to the stream for the connected user to read it from there.

**Searching**

When it comes to searching, this software has a lot to offer to its users. Users have two options for searching information: a local search or a global search.
Figure 2-13: The upload procedure flow

1. START
2. Does the folder have upload rights?
   - YES: Select the file to upload
     - Extract the Filename and make a string which represents the remote filename
     - Make an XML request for uploading that file
     - Send the request to the Listener
     - Listener will open a stream for the file to upload
     - Read a block from the file that needs to be uploaded
     - Write the block to the listener's network stream
     - Move blocks to follow
   - NO: Displays an error message
3. STOP
Chapter 2: Designing the P2P Application

After getting connected to a listener, you may want to search for files on your computer or to refine the list to show only a particular type of resources. This can be done by deploying a local search. Although the area to be searched for the resource in this case is restricted to the machine you are connected to, this search is powerful and works very effectively inside these demarcations. You are provided with all the search options you can use with the DIR command in DOS.

When you wish to use this option, a request XML is created with information about the target to search and the search criteria. The rest of the process takes place on the listener's side. What happens can be seen in the section titled "Listener Design" in this chapter.

This service is the most conspicuous utility in this application. Before you connect to a listener listed in the list of connected listeners, you can search for the information of your interest and its presence on these listeners. This option helps you in deciding which listener to connect to.

When you initiate this kind of search, an ASP called Search.asp is called, passing the search criteria as the parameter to this ASP. This ASP provides results in an XML format that contains information about listeners, where the concerned information is spotted, and the number of instances of information on each listener. The global search operation is depicted in the Figure 2-14.

Having discussed in detail all programming techniques, tools, intercomponent communication, and architecture, the user interface that finally presents the application to you is discussed here.

Now we are in position to appreciate what is required of the user interface for this application. Every constituent of the user interface displayed to you at various stages of the life cycle of this application has been meticulously worked out to help you derive the maximum benefit from this application. Every attempt has been made to render this interface as powerful and intuitive as possible.

Figure 2-14: Global search flow

Client: A Comprehensive View

The following sequence depicts the entire flow of the application with all possible information. In our P2P application, the listener holds the responsibility for the execution of the application. Unless the listener logs on to the server, the application cannot move ahead, as the listener caters to all requirements coming its way and initiates the requesting peer in data transfer, downloading, uploading, and searching.
An overview of the application is given here. However, technical aspects are not discussed here, as they are detailed in upcoming chapters. Before proceeding, a few points must be clarified to achieve a better understanding of the application.

Throughout the lifecycle of this application, the listener can declare desired resources (files/folders) as *shared* without logging on to the server. The sharing process can be carried out after logging on to the server as well.

Any file or folder can be declared as *shared* only for once. Once declared as *shared*, these files/folders are available to all incoming clients.

At the start up of the application, a window is displayed (see Figure 2-15) prompting the client to enter the *Login ID* in the textbox.

![Peer to Peer Login Window](image)

**Figure 2-15:** The peer to peer login window

You will find a few more buttons on the displayed window apart from Login Button (that is, Share Files/Folder, Quit, and a Checkbox named Remember My Login ID. After typing the Login ID and clicking the Login button, an icon appears in the system tray, indicating that the user has successfully logged in (see Figure 2-16). However, this tray icon will not be available in the Java version of this application.

![System Tray Icon](image)

**Figure 2-16:** This icon in the system tray indicates that you have successfully logged in.

You may declare some file or folder as *shared* before logging on to the server. For this, execute the following steps:

1. Click the Share Files/Folders button at the start up window. A Share dialog box is displayed on the screen (see Figure 2-17).
Designing the P2P Application

Chapter 2: Designing the P2P Application

There is a Checkbox on the Share File/Folder dialog box named Add this entry only, which helps you determine whether you would like to continue with sharing resources or terminate the process (when you are through with sharing a particular file/folder). By checking it, you close the dialog box and share only the currently selected file or folder; otherwise, you can continue the sharing process.

2. From the Share dialog box, choose the Share File box and click the Browse button. The Select a file to share dialog box is displayed (see Figure 2-18), from which you can choose the file for sharing.

Figure 2-17: Share Dialog allows the user to share files of folders.

Figure 2-18: Select the file you want to share.
3. From the File dialog box, choose the file you’d like to share, and click the Open button. Again, the Share dialog (refer to Figure 2-17) box is displayed, and this time you are required to assign some rights or properties to the selected file. You can do so by checking the appropriate Radio button from the Rights option field.

4. Finally, click the Share it button.

A confirmation message is shown, indicating whether the file has been successfully shared or not (see Figure 2-19).

To set any folder as shared, you need not take any step apart from those mentioned for setting any file as shared. For the convenience of all users who use this application, the steps to declare a folder as shared are as follows:

1. On the Share File Folder dialog box choose, the Share Folder tab, and click the Browse button (see Figure 2-20).

2. A window is displayed (see Figure 2-21) from which you are required to select a folder. Apart from selecting the already available folders, you can create a new folder instantly by clicking the New folder button placed on the current window. After selecting the appropriate folder, click the Open button.
3. Once again, the Share dialog window is displayed, out of which the selected folder has to be assigned some rights from the Rights option field (refer to Figure 2-20).

4. Finally, click the Share it button placed on the Share dialog window.

A confirmation message is displayed, indicating whether the folder has been successfully shared or not, just as in the case of the sharing file. If the folder has been shared already, an error message is shown (see Figure 2-22).

Once you are through with sharing and login, you are listed on the server as a connecting peer with some shared resources to distribute among users who require them. As a result of successful login on the server, a window comes up showing the list of all other peers who are already connected with the server along with their IP addresses (see Figure 2-23).
Chapter 2: Designing the P2P Application

Now an end user can establish connection with the listed peers. Any requesting peer can search for the contents of his or her interest before establishing a connection with one of the listed peers. The following steps establish such a connection:

1. Click the Search button (see Figure 2-23). A Search dialog box is displayed, containing Search on and Search for fields representing “search to be made on which peer” and “what content should be searched,” respectively (see Figure 2-24). A search can be made either by passing full phrases or by using wildcard patterns.

2. After passing the desired searching criteria, click the Search button.

3. Search results are displayed, along with the number of instances found on corresponding peers (see Figure 2-25).

In case the requesting peer does not want to go through the search option but would like to initiate a direct communication with the listed peers, the following steps suffice:

1. Select the peer name from the computer name list displayed in the window and click the Open button to start communication (see Figure 2-23).

2. Or simply double click the peer name from computer name list in the window and move forward.
Figure 2-25: The number of times the information (satisfying the search criteria you specified) was found and peers (other than you) this information was found on are listed in this window.

After selecting and opening the peer, the shared contents on the window shows all the *shared resources* of the corresponding peer (see Figure 2-26).

Figure 2-26: Resources shared by the peer you are connected with are shown in this window.

From this window, you perform the following:

1. Select any file you want to download.
2. Click the Download button.
Upon clicking the Download button, the Select Location dialog box window appears for you to mention the location at and the name with which the user wants the downloaded file to get saved on the hard disk. From the Select Location dialog box window, click the Save Button to download the shared content at the desired location (see Figure 2-27).

![File dialog box to specify the location at and the name with which the user wants the downloaded file to get saved on the hard disk.](image)

**Figure 2-27:** File dialog box to specify the location at and the name with which the user wants the downloaded file to get saved on the hard disk.

Just as content can be downloaded from a peer, it can easily be uploaded to the peer you are connected with. To do this, follow these steps:

1. Decide what you want to upload on the connected peer’s machine:
   1. Click the Upload button from the Shared contents window.
   2. The Select File dialog box is displayed.
   3. Select a file from the box, and click the Open button on the current window.

This uploads the selected file to the selected folder in the shared resource list or to the remote folder that is currently open. It is also worthwhile to mention that files can be uploaded to specific remote folders only if they are user-shared with write permission assigned (see Figure 2-28).
Once you are connected to a peer, you may find yourself in a situation in which you are looking at a list of shared resources that is too long for you to find the desired information. You can narrow down this list to those entries that concern you (see Figure 2-29). To do this, you can do the following:

1. Click the Search button. It shows you a small Search dialog box similar to the one we encounter while doing a global search, with the first Edit box disabled. The second textbox allows you to type a search criteria, for example, *.exe.

2. Click the Search button of current window (Search dialog box).

3. Results satisfying the criteria are displayed.

To quit the application, simply right-click the Application icon placed on the system and select quit from the pop-up menu (see Figure 2-30).
Chapter 2: Designing the P2P Application

A confirmation dialog box is displayed to indicate that the application has successfully logged out (see Figure 2-31). This means the application is no longer connected and is no longer on the server list.

![Confirmation Dialog Box](image)

Figure 2-31: User is notified that the listener has successfully logged out.

Summary

In this chapter, we have presented the complete design of our P2P application. The architecture of the software is such that the design is modular with features of expandability and scalability. The communication mechanism involved among the three components (server, listener, and browser) was discussed in detail. The various classes and methods used for the implementation in both Java and C# were also discussed. The user interface for the application is also described so that you get a feel for running the application.

Now that you are familiar with the design of our application, the following chapters are dedicated to its implementation aspects.
Chapter 3

Understanding the P2P Server

SQL Server 2000 (Structured Query Language Server 2000) is a novel concept in database management and data-accessing methods. Although the previous version of SQL Server (SQL 7.0) continues to challenge other RDBMS, SQL Server 2000 can claim an edge over it. SQL Server 2000 has reduced the need for lengthy programming to access data, and it addresses data in sets rather than by constructing typical tools. The language SQL Server 2000 provides to access data from the server is known as SQL, a short form of Structured Query Language. There are numerous benefits to using SQL Server 2000, some of which are the following:

♦ It translates a logical name into a linked set of physical locations, thereby reducing the need to refer to the physical location of data.
♦ It manages all locks and, as a result, eliminates the need for explicit locking statements.
♦ It identifies the most efficient method of finding the required data without specifying the index of search strategy.

As mentioned previously, SQL Server 2000 is more versatile than SQL 7.0. Apart from reducing the length of the programming required for accessing data, SQL Server 2000 has come up with a technique for accessing data over HTTP by passing URLs to it. The main reason for using SQL Server 2000 in our application is its compatibility with XML language, the feature which enhances the usability of SQL Server 2000 in Web-development environments. Various new features are incorporated in SQL Server 2000 to extend its role in Web development while supporting XML. Some of these features are the following:

♦ The ForXML clause has been equipped to the select statement for retrieval of data in XML format.
♦ With OpenXML rowset provider, SQL Server 2000 can write XML data. This enables responding in XML format.
♦ SQL Server 2000 can be accessed through URL.
♦ To set an XML document as command text, SQL Server 2000 OLEDB has been enhanced.

Creating Tables in SQL Server 2000

A table in an SQL Server is the object that contains data residing on a database. A database can have more than one table. Similarly, a table can have one or more than one attribute, commonly known as fields and columns. The structure designs of database tables used in P2P applications are shown in Figures 3-1 and 3-2.
To create tables in SQL Server 2000, simply type the name of your choice preceded with the key words "Create Table." The complete description follows:

To create tables, follow this path: "Start ➔ Programs ➔ MicrosoftSQLServer ➔ Query Analyzer".

You find the screen divided into two parts. In the upper part, start creating the table by following this example where we are creating the peer table used in our P2P database:
Chapter 3: Understanding the P2P Server

```
cREATE TABLE peer
(
ip_address CHAR(20) PRIMARY KEY,
user_name CHAR(50),
status CHAR(10),
connected_time DATETIME(8)
)
```

In the previous example, `CREATE TABLE` is the key word and `peer` is the name of the table. Notice that every table requires information such as the following: names of the columns, their data types, and column properties. These properties can convey whether or not a column allows null values and so on. While mentioning data types for a column, its length should be mentioned against them.

A key identifies and defines a specific set of one or more columns in a table that serves a specific purpose. In SQL Server 2000, two types of keys exist:

- Primary key
- Foreign key

**Primary Key**

A primary key refers to one or more columns in a table that uniquely identify its rows. There can’t be more than one primary key in any table. If you take a look at the structure of the peer table explained in the preceding section, you will notice that the `ip_address` field has a key sign preceding it, which indicates that this field is the primary key for the table it belongs to. The field can be declared as primary at the time of creation of the table by adding the key word `primary key` after it.

**Foreign Key**

When the primary key of one table appears as an attribute in another table, it is called a foreign key in the second table. A foreign key is used to relate two tables. For every value of the foreign key, there is a matching value of the primary key. Unlike the primary key, a table can contain any number of foreign keys, depending on database design and data-access requirements.

**Select Statement in SQL Server 2000**

SQL Server 2000 provides the `SELECT` statement to access and retrieve data from the server. A `SELECT` statement prompts the server to prepare a result and to return it to the client application. The key words `SELECT`, `FROM`, and `WHERE` make up the basic `SELECT` statement.

**Selecting a column**

The `SELECT` statement can be used to retrieve specific columns from the table by specifying the column names from the table. The column names specified in the `SELECT` statement are separated by a comma (,). There is no need to insert a comma after the last column name. The syntax is the following:

```
SELECT column1, column2 FROM <tablename>
```

**Selecting all columns**

The `SELECT` statement, along with the asterisk symbol (*), produces the result in the form of detailed data. The asterisk symbol, when used with the `SELECT` statement, ensures that the column listing of data is in same sequence as the one used at the time of table creation. For instance, for retrieving the records from the peer table, the statement would be:

```
SELECT * FROM peer
```
Description of the ASP Files with Flowcharts

This section explains the ASP files at the server end that enable the user to interact with the underlying database apart from bearing the logic for user authentication, resource sharing, and data integrity.

login.asp

This file is used to make entries in the peer table of the user who has just logged in to our P2P application. This file generates an XML document that communicates whether or not the logging of that user has been successful.

This program enables users to enter our server’s peer table by using our P2P application. The result of the login, whether it has been successful or not, is displayed to the user by using XML.

Flowchart description of login.asp

This flow chart (see Figure 3-3) explains how the SQL server enters the user in the peer table of the database, which is requesting for login from the P2P application. The result of login, whether it is successful or not, is in XML format.

The program fills the variables sUserID, sIPaddress, and sharestring with the values USERID, IP, and SHARE, respectively, extracted from the querystring of login.asp. It then obtains the remote IP address of the client machine from where this login.asp page has been called. Next, it checks the values of sUserID, sIPaddress, and remote-IP. If any of these values are empty, the program generates an XML response for unsuccessful login, and the program is terminated. If the values of these variables are valid, the program creates new instances of ADODB connection to establish the connection with the P2P database residing in SQL Server 2000. After that, the program executes an SQL query that selects all the records from the peer table in which the ip_address is equal to the value of remote_IP variable. If any record exists in the recordset opened with the preceding SQL query, the program deletes all records from the peer table in which the IP_address is equal to remote_IP. Then a new record is added to the peer table in which the IP_address and user_name are added, respectively, to the values of remote-IP and sUserID. The connected_time field is set to the current time. Now the program deletes all the entries of remote_IP in the share table and makes new entries for this remote_IP with the list of shared filenames in the share table. Lastly, the program generates an XML document for successful login and terminates the process.

Correct Usage

The correct syntax for login.asp is:


If the USERID parameter is missed in the preceding address, the following error XML is generated:

```xml
<?xml version='1.0' encoding='utf-8'?>
<p2p_lng>
  <response type='AUTH'>
    <connection code ="1" status="User Id can't be blank." ip="0" />
  </response>
</p2p_lng>
```
Figure 3-3: Flowchart of login.asp
Listing 3-1 contains the code for login.asp.

**Listing 3-1: login.asp**

//© 2001 Dreamtech Software India Inc.
// All rights reserved.

```vbscript
1 <%@ Language=VBScript %>
2 <!--#include file="adovbs.inc"-->
3
4 '<//Variable declaration----------//
5 dim sUserId,sIPaddress,sError,str,sharestring,count,iloop,remoteIP
6 dim conn,rsUser,rsIP,rsPeer
7 dim sqlUser,sqlIP,sqlPeer
8 dim bError,bUserExist,bIPexist
9 dim tConnectedTime
10 '<//------------------------------//
11 '<//Getting values from Query String variables-------------//
12 sUserId = Ucase(trim(Request.QueryString("USERID")))
13 sIPaddress = trim(Request.QueryString("IP"))
14 sharestring = trim(Request.QueryString("share"))
15 '<//Getting the remoteIP address of the machine--------//
16 remoteIP = trim(Request.ServerVariables("REMOTE_ADDR"))
17 '<//Check for userid, if empty initialize variables for error message-----//
18 if sUserId ="" then
19 bError=true
20 sError="User Id can't be blank."
21 end if
22 '<//Check for IPaddress and remoteIP, if both are empty then initialize variables for error message----//
23 if (sIPaddress="" and remoteIP ="")then
24 bError=true
25 sError=sError & "IP can't be blank."
26 end if
27 '<//Writing XML if either userid or IPaddress does not exist----//
28 if bError then%
29 <?xml version='1.0' encoding='utf-8'?>
30 <p2p_language
```
58 Chapter 3: Understanding the P2P Server

50 <response type='AUTH'>
51 <connection code ="1" status="<%=sError%>" ip="0" />
52 </response>
53 </p2p_language>
54
55 <%
56 Response.End
57 end if
58
59 '//-----------------------------//
60
61
62 '//Creating object for connection string--------//
63 set conn = Server.CreateObject("ADODB.Connection")
64 conn.ConnectionString = 'Provider=SQLOLEDB.1;Persist Security
65 Info=False;User ID=sa;Initial Catalog=p2p;Data
66 Source=developers"
67 conn.Open
68 '//-------------------------------------------//
69
70
71
72 '//Creating objects for recordset-------------//
73 Set rsIP= Server.CreateObject("ADODB.Recordset")
74 Set rsPeer = Server.CreateObject("ADODB.Recordset")
75 Set rsShare = Server.CreateObject("ADODB.Recordset")
76 Set rsDeleteShare = Server.CreateObject("ADODB.Recordset")
77 rsPeer.CursorType=adOpenDynamic
78 rsIP.CursorType=adOpenStatic
79 '//-------------------------------------------//
80
81
82
83 '//Writing SQL queries and open recordset--------//
84 sqlIP = "SELECT * from peer where ip_address=" & remoteIP & ""
85 rsIP.Open sqlIP,conn
86 rsPeer.open "peer",conn,adOpenDynamic,adLockOptimistic
87 '//-------------------------------------------//
88
89
90
91
92
93
94 '//Check for no. of records in both the recordsets-----//
95 if rsIP.RecordCount<>0 then bIPexist=true
96 '//-------------------------------------------//
97
98
99
100
101 '//Delete the record if IP address already exists------//
102 if(bIPexist) then
103     conn.Execute "Delete from peer where ip_address=" & remoteIP & ""

104    end if
105  '//----------------------------------------//
106
107
108
109  '//Close and destroy objects of recordsets--------//
110  rsIP.Close
111  set rsIP=Nothing
112  '//----------------------------------------//
113
114
115
116  '//Add a new record in peer table--------------//
117  rspeer.AddNew
118  rsPeer("ip_address")=remoteIP
119  rsPeer("user_name")=sUserId
120  rsPeer("status")=1
121  rsPeer("connected_time")=now()
122  rsPeer.Update
123  '//-------------------------------------------//
124
125
126
127  '//Writing XML if neither userid nor IPaddress already exists--//
128  %>
129  <?xml version='1.0' encoding='utf-8'?>
130  <p2p_language>
131    <response tag='AUTH'>
132      <connection code ="0" status="Successful" ip="<%=remoteIP%>" />
133    </response>
134  </p2p_language>
135
136  <%
137
138  '//----------------------------------------//
139
140
141
142
143  '//Delete records from share table where that IPaddress already exists--//
144  Set rsShare = conn.Execute("delete share where ip_address='" & remoteIP & 
"'")
145  '//----------------------------------------//
146
147
148
149  '//Find out the filenames from the QuerySting variable------//
150  sharestring = CStr((trim(sharestring)))
151  count = 0
152  str = split(sharestring, "*")
153  '//----------------------------------------//
154
155
156
157
158
Chapter 3: Understanding the P2P Server

Code description

♦ Lines 6-10: Variables are declared here to store the values of the username, IP address, and so on.
♦ Lines 15-17: Variables are initialized here with the values of the QueryString variables USER ID, IP, and share.
♦ Lines 22: Value of REMOTE_ADDR ServerVariable is set into remoteIP. REMOTE_ADDR returns the remote IP address of the machine requesting for the ASP page.
♦ Lines 28-31: If sUserId is empty, sError variable is initialized with an error message. It generates an error message by setting the boolean variable bError as true and setting the variable sError as User Id can’t be blank.
♦ Lines 36-39: If both sIPAddress and remoteIP are empty, sError variable is initialized with an error message. This checks if both the variables sIPAddress and remoteIP (value acquired in line 22 as remote IP address of the machine) are not blank. In case both the variables are blank, it presents an error message by setting the boolean variable bError as true and setting the value of variable sError as IP address can’t be null.
♦ Lines 46-57: If bError is equal to TRUE, XML is generated with the message for unsuccessful login, and the process ends. This code checks if the value of boolean variable bError is set to true; then an error message is generated using XML. After this, the program stops. (This is required, as both userId and IP address are null in lines 36-39).
♦ Lines 64-66: An ADODB connection object with the name conn is created here, which is used to establish the connection with the P2P database that is on SQL Server 2000.
A connection object represents an open connection to a data source. Here, the ADODB connection object is used to establish a connection with SQL Server 2000. A connection object is needed to access data using the data environment and represents a connection to a database residing on the server that is used as a data source. An alternative way of establishing the connection is through DSN. For this, replace the connection string statement and while calling the open function on the connection object, pass the DSN name as the only parameter.

```
conn.Open ("P2Pdsn")
```

Replace the P2Pdsn with the name of the DSN created by you.

♦ Lines 73-78: The new instances of the ADODB recordset are created here and set their CursorType as adopenstatic.

```
(A cursor type is a way to cache data on the client machine and to provide local scrolling, filtering, and sorting capabilities. adopenstatic is a static copy of a set of records that can be used to find data or generate reports; changes made by other users are not visible.)

- Lines 84-110: The following section checks for instances of multiple records in which the IP address is equal to the remote IP value (84-86). Eventually, our program deletes all records from the peer table having an ip_address equal to the remote IP.
  - 84-86: An SQL query is defined in the variable sqlIP, which has selected all the records from the peer table in which the ip_address field value is equal to the remote_IP value. Then the selected record is opened in the rsPeer recordset.
  - 95: If more than one record is in the rsPeer recordset, the bIPexist variable is set to true.
- Lines 102-104: If bIPexist is true, an SQL is executed that deletes all the records from the peer table in which ip_address is equal to remoteIP value.
- Lines 110-111: An instance of the recordset rsIP is closed and destroyed here.
- Lines 117-122: A new record is added in the peer table by using rsPeer recordset object in which ip_address and user_name are added with the values of the remoteIP and sUserId, respectively. connected_time field is equal to the current time.
- Lines 129-134: An XML response is generated here, having the message for successful login. These lines generate a message by using XML to inform the user that the login is successful.
- Lines 144: An SQL query is executed here that deletes all the entries from the share table where ip_address is equal to the remoteIP value.
- Lines 150-152: A data type of the sharestring variable is converted into a string type by using the CStr function. After that, the value of the sharestring variable is split on the * in that string by using the split function. The resulting values are stored in the str variable.
- Lines 160-163: The number of filenames stored in the str variable is entered into the share table, with the ip_address stored in the remote_IP variable, by executing an SQL query.
- Lines 169-170: The instance of the connection with the name conn is closed and destroyed here.

These lines help close the connection object and set the value of conn to null.

**logout.asp**

This file is used to delete the entry of a user from the peer table at the time of logout.

**Flowchart description of logout.asp**

The flowchart shown in Figure 3-4 explains how to delete the entry from the peer table of the user, requesting for logout from the P2P application.

First, the program gets the value of the IP address from the QueryString variable and assigns it to the sIPaddress variable. Next, the program checks the value of sIPaddress. If it is empty, the program terminates the process; otherwise, it creates the ADODB connection and recordset objects to establish the connection with the P2P database residing on SQL Server 2000, and also the connection with the tables of that P2P database. After that, the program executes an SQL query to delete the entry of the user from the peer table, which is requesting for logout from the P2P application. After deleting the records from the peer table, the program terminates the process.
The correct syntax of the logout.asp page is

http://p2pserver/logout.asp&IP='IP address of the machine'.

Listing 3-2 describes the code of logout.asp.

**Listing 3-2: logout.asp**

//© 2001 Dreamtech Software India Inc.
// All rights reserved

```
<%@ Language=VBScript %>
<!--#include file="adovbs.inc"-->

1
2 '<!--Variable declaration--------//
3 Dim sIPaddress
4 Dim conn
5 '<---------------------------//
6
7 '<--Getting the IP address from queryString variable----//
8 sIPaddress = trim(Request.QueryString("IP"))
9 '<---------------------------------//
10
11 '<--If sIPaddress variable is empty then end the process--//
12 if sIPaddress="" then
13   Response.End
14 end if
15 '<------------------------------------------------------------------------------------//
```
Chapter 3: Understanding the P2P Server

21
22
23 ' //Create the connection object and open it '--------------//'
24 set conn = Server.CreateObject("ADODB.Connection")
25 conn.ConnectionString = "Provider=SQLOLEDB.1;Persist Security
Info=False;User ID=sa;Initial Catalog=p2p;Data Source=developers"
26 conn.Open
27 ' //------------------------------------------------------//'
28
29 ' //Delete the records from peer table for that ip address--//'
30 conn.execute "delete from peer where ip_address='" & sIPaddress & "]
31 ' //------------------------------------------------------//'
32
33
34 ' //Close and destroy the connection object-------------------//'
35 conn.Close
36 set conn=Nothing
37 ' //------------------------------------------------------//'
38 %>

Code description

♦ Lines 6-7: Variables sIPaddress and conn are declared here. sIPaddress is for storing the value ip_address. conn is used to establish connection with the database kept on the server.

♦ Line 12: Variable sIPaddress is initialized with the value of QueryString variable IP. These lines initialize the value of variable sIPaddress with the value of QueryString variable IP retrieved from the client.

♦ Lines 17-18: The value of sIPaddress variable is checked. If it is empty, the process is terminated. These lines check whether there is a value for IPaddress; if the value of sIPaddress is null, the process of logout is terminated.

♦ Lines 24-26: An ADODB connection object with the name conn is created here, which is used to establish the connection with the P2P database on SQL Server 2000.

♦ Line 30: The connection object executes an SQL query, in which all the records in the peer table are deleted that have an IP address equal to the value of the sIPaddress variable.

♦ Lines 35-36: The established connection object conn is closed and destroyed here. These lines close the connection object conn and the value of the conn variable is set to null.

userlist.asp

This program is used to generate an XML document that shows the userlist in XML format that has the information username and the IP address from the peer table.

This code displays a list of users currently logged on to the P2P application by using XML. This userlist is displayed in XML format, showing user information including the username and the IP address from the peer table.

Flowchart description of userlist.asp

The flowchart shown in Figure 3-5 explains that the currently logged user information on the P2P application is displayed in XML format. User information includes the username and the IP address of the logged users from the peer table.
First, the program declares all the variables used in the file. Then the ADODB connection and recordset objects are created to establish the connection with SQL Server 2000 and with the peer table present in the P2P database. After that, the program executes an SQL query to select all records from the peer table of the database having IP address, user_name, and connected_time fields. All the records selected by the query are opened by a recordset object and then displayed in XML format.

The correct syntax of the userlist.asp is

```
http://p2pserver/userlist.asp
```

It does not have any parameters to be passed to it.

Listing 3-3 displays the code of userlist.asp.

**Listing 3-3: userlist.asp**

```
//© 2001 Dreamtech Software India Inc.
// All rights reserved.
1<%@ Language=VBScript %>
2 <%Option Explicit%>
3 <!--#include file="adovbs.inc"-->
4 <%
5 '//Variables declaration-----------------------//
6 dim conn,sqlPeer,rsPeer
7 dim sUserId,sIP,sString
8 '//-------------------------------//
9
10 '//Establish the connection creating connection object--------//
11 set conn = Server.CreateObject("ADODB.Connection")
```
conn.ConnectionString = "Provider=SQLOLEDB.1;Persist Security Info=False;User ID=sa;Initial Catalog=p2p;Data Source=developers"
conn.Open
'//----------------------------------------------------//
//'create a recordset object and open it with peer table--------//
Set rsPeer = Server.CreateObject("ADODB.Recordset")
rsPeer.CursorType = adOpenStatic
sqlPeer="select * from peer"
rsPeer.Open sqlPeer, conn
'//-----------------------------------------------------------//
//'Creating XML for no. of users present in the peer table------//
<%><?xml version='1.0' encoding='utf-8'?><p2p_lng><response type="USERLIST">
<% while not(rsPeer.EOF)
 sUserId=trim(rsPeer.Fields("user_name"))
 sIP=trim(rsPeer.Fields("ip_address"))
 %>  <userinfo username='<%=sUserId%>' ip='<%=sIP%>' />
<%
 rsPeer.MoveNext
 wend
%></response></p2p_lng>
'//-------------------------------------------------------//
//'Close and destroy the connection and recordset objects---------------//
rsPeer.Close
set rsPeer=Nothing
conn.Close
set conn=Nothing
Response.End
'//---------------------------------------------------------------------//
%>

Code description
♦ Lines 6-7: Variable names are declared here.
♦ Lines 12-14: An ADODB connection object with the name conn is created here, which is used to establish the connection with the P2P database on SQL Server 2000. A connection object
Chapter 3: Understanding the P2P Server

represents an open connection to a data source. Here, the ADODB connection object is used to establish a connection with SQL Server 2000. A connection object is needed to access data using the data environment and represents a connection to a database residing on server used as a data source.

♦ Lines 19-22: A new instance of the ADODB recordset is created with the name rsPeer and CursorType adopenstatic. An SQL query is defined in the variable sqlPeer, which selects all the records from the peer table. Then the selected record is opened in the rsPeer recordset.

♦ Lines 30-42: An XML document is generated here on the basis of records selected in the rsPeer recordset. The XML document has the username and IP address of the users present in the peer table.

This code displays a list of currently logged on peers to the user using XML. This user list is displayed in XML format that has the information username and IP address from the peer table using the object rsPeer of recordset.

♦ Lines 51-55: The instances of the recordset and connection created in the beginning of the file are closed and destroyed here.

These lines help in closing the connection conn and recordset rsPeer created earlier. Both of these are initialized to a null value.

userlist.xml

The other way of getting the list of currently logged in users in XML format is to configure SQL Server 2000 for XML query support. This is done by creating a virtual directory and mapping it to the server path where the XML query files (userlist.xml in this case) are kept. Now the virtual name template for this virtual directory is again mapped to the directory to which the virtual directory was mapped. For more generic discussions, refer to the SQL documentation.

http://<SERVERNAME>/<virtual_directory_name>/template/userlist.xml

Having done this when userlist.xml runs on this virtual path, it communicates with the peer table of the P2P database and shows the user information present in the peer table, such as the username, ip_address, and so on. Listing 3-4 shows the code of userlist.xml.

Listing 3-4: userlist.xml

//© 2001 Dreamtech Software India Inc.
// All rights reserved.

1 <?xml version = '1.0' encoding = 'UTF-8'?>
2 <p2p_lng xmlns:sql="urn:schemas-microsoft-com:xml-sql">
3   <response type = "USERLIST">
4     <sql:query>
5       SELECT 1 as Tag,
6       null as Parent,
7       user_name as [userinfo!1!username],
8       ip_address as [userinfo!1!ip]
9       FROM peer
10      FOR XML EXPLICIT
11     </sql:query>
12   </response>
13 </p2p_lng>
**Code description**

- Line 1: This line describes the processing of the instruction that shows that the code written in this file is an XML format.
- Line 2: There is a tag defined with the name `<p2p_lng>`.
- Line 3: A tag named `<response>` is defined here with an attribute named `type` having the value `USERLIST`.
- Line 4: The `<sql:query>` tag is defined here. This is an XML built-in tag used to define the SQL query in an XML file.
- Lines 5-10: An SQL query is defined here in explicit mode of the response. This query selects `user_name` and `IP address` from the peer table of the P2P database and shows the resulting data in XML format set by the developer.
- Line 11: The `</sql:query>` tag is defined here that indicates the end of the `<sql:query>` tag.
- Line 12: The `</response>` tag is defined here that indicates the end of the `<response>` tag.
- Line 13: The `</p2p_lng>` tag is defined here that indicates the end of the `<p2p_lng>` tag.

**search.asp**

This file is used to generate an XML response on the basis of search criteria. This XML shows the listing of the usernames and the shared file or folder names in XML format.

**Flowchart description of search.asp**

The flow chart shown in Figure 3-6 explains how the list of user names and file names shared by the users, using some match criteria, is obtained in XML format from the peer and the share tables.

First, the program gets the value of `user_name` and filenames from the Querystring variables and then assigns them to the `UserID` and filename variables, respectively. After that, the program calls the function `replacestar`, which is defined to replace `*` into `%` in the string of the filename variable, which holds the value of the search string sent by the user who has been allowed to use `*` as the wildcard. This is because `*` is a common wildcard in various forms of search utilities, including the windows file search and in many of the better known search engines. This search string thus has to be parsed to replace `*` (which is not legible to SQL queries) with `%` (which is legible). The program then creates the ADODB connection and recordset objects to establish connection with the P2P database residing in SQL Server 2000 to be able to access the peer and the share table. Next, the program executes an SQL query that selects every `user_name` from the peer table having the `userID` similar to the one specified in the search string. Then the program executes another SQL query that selects all the filenames shared by the users selected in the previous query. The output of this program is in XML format having the list of user names and the shared filenames. After throwing this XML document, the program terminates the process. Listing 3-5 contains the code for search.asp.
Figure 3-6: Flowchart of search.asp
Listing 3-5: search.asp

//© 2001 Dreamtech Software India Inc.
// All rights reserved.

```vbscript
<%@ Language=VBScript %> <!--#include file="adovbs.inc"-->

//Getting the values of variables from QueryString--------//

UserId = trim(Request.QueryString("US"))
filename = UCase(trim(Request.QueryString("FS")))

//Calling the function replacestar-------------//
filename = replacestar(filename)
UserId = replacestar(UserId)

//Definition of the function replacestar that replace "*" with "%" -------

Function replacestar(tmpstring)
    tmpstring = REPLACE(tmpstring,"*","%")
    replacestar = tmpstring
End Function

//Create and open the connection object-----------------//
set conn = Server.CreateObject("ADODB.Connection")
conn.ConnectionString = "Provider=SQLOLEDB.1;Persist Security Info=False;User ID=sa;Initial Catalog=p2p;Data Source=developers"
conn.Open

//Create and open the recordset object with peer table-----------------//
Set rsUser= Server.CreateObject("ADODB.Recordset")
Set rsFile= Server.CreateObject("ADODB.Recordset")
sqlIP = "SELECT ip_address,user_name from peer where user_name like " & UserId & ""
rsUser.Open sqlIP,conn

//Create an XML for the search result-------------------//
```
```
Chapter 3: Understanding the P2P Server

```p2p_language
<p2p_language>
<response type='SERVERSEARCH'>
<%while not(rsUser.EOF)
sqlFile = "SELECT * from share where ip_address='' & 
rsUser.Fields("ip_address") & '' and file_name like '' & filename & ''
rsFile.Open sqlFile,conn
ip = rsUser.Fields("ip_address")
uname = rsUser.Fields("user_name")
while not(rsFile.EOF)
fname = rsFile.Fields("file_name")%
<result ip="<%=ip%>" username="<%=uname%>" filename="<%=fname%>" />
<%rsFile.MoveNext
wend
rsFile.Close
rsUser.MoveNext
wend
%
</response>
</p2p_language>

```p2p_language>

Code description

♦ Lines 6-7: The values of the userId and filename variables are set with the values of the QuerySting variables US and FS, respectively.

♦ Lines 13-14: The user-defined function replacestar is called here to replace * text with % text in the string of the filename and userID variables. Resulting strings are stored in their respective variable userIDs and filenames.

♦ Lines 18-21: The function replacestar is defined here. This function stores the passed parameter’s value in the tmpstring variable. Then the built-in function REPLACE is used to replace * with % in the tmpstring. The resulting string is returned.

These lines define the function replacestar, called in lines 13-14. This function is used to replace * with %.

♦ Lines 27-29: An ADODB connection object with the name conn is created here, which is used to establish the connection with the P2P database on SQL Server 2000.

A connection object represents an open connection to a data source. Here, the ADODB connection object is used to establish a connection with SQL Server 2000. A connection object is needed to
access data using the data environment and represents a connection to a database residing on the
server used as a data source.

♦ Lines 34-37: The new instances of the ADODB recordset are created with names rsUser and
rsFile. An SQL query is defined in the variable sqlIP, which selects all the records from the
peer table in which username is similar to the value of the userId variable. Then the selected
record is opened in the rsUser recordset.

♦ Lines 46-71: An XML response is generated on the basis of the records selected in the User
recordset. An SQL query is defined in the variable sqlFile, which selects all the records from the
share table in which file_name is similar to the value of the filename variable. Then the
selected records are opened in the rsFile recordset. The search result having user names and
shared filenames is displayed in XML format.

♦ Lines 78-79: The instance of the connection created in the beginning of the file is closed and
destroyed here.

♦ These lines help in closing the connection conn created earlier and set it to a null value.

This asp has two parameters that are to be passed to it. These parameters are US and FS. The US and FS
parameters are for User Search and File Search, respectively. In the US parameter, the scope for
the computer name is passed (for example US=f* means all the computer names starting with f),
similarly the parameter FS is used. The correct syntax for the search.asp page is

http://p2pserver/search.asp?US=f*?FS=*

This will search for all the files shared by the computers with names starting with the letter f.

Summary

In this chapter, we have studied the details of server-side coding required for the full and proper
functioning of the server. The server used here has Microsoft SQL Server 2000 installed on it. You can
use any database you want and can change the ASPs accordingly for the purpose of making this system
run. This is because no special feature of SQL Server 2000 is being used as such, except for the example
userlist.xml that emphasizes the usage of XML awareness of SQL Server 2000. In addition, we have
discussed programming of ASPs in detail as we have walked the user through the process of how
different ASP programs responsible for different tasks are coded. The flowcharts in this chapter help you
understand coding details and the logical flow of the programs.
Chapter 4

The P2P Application in Java

We have selected Java to build our P2P application in this chapter. The same application can be built by using C#. Building the application in C# can meet the purpose quite well and has the advantage that the final application will be in the form of an EXE file. However, there are a few disadvantages with C#. A C# application is not available to the user as a compiled file, because not all people have the C# compiler. Moreover, such a compiler can run only with Visual Studio 6. This means that the user is required to install Visual Studio 6 so as to provide the Common Language Runtime (CLR) required to run the C# code. Of course, this situation changes as C# gains more acceptance. Because C# has the patronage of Microsoft behind it, we can reasonably expect that running C# applications without having to first compile them will become quite common in the near future. But for the time being, Java is the best option.

It is not impossible to build P2P applications using other programming languages, but Java gives us all the advantages associated with it, the major one being portability across all platforms. One argument that may slightly go against Java is performance, but the difference in performance between C# and Java is not significant. Java provides us with a fairly well-established platform with an exhaustive range of APIs. One more feature in favor of Java is that its combination with XML is considered ideal by many people because Java is a portable language and XML is portable data which can be easily ported to any platform. This makes Java and XML a perfect match.

We have used the same design for our application for both Java and C#. The architectures, as well as the programming logic, are similar. Only the implementation is different. XML is used as the data carrier in both the cases. The same project has been implemented using the Java 2Micro Edition platform. To have a look at the application, you can refer to Wireless Programming with J2ME: Cracking the Code. We discuss implementation of this application by using C# in the next chapter.

This chapter introduces you to the task of cracking code and illustrates all aspects of high-level programming. All programs carry a detailed explanation wherever needed so that you are able to understand and appreciate the code better.

The code explanation starts with a discussion of the XML classes involved in the application. The two major XML classes are SParser and xmlwriter. The Sparser class is a wrapper to the XML parser class. The name of the XML file to be parsed is to the Sparser.java class, and the XML parser is called from it. The XML parser parses the file and stores the parsed data in an object of the class Vector, which is returned to the calling program.

The xmlwriter class, on the other hand, is used to handle responses and requests to and from the browser, as well as to and from the listener, in XML format. The functions of this class take some parameters, and, depending upon the parameters, a request or response is generated.

After discussing the building blocks of our application, we go on to the application itself. This consists of a listener software and a browser software (client). These two interact with each other to give the final shape to this P2P application. The listener, as the name suggests, listens to the request sent by the browser (client) and then sends the response back to the client in XML format. The client, on the other hand, is responsible for showing the response to users.
Listener Source Code: XMLParserServer.java

The input to this Class is the name of the XML file passed as a string, and the output of the class is a Vector object containing all the information from a parsed XML file. This class acts as a wrapper class to the XML parser (it calls the XML parser and stores the results generated by the parser in a Vector). Listing 4-1 contains the source code for XMLParserServer.java.

Listing 4-1: XMLParserServer.java

// © 2001 Dreamtech Software India Inc.
// All rights reserved

1. // parses XML File
2. 3. import java.awt.*;
4. import java.io.*;
5. import java.util.*;
6. import org.xml.sax.*;
7. import org.apache.xerces.parsers.SAXParser;
8. // This is the wrapper class of the XML parser class as it calls the Xml parser and in turn when
9. the xml parser, parses the xml file it generates the call backs on the class MyContentHandler...
10. the various parsed documents can be used by using the varoius functions in this class.
11. This class returns a vector to the class which calls this class XMLParserServer.java the vector
12. consists of the data provided by the XML document.
13. 14. */
15. public class XMLParserServer
16. { 17. String attributevalue;
18. Vector value1;
19. // Function used for calling the parser it has a parameter called "uri" which has the
20. // information of the file to be parsed...
21. public void perform(String uri)
22. { 23. System.out.println("Parsing XML File : " + uri + "\n\n");
24. try
25. { 26. // Generate an object of the XMLParser class..
27. XMLReader parser = new SAXParser();
28. // Generate an Object of the MyContentHandler Class it is in this class that
29. // the xml parser generates the call backs...
30. MyContentHandler contHandler = new MyContentHandler();
31. parser.setContentHandler(contHandler);
32. // call the parse function of the XMLParser class with the file information
33. // as the parameter...
34. parser.parse(uri);
35. value1 = contHandler.returnvector();
36. }
37. catch(IOException e)
38. {
39.     System.out.println("Error reading uri : "+e.getMessage());
40. }
41. catch(SAXException e)
42. {
43.     System.out.println("Error in parsing : "+e.getMessage());
44. }
45. }
46. // This function returns the vector generated after xmlparsing is complete..
47. public Vector yakreturn()
48. {
49.     return value1;
50. }
51. }
52. }
53. class MyContentHandler implements ContentHandler
54. {
55.    private Locator locator;
56.    Vector value = new Vector();
57.    public Vector returnvector()
58.    {
59.        return value;
60.    }
61. }
62. // Only this function is used by us for our purpose....
63. public void startElement(String namespaceURI, String localName, String
64. rawName, Attributes atts) throws SAXException
65. {
66.     System.out.println("Name of the tag "+localName);
67.     System.out.println(" NO of Attributes "+atts.getLength());
68.     for(int i=0;i<atts.getLength();i++)
69.     {
70.         System.out.println("Value of i :"+i);
71.         if(atts.getValue(i) == null)
72.             {
73.                 System.out.println("Entered if Statement");
74.             }
75.         else
76.             {
77.                 System.out.println(" Attribute :"+atts.getLocalName(i)+
78.                     atts.getValue(i));
79.                     value.add(atts.getValue(i));
80.             }
81.         }
82.     }
83. }
84. if(!namespaceURI.equals(""))
85. {
86. }
87. else
88. {
89. }
Chapter 4: The P2P Application in Java

91. }
92. 
93. 
94. public void characters( char[] ch, int start , int end )
95. {
96.   //empty
97. }
98. public void startDocument() {}
99. public void endDocument() {}
100. public void endElement(String nameSpaceURI, String localName, String rawName) {}
101. public void startPrefixMapping(String prefix, String uri) {}
102. public void endPrefixMapping(String prefix) {}
103. public void ignorableWhitespace(char[] ch, int start, int end) {}
104. public void processingInstruction(String target, String data) {}
105. public void setDocumentLocator(Locator locator) {}
106. public void skippedEntity(String name) {}

Code description

♦ Lines 3-5: This portion includes the basic packages used by the various classes to build this application. The packages include java.io (file streams), java.util (various utilities), and so on.

♦ Lines 6-7: This part lists the basic packages used by the various classes of the XML parser. These packages are org.xml.sax.* and org.apache.xerces.parsers.SAXParser.

♦ Lines 15-18: This code declares a public class (XMLParserServer) that represents the base class for this file and declares the user-defined variables, which are used for various purposes during the course of the program. These include Vector-type variables into which the result of the parsed XML file is stored.

♦ Lines 21-45: The code here defines a function of the name perform. This function is used for calling the parser. It take a parameter called uri, which has the information of the file to be parsed.

• 27: This generates an object of the XMLParser class.

• 30-31: This generates an object of the MyContentHandler class. In this class, the XML parser generates the callbacks. The method setContentHandler is called, with object of MyContentHandler class as parameter. This assigns the handler to the parser.

• 34: This calls the parse function of the XMLParser class with the file information as the parameter.

• 35: The content handler’s attributes are stored in the variable called value1.

• 37-44: Exception Handling code. The code makes sure the appropriate message is shown, depending on whether the error is in parsing or reading.

♦ Lines 47-50: The code here defines a function of the name yakreturn(). This function returns a vector object value1 containing data after XML parsing.

♦ Lines 54-108: The code here pertains to the declaration of a MyContentHandler class. In this class, the callback from the XML parsing is handled and stored in appropriate data structures. For our requirements, we use a startElement function (69-77) and store the values in the Vector object, which is returned to the calling class in the end.

• 57: A new Vector object is instantiated.

• 58-61: This defines the method returnvector, which returns the Vector values.
Chapter 4: The P2P Application in Java

- 63: A method `startElement` is defined. It takes the parameters `namespaceURI`, `localname`, `rawName` and `atts` (attributes).
- 68 - 83: Values of attributes are obtained by calling the `getValue` method and are added to the `Vector` object after casting them as object. A `for` loop runs till the count becomes equal to the length of `atts` parameter.
- 94-108: This is the implementation of the methods of the `ContentHandler` interface.

**Login.java**

Login is the first class to be invoked when the application starts. This is a GUI class. The user is shown a window containing a text field for entering the user name, three buttons, and a checkbox.

- **Shared Files button**: This button is used to share files. When the user presses this button, the `Shareddialog` class is called. This contains a new window through which the user can select the file or folder to be shared.
- **Login button**: This is used to invoke the listener. Pressing this button calls `Login.asp` on the server, which makes an entry of the user name entered and the IP address of the system. This also invokes listener in a new thread by calling the `IDGMultiServer` class.
- **Cancel button**: This is used to close the application. Pressing this button calls `Logout.asp` on the server, which removes the user entry from the server.
- **The checkbox “Remember my Login ID”**: This checkbox is used to store user information. Checking the box creates a `UserInfo.ini` file, which contains the user name. If unchecked, the file is deleted (see Figure 4-1).

![Figure 4-1: The Login dialog box](image)

Listing 4-2 describes the code of `login.java`.

**Listing 4-2: Login.java**

```
// © 2001 Dreamtech Software India Inc.
// All rights reserved

/*
  * This is login class. This class is used to login. The user has to enter
  * his login name
  * and through this Frame user can open the Share frame. The ip address of
  * the System and
  * login is sent to the Server (Login.asp). When the user wants to logout
  * again the
  * information is sent to the Server (Logout.asp)
  */
```
8. import java.util.*;
9. import java.net.*;
10. import java.io.*;
11. import javax.swing.*;
12. import javax.swing.event.*;
13. import java.awt.*;
14. import java.awt.event.*;
15.
16. public class Login extends JFrame implements ActionListener
17. {
18.     // This is Constructor part of this login class
19.     public Login()
20.     {
21.         // set the title of this frame
22.         setTitle("Login");
23.         // set layout to null
24.         getContentPane().setLayout(null);
25.         // Set size to 310 and 150
26.         setSize(310, 150);
27.         // get the insets
28.         Insets insets = getInsets();
29.         // write the window closing event.
30.         addWindowListener(new WindowAdapter()
31.         { public void WindowClosing(WindowEvent e)
32.             { System.exit(0);
33.             }
34.         });
35.         //This is the label displayed just before the TextFeild
36.         l_login = new JLabel("Login ID :");
37.         //This method sets the coordinates where this label will be drawn
38.         l_login.setBounds(10 + insets.left, insets.top + 10, 60, 20);
39.         //Label is added to the Frame
40.         getContentPane().add(l_login);
41.     }
42. }
43. //This text field is for the user to enter his login name
44. login_field = new JTextField(50);
45. //This method sets the coordinates where this text field will be drawn
46. login_field.setBounds(70 + insets.left, insets.top + 10, 220, 20);
47. //TextFeild is also added to the Frame
48. getContentPane().add(login_field);
49. //*CheckBox is used to store login information
50. * If Checked UserInfo.ini file is created which will store the user name
51. */
52. c_remember = new JCheckBox("Remember my Login ID");
53. //This method sets the coordinates where this textfeild will be drawn
54. c_remember.setBounds(70 + insets.left, insets.top + 40, 220, 20);
//Registering this component for user events
register.addActionListener(this);

//Text Field is added to the Frame
register.add(c_remember);

/* This button is for sharing a file. Pressing this button will open a
filechooser dialog
* where the user can choose the files to be shared
*/
shared_files = new JButton("Shared Files");
//This method sets the coordinates where this text field will be drawn
shared_files.setBounds(0 + insets.left, insets.top + 70, 110, 40);
//Registering this component for user events */
shared_files.addActionListener(this);
//TextField is added to the Frame
register.add(shared_files);

//This button is for Login. Pressing this button will register the user
with the server
login = new JButton("Login");
//This method sets the coordinates where this textfield will be drawn
login.setBounds(111 + insets.left, insets.top + 70, 90, 40);
//Registering this component for user events
login.addActionListener(this);
//TextField is added to the Frame
register.add(login);

/*This button is for Logout. Pressing this button will make the user
logout and his entry will
* be removed from the server
*/
cancel = new JButton("Cancel");
// This method sets the coordinates where this textfield will be drawn
cancel.setBounds(202 + insets.left, insets.top + 70, 90, 40);
// Registering this component for user events
cancel.addActionListener(this);
//TextField is added to the Frame
register.add(cancel);

try {
    // Store the ip address of local host
    InetAddress localHostAddress = InetAddress.getLocalHost();
    // local ip address converted into string
    String local_address = localHostAddress.toString();
    // its local ip address seperated by "/"
    st = new StringTokenizer(local_address, "/");
    // While loop run till tokens are present
    while(st.hasMoreTokens())
    {
        // login computer name stored in this variable

Chapter 4: The P2P Application in Java

117.   machine_owner_name = st.nextToken();
118. 
119.   // login computer ip address stored in this variable
120.   ip_address = st.nextToken();
121. }
122. }
123. }
124. catch (Exception e)
125. { System.out.println("Error: " + e);
126. }
127. }
128. 
129. try
130. {
131. 
132.   String r_line = ""
133. 
134.   // Opened a Stream to read from UserInfo.ini file
135.   user_stream = new DataInputStream(new BufferedInputStream(new FileInputStream("UserInfo.ini")));
136.   
137.   // Buffered reader object instantiated to read from the stream
138.   name_in_buffer_stream = new BufferedReader(new InputStreamReader(user_stream));
139. 
140.   // This loop runs until the end of the stream
141.   while((r_line = name_in_buffer_stream.readLine()) != null)
142.     {
143.       // string divide on the basis of " = 
144.       r_st = new StringTokenizer(r_line, " = ");
145.       
146.       // while loop run till any token present
147.       while(r_st.hasMoreTokens())
148.         {
149.           // store the text "username"
150.           user = r_st.nextToken();
151.           
152.           // store the name of the user
153.           name = r_st.nextToken();
154.           
155.           // display the name of login user in textfiled
156.           login_field.setText(name);
157.           
158.           // checkbox c_remember set true
159.           c_remember.setSelected(true);
160.         }
161.       }
162.     }
163.   
164.   // close the user_stream
165.   user_stream.close();
166.   
167.   // close the name_in_buffer_stream
168.   name_in_buffer_stream.close();
169. }
170. catch (IOException e)
/* This method is used to handle the user generated events */
public void actionPerformed(ActionEvent e) {
    // The getSource method returns the object of the component which has generated an event
    Object pan = e.getSource();
    // The code in the if condition is executed if the user clicks the login button
    if (pan.equals(blogin)) {
        try {
            int count = 0;
            full_string = "";
            String search_name = "";
            // First time search the "Share.ini" file open the stream for it
            BufferedReader first_search = new BufferedReader(new FileReader("Share.ini"));
            // while loop run till first_search stream not become null
            while ((search_name = first_search.readLine()) != null) {
                count++;
                String filename = "";
                String filesize = "";
                String mask = "";
                // String divided in the basis of "="
                st_xml = new StringTokenizer(search_name, ";=");
                // while loop run till tokens present in StringTokenizer
                while (st_xml.hasMoreTokens()) {
                    // in this variable store the file name and directory
                    filename = st_xml.nextToken();
                    /* if filename string not end by "\" then enter in this condition */
                    /* otherwise go to else part */
                    if (!filename.endsWith("\\\") ) {
                        // store the rights of file
                        mask = st_xml.nextToken();
                        // Store the file size in this variable
                        filesize = st_xml.nextToken();
                        // take only file name
                        String word = filename.substring(filename.lastIndexOf("\\") + 1 ));
                    }
                }
            }
        } catch (IOException e1) {
            // TODO Auto-generated catch block
```
```
224. // store the file name with ***
225. full_string = full_string + word + "***";
226.
227.
228. }
229. else
230. {
231. // store the rights of file
232. mask = st_xml.nextToken();
233.
234. // store the directory with full path name ***
235. full_string = full_string + filename + "***";
236.
237. }
238. }
239. }
240. }
241. }
242. }
243. catch(IOException ex)
244. {
245. }
246. }
247. try
248. {
249. // store the user name which is enterd by user in textfield
250. name = login_field.getText();
251. }
252. }
253. // full_string value is null means first time *Share.ini* file does not exist
254. if (full_string.equals(""))
255. {
256. full_string = "";
257. }
258. else
259. {
260. // make one string for share.ini file list
261. full_string = full_string.substring(0, (full_string.length()- 1));
262. }
263. /* Login.asp file on the server is called and three parameterd are passed ip_address,share file
264. * and directory list
265. */
266. *
267. urlName = "http://www.s-cop.com/login.asp?USERID=" + name + "&IP=" + ip_address + "&share=" + full_string;
268. // make the object for URL and send the asp path
269. URL url = new URL(urlName);
270. }
271. // open the connection of url
272. URLConnection connection = url.openConnection();
273. }
274. // connect with url
275. connection.connect();
276.
277.    // Open a stream to read from the URL
278.    BufferedReader response_stream = new BufferedReader(new InputStreamReader(connection.getInputStream()));
279.
280.    // Character array created
281.    char[] b = new char[32];
282.    int i = 0;
283.
284.    // Make the string buffer object which stores the response sent by server
285.    StringBuffer sb = new StringBuffer();
286.    while ((i = response_stream.read(b, 0, 32)) > 0)
287.    {
288.      // character by character append in the string buffer
289.        sb = sb.append(b, 0, i);
290.    }
291.
292.    // make the object of new start class
293.    newstartclass = new StartNewClass();
294.
295.    // call the constructor of that class
296.    newstartclass.startnew();
297.
298.    }
299.  catch (Exception exception)
300.  {
301.    System.out.println("Error : "+ exception);
302.  }
303.
304.  }
305.  }
306.
307.  // if user clicks on "shared files" button then enter in this condition
308.  else if (pan.equals(bshared_files))
309.  {
310.    // make the object of Shareddilog class
311.    sh_files = new Shareddilog();
312.    // calling the function of this Shareddilog class
313.    sh_files.shared_files();
314.    }
315.
316.  // if user click on "cancel" button this code is executed
317.  else if (pan.equals(bcancel))
318.  {
319.    try
320.    {
321.      // Call the logout.asp file giving parameter the ip address of the system
322.      urlName = "http://www.s-cop.com/logout.asp?IP=" + ip_address;
323.      url = new URL(urlName);
324.      url.openConnection();
catch(Exception exception) {
    System.out.println("Error : "+ exception);
}
System.exit(0);

// if user checks "Remember my Login ID" checkbox the code in this condition is executed
else if(pan.equals(c_remember)) {
    if(c_remember.isSelected()) {
        String u_name = ";
        /* login text field is not null then enter in this condition otherwise enter in 
        * else condition */
        if((u_name = login_field.getText()) != null) {
            try {
                // open the stream of "UserInfo.ini" for writing the file
                out = new DataOutputStream(new FileOutputStream("UserInfo.ini"));
                // write in this file Byte by Byte
                out.writeBytes("username = "+ u_name);
            } catch (IOException tr) {
            } else {
                // Make the file object store the "UserInfo.ini" data into this variable
                File user_list = new File("UserInfo.ini");
                // Find the path of the file
                String actual_path = user_list.getAbsolutePath();
                // Make the object for deleting the file
                File delete_file = new File(actual_path);
                String d_file = ";
                if((d_file = login_field.getText()) != null) {
                    // delete the file
```java
386.         boolean io = delete_file.delete();
387.
388.     // The textfield is cleared (blank)
389.     login_field.setText("");
390. }
391. }
392. }
393. }
394. // The main function of the application
395. public static void main(String[] args)
396. {
397.         
398.         
399.         JFrame login = new Login();
400.         login.show();
401.         }
402. }
403. // Declaring login button
404.     JButton blogin;
405. 
406. // Declaring cancel button
407.     JButton bcancel;
408. 
409. // Declaring ShareFile button
410.     JButton bshared_files;
411. 
412. // Declaring TextFeild for entering userame
413.     JTextField login_field;
414. 
415. // Label declared
416.     JLabel l_login;
417. 
418. // declaring Checkbox
419.     JCheckBox c_remember;
420. 
421. // This string will contain the url address
422.     String urlName;
423. 
424. // Declaring StringTokenizer
425.     StringTokenizer st_xml;
426. 
427. // Declaring StringTokenizer
428.     StringTokenizer st;
429. 
430. // Declaring StringTokenizer
431.     StringTokenizer r_st;
432. 
433. // This string contains the ip address of the User’s System
434.     static private String ip_address;
435. 
436.     static private String machine_owner_name;
437. 
438. // Store the "Username"
439.     static private String user;
440. ```
442.   // Store the login name
443.       static private String name;
444.   //Declaring DatadInputStream
445.       private DataOutputStream out;
446.   //Declaring DatadInputStream
447.       private DataInputStream user_stream;
448.   //Declaring BufferedReader
449.       private BufferedReader name_in_buffer_stream;
450. 
451.   // Declaring StartNewClass which call the listener
452.       StartNewClass newstartclass;
453. 
454.   // Declaring Shareddialog which call the filechooser frame for sharing
455.       String full_string;
456. } 
457. 
458. } 
459. 
460. /* this class starts another Thread.In the application login screen runs
461. on one thread and listener
462. * runs on another Thread */
463. class StartNewClass extends Thread {
464.     public void startnew()
465.     {
466.         // call the run function
467.         start();
468.     }
469. }
470. 
471. public void run()
472. {
473.     try
474.     {
475.         // make the object for MultiServer class
476.         server = new MultiServer();
477.         // call the function of this class
478.         server.multiaccess();
479.     }
480.     }
481. 
482. } 
483. 
484. } 
485. public MultiServer server; 
486. } 

**Code description**

- Lines 8-14: Import statements. This imports basic classes needed for the application.
- Line 20: The constructor is called.
- Lines 24-33: This sets the size of the Frame (Container) and the text, which is displayed as the title of the Frame.
- Lines 36-41: This part handles window-closing events.
Chapter 4: The P2P Application in Java

- Lines 44-100: This declares GUI objects, defines the coordinates where the component is drawn on the screen, registers the component to handle user events, and draws all GUI objects on Frame.
  - 44-48: A label is instantiated, its bounds (coordinates) are set, and it is added to the content pane.
  - 52-56: A text field for login is instantiated, its bounds are set, and it is added to the content pane.
  - 61-67: A checkbox is instantiated, its bounds (coordinates) are set, and it is added to the content pane. An ActionListener is added to it.
  - 73-100: Three buttons (Shared Files, Login, and Cancel) are added in the same way as previously. An ActionListener is added to this button, too.

- Lines 102-123: A try… catch block is used to store the IP address of the local host and to convert it into a string. StringTokenizer and a while loop are used to store the login computer name and the login computer’s IP address in variables. The Code also handles any exception.

- Lines 129-172: This code reads the UserInfo.ini file and gets the user name from that file and stores it in a string, which is displayed in the textfield. This is done inside a try…catch block. A stream is opened to read from the UserInfo.ini file, and a BufferedReader reads from this stream. A StringTokenizer and a while loop are used to divide the string and store the user name. The name of the login user is displayed in the textfield, and the state of the checkbox c_remember is set to true.

- Lines 175: This method is used to handle action events usually generated when the user clicks some button. Any event generated invokes this method.

- Lines 181-305: This if condition is true when the user presses the Login button. In this if condition, the Share.ini is read, and the entries are appended in a String object. This String object is sent as the third parameter to Login.asp. The other two parameters are IP address and user name. If the Shared.ini does not exist, the the String is blank. This also invokes the listener by instantiating a MultiServer class.
  - 183-245: This is a try…catch block, inside which the Share.ini file is searched and a stream is opened for it. Then a StringTokenizer inside a while loop is used to divide the string. The rights of the file, the file size, the file name, and so on are stored.
  - 247-302: This try…catch block stores the user name entered in the textfield. It then checks whether the full_string has a null value. If it has, the Share.ini file doesn’t exist. The Login.asp file on the server is called and three parameters are passed to it — the IP address, the share file, and the directory list. A connection is opened with the URL, and a stream is opened to read from the URL. A new start class object is created.

- Lines 308-315: This if condition is true when the user presses the Shared File button. In this if condition, the Shareddialog class shared_files method is called, which will generate the Shareddialog frame.

- Lines 318-338: This if condition is true when the user presses the Cancel button. In this if condition, Logout.asp is called and the IP address is passed as a parameter, which removes the entry from the server.

- Lines 341-392: This if condition is true when the user clicks the checkbox. If the checkbox is checked, the user name is entered in the UserInfo.ini file; if it is unchecked, the UserInfo.ini file is deleted.

- Lines 396-458: This is the main method. In this method, the login class is instantiated and objects are declared.

- Lines 464-486: This is an inner class and invokes the listener in a new thread. This class calls the object of the class MultiServer.
Shareddilog.java

This class is used for sharing files and folders and is GUI class (see Listing 4-3). The window contains five buttons, a textfield, two radio buttons, and one checkbox.

♦ **ShareFile Button**: The user presses this button if he or she wants to share files only. After pressing this button, if the user presses the Browse button, the file is displayed (see Figure 4-2).

♦ **ShareFolder Button**: The user presses this button if he or she wants to share folders only. After pressing this button, if the user presses the Browse button, folders are displayed.

♦ **Browse Button**: The user presses this button to select the files and folders to be shared. Pressing this button displays a file chooser dialog box from which the user can select the file or folder.

♦ **Shareit Button**: Pressing this button makes the file and folder entries in Share.ini file. If the Share.ini file does not exist, this creates it.

♦ **Close Button**: This button closes the window.

♦ **TextField**: The textfield displays the full path of the files or folder selected.

♦ **Radio Buttons**: There are two radio buttons. One is read only. If this radio button is checked, it means that the shared file is given a ReadOnly permission. The second radio button is Read/Write; if this button is checked, it means that the shared file is given Read/Write permission.

♦ **CheckBox**: If the checkbox “Add this entry only” is checked, after pressing Share button, the window closes. The user doesn't have to press the Close button.

![Share Dialog](image)

Figure 4-2: Share file/folders

**Listing 4-3: Shareddilog.java**

//© 2001 Dreamtech Software India Inc.
// All rights reserved

1. import java.util.*;
2. import java.awt.*;
3. import java.awt.event.*;
4. import java.io.*;
5. import javax.swing.*;
6. import javax.swing.event.*;
7. import javax.swing.border.*;
8. /*
9. * This class mainly used for sharing files and directories for user to use. Through this class
10. */
Chapter 4: The P2P Application in Java

11. * the listener can add the files and directory. The major disadvantage is
12. * cannot remove the already shared file/folder.
13. */
14.
15. class Shareddilog extends JDialog implements ActionListener
16. {
17.  public void shared_files()
18.  {
19.      // Create an object of the class JDialog
20.      sharedilog = new JDialog();
21.      // Set the Layout as null then draw the gui components according to our
22.      need
23.      sharedilog.getContentPane().setLayout(null);
24.      sharedilog.setTitle("Share Dialog");
25.      // Adding the window Listener to the gui window
26.      // ie. Code for the "cross"
27.      addWindowListener( new WindowAdapter()
28.          { public void windowClosing(WindowEvent e)
29.              {
30.                dispose();
31.              }
32.          });
33.      share_file = new JButton("Share File"); // Initializing the GUI
34.  }
35.      Component.
36.      share_file.setBounds(10,7,100,20); // Positioning the GUI Component.
37.      share_file.addActionListener(this);// Add action listener
38.      sharedilog.getContentPane().add(share_file); // Adding the GUI
39.      Buttons....
40.      share_folder = new JButton("Share Folder"); // Initializing the GUI
41.      Component.
42.      share_folder.setBounds(110,7,150,20);// Positioning the GUI Component.
43.      share_folder.addActionListener(this);// Add action listener
44.      sharedilog.getContentPane().add(share_folder); // Adding the GUI
45.      Buttons.....
46.      l_type = new JLabel(); // Initializing the GUI Component.
47.      l_type.setText("Type a filename here or click browse to select");
49.      sharedilog.getContentPane().add(l_type);// Adding the GUI Component
50.      t_type = new JTextField(150); // Initializing the GUI Component
51.      t_type.setBounds(15,60, 380, 20);// Positioning the GUI Component.
52.      sharedilog.getContentPane().add(t_type);// Adding the GUI Component
53.      browse = new JButton("Browse"); // Initializing the GUI Component
54.      browse.setBounds(315,85, 80, 20);// Positioning the GUI Component.
55.      browse.addActionListener(this);// Add action listener
56.      sharedilog.getContentPane().add(browse); // Adding the GUI Component
Chapter 4: The P2P Application in Java

60.  c_entry = new JCheckBox("Add this entry only"); // Initializing the GUI Component
61.  c_entry.setBounds(275,110, 150, 20); // Positioning the GUI Component.
62.  sharedilog.getContentPane().add(c_entry); // Adding the GUI Component
63.  
64.  // When this button is pressed then the information regarding the file/folder shared will be added on to shard.ini file.
65.  shared_it = new JButton("Share it"); // Initializing the GUI Component
66.  shared_it.setBounds(250,165, 80, 20); // Positioning the GUI Component.
67.  shared_it.addActionListener(this); // Add action listener
68.  sharedilog.getContentPane().add(shared_it); // Adding the GUI Component
69.  
70.  // This button for close the Dialog
71.  close = new JButton("Close"); // Initializing the GUI Component
72.  close.setBounds(340,165, 80, 20); // Positioning the GUI Component.
73.  close.addActionListener(this); // Add action listener
74.  sharedilog.getContentPane().add(close); // Adding the GUI Component
75.  
76.  // This RadioButton to give read only permission to the file
77.  read_only = new JRadioButton("Read only", true); // Initializing the GUI Component
78.  read_only.setBounds(10,140,80,20); // Positioning the GUI Component.
79.  sharedilog.getContentPane().add(read_only); // Adding the GUI Component
80.  
81.  // This RadioButton to give the read & write permission to the file
82.  read_write =new JRadioButton("Read/Write", false);
83.  read_write.setBounds(10,165,80,20); // Positioning the GUI Component.
84.  sharedilog.getContentPane().add(read_write); // Adding the GUI Component
85.  group = new ButtonGroup();
86.  
87.  group.add(read_only); // Add the radiobutton to the radiobuttongroup
88.  group.add(read_write); // Thereby enabling only one button at a time.
89.  
90.  // Make vector to read Share.ini file and insert into it.
91.  v_file_list = new Vector();
92.  
93.  try
94.  {
95.    // String r_line iniatialize as null;
96.    String r_line = "";
97.    
98.    // Open a DataInputStream to the file "Share.ini" for Reading the data from the file.
99.    data_in = new DataInputStream(new BufferedInputStream(new FileInputStream("Share.ini")));
100.   // Open a bufferedreader to the file "Share.ini" for Reading the data from the file.
101.   data_buffer_in = new BufferedReader(new InputStreamReader(data_in));
102.   
103.   // Reading the file buffer
104.   while((r_line = data_buffer_in.readLine()) != null)
105.     {
106.       v_file_list.add(r_line);
107.     }
108.  }
// close DataInputStream
data_in.close();

// close the input Buffer
data_buffer_in.close();

} catch (IOException e) {}
165. // Readonly right means the mask is set to 0;
166. if (read_only.isSelected())
167. {
168.     r_rights = "0";
169. }
170. // readwrite right means the mask is set to 1;
171. else if (read_write.isSelected())
172. {
173.     r_rights = "1";
174. }
175. } // show the JFileChooser
176. int pp = fileselection.showOpenDialog(this);
177. if (pp == 0) {
178.     // selected the current file or directroy
179.     file_list = fileselection.getSelectedFile();
180.     // Find the length of the file
181.     lengthoffile = file_list.length();
182.     // Display the text on to the JTextField..
183.     t_type.setText(file_list.toString());
184. }
185. try {
186.     s_line = "";
187.     // Open a DataInputStream to the file "Share.ini" for Reading the data from
188.     // the file.
189.     data_in = new DataInputStream(new BufferedInputStream(new
190.     FileInputStream("Share.ini")));  
191.     // Open a Inputbuffer to the file "Share.ini" for Reading the data from
192.     // the file.
193.     data_buffer_in = new BufferedReader(new InputStreamReader(data_in));
194.     // value_all_ready_present = false;
195.     while((s_line = data_buffer_in.readLine()) != null) {
196.         // Divide String on the basis of "=
197.         st = new StringTokenizer(s_line, "=");
198.         // while loop run tile token present
199.         while(st.hasMoreTokens()) {
200.             if (st.nextToken().equalsIgnoreCase(file_list.toString()) == true )
201.                 value_all_ready_present = true;
202.             }
203.         }
204.         first_time_entry = true;
205.         while((s_line = data_buffer_in.readLine()) != null) {
206.             // Divide String on the basis of ";=
207.             st = new StringTokenizer(s_line, "=");
208.             // while loop run tile token present
209.             while(st.hasMoreTokens()) {
210.                 if (st.nextToken().equalsIgnoreCase(file_list.toString()) == true )
211.                     value_all_ready_present = true;
212.             }
213.         }
214.         }
215.         // value_all_ready_present = true;
216.         Box b = Box.createVerticalBox();
b.add(Box.createGlue());
b.add(new JLabel("This file is all ready exists"));
getContentPane().add(b, "Center");
setSize(180, 100);
setVisible(true);
JPanel p2 = new JPanel();
// Press the "OK" button then close the messagebox
JButton ok = new JButton("OK");
p2.add(ok);
getContentPane().add(p2, "South");
ok.addActionListener(new ActionListener()
{
  public void actionPerformed(ActionEvent evt)
  {
    setVisible(false);
  }
});
System.out.println("This file is already exists");
else
{

}
// close DataInputStream
data_in.close();
// close the input Buffer
data_buffer_in.close();
catch (IOException ex)
{}

// if files and directories are not present in share.ini
if(value_all_ready_present == false || first_time_entry == false)
{
// Check whether file_list is directory or file
if (!file_list.isDirectory())
{
// Add Files list in vector
v_file_list.add(file_list.toString() + "=" + r_reights + "=" + lengthoffile);
}
else
{
    // Add directories list in vector
    v_file_list.add(file_list.toString() + "\=" + r_reights );
}
}
else if(source == shared_it) // When share button is pressed.
{
try
{
Chapter 4: The P2P Application in Java

272. // Open a DataOutputStream to the file "Share.ini" for Reading the data from the file.
273. data_out = new DataOutputStream(
274. (new FileOutputStream("Share.ini")))
275. 
276. for (int t = 0; t < v_file_list.size() ; t++)
277. {
278. // Write the all data present in the vector byte-by-byte in the Share.ini file
279. data_out.writeBytes(v_file_list.get(t) + \"\n\");
280. }
281. // Close the object of DataOutputStream
282. data_out.close();
283. }
284. catch (IOException ex)
285. {
286. 
287. if (c_entry.isSelected()) // When add this file only checkbox is checked
288. {
289. // Close the share dialog box
290. sharedilog.dispose();
291. }
292. }
293. }
294. }
295. }
296. else if(source == close) // Close button is pressed.
297. {
298. 
299. sharedilog.dispose();
300. }
301. }
302. }
303. }
304. 
305. JFileChooser fileselection;
306. JButton share_file;
307. JButton share_folder;
308. JLabel l_type;
309. JTextField t_type;
310. JButton browse;
311. JCheckBox c_entry;
312. JButton shared_it;
313. JButton close;
314. JRadioButton read_only;
315. JRadioButton read_write;
316. ButtonGroup group;
317. static boolean b_cho_f = false;
318. static boolean b_cho_d = false;
319. static boolean first_time_entry = false;
320. static boolean value_all_ready_present = false;
321. DataOutputStream data_out;
322. DataInputStream data_in;
323. BufferedReader data_buffer_in;
324. StringTokenizer st;
325. // list of files
static File file_list;
// Store the length of files
long lengthoffile;
// Store the list all files which are all ready shared
Vector v_file_list;
// Give the rights to the file and Directroy, "0" for read only and "1" for read and write
String r_reights;
// Main Dialog box
JDialog  sharedilog;
String s_line;

Code description
♦ Lines 1-7: Import statements to import basic classes for the application.
♦ Line 18: shared_files method declared.
♦ Lines 21-24: The Jdialog class is instantiated and the title of the dialog is set, which is displayed as the title of the window.
♦ Lines 28-33: This handles the window-closing event.
♦ Lines 34-91: This code declares and initializes the components. The components inset are set. This sets the coordinates where the Component is drawn and also registers these components to handle user events and the Component is then added to the container.
♦ Lines 94-125: This appends the data in the Share.ini file.
♦ Line 129: This method handles the events generated on the GUI components.
♦ Lines 134-139: This if condition is executed if the user presses the ShareFile button.
♦ Lines 141-145: This if condition is executed if the user presses the ShareFolder button.
♦ Lines 146-266: This if condition is executed if the user presses the Browse button. This opens a file-chooser dialog box from which the user can select the files to be shared. Every entry is stored in the Vector with full path, permissions, and size.
• 154-164: File selection mode is set depending on which of the b_cho_f and b_cho_d variables is true.
• 168-175: File permissions are checked and the mask is set to 0 or 1 accordingly. Setting the mask to 1 means read and write permission.
• 180-187: The currently selected file or directory is found out, and its size is obtained with getLength method. The text is displayed on the text field.
• 196-247: A stream is opened to read from the file Share.ini. It is divided up on the basis of “=” by using a StringTokenizer. The OK button is created, and an ActionListener is attached to it. The stream is closed when the while loop responsible for reading runs out.
• 251-266: This checks whether files and directories are not present in Share.ini file. If they are not, they are added to Vector.
♦ Lines 268-294: This if condition is true if the user presses the Share It button. In this case, the values from the Vector are read and written in Share.ini.
♦ Lines 296-302: This if condition is true if the user presses the Close button. It also disposess of the dialog box.
♦ Lines 305-335: Objects and variables declaration.
MultiServer.java

This class handles the client requests such as show file, download file, Search files and directories, and upload (see Listing 4-4). To read the request and to respond to clients, this class has to parse and write an XML file. For parsing, it calls the Sparser class and for writing it calls the XMLWriter class. For searching requests, this class calls the check_directory class.

Listing 4-4: MultiServer.java

//© 2001 Dreamtech Software India Inc.
// All rights reserved

1. import java.io.*;
2. import java.net.*;
3. import java.util.*;
4. 
5. /*  This Class is the listener class of the application. This class handles
6. * the request from clients
7. * the class extends Thread class because any new request runs in a new
8. * thread 
*/
9. class  OneServer extends Thread {
10.     // Make object for client socket
11.     private Socket socket;
12.     
13.     // Make object for tokenizer
14.     private StringTokenizer st_xml;
15.     
16.     // Make object for parser
17.     static  SParser sp;
18.     private BufferedOutputStream data_write;
19.     
20.     // Return files and directory names.Return the file size and rights of
21.     String[][] returnvalueoffiles;
22.     
23.     // Vector Declared
24.     static Vector v_file_name;
25.     //This class is used to write xmldata
26.     private XmlWriter xmlwriter;
27.     
28.     private check_directory check;
29.     
30.     //Declaring BufferedInputStream
31.     private BufferedInputStream in;
32.     
33.     // Declaring BufferedOutputStream
34.     private BufferedOutputStream out;
35.     
36.     // This String will contain the FileName
37.     String left_half;
38.     
39.     // Store right part of the file it means after "."
40.     String right_half;
41.     
42.     // Store detail path name of file & directories in file object
43. File path_file;
44.
45. // Store files and directories names in string array
46. String[] files;
47.
48. /* when search any word if any character present in left of file extension then in that
49. * condition this flag becomes true */
50. boolean left_half_flag = false;
51.
52. /* when search any word if any character present in right of file extension then in that
53. * condition this flag becomes true */
54. boolean right_half_flag = false;
55.
56. // file and directory stored in this variable
57. String filename;
58.
59. // file size stored in this variable
60. String filesize;
61.
62. // file rights stored in this variable
63. String filemask;
64.
65. String stemp = "";
66.
67. /* from this function listener takes request from client and according to the
68. * request listener gives the response */
69. public OneServer(Socket soc) throws IOException
70. {
71.
72. // store the socket value which is connected by user
73. socket = soc;
74.
75. // take the request from the user in the form of BufferedInputStream
76. in = new BufferedInputStream(socket.getInputStream());
77.
78. // give the response to the user from listener in form of BufferedOutputStream
79. out = new BufferedOutputStream(socket.getOutputStream());
80.
81. // start the new thread for new user
82. start();
83. }
84.
85.
86. // when any new user connects then this function call by start()
87. public void run()
88. {
89. try
90. {
91. int i = 0;
92. String value;
93. byte[] a = new byte[1024];
95. // open main.xml file as BufferedOutputStream for writing data into it
96. data_write = new BufferedOutputStream(
97. (new FileOutputStream("main.xml")));
98.
99. // read data from user DataInputStream
100. in.read(a,0,1024);
101.
102. // Store reading value in temp String
103. String temp = new String(a);
104.
105. // Break the temp till last ">" + 1 value
106. temp = temp.substring(0,(temp.lastIndexOf("">")+1));
107.
108. // convert temp string value in Byte
109. byte d[] = temp.getBytes();
110.
111. // write converted value in "main.xml" file
112. data_write.write(d, 0, d.length);
113.
114. // Close the data_write stream
115. data_write.close();
116.
117. // Make the object of XmlWriter class
118. xmlwriter = new XmlWriter();
119.
120. // Make the object of SParser class
121. sp = new SParser();
122.
123. // call the function of class SParser and pass the argument as string
124. sp.perform("main.xml");
125.
126. // This function returns the file names as vector
127. v_file_name = sp.yakreturn();
128.
129. // This for loop runs the size of the vector
130. for(int t = 0; t < v_file_name.size(); t++)
131. {
132.
133. // vector value stored is showfiles type then enter in this condition
134. if((v_file_name.get(t).toString()).equalsIgnoreCase("SHOWFILES"))
135. {
136.     String s = ";
137. 
138. // open stream of "Share.ini" file for reading
139.     BufferedReader data_read = new
140.     BufferedReader(new FileReader("Share.ini"));
141.
142. // This while loop runs till stream "data_read" not become null
143. while ((s = data_read.readLine()) != null)
144. {
145.     String filename = "";
146.     String filesize = "";
147.     String mask = "";
149. // divide string "s" on the basis of "=" and store in st_xml
150. st_xml = new StringTokenizer(s, ";");
151.
152. // This while loop runs till that tokenizer present in st_xml
153. while(st_xml.hasMoreTokens())
154. {
155. // Here store first token in variable filename
156. filename = st_xml.nextToken();
157.
158. /* This filename string ends with "\" then enter in this part otherwise enter
159.  * in else part. if filename string ends with "\" it means that is directory there
160.  * otherwise that is file */
161.  if( !filename.endsWith("\\") )
162.   {
163. // Here store second token in variable mask. it is rights of file
164.   mask = st_xml.nextToken();
165. 
166. // Here store third token in variable filesize. it is writes of file
167.   filesize = st_xml.nextToken();
168. 
169. /* Call the returnHeader function from XmlWriter class this function
170.  * returns the header of xml file as string and stores this value in stemp variable. */
171.   stemp = xmlwriter.returnHeader(v_file_name.get(t).toString());
172.   
173. /* Call the responseFString function from XmlWriter class this function
174.  * writes the xml file for files. */
175.   xmlwriter.responseFString(v_file_name.get(t).toString(), filename, filesize, mask);
176. }
177. else
178. {
179. // Here store second token in variable mask. it is rights of file
180.   mask = st_xml.nextToken();
181.
182. /* Call the returnHeader function from XmlWriter class this function return
183.  * the header of xml file as string and store this value in stemp variable. */
184.   stemp = xmlwriter.returnHeader(v_file_name.get(t).toString());
185. 
186. /* Call the responseFString function from XmlWriter class this function
187.  * writes the xml file for Directroy. */
188.   xmlwriter.responseFString(v_file_name.get(t).toString(), filename, "", mask);
189. }
190. }
191. }
/* Call the returnResponse function this function returns whole xml except header of xml as */
String wholexmlwithoutheader = xmlwriter.returnResponse();

/* Add two string variable and store in any third string variable. This variable stores whole xml file */
wholexmlwithoutheader = stemp+wholexmlwithoutheader;

// Find the length of xml file and send 0 to length of file xml file bytes to user
out.write(wholexmlwithoutheader.getBytes(),0,wholexmlwithoutheader.length());

// Close the data_read stream which read from file.
data_read.close();

// Close the out stream which connected to user.
out.close();

// In this condition we do all download work related to user request
else if((v_file_name.get(t).toString()).equalsIgnoreCase("DOWNLOAD"))
{

// Store the file name in variable f_name. This file downloaded by user
String f_name = v_file_name.get(1).toString();

// initilize the variable of len
int len = 0;

// Open the file stream of stored file name which is present in f_name.
FileInputStream fstream = new FileInputStream(f_name);

// Make variable c_write as Byte array which is sent to user
byte[] c_write = new byte[32];

// While loop run upto 32 Byte of all stored array value
while ((len = fstream.read(c_write,0,32))>0)
{

// Send the out stream to user every 32 Byte
out.write(c_write, 0, len);
}

// Close the out Stream
out.close();

}
/* Make the object of check_directory which search file and directory
 * which is requested by the user */
check = new check_directory();

// Store file & directory name with path in whole_string variable
String whole_string = v_file_name.get(1).toString();

// Store file & directory path in full_path
String full_path = whole_string.substring(0,
whole_string.lastIndexOf("\") + 1));

// Store file & directory name without path in word variable
String word = whole_string.substring((whole_string.lastIndexOf("\") + 1));

// Make file object of file which path present in full_path
path_file = new File(full_path);

// Find the position of "." in file
int dot_index = word.indexOf('.');

/* When "." not present in file then return -1 then enter in this condition
 * otherwise enter */
if (dot_index == -1) {
    // whole word value store in left_half variable
    left_half = word;
    // write_half variable value become blank(" ")
    right_half = " ";
    // Find the position of "*" in left_half variable and store that variable
    // in asterix_index
    int asterix_index = left_half.indexOf("*");
    // if left_half variable not content any "*" then its return -1 then not
    // enter in this condition
    if (asterix_index != -1) {
        /* Store value in left_half first position to "*" position when check the
         * left_half_flag
         * flag to true */
        left_half = left_half.substring(0,asterix_index);
        left_half_flag = true;
    }
    /* Store value in left_half first position to "*" position when check the
     * left_half_flag
     * flag to true */
    left_half = left_half.substring(0,asterix_index);
    left_half_flag = true;
}
else {
    // Store file name beginning to "." position left part
    left_half = word.substring(0,word.indexOf('.'));

    // Store file name last to "." position of right part of that file
    right_half = word.substring(word.indexOf('.') + 1);
300. // left_half is equal to "**" or left_half is equal to "" then enter in this condition
301. if ((left_half.equals("**"))||(left_half.equals("")))
302. {
303.  // left_half string value initialize by null(" ")
304.  left_half = " ";
305. }
306. else
307. {
308.  // Find the position of "*" in left_half variable and store that variable in asterix_index
309.  int asterix_index = left_half.indexOf("*");
310.  // if left_half variable not contain any "***" then its return -1 then not enter in this condition
311.  if (asterix_index != -1)
312.  {
313.    // Store value in left_half first position to "***" position when check the left_half_flag
314.    * flag to true */
315.    left_half = left_half.substring(0,asterix_index);
316.    left_half_flag = true;
317.  }
318.  }
319. }
320.  // right_half is equal to "**" or right_half is equal to "" then enter in this condition
321. if ((right_half.equals("**"))||(right_half.equals("")))
322. {
323.  // right_half string value initialize by null(" ")
324.  right_half = " ";
325.  }
326. else
327. {
328.  // Find the position of "***" in right_half variable and store that variable in asterix_index
329.  int asterix_index = right_half.indexOf("***");
330.  // if right_half variable does not contain any "***" then do not enter in this condition
331.  if (asterix_index != -1)
332.  {
333.    // Store value in right_half first position to "***" position when check the right_half_flag
334.    * flag to true */
335.    right_half = right_half.substring(0,asterix_index);
336.    right_half_flag = true;
337.  }
338.  }
339.  }
340.  // Store files name which are present in this path_file in files array
341.  files = path_file.list();
342.  }
343.  }
344.  }
345.  // Store files name which are present in this path_file in files array
346.  files = path_file.list();
347.  }
// make object of String array which contains files & directories name filesize and mask
returnvalueoffiles = new String[files.length + 1][3];

// Store all values in returnvalueoffiles array which return by wild_card function
returnvalueoffiles =
    check.wild_card(left_half,right_half,left_half_flag,right_half_flag,
    path_file);

for(int y = 0; y < files.length + 1; y++)
{
    // data in this array returnvalueoffiles[y][0] not present in this then break the loop otherwise go to else part
    if(returnvalueoffiles[y][0] == null)
    {
        break;
    }
    else
    {
        // Store full path with file & directories name in filename
        filename = path_file + "\" + returnvalueoffiles[y][0];

        // Store size of file
        filesize = returnvalueoffiles[y][1];

        // Store the rights of the files
        filemask = returnvalueoffiles[y][2];

        /* Call the returnHeader function from XmlWriter class this fuction returns the header of xml file as string and stores this value in stemp variable. */
        stemp = xmlwriter.returnHeader("SHOWFILES");

        /* Call the responseFString function from XmlWriter class this fuction writes the xml file for Directroy. */
        xmlwriter.responseFString("SHOWFILES", filename, filesize, filemask);
    }
}

String wholexmlwithoutheader = "";

/* data in this array returnvalueoffiles[0][0] not present in this then enter in this condition it means there is no file and diretory otherwise go to else part of this condition*/
if(returnvalueoffiles[0][0] == null)
{

396.  // Make one xml file without any files & directories list
397.  stemp = xmlwriter.returnHeader("SHOWFILES");
398.  wholexmlwithoutheader = "</response></p2p_lng>*;
399.  }
400.  else
401.  {
402.  /* Call the returnResponse function this function returns whole xml
except header
403.   * of xml as string. Store this value in wholexmlwithoutheader variable */
404.  wholexmlwithoutheader = xmlwriter.returnResponse();
405.  }
406.
407.  /* Add two string variables and store in any third string variable. This
variable stores whole
408.   * xml file */
409.  wholexmlwithoutheader = stemp + wholexmlwithoutheader;
410.
411.  // Find the length of xml file and send 0 to length of file xml file
bytes to user
412.  out.write(wholexmlwithoutheader.getBytes(0), 0, wholexmlwithoutheader.length());
413.
414.  // Close the data_read stream which read from file.
415.  out.close();
416.
417.  }
418.
419.
420.  // vector value store is upload type then enter in this condition
421.  else if((v_file_name.get(t).toString()).equalsIgnoreCase("UPLOAD"))
422.  {
423.   // Store file & directory name with path in whole_string variable
424.   String upload_name = v_file_name.get(1).toString();
425.
426.  }
427.  // initialize the variable len
428.  int len = 0;
429.
430.  // Make variable c_write as Byte array which is sent to user
431.  byte[] c_write = new byte[32];
432.  // open stream of upload_name file for writing
433.  data_write = new BufferedOutputStream(
434.   (new FileOutputStream(upload_name)));
435.  // While loop run upto 32 Byte of all stored array value
436.  while ((len = in.read(c_write, 0, 32)) > 0)
437.  {
438.    // Send the out stream to user every 32 Byte
439.    data_write.write(c_write, 0, len);
440.  }
441.  // Close the data_write stream
442.  data_write.close();
443.
444.  }
448.  )
449.  )
450.  }  
451.  catch (IOException e)
452.  {  
453.      System.out.println("Exception occurred" + e);
454.  }
455.  
456.  }
457.  
458.  }
459.  
460.  public class MultiServer
461.  {
462.  
463.  // Here initialize the PORT
464.  static final int PORT = 7070;
465.  
466.  MultiServer()
467.  {
468.  }
469.  
470.  void multiaccess() throws IOException
471.  {
472.  
473.  /* Create an object of server socket on this port any client can connect and they can send
474.   * its request */
475.   ServerSocket s = new ServerSocket(PORT);
476.   System.out.println("Server Started");
477.   try
478.   {
479.     while (true)
480.     {
481.       // Create a new socket for every client
482.       Socket soc = s.accept();
483.       try
484.       {
485.         // Call the OneServer class and pass the connected client socket
486.         new OneServer(soc);  
487.         catch (Exception e)
488.         {
489.           // Close the created socket
490.           soc.close();
491.         }
492.       }  
493.  
494.  
495.  }
496.  
497.  catch (Exception e)
498.  {}  
499.  
500.  catch (Exception e)
501.  {}  
502.  }
Code description

- Lines 1-3: Import statements for importing basic class files.
- Lines 11-65: Declares variables and objects.
- Lines 69: Constructor is defined for OneServer, an inner class of MultiServer.
- Lines 72-82: Objects are defined to store the socket value, receive requests, and respond to them. The start method is called.
- Line 87: The run method is defined.
- Lines 89-134: The main.xml is parsed. A byte stream is read from the client, and the data read is written in main.xml; then this XML file is parsed, which provides the request type.
- Lines 135-215: This if condition is executed if the request type is SHOWFILES. In this case, the Share.ini file is read, and the entries (File Name, File Size, Permissions) are sent to the client as XML data, which is written by the XmlWriter class. The same strategy of using a while loop and StringTokenizer is applied to break up the string and to read it.
- Lines 218-244: This if condition is executed if the request type is DOWNLOAD. In this case, a stream is opened from the file to be downloaded, and the object of the stream is sent to the client.
- Lines 247-417: This if condition is executed if the request type is SEARCH. In this case, the search string is compared with the files or directory name in Share.ini file, and the entries found are sent as XML data to the client.
- Lines 421-504: The MultiServer class is defined. A port is provided at which the listener runs. Method multiaccess is defined, in which the inner class OneServer is called.

check_directory.java

This class is used for searching. The search string is passed to this class, and this class searches for the file and directory with the search string and returns a 2D array containing the file name, file size, and permission on files and the directory (see Listing 4-5).

Listing 4-5: check_directory.java

//© 2001 Dreamtech Software India Inc.
// All rights reserved
1. import java.io.*;
2. /*
3. * This class is only for searching purpose . namely
5. * case it can
6. * search file as well directory .
7. */
8. 
9. public class check_directory
10. {
// To store the list of files
String[] files;

// to store the fillter files only
String[] fillterfiles;

String[][] all_details;

// Number of files.
int count = 0;

int sizeoffile = 0;

public String[][] wild_card(String left_half,String right_half, boolean left_half_flag, boolean right_half_flag, File path_file) {
    // store all the files in files variable
    files = path_file.list();

    fillterfiles = new String[files.length + 1];

    // If search criteria is ".*"
    if ((left_half.equals(" "))&&(right_half.equals(" ")))
    {
        // Total number of files..
        for (int i = 0;i<files.length ;i++ )
        {
            // Search result..
            fillterfiles[count] = files[i];
            count++;
        }
    }

    // right side of file is "" and "*"
    else if (right_half.equals(" "))
    {
        String temp = "";

        // Total number of files..
        for (int i = 0;i<files.length ;i++ )
        {
            // Find the position of "." in file
            int index_dot = files[i].indexOf(".");

            if (index_dot != -1) // "."  is not found..
                {
                    // Store the substring till the index of dot is reached.
                    temp = files[i].substring(0,index_dot);
                }
        }

        // Store the first[i] in temp variable
temp = files[i];

// temp variable is equal to left_half and "*" not present in left side of file
if ((temp.length() == (left_half.length())) && (temp.equalsIgnoreCase(left_half)) && (!left_half_flag)) {
    // files satisfying the search criteria.
    fillterfiles[count] = files[i];
    // increment the count value by 1
    count++;
}

// * is present in left half
if ((left_half_flag) && (temp.length() >= left_half.length())) {
    // files satisfying the search criteria
    fillterfiles[count] = files[i];
    // increment the count value by 1
    count++;
}

// left side of file is "*" and "**"
else if (left_half.equalsIgnoreCase(temp.substring(0, left_half.length()))) {
    // files satisfying the search criteria
    fillterfiles[count] = files[i];
    // increment the count value by 1
    count++;
}

// left side of file is "*" and "***"
else if (left_half.equalsIgnoreCase(" ")) {
    String temp = "";

    // total number of files
    for (int i = 0; i < files.length; i++) {
        // Find the position of "." in file
        int index_dot = files[i].indexOf(".");
        if (index_dot != -1) // if dot is found
            { 
                // Store value in temp "." position + 1 to last charachter
                temp = files[i].substring(index_dot + 1);
            }
    }
}

// temp variable is equal to right_half and "*" not present in left side of file
108 Chapter 4: The P2P Application in Java

```java
if ((temp.length() ==
(right_half.length()))&&
(temp.equalsIgnoreCase(right_half))&&(!right_half_flag))
{
    // files satisfying the search criteria
    filterfiles[count] = files[i];

    // increment the count value by 1
    count++;
}

/* "*" present in right side and temp (length) is greater than right_half length */
if ((right_half_flag)&&(temp.length() >= right_half.length()) )
{
    if (right_half.equalsIgnoreCase(temp.substring(0,right_half.length())))
    {
        // files satisfying the search criteria
        filterfiles[count] = files[i];

        // increment the count value by 1
        count++;
    }
}

} // End for loop....

else
{
    /* some character and "*" present in left side and some character and 
    "*" present in right side */
    if ((right_half_flag)&&(left_half_flag))
    {
        // total number of files..
        for (int i = 0;i<files.length ;i++ )
        {
            /* if file files[i] length is grater and equal to left_half length 
            * then enter in this condition */
            if (files[i].length() >= left_half.length())
            {
                if (left_half.equalsIgnoreCase(files[i].substring(0,left_half.length())))
                {
                    // Find the position of "." in file
                    int index = files[i].indexOf(".");

                    if (index != -1)
                    {
                        // Store value in temp "." position + 1 to last character
                        String temp = files[i].substring(index+1);

                        /* if file temp length is grater and equal to right_half length */
                        if (temp.length() >= right_half.length())
                        {
                        }
                    }
                }
            }
        }
    }
}
```
if (temp.substring(0, right_half.length()).equalsIgnoreCase(right_half)) {
    // files satisfying the search criteria
    filterfiles[count] = files[i];
    // increment the count value by 1
    count++;
}

/* only characters are present in left side and some character and */
else if ((right_half_flag) & (!left_half_flag)) {
    // make complete filename out of the criteria.
    String filename = left_half + "." + right_half;
    // total number of files.
    for (int i = 0; i < files.length; i++)
        if (files[i].length() >= filename.length())
            if (filename.equalsIgnoreCase(files[i].substring(0, filename.length())))
                // files satisfying the search criteria
                filterfiles[count] = files[i];
                // increment the count value by 1
                count++;
}

/* only characters are present in right side and some character and */
else if (!(right_half_flag) & (left_half_flag)) {
    // total number of files.
    for (int i = 0; i < files.length; i++)
        if (files[i].length() >= filename.length())
            if (filename.equalsIgnoreCase(files[i].substring(0, filename.length())))
                // files satisfying the search criteria
                filterfiles[count] = files[i];
                // increment the count value by 1
                count++;
}

/* only characters are present in left side and some character and */
else if ((!right_half_flag) & (left_half_flag)) {
    // total number of files.
    for (int i = 0; i < files.length; i++)
        if (files[i].length() >= filename.length())
            if (filename.equalsIgnoreCase(files[i].substring(0, filename.length())))
                // files satisfying the search criteria
                filterfiles[count] = files[i];
                // increment the count value by 1
                count++;
}
if length of files[i] is greater and equal to left_half length
if (files[i].length() >= left_half.length())

if left_half is equal to substring of files[i] first position to length of left_half */
if (left_half.equalsIgnoreCase(files[i].substring(0, left_half.length())))

Find the position of "." in file
int index = files[i].indexOf(".");

if (index != -1)

if (temp.equalsIgnoreCase(right_half))

// files satisfying the search criteria
fillterfiles[count] = files[i];
// increment the count value by 1
count++;
Chapter 4: The P2P Application in Java 111

long selectedfilesize = 0;

String filesizereturn = "";

String maskoffile = "";

all_details = new String[fillterfiles.length][3];

// run till the length of fillterfiles array..
for(int i = 0; fillterfiles[i] != null; i++)
{
    //Make the file object by appending path of files
    File finallist = new File(path_file + "\" + fillterfiles[i]);

    // if current file object is directory then
    if (finallist.isDirectory())
    {
        // no filesize
        filesizereturn = "";

        //mask = 0
        maskoffile = "0";

        // in case of directroy add "\" in last
        fillterfiles[i] = fillterfiles[i] + "\" ;
    }

}

// if a file then
else
{
    if (finallist.canRead())
    {
        // read mask
        maskoffile = "0";
    }

}

else if(finallist.canWrite())
{
    // write mask
    maskoffile = "1";
}

// find the file size
selectedfilesize = finallist.length();
Chapter 4: The P2P Application in Java

```java
334.     filesizereturn = Long.toString(selectedfilesize);
335. }
336.
337.     /// Store the information in array..
338.
339.     all_details[i][0] = fillterfiles[i];
340.     all_details[i][1] = filesizereturn;
341.     all_details[i][2] = maskoffile ;
342. }
343.     // Return the array to the calling programme..
344.     return all_details;
345. }
346.
347. }
```

**Code description**
- Line 1: A single import statement that imports the java.io class.
- Lines 14-23: Arrays and variables are declared.
- Lines 28-345: This is the method `wild_card` definition. This method searches for files and directories based on the search string and stores the entries in a 2D array and returns that array. The array contains the full path of the file or directory, the file size, and permissions.
  - 36-49: This is an if block to show search results if the search criterion is `*.*`.
  - 52-70: This shows results if the search criterion is that the right side of the file specified is `""` and the left part is `"*"`. The left side of a file is the file name and the right side is the extension of the file. Say, for example, if the file name is `Help.doc` the left side is “Help” and the right side is “.doc”.
  - 71-79: The code for the case when `"*"` is not present in the left half.
  - 81-94: The code for the case when `"*"` is present in the left half.
  - 98-145: The code for the case when the left side of the file is `""` and the right side is `"*"`.
  - 149-190: The code for the case when some characters and `"*"` are present on both the right and the left side.
  - 192-217: The code for the case when only characters are present on the left side and characters as well as `"*"` are present on the right side.
  - 220-256: The code for the case when only characters are present on the right side and characters as well as `"*"` are present on the left side.
  - 257-280: Complete filenames are formed, and files satisfying the preceding search criteria are stored in an array.
  - 294-347: The second dimension of the array is used to store information about the files found during a search.

**Browser/Client Source Code: SParser.java**

The input to the `SParser` class is the name of the XML file passed as a string, and the output of the class is a `Vector` object containing all the information from a parsed XML file. This class acts as a wrapper class to the XML parser (it calls the XML parser and stores the results generated by the parser in a `Vector`). Listing 4-6 contains the source code for `SParser.java`. 
Listing 4-6: SParser.java

//© 2001 Dreamtech Software India Inc.
// All rights reserved

1. import java.awt. *;
2. import java.io. *;
3. import java.util. *;
4. import org.xml.sax. *;
5. import org.apache.xerces.parsers.SAXParser;
6. 
7. /* This is the wrapper class of the XML parser class as it calls the Xml parser and in turn
8. when the xml parser, parses the xml file it generates the call backs on the class
9. MyContentHandler... the various parsed documents can be used by using the various functions in this class.
10. */
11. This class returns a vector to the class which calls this class SParser.java the vector
12. consists of the data provided by the XML document.
13.
14. * /
15.
16. public class SParser
17. {
18. Vector values = new Vector();
19. // Initializing a object(values) of the class vector...
20. // It is this object which is returned to the class called...
21. public Vector perform(String uri)
22.     { // it has a parameter called "uri" which has the
23.     // information of the file to be parsed...
24.         try
25.         {
26.             XMLReader parser = new SAXParser();
27.             // Generate an object of the
28.             // XMLParser class...
29.             // Generate an Object of the MyContentHandler Class it is in this class that the xml parser generates the call backs...
30.             MyContentHandler contHandler = new MyContentHandler();
31.             parser.setContentHandler(contHandler);
32.             parser.parse(uri); // call the parse function of the XMLParser class with the
33.                     // file information as the parameter...
34.             values = contHandler.values_attributes();
35.         }
36.         catch (IOException e)
43.     
44.     System.out.println("Error reading uri : " + e.getMessage());
45.   }
46.   
47.   catch (SAXException e)
48.   {
49.     System.out.println("Error in parsing : " + e.getMessage());
50.   }
51.   return values; // Return the vector generated after xml parsing is complete.
52. }
53. }
54. }
55. }
56. }
57. }
58. class MyContentHandler implements ContentHandler
59. {
60.   private Locator locator;
61. 
62.   Vector values = new Vector();
63.   int j = 0;
64. 
65.   public Vector values_attributes()
66.   {
67.     return (values);
68.   }
69.   public void startElement(String namespaceURI, String localName, String rawName, Attributes atts) throws SAXException // Only this function is used by us for our
70.   // purpose....
71.   {
72.     for(int i = 0; i<atts.getLength(); i++)
73.     {
74.       values.add(j,(Object)atts.getValue(i));
75.       j++;
76.     }
77.   }
78.   public void characters(char[] ch, int start, int end) {
79.   }
80.   public void startDocument() {
81.   public void endDocument() {
82.   public void endElement(String nameSpaceURI, String localName, String rawName) {
83.   public void startPrefixMapping(String prefix, String uri) {
84.   public void endPrefixMapping(String prefix) {
85.   public void ignorableWhitespace(char[] ch, int start, int end) {
86.   public void processingInstruction(String target, String data) {
87.   public void setDocumentLocator(Locator locator) {
88.   public void skippedEntity(String name) {
89.   }
90. }
**Code description**

- Lines 1-3: This portion includes the basic packages used by the various classes to build this application. The packages include `java.io` (file streams), `java.util` (various utilities), and so on.
- Lines 4-5: This part lists the basic packages used by the various classes of the XML parser. These packages are `org.xml.sax.*` and `org.apache.xerces.parsers.SAXParser`.
- Lines 16-18: This code declares a public class (`SParser`) that represents the base class for this file and declares the user-defined variables, which are used for various purposes during the course of the program. These include Vector-type variables into which the result of the parsed XML file is stored.
- Lines 21-53: The code here defines a function of the name `perform`. This function is used for calling the parser. It take a parameter called `uri`, which has the information of the file to be parsed.
  - 27: This generates an object of the `XMLParser` class.
  - 33-35: This generates an object of the `MyContentHandler` class. In this class, the XML parser generates the callbacks. The method `setContentHandler` is called, with object of `MyContentHandler` class as parameter. This assigns the handler to the parser.
  - 36: This calls the `parse` function of the `XMLParser` class with the file information as the parameter.
  - 39: The content handler’s attributes are stored in the variable called `value`.
  - 42-49: Exception Handling code. The code makes sure the appropriate message is shown, depending on whether the error is in parsing or reading.
  - 51: Once XML parsing is complete, the Vector object generated is returned.
- Lines 58-90: The code here pertains to the declaration of a `MyContentHandler` class. In this class, the callback from the XML parsing is handled and stored in appropriate data structures. For our requirements, we use a `startElement` function (69-77) and store the values in the Vector object, which is returned to the calling class in the end.
  - 62: A new `Vector` object is instantiated.
  - 65-68: This defines the method `values_attributes`, which returns the Vector values.
  - 69: A method `startElement` is defined. It takes the parameters `namespaceURI`, `localname`, `rawName` and `atts` (attributes).
  - 72-76: Values of attributes are obtained by calling the `getValue` method and are added to the Vector object after casting them as object. A for loop runs till the count becomes equal to the length of `atts` parameter.
  - 78-88: This is the implementation of the methods of the `ContentHandler` interface.

**XMLWriter.java**

The `XMLWriter` class is common to both the client and the listener, as it is used to send requests and responses to the listener and the client, respectively (see Listing 4-7). It has four important functions to perform. Two are used to send requests, and two are used to send responses.

Two functions are used for sending requests:

- `requestFString()`
- `returnRequest()`
Two functions are used for sending responses:
- responseFString()
- returnResponse()

Listing 4-7: XMLWriter.java

//© 2001 Dreamtech Software India Inc.
// All rights reserved

1. // class for xml writer (returns a String for generating an XML File)
2. import java.awt.*;
3. import java.io.*;
4. import java.util.*;
5. import java.lang.*;
6.
7. public class XmlWriter
8. {
9.   int flag = 1;
10.  StringBuffer sbuf = new StringBuffer();
11.  String name = "", fame = "", type = "", size = "", mask = "";
12.  void requestFString(String filetype, String filename)
13.  {
14.   sbuf = sbuf.delete(0, sbuf.capacity());
15.   if (filetype.equals("SEARCH") || filetype.equals("DOWNLOAD") ||
16.     filetype.equals("UPLOAD") )
17.     {
18.       sbuf.append("<scope type="" +filename+"" mask=''>");
19.     }
20.   }
21.  void responseFString(String filetype, String filename, String filesize,
22.                         String mask)
23.  {
24.    if (filetype.equals("SHOWFILES") || filetype.equals("SEARCH") )
25.      {
26.        sbuf.append("<scope type="" +filename+"" mask='">");
27.        sbuf.append("</scope>");
28.      }
29.  }
public String returnHeader(String filetype) throws IOException
{
    StringBuffer sb = new StringBuffer();
    sb.append("<?xml version="1.0" encoding="utf-8"?>");
    sb.append("<p2p_lng>");
    sb.append("<response type=""+filetype+"/>");
    String temp_s = sb.toString();
    sb = sb.delete(0,sb.capacity());
    return(temp_s);
}

public String returnResponse() throws IOException
{
    String tt;
    tt = sbuf.toString();
    //System.out.println(sbuf.toString()) ;   // last response Statement to print
    sbuf = sbuf.delete(0, sbuf.capacity());
    return tt;
}

public String returnRequest() throws IOException
{
    String tt;
    tt = sbuf.toString();
    //System.out.println(sbuf.toString()) ;   // last request Statement to print
    sbuf = sbuf.delete(0, sbuf.capacity());
    return tt;
}
Chapter 4: The P2P Application in Java

**Code description**

- Lines 3-6: This includes the basic packages used by various classes included to build this application. The packages include `java.io` (file streams), `java.util` (various utilities), `java.lang` (string), and so on.
- Lines 11-14: This declares the user-defined variables, which are used for various purposes during the course of the program. These include String types (`name`, `fame`, `type`, `size`, `mask`, `int flag`, `stringbuffer sbuf`, and so on).
- Lines 16-29: This includes the code for generating the XML request sent by the client to the listener (`requestFString`); the client passes parameters according to the request to be generated: Search, Download, Upload, ShowFiles, and so on.
  - 18: Delete the String buffer initially.
  - 20-21: Append to the String buffer the initial tags common to all the requests, such as XML tags, `p2p_lng` tags, and so on.
  - 22: Append the request type according to the parameter passed.
  - 24-28: Whether or not the filetype equals SEARCH, DOWNLOAD or UPLOAD, the scope type is appended accordingly.
- Lines 31-57: This includes the code for generating the XML response sent by the listener to the client (`responseFString`). The listener passes the parameters according to the response to be generated. For example: Search, Download, Upload, ShowFiles, and so on.
  - 40-45: If the response to be generated is of the type showfiles or Search and the file size is zero, (that is, it is a folder), append `fileinfo` tag in the string buffer, with a filename mask written to it. This information is passed as parameter to the `responseFString` function.
  - 46-50: Otherwise, append the string buffer with a `fileinfo` tag with a filename, mask, and file size written to it.
  - 52-57: This shows an error message if the filetype is ERROR.
- Lines 59-69: This includes the code for generating the XML response sent by the listener to the client (`returnHeader`). This code returns the header information of the XML file to be generated.
- Lines 71-81: This includes the code for generating the XML response sent by the listener to the client (`returnResponse`). This code returns the end tag information of the XML file to be generated.
  - Therefore, a complete XML response is generated by combining three functions: `returnHeader` (59-69), `responseFString` (31-57), and `returnResponse` (59-69).
- Lines 83-93: This includes the code for generating the XML request sent by the client to the listener (`returnRequest`). This code returns the end tag information of the XML file to be generated.
  - Therefore, a complete XML request is generated by combining two functions: `requestFString` (16-29) and `returnRequest` (83-93).

**client.java**

This is the main class of the client. It requests the server for the list (userlist) of all the listeners connected at a particular time (see Listing 4-8). After receiving the response from the server, the `client.java` class displays the list of listeners connected to the user (see Figure 4-3) and provides the user with certain options:

- To connect to a particular listener
- To refresh the list so as to see whether any other entry is added to the list or not
To search for files on all the listeners’ (server-level search) machines or to search by specifying search criteria
♦ To view the files (which are searched during the search option)
♦ To exit the client application

Figure 4-3: Starting screen of the peer-to-peer client

Listing 4-8: client.java

// © 2001 Dreamtech Software India Inc.
// All rights reserved

1. import javax.swing.*;
2. import javax.swing.event.*;
3. import java.awt.event.*;
4. import java.awt.*;
5. import java.net.*;
6. import javax.swing.table.*;
7. import java.io.*;
8. import java.util.*;
9. 
10. /* This is the main Class: it caters to the GUI and the starting the Peer 2 Peer Client */
11. 
12. public class client extends JFrame implements ActionListener
13. {
14.  private JButton client_open; //
15.  private JButton client_refresh; //
16. private JButton client_search;  
17. private JButton client_quit;    // GUI Components for 
18. various purposes..
19. private JButton client_view_files; //
20. private JTable client_listing;     //
21. private JScrollPane client_scroller; //
22. int count = 0;
23. Container contentpane;        // For placing 
24. the GUI Components
25. TableModel default_table;      //
26. String names[] = {"Users Connected"}; // For JTable GUI Component
27. Object data[][] ={{null},{null},{null},{null},{null},{null},
28. (null),(null),(null),(null),(null),(null),(null),(null),(null)},
29. (null),(null),(null),(null),(null),(null),(null),(null),(null)};
30. private URLConnection urlconnection; //
31. private InputStream url_inputstream; // For Connecting and gaining the 
32. information..
33. private Socket client_socket;
34. boolean go_on = true;
35. boolean check = true;
36. Vector values = new Vector();
37. String information[][];
38. int g;
39. add_on connection;
40. /* The constructor for the Main Class Takes Four Arguments.....*/
41. // param ... For the state in which this class is called (Search / Other 
42. ).
43. // us ... Will come into use when param is in Search Mode..
44. // fs ... Will come into use when param is in Search Mode..
45. // present_users ... Will come into use when param is in Search Mode..
46. client(String param, String us, String fs, String present_users[][])
47. {
48.          setTitle("Peer 2 Peer Client...");
49.          contentpane = getContentPane();
50.          contentpane.setLayout(null); // Setting the Layout to Absolute 
51. Layout..
52.          client_open = new JButton("Open"); // Initializing the GUI 
53. Component.
54.          client_open.setMnemonic('O'); // Setting the Mnemonic..
55.          client_open.setBounds(20,20,80,35); // Positioning the GUI Component.
Chapter 4: The P2P Application in Java 121

63. client_refresh = new JButton("Refresh"); // Initializing the GUI Component.
64. client_refresh.setMnemonic('R'); // Setting the Mnemonic..
65. client_refresh.setBounds(100,20,80,35); // Positioning the GUI Component.
66. client_search = new JButton("Search"); // Initializing the GUI Component.
67. client_search.setMnemonic('S'); // Setting the Mnemonic..
68. client_search.setBounds(180,20,80,35); // Positioning the GUI Component.
69. client_view_files = new JButton("View Files"); // Initializing the GUI Component.
70. client_view_files.setMnemonic('V'); // Setting the Mnemonic..
71. client_view_files.setBounds(260,20,100,35); // Positioning the GUI Component.
72. client_view_files.setEnabled(false);
73. client_quit = new JButton("Quit"); // Initializing the GUI Component.
74. client_quit.setMnemonic('Q'); // Setting the Mnemonic..
75. client_quit.setBounds(360,20,80,35); // Positioning the GUI Component.
76. // Initializing the Table
77. default_table = new AbstractTableModel()
78. {
79.     // These methods always need to be implemented.
80.     public int getColumnCount() { return names.length; }
81.     public int getRowCount() { return data.length; }
82.     public Object getValueAt(int row, int col) {return data[row][col];}
83.     public String getColumnName(int column) {return names[column];}
84.     public Class getColumnClass(int col) {return getValueAt(0,col).getClass();}
85.     public boolean isCellEditable(int row, int col) {return (col==4);}
86.     public void setValueAt(Object aValue, int row, int column) {
87.         data[row][column] = aValue;
88.         fireTableCellUpdated(row, column);
89.     }
90. }
91. // Positioning and Initializing the GUI Component (Table)...
92. client_listing = new JTable(default_table);
93. client_listing.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
94. client_listing.getTableHeader().setReorderingAllowed(false);
95. client_listing.setBounds(10,55,440,300);
96. client_listing.setGridColor(new Color(255,255,255));
97. client_listing.setBackground(new Color(255,255,255));
109. // Adding Scroll Facility to the JTable by adding JScrollPane Component
110.
111. client_scroller = new JScrollPane();
112. client_scroller.setBounds(10, 55, 440, 300);
113. client_scroller.setViewportView(client_listing);// Adding the Table...
114.
115. contentpane.add(client_scroller); // Placing the scroller on to the gui window..
116.
117. // The Rest of the GUI components will be placed on the window as and when needed.
118.
119. information = present_users; // A String array(2 D) is initialized with the
120. // present_users for reference..
121.
122. // The Class add_on is a multi utility class which has functions defined in it which
123. // are used by many other classes in this peer 2 peer client project...
124.
125. connection = new add_on();//Object Connection of class add_on is created..
126.
127. // A function start_connection of the class add_on is called... which performs
128. // 2 tasks on the basis of parameters passed on to it...
129.
130. // param ... For the state in which this class is called (Search / Other ).
131. // us ... Will come into use when param is in Search Mode..
132. // fs ... Will come into use when param is in Search Mode..
133. // present_users ... Will come into use when param is in Search Mode..
134.
135. information = connection.start_connection(param, us, fs, present_users);
136.
137. // The function returns a String 2 D Array which has the users list along with the
138. // shared files (if the param was Search) and the ip address of the user..
139.
140. // Another function of this class is called which is responsible for placing the
141. // information provided by the above function on to the Table...
142.
143. place_info_table(information);
144.
145. // Now according to the parameters passed in the class the various GUI components
146. // are enabled or disabled....
147.
148. if (param.equalsIgnoreCase("search"))
150. (  
151.     client_view_files.setEnabled(true); // View Files Button is  
152.     Enabled...)  
153. 
154.     // Adding the window Listener to the gui window  
155.     // ie. Code for the "cross"...  
156.     addWindowListener (new java.awt.event.WindowAdapter () {  
157.         public void windowClosing (java.awt.event.WindowEvent evt) {  
158.             System.exit(0);  
159.         }  
160.         }  
161.     );  
162. 
163.     contentpane.add(client_open); //  
164.     contentpane.add(client_refresh); //  
165.     contentpane.add(client_search); // Adding the GUI  
Buttons.....  
166.     contentpane.add(client_view_files); //  
167.     contentpane.add(client_quit); //  
168.  
169. } // End of the constructor.......(client)...
170. }  
171. // The part Below pertains to the Action Performed when a Button is  
presseed...  
172. 
173. public void actionPerformed(ActionEvent ae)  
174. {  
175.   if (ae.getSource() == client_open) // When open Button is pressed...  
176.   {  
177.     int row = 0; // Temporary  
variable to get the index...  
178.     check = true;  
179.     row = client_listing.getSelectedRow(); // To get the index of the  
row which  
180.     // is selected by the user...  
181.     try  
182.     {  
183.       if(information[row][1] != null) // If the user index is not  
null...  
184.         {  
185.           try  
186.           {  
187.             InetAddress inet =  
InetAddress.getByName(information[row][1]);  
188.             client_socket = new Socket(inet,7070); // Create a  
client_socket on the  
189.             // Listener's machine at port 7070.  
190.             }  
191.             // Get The output as well as the input Streams on that socket...  
192.             BufferedOutputStream out = new  
BufferedOutputStream(client_socket.getOutputStream());  
193.             

Chapter 4: The P2P Application in Java

194. BufferedInputStream br_socket = new BufferedInputStream(client_socket.getInputStream());
195. // Now a request is sent to the Listener to show all the shareable files
196. // of the particular user which was selected by the user..
197.
198. // To send a request a class of the name xmlwriter is used which has two
199. // functions of interest they are... requestFString(String, String) and
200. // returnRequest(), this xmlwriter is a versatile class as it is used to
201. // generate xml requests for various purposes... SEARCH, UPLOAD, DOWNLOAD
202. // and SHOWFILES... therefore accordingly the requestFString takes two
203. // parameters in case of SHOWFILES the second parameters is not used.
204.
205. XmlWriter writer = new XmlWriter(); // Initializing a object of
206. // xmlwriter..
207. writer.requestFString("SHOWFILES"," "); //calling the function...
208. String file_data = writer.returnRequest(); // getting the request in a
209. // temp variable file_data.
210.
211. byte file_bytes[] = file_data.getBytes(); // getting byte array of string
212. int file_size = file_bytes.length; // Getting the length of byte array
213.
214. byte b[] = new byte[1024]; // Initializing a new byte array of 1024.
215.
216. // another method of the class add_on is used now (appropriatelength) this
217. // is used so as to make the request sent by the client to the listener
218. // 1024 in length...(for c#) listener's...
219.
220. // The method takes a byte array and its length as parameters and return
221. // a byte array of length 1024 bytes....
222.
223. add_on upload = new add_on();
224. b = upload.appropriatelength(file_bytes, file_size);
225.
226. out.write(b,0,1024); // The byte array is written on the output stream
227.
228.
229. // An output stream is also initialized this is used to store all the
230. // response
231. // from the listener..
232.
233. BufferedOutputStream out_file = new BufferedOutputStream(new
234. FileoutputStream("response.xml");
235.
236. int y = 0; // Temporary variables....
237. byte f[] = new byte[32]; // Temporary variables...
238.
239. while ((y = br_socket.read(f,0,32))>0) // the socket input stream is
240. // read
241.  {
242.    out_file.write(f,0,y); // written on to the file output
243.    stream...
244.  }
245.
246. out.close(); //
The filestream and socket streams are closed...

try
{
    client_socket.close(); // Close the Client_socket...
}
catch (Exception e)
{
    row = 0;
}

if (check) // If the exception occurs then do not come here....
else
{
    Vector parameters = new Vector(); // Temp Vector Declaration..
    // A class SParser is also used here this class has a function/method of
    // the name perform which calls the xml parser to parse the xml file
    // generated by the response from the client soket...
    // the function perform returns a Vector which has the files/directories,
    // along with their flag information and size in case of files....
    SParser sp = new SParser();
    parameters = sp.perform("response.xml");

    // The vector value returned by the xml parser is then passed as one of
    // the parameters to a class named file_gui this class is responsible for
    // displaying GUI consisting of a table and some buttons along with the
    // root information and flag..
    // Initially since the class is called for the first time the parameter
    // for the root is given the name "ROOT" and the Flag is set to "0"..
    file_gui showfiles = new
    file_gui(parameters,information[row][1],"Root","0");
    showfiles.show();
    check = false;
}

// End Try....
297. else if (ae.getSource() == client_refresh) 
298. {
299.   this.setVisible(false); // hide the present window....
300.   String present_users[][] = { {" ", " "}, {" ", " "} }; // make the user
301.   list empty
302.   // Call the main class a new ....
303.   client client = new client("start", ", ", ", ", present_users);
304.   client.setSize(465, 410);
305.   client.show();
306.
307. } // End if of ae.getSource()...
308.
309. else if (ae.getSource() == client_quit) 
310. {
311.   System.exit(0); // Close the connection and exit to system...
312. } // End if of ae.getSource()...
313.
314. else if (ae.getSource() == client_search) 
315. {
316. // If search button is pressed a new class called search_window is called which
317. // is responsible for client searching it caters to the gui of the
318. // search_window
319. // as well....
320.   search_window search_users = new search_window(this, information);
321.   search_users.show();
322. 
323. } // End if of ae.getSource()...
324.
325. else if (ae.getSource() == client_view_files) 
326. {
327. // This button activates only after the client_search button is used for
328. // searching the listener or a particular file name on all the possible
329. // users...
330. 
331. // When this button is activated the user will see all the names of the
332. // connected at that instant along with the number of files (satisfying a
333. // particular search criteria) enclosed in bracket to see those files the
334. // user
335. // presses the viewfiles button after selecting a particular user and the
336. // list of the files is displayed to the user. The user is however not able to
337. // download the files from that location...
338.   int row = 0;
339. 
340.   row = client_listing.getSelectedRow();
341. // for this another function of the add_on class is used returnfilenames
342. // which
343.   returnfilenames vector along with the name of the user..
344.   add_on search_result = new add_on();
345.   Vector filenames = search_result.returnfilenames();
346.   Vector results = new Vector();
347.   results.add(0,(Object)"files");
for (int i = 1; i < filenames.size(); i++)
{
  String temp = (String) filenames.get(i);
  String name = " ";
  name = temp.substring(0, temp.indexOf("?"));
  name = name.substring(0, information[row][1].length());
  if (name.equals(information[row][1]))
  {
    temp = temp.substring(temp.indexOf("?") + 1);
    StringTokenizer st = new StringTokenizer(temp, "?");
    while (st.hasMoreTokens())
    {
      results.add((Object) st.nextToken());
    }
    temp = " ";
  }
}
// when the vector is generated (results) it is passed on to the the file_gui
// class which is responsible for displaying the files..., the root parameter
// is given the value as search result and the flag information as "0"...
file_gui showfiles = new file_gui(results, information[row][1], "Search Results", "0");
showfiles.show();

public void place_info_table(String information[][]) {
  if (!(information.length > 1)) // If no information[][] array is generated..
  {
    JOptionPane.showMessageDialog(this, "Sorry There is no server at present to satisfy ur request. ", "Peer 2 Peer Client", JOptionPane.ERROR_MESSAGE);
    client_open.setEnabled(false);
    client_refresh.setEnabled(false); // all buttons except quit are disabled.
    client_search.setEnabled(false);
    client_view_files.setEnabled(false);
    client_quit.setEnabled(true);
Lines 1-8: This code is for including the basic packages used by various classes to build this application. The packages include java.net (sockets), java.io (file streams), javax.swing (frames), javax.swing.event (event handling), and so on.
Chapter 4: The P2P Application in Java

♦ Lines 12-27: This declares a public class (Client) that represents the base class for this file. All the variables defined here represent their respective controls pasted on the frame. These variables include client_open, client_refresh, client_search, client_viewfiles, and client_quit that are of the type JButton, client_listing that is of the type Jtable, and so on.

♦ Lines 28-48: This declares the user-defined variables used for various purposes during the course of the program; these are variables for URLConnection, InputStream, and variables of the type boolean for various checks during the program.

♦ Lines 49-125: The code between these lines pertains to initializing the GUI components (that is, the buttons, the table, and the scrollbar positioning). These GUI components on the frame are supplied with their mnemonics for faster access through the keyboard, and so on.

♦ Lines 126-136: Object connection of class add_on is created, and a method of the class start_connection is called, which returns a list of all the listeners connected at a particular time.

♦ Lines 144: The information, returned by the method, is then passed on to the function place_information, which is responsible for displaying the information in the Table for the user to understand easily.

♦ Lines 163-169: The code here pertains to adding the various GUI components to the frame.

♦ Lines 173-378: This includes the code for actions performed on various buttons.

♦ Lines 175-296: This includes the code executed when the client selects a particular listener from the list and presses the Open button (client_open).

• 179: The row selected by the client is identified and based on the row from which the IP address of the listener is obtained.
• 187-188: Create a client_socket on the listener’s machine on the port 7070.
• 192-195: Get the output as well as the input streams on that socket.
• 206-208: Now a request is sent to the listener to show all the shareable files.
• 224-225: A method of the class add_on is used now (appropriatelength). This is used to make the request sent by the client to the listener 1024 bytes in length (for C# listeners).
• 227: The byte array is written to the output stream.
• 233: An output stream is also initialized. This is used to store all the responses from the listener.
• 238-241: The socket-input stream is read and written to the file-output stream.
• 243-263: This closes the file streams as well as the sockets and handles the exceptions through the try...catch blocks.
• 266-276: A Temporary Vector (parameters) is declared. The SParse class is also used here. This class has a function/method of the name perform, which calls the XML parser to parse the XML file generated by the response from the client socket. The function returns a Vector, which is stored in the Temp Vector.
• 286-287: The Vector value returned by the XML parser is then passed as one of the parameters to a class named file_gui. This class is responsible for displaying GUI consisting of a table and some buttons, along with the root information and flag.

♦ 298-308: This includes the code executed when the client presses the Refresh button, in which case the current frame is made to disappear and a new request is sent to the server for the list of all the available listeners.

♦ 310-313: This includes the code executed when the client presses the Quit button to quit the application.
Chapter 4: The P2P Application in Java

- 315-323: This includes the code executed when the client presses the Search button on the main screen. Here a GUI appears above the main screen, which is responsible for performing server-level searches on file names and on users connected at a particular time.

- 325-376: This includes the code executed when the client presses the Viewfiles button. This button is initially disabled and only comes into use view when the user has used the Search button to perform a server-level search. This code is responsible for displaying the files of a particular user, which satisfy the search criteria.

  - 339: The row selected by the client is identified and based on the selected row; the IP address of the listener is obtained.
  - 342-343: Another function returnfilenames of the add_on class is used, which returns the filenames Vector along with the name of the user.
  - 344-369: Out of this Vector, another Vector is generated that consists of files or folders pertaining to a particular user who has been highlighted when the Viewfiles button is pressed. The delimiter used is "?".
  - 373-374: When the Vector is generated (results), it is passed to the file_gui class, which is responsible for displaying files; the root parameter is given the value as a search result and the flag information as "0."

- 383-420: This includes the code of a function, which places the information (number of users connected at a particular time) on the GUI and enables appropriate buttons in between.

  - 406-412: This loops through information and puts the value on the table.
  - 414-418: This adds an action listener to all the buttons.

- 424-432: The main function is declared, which is called at the start.
  - 426: Because the client is called for the first time, the present users array is left empty.
  - 429-431: The constructor is called, the size of the frame is set, and the frame is displayed.

search_window.java

This class is used when the user presses the Search button on the main screen, the class helps in server-level search after specifying certain search criteria (see Listing 4-9). The search criteria can be a particular file on all the listeners connected or can be a particular file with a subset of all listeners connected. The search window is shown in Figure 4-4.

Figure 4-4: Server level search screen

Listing 4-9: search_window.java

//© 2001 Dreamtech Software India Inc.
//All rights reserved

1. import javax.swing.*;
2. import javax.swing.event.*;
Chapter 4: The P2P Application in Java

3. import java.awt.event.*;
4. import java.awt. *;
5. import java.util.*;
6. import java.net. *;
7. import java.io. *;
8. 
9. /* This Class is used to implement Client search on the p2p client... This class has
10.   two main features....
11.   1. To search for a file on all the connected listeners...
12.   2. To search for a file on a particular subset of the connected listeners...
13. */
14. 
15. public class search_window extends JFrame implements ActionListener
16. {
17.     JLabel label_file_name;   //
18.     JLabel label_computer_name;  //
19.     JTextField search_file_name; // GUI Components for various purposes..
20.     JTextField search_computer_name; //
21.     JButton search_ok,search_cancel; //
22.     
23.     Container contentpane;
24.     
25.     String information[][]; // information about the parent frame..
26.     JFrame parent_window; // To Keep
27.     
28.     // The constructor of the class search window takes two arguments..
29.     // 1..  The parent frame...
30.     // 2..  The names of all the present listeners and their IP addresses stored in 2-D array info..
31.     //
32.     // The arguments are then initialized to the objects parent_window and String 2-D array information....
33.     
34.     search_window(JFrame ancestor, String info[][])
35.     {
36.         super("Search Window...");
37.         setSize(375,160); // Set The Size of the Frame...
38.         parent_window = ancestor;
39.         contentpane = getContentPane();
40.         contentpane.setLayout(null); // Setting the Layout to Absolute Layout..
41.         label_file_name = new JLabel("Search for File Names "); // Initializing the GUI.
42.         label_file_name.setBounds(10,10,130,25); // Positioning the GUI Component.
search_file_name = new JTextField(30); // Initializing the GUI Component.
search_file_name.setBounds(180,10,160,25); // Positioning the GUI Component.

label_computer_name = new JLabel("Search On Computer Names "); // Initializing the // GUI Component.
label_computer_name.setBounds(10,45,170,25); // Positioning the GUI Component.

search_computer_name = new JTextField(30); // Initializing the GUI Component.
search_computer_name.setBounds(180,45,160,25);// Positioning the GUI Component.

search_ok = new JButton("Search"); // Initializing the GUI Component.
search_ok.setMnemonic('S'); // Setting the Mnemonic..
search_ok.setBounds(75,90,80,25); // Positioning the GUI Component.

search_cancel = new JButton("Cancel"); // Initializing the GUI Component.
search_cancel.setMnemonic('C'); // Setting the Mnemonic..
search_cancel.setBounds(180,90,80,25);  // Positioning the GUI Component.

search_ok.addActionListener(this); // Add action listener to all the
search_cancel.addActionListener(this); // Add action listener to buttons...

addWindowListener (new java.awt.event.WindowAdapter () {
public void windowClosing (java.awt.event.WindowEvent evt) {
setVisible(false);
}
});

contentpane.add(label_file_name); // Adding the GUI Buttons.....
contentpane.add(search_file_name); // Adding the GUI Buttons.....
contentpane.add(label_computer_name); // Adding the GUI Buttons.....
contentpane.add(search_computer_name); // Adding the GUI Buttons.....
contentpane.add(searchOk); // Adding the GUI Buttons.....
contentpane.add(search_cancel); // Adding the GUI Buttons.....
Chapter 4: The P2P Application in Java 133

90. information = info;  // Initializing the variable information with
91. // the parameter
92. // info...
93. }
94.
95. public void actionPerformed(ActionEvent ae)
96. {
97. if (ae.getSource() == search_ok) // When Ok Button Is pressed....
98. {
99. String us = " "; // Two variables namely us(user search) and fs
100. (file search)
101. String fs = " "; // are used to store the value that the user
102. // enters in the
103. // appropriate textfields...
104. boolean search = true; // Temporary variable used in
105. computation..
106. us = search_computer_name.getText();  // Storing the value
107. fs = search_file_name.getText(); // Storing the value
108. if ((us.equals(""))&&(fs.equals(""))) // Apply check
109. condition
110. { //
111. JOptionPane.showMessageDialog(this,"Please Enter some search
112. Criteria ","Peer 2 Peer Client",JOptionPane.ERROR_MESSAGE);
113. search = false;
114. }
115. }
116. else if (us.equals("")) // if any of the field is
117. left empty..
118. { //
119. us = "**";
120. }
121. else if (fs.equals("")) // if any of the field is
122. left empty..
123. { //
124. fs = "**";
125. }
126. if (search) // Start the search procedure... as both
127. the variables
128. { // have been assigned the
129. value...
130. this.setVisible(false);  // hide the search window...
131. parent_window.setVisible(false); // Hide the parent window as well...
132. // Call the main class client with the parameters...
133. // 1.. Search..  
134. // 2.. user search criteria..
135. // 3.. file search criteria..
136. // 4.. Information about the present users...
137. client search_result = new client("Search",us,fs,information);
138. search_result.setSize(465,410);  // Set the size of the GUI called..
136. search_result.show();  // Display the GUI...
137.
138.
139.
140. else if (ae.getSource() == search_cancel) // When Cancel Button Is pressed....
141. {
142. this.setVisible(false);  // Hide this GUI screen
143. }
144. }
145.
146. }

**Code description**

♦ Lines 1-7: This includes the basic packages used by the various classes used to build this application. The packages are java.net (sockets), java.io (file streams), javax.swing (frames), javax.swing.event (event handling), and so on.

♦ Lines 15-21: This declares a public class (`search_window`) that represents the base class for this file. All the variables defined here represent their respective controls pasted on the frame. These variables are `label_file_name` of the type Jlabel; `search_ok` of the type JButton; `search_cancel` of the type JButton; `search_file_name` of the type JTextField, and so on.

♦ Lines 36-70: The code between these lines pertains to initializing the GUI components (that is, the buttons, the table, the scrollbar positioning these GUI components on the frame, and their mnemonics for faster access through the keyboard, and so on.

♦ Lines 76-81: Adding window Listener.

♦ Lines 83-88: The code here pertains to adding the various GUI components to the frame.

♦ Lines 95-144: This includes the code for actions performed on various buttons.

♦ Lines 97-139: This includes the code executed when the client, after entering certain search criteria in the two JTextField, presses the Search button to commence the searching operation.
  • 110-115: If both the JTextField are left empty and the Search button is pressed, the Alert Box requisite pops up.
  • 116-123: If any of the JTextField are left empty, a "*" is assigned to the variable of that field type.
  • 124-137: Start the search procedure by hiding the search GUI and the frame that has invoked the GUI. Then call the main class client with the parameters: 1. Search. 2. User search criteria. 3. File search criteria. 4. Information about present users.

♦ Lines 140-143: If the Cancel button is pressed, the search window hides.

**file_gui.java**

This class is used when the client/user decides to visit a particular listener to download/upload/search files (see Figure 4-5). This class, apart from showing the shared files of a particular listener, provides buttons to let the client perform various operations (see Listing 4-10). The operations might be the following:

♦ Open option (used when the user wants to browse the folders)

♦ Download option (used when the user wants to download a particular file)

♦ Upload option (used when the user, after having the proper permissions, wants to upload a file to the listener’s machine)
♦ Search option (used when the user wants to search for a particular file on the listener’s machine by specifying certain search criteria).

Figure 4-5: file_gui.java

Listing 4-10: file_gui.java

//© 2001 Dreamtech Software India Inc.
// All rights reserved

1. import javax.swing.*;
2. import javax.swing.event.*;
3. import java.awt.event.*;
4. import java.awt.*;
5. import java.net.*;
6. import java.io.*;
7. import javax.swing.table.*;
8. import java.util.*;
9.
10. /* When the Client selects a particular Listener's name from the list of the users connected and presses the button Open then this class is called. This class helps the client directly connect to the Listener's machine and browse through, download, upload, search for the files/folders shared by that particular Listener....
11.  This class provides the client facilities for download, upload, search...
16. through shared folders facility...
17. */
18.
19. public class file_gui extends JFrame implements ActionListener
20. {
21. private JButton file_open; //
22. private JButton file_search; //
23. private JButton file_close; //
24. private JButton file_upload; // GUI Components for various purposes..
25. private JButton file_download; //
26. private JTable file_listing; //
27. private JScrollPane file_scroller; //
28. private JLabel status; //
29. private Container contentpane; // For placing the GUI Components
30. private Socket file_socket; // To Declare the client socket...
31. Vector values = new Vector(); // To declare a vector "values" for holding the results of the xml response from
32. TableModel default_table; // For JTable GUI Component
33. String names[] = {"Files / Folders..", "Size", "Type"}; // Name of the Columns in JTable
34. Object data[][] ={{" ", " ", " "},{" ", " ", " "}}; // Initializing the JTable Columns..
35. String ip_address; // Variable to store the ip_address.
36. String information[][];
37. String file_name = new String();
38. String status_text; // Variable to store the root information
39. String flag_info; // Variable to store the flag information
40. boolean done = false;
41.
42. /* Constructor is called with 4 parameters they are as follows...
43. 1. This is the result of XML parsing done (and stored in a vector) at the class
44.    files which calls this class.
45. 2. The IP address of the Listener to which the client wants to connect...
46. 3. The root information (i.e., the directory in which the user is currently browsing...). Initially this information is kept as "ROOT".
47. 4. The flag Information about the directory in which the user is currently browsing, this is used for upload purposes... Initially this is kept as 0 as 0 stands for no uploads and client cannot upload on listener's root directory..
48. */
public file_gui(Vector parameter, String ip, String stat_text, String flag_info)
{
    values = parameter;  // Initializing the variables with parameters...
    ip_address = ip;  //
    status_text = stat_text;  //
    this.flag_info = flag_info;  //

    setTitle("Peer 2 Peer Client");
    setSize(475, 405);  // Set The size of the frame...

    contentpane = getContentPane();  // Initialize the window for placing the
    // components..
    contentpane.setLayout(null);  // Setting the Layout to Absolute Layout..

    file_open = new JButton("Open");  // Initializing the GUI Component.
    file_open.setMnemonic('O');  // Setting the Mnemonic..
    file_open.setBounds(20, 20, 80, 35);  // Positioning the GUI Component.

    file_upload = new JButton("Upload");  // Initializing the GUI Component.
    file_upload.setMnemonic('U');  // Setting the Mnemonic..
    file_upload.setBounds(100, 20, 80, 35);  // Positioning the GUI Component.

    file_download = new JButton("Download");  // Initializing the GUI Component.
    file_download.setMnemonic('D');  // Setting the Mnemonic..
    file_download.setBounds(180, 20, 100, 35);  // Positioning the GUI Component.

    file_search = new JButton("Search");  // Initializing the GUI Component.
    file_search.setMnemonic('S');  // Setting the Mnemonic..
    file_search.setBounds(280, 20, 80, 35);  // Positioning the GUI Component.

    status = new JLabel(status_text);  // Initializing the GUI Component.
    status.setBounds(10, 355, 300, 25);  // Positioning the GUI Component.

    // Initializing the Table
    default_table = new AbstractTableModel()
    {
        // These methods always need to be implemented.
        public int getColumnCount() { return names.length; }
        public int getRowCount() { return data.length; }
    }
public Object getValueAt(int row, int col) {return data[row][col];}

// The default implementations of these methods in
// AbstractTableModel would work, but we can refine them.
public String getColumnName(int column) {return names[column];}
public Class getColumnClass(int col) {return getValueAt(0,col).getClass();}
public boolean isCellEditable(int row, int col) {return (col==4);}
public void setValueAt(Object aValue, int row, int column) {
data[row][column] = aValue;
fireTableCellUpdated(row, column);
}

/* This condition is applied so as to disable all the buttons except
the close button when view files button is pressed as while viewing the
files the client cannot make downloads/uploads etc...
*/

if (stat_text.equalsIgnoreCase("SEARCH RESULTS")) {

    file_open.setEnabled(false);
    file_download.setEnabled(false);
    file_upload.setEnabled(false);
    file_search.setEnabled(false);
}

contentpane.add(file_open); //
contentpane.add(file_close); //
contentpane.add(file_download); //
contentpane.add(file_upload); // Adding the GUI Components...
contentpane.add(file_search); //
contentpane.add(status); //

/* An important function "formating" is called with the parameter a
vector value this helps in extracting the information stored in the vector and
placing them in a string 2-Dimensional array in proper format for later
reference also placing the information on to the JTable...
*/

formating(values);

// Positioning and Initializing the GUI Component (Table)...
file_listing = new JTable(default_table);
file_listing.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
//file_listing.getTableHeader().setReorderingAllowed(false);
//file_listing.setBounds(10,55,650,300);
file_listing.setGridColor(new Color(255,255,255));
151.  //file_listing.setPreferredScrolledViewPortSize(new Dimension(500, 70));
152.
153.  // Adding Scroll Facility to the JTable by adding JScrollPane
154.  file_scroller = new JScrollPane(); //file_listing);
155.  file_scroller.setBounds(10, 55, 450, 300);
156.  file_scroller.setViewportView(file_listing);
157.  file_scroller.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
158.  file_listing.setAutoResizeMode(JTable.AUTO_RESIZE_ALL_COLUMNS);
159.  contentpane.add(file_scroller); // Add the Scroll Bar...
160.
161.  // Adding the window Listener to the gui window
162.  // ie. Code for the "cross"...
163.  addWindowListener (new java.awt.event.WindowAdapter () {
164.          public void windowClosing (java.awt.event.WindowEvent evt) {
165.     setVisible(false);
166.          }
167.          });
168.          
169.  file_open.addActionListener(this); // When open Button is pressed...
170.  file_upload.addActionListener(this); // Add the ActionListener...
171.  file_download.addActionListener(this);
172.  file_search.addActionListener(this);
173.  file_close.addActionListener(this); // End of Constructor file_gui...
174.  
175.  // The part Below pertains to the Action Performed when a Button is pressed...
176.  public void actionPerformed(ActionEvent ae)
177.  {
178.          if (ae.getSource() == file_open) // When open Button is pressed...
179.          {
180./* This button will function only in case when the user selects a folder and
181.  opens it for knowing the details of the folder.. If the client tries to
182.  open
183.  a file a MessageBox is shown....
184.  */
185.          try
186.          {
187.              int row = 0;
188.              row = file_listing.getSelectedRow(); // Get the Selection...
189.              if (information[row][2].equalsIgnoreCase("Folder")) // Check whether
190.                  Folder
191.                  { 
192.                      /* If a folder is selected and then pressed the open button then an object
of the class add_on is created (request) This is responsible for sending
the request to the Listener by using a function of the class of the name
"search_request". The parameters passed on to the function are the folder
name, followed by the search criteria here in this case it is ".\.*", foll-
owed by the ip_address of the listener, followed by the flag
information of the folder searched for...
*/

```java
add_on open_request = new add_on();
open_request.search_request(information[row][0],"\.*",ip_address,information[row ][3]);
}
else // If a file is selected...
{
    // Alert / Message Box is displayed...
    JOptionPane.showMessageDialog(this,"Cannot Open a File
Over Network. Try Downloading it. ","Peer 2
Peer...",JOptionPane.INFORMATION_MESSAGE);
}
    catch(Exception es)
{
    // End File Open ....
}
else if (ae.getSource() == file_close) // If close button is pressed
{
    this.setVisible(false); // Hide this frame...
}
else if (ae.getSource() == file_download) // If Download button is
pressed...
{
    /* When download button is pressed a class called SwingWorker provided by
Sun is
called this class helps in running a task in separate thread thus helps
in gui.
*/
    final SwingWorker worker = new SwingWorker()
{
    public Object construct()
    {
        // The function of the class is given a task to
perform (downloading).
        // By calling a class download_file..
        // This downloading is done in a separate thread...
        return new download_file();
    }
    worker.start(); // Calling the start of the swingworker
}
else if (ae.getSource() == file_upload)
{
    done = false;
    /* When Upload button is pressed a class called SwingWorker provided by Sun is

called this class helps in running a task in separate thread thus helps in gui.

final SwingWorker upload = new SwingWorker()
{
  public Object construct()
  {
    // The function of the class is given a task to perform (Uploading).
    // By calling a class upload_file..
    // This uploading is done in a separate thread...
    return new upload_file();
  }
};
upload.start(); // Calling the start of the swingworker
if (done)
{
  this.setVisible(false); // Hide the window after the upload...
}
} // End Upload.....
else if (ae.getSource() == file_search) // When Search Button is pressed...
{
  /* When search button is pressed by the client then root information is checked
  if root information is is "ROOT" then a message box is shown indicating that
  no search can be done...
  else a class search screen is called... which caters to the search ...
  */
  if (status_text.equalsIgnoreCase("Root")) // If status_text is "ROOT"
  {
    JOptionPane.showMessageDialog(this,"Cannot Search on Root. Try searching in sub directories . ","Peer 2 Peer...",JOptionPane.INFORMATION_MESSAGE);
  }
  else // If search Text is not root...
  {
    /* Create an object of the class search_screen ..
    the parameters passed on to the constructor are...
    1. Status_text...
    2. the ip address of the listener...
    3. The flag_info of the folder on which search is being performed...
    */
    search_screen search_now = new search_screen(status_text,ip_address,flag_info);
    search_now.show(); // Show the frame...
  }
290. } // End of Action Listener....
291.
292.
293.
294. /* An important function "formatting" is called with the parameter a vector value
295.   this helps in extracting the information stored in the vector and placing them
296.   in a string 2-Dimensional array in proper format for later reference also
297.   placing the information on to the JTable...
298. */
299.
300. void formatting(Vector values)
301. {
302.     // To check whether the vector has more than one value or not.
303.     if (values.size() > 1)
304.     {
305.         int array_size = values.size();
306.         information = new String[array_size][4]; // Array in to which the information
307.         // extracted is added...
308.         // Information to be placed on the Table is put into the array (2-D) data...
309.         if (array_size > 19)
310.             data = new Object[array_size][3];
311.         } else
312.             data = new Object[19][3]; // Minimum size of array...
313.         boolean file_or_Folderectory;
314.         int i = 1;
315.         int g = 0;
316.         while(i<array_size)
317.             try
318.                 String temp = (String)values.get(i);
319.                 file_name = temp.substring(temp.lastIndexOf(\\)+1);
320.                 if (file_name.equals(""))
321.                     { information[g][0] = (String)values.get(i);
322.                         data[g][0] = values.get(i);
323.                         information[g][2] = "Folder";
324.                         data[g][2] = "Folder";
325.                         i++;
326.                     }
327.             catch (Exception e)
328.                 file_name = temp.substring(temp.lastIndexOf(\\)+1);
329.                 if (file_name.equals(""))
330.                     { information[g][0] = (String)values.get(i);
331.                         data[g][0] = values.get(i);
332.                         information[g][2] = "Folder";
333.                         data[g][2] = "Folder";
334.                         i++;
335.                     }
336.             }
337.         }
338.     }
339. }
340.
information[g][0] = (String)values.get(i);

int index = file_name.lastIndexOf(".");
String gh = ";
if (index == -1)
{
    information[g][2] = ";
data[g][2] = information[g][2]+ File";
    gh = file_name;
}
else
{
    information[g][2] = file_name.substring(file_name.lastIndexOf("."))+1);
data[g][2] = information[g][2]+ File";
    gh = file_name.substring(0,index);
}
data[g][0] = gh;
information[g][2] = file_name.substring(file_name.lastIndexOf("."))+1);
data[g][2] = information[g][2]+ File";
    i++;
information[g][3] = (String)values.get(i);

    i++;
information[g][1] = (String)values.get(i);
data[g][1] = values.get(i);
i++;
}
else // end if..
{
    information = new String[1][3];
    information[0][0] = "No Files are shareable";
    information[0][1] = " ";
    information[0][2] = " ";
    data[0][0] = (Object)"No Files are shareable";
public class download_file extends JFrame
{
    download_file() // Constructor...
    {
        try
        {
            int row = 0;
            row = file_listing.getSelectedRow(); // Get the
            selection of the user...
            InetAddress inet = InetAddress.getByName(ip_address);
            file_socket = new Socket(inet,7070); // Establish a
            socket connection with
            // the Listener on the port 7070
            // address -- inet..
            // Get The output as well as the input Streams on that
            socket...
            BufferedOutputStream out = new
            BufferedOutputStream(file_socket.getOutputStream());
            BufferedInputStream br_socket = new
            BufferedInputStream(file_socket.getInputStream());
            // if the selection is a folder... then pop up a message for
            denial of
            // download..
            if (information[row][2].equalsIgnoreCase("Folder"))
            {
                JOptionPane.showMessageDialog(this,"Cannot Download a Folder.
                Try Opening it. ","Peer 2 Peer...",JOptionPane.INFORMATION_MESSAGE);
            }
            else // If the request is that of a file..
            {
                XmlWriter writer = new XmlWriter(); // Call a class
                XMLWRITER to generate
                // request by using a function
                writer.requestFString("DOWNLOAD",information[row][0]); // requestFString...
                String file_data = writer.returnRequest();
                byte file_bytes[] = file_data.getBytes(); // get the Number
                of bytes from the
432. // request...
433. String temporary = information[row][0].substring(information[row][0].lastIndexOf("\")+1);
434. JTextField jfc = new JTextField(); // Call an object of JTextField
435. // File Dialog to place the file.
436. File file = new File (temporary);
437. jfc.setSelectedFile(file);
438. jfc.ensureFileIsVisible(file);
439. int button_pressed = jfc.showSaveDialog(this);
440. String str1 = "";
441. File file_final = jfc.getSelectedFile();
442. if (button_pressed == JFileChooser.APPROVE_OPTION)
443. {
444.  str1 = file_final.getPath(); // Get the path where the file is being saved..
445. }
446. }
447. }
448. File file = new File (temporary);
449. jfc.setSelectedFile(file);
450. jfc.ensureFileIsVisible(file);
451. int button_pressed = jfc.showSaveDialog(this);
452. String str1 = "";
453. File file_final = jfc.getSelectedFile();
454. if (button_pressed == JFileChooser.APPROVE_OPTION)
455. {
456.  str1 = file_final.getPath(); // Get the path where the file is being saved..
457. }
458. }
459. File file = new File (temporary);
460. jfc.setSelectedFile(file);
461. jfc.ensureFileIsVisible(file);
462. int button_pressed = jfc.showSaveDialog(this);
463. String str1 = "";
464. File file_final = jfc.getSelectedFile();
465. if (button_pressed == JFileChooser.APPROVE_OPTION)
466. {
467.  str1 = file_final.getPath(); // Get the path where the file is being saved..
468. }
469. }
470. }
471. BufferedOutputStream out_file = new BufferedOutputStream(new FileOutputStream(str1)); // Create an outputstream to that path...
472. int file_size = file_bytes.length;
473. // Adjust the request length to 1024
474. // (for c#) listener's...
475. byte b[] = new byte[1024];
476. // another method of the class add_on is used now (appropriate length) this
477. // is used so as to make the request sent by the client to the listener
478. // 1024 in length (for C#) listeners...
479. // The method takes a byte array and its length as parameters and return
480. // a byte array of length 1024 bytes....
481. add_on download = new add_on();
482. b = download.appropriate_length(file_bytes, file_size);
483. out.write(b,0,1024); // The byte array is written on the output stream
484. int y = 0;
485. byte f[] = new byte[32];
486. int file1_size = Integer.parseInt(information[row][1]);
487. // Generate a progress monitor to monitor the request...
488. ProgressMonitor pm = new ProgressMonitor(this, "Downloading File...", "Downloading Please Wait...", 0, file1_size);
489. int current = 0;
490. pm.setMillisToPopup(5);
while ((y = br_socket.read(f,0,32))>0) // Read the socket and
write on to the
// file...
{
  out_file.write(f,0,y);
  current = current + y;
  pm.setProgress(current); // Monitoring the progress
  // monitor...
}
out.close(); //
br_socket.close(); // The filestream and
socket streams are
out_file.close(); // Closed

} // End Constructor..
} // End Class Download_file...

/*  This class is used to upload the file from the Client on to the
Listener's
machine....
*/
public class upload_file extends JFrame
{
  upload_file() // Constructor...
  {
    String str1 ="";
    Vector parameters = new Vector();
    try
    {
      InetAddress inet = InetAddress.getByName(ip_address);
      file_socket = new Socket(inet,7070); // Establish a socket
connection with
      // the Listener on the port 7070
      // address -- inet..
521.  // Get the output as well as the input streams on that
socket...
522.
523.  BufferedOutputStream out = new
BufferedOutputStream(file_socket.getOutputStream());
524.
525.  BufferedInputStream br_socket = new
BufferedInputStream(file_socket.getInputStream());
526.
527.  // if the upload is in a root... then pop up a message for
denial of
528.  // download..
529.
530.  // if the flag is a 0... then pop up a message for denial of
531.  // download..
532.
533.  if (status_text.equalsIgnoreCase("ROOT"))
534.  {
535.    JOptionPane.showMessageDialog(this,"Cannot Upload In Root. Try sub Folders.","Peer 2 Peer...",JOptionPane.INFORMATION_MESSAGE);
536.  }
537.  else if (flag_info.equals("0"))
538.  {
539.    JOptionPane.showMessageDialog(this,"Cannot Upload In Read Only Folder. Try other Folders.","Peer 2 Peer...",JOptionPane.INFORMATION_MESSAGE);
540.  }
541.  else // Else if flag = 1
542.  {
543.    JFileChooser jfc = new JFileChooser(); // Call an object of JFileChooser
544.     // File Dialog to choose the file.
545.     int button_pressed = jfc.showOpenDialog(this);
546.     File file_final = jfc.getSelectedFile();
547.     if (button_pressed == JFileChooser.APPROVE_OPTION)
548.     {
549.       str1 = file_final.getPath(); // Get the path of the file
550.     }
551.     String temp = str1.substring(str1.lastIndexOf("\")+1);
552.     temp = status_text+temp;
553.     XmlWriter writer = new XmlWriter(); // Call a class
XMLWRITER to generate
554.     writer.requestFString("UPLOAD",temp); // request by using a function
555.     String file_data = writer.returnRequest();
556.     byte file_bytes[] = file_data.getBytes(); // get the bytes from the
563. // request...
564.  
565.  BufferedInputStream file_read = new BufferedInputStream(new
566.  FileInputStream(str1)); // Create an inputstream to that path...
567.  
568.  int upload_file_size = file_read.available();
569.  int file_size = file_bytes.length;
570.  
571.  byte b[] = new byte[1024];
572.  // Adjust the request length to 1024
573.  // (for c#) listeners...
574.  
575.  // another method of the class add_on is used now
576.  (appropriate length) this
577.  // is used so as to make the request sent by the client to
578.  // the listener
579.  // 1024 in length..(for c#) listeners...
580.  
581.  // The method takes a byte array and its length as parameters
582.  // and returns
583.  // a byte array of length 1024 bytes....
584.  
585.  add_on upload = new add_on();
586.  
587.  b = upload.appropriateLength(file_bytes, file_size);
588.  
589.  out.write(b,0,1024); // The byte array is written on the
590.  // output stream
591.  
592.  int y = 0;
593.  byte f[] = new byte[1024];
594.  // Generate a progress monitor to monitor the request...
595.  
596.  ProgressMonitor pm = new ProgressMonitor(this,"Uploading
597.  File..","Uploading Please Wait...",0,upload_file_size);
598.  
599.  int current = 0;
600.  //pm.setMillisToPopup(5);
601.  
602.  while ((y = file_read.read(f,0,1024))>0)// Read the file and
603.  // write on socket
604.  {
605.    
606.    out.write(f,0,y);
607.    current = current + y ;
608.    pm.setProgress(current); // Monitor the current
609.  // activity...
610.  }
611.  
612.  out.close();
613.  
614.  br_socket.close(); // Close the streams...
615.  
616. }
617.  
618. try
619.  
620. catch(Exception es)
Chapter 4: The P2P Application in Java 149

610.   {
611.     file_socket.close();  // Close the socket...
612.   }
613.  catch (Exception e)
614.   {
615.    System.out.println( "Some Error while Closing : " +e );
616.   }
617.
618.   }  // End Constructor....
619. }   // END UPLOAD FILE...
620.
621. }   // End File_Gui...

Code description

♦ Lines 1-8: This includes the basic packages used by the various classes to build this application. The packages include java.net (sockets), java.io (file streams), javax.swing (frames), javax.swing.event (event handling), and so on.

♦ Lines 20-31: This declares a public class (file_gui) that represents the base class for this file. All the variables defined here represent their respective controls that on the frame. These variables include file_open; file_download; file_upload; file_search; and file_close that are of the type JButton; file_listing, which is of the type JTable, and status, which is of the type JLabel, and so on.

♦ Lines 32-45: This declares the user-defined variables used for various purposes during the course of the program. This includes variables for URLConnection; InputStream; sockets; string variables (ip_address, information[][]), status_text, flag_info); and variables of the type boolean for various checks during the program.

♦ Lines 59-114: The code between these lines pertains to initializing the variables with parameters and initializing the GUI components (that is, the buttons, the table, and the scrollbar positioning). Mnemonics are supplied for faster access through the keyboard.

♦ Lines 121-127: A condition is applied to disable all buttons except the Close button when the View Files button is pressed; while viewing the files, the client cannot make downloads or uploads.

♦ Lines 129-134: This code pertains to adding all the GUI components to the frame.

♦ Lines 142: An important formatting function is called with the parameter of a Vector value. This helps in extracting information stored in the Vector and placing information in a string two-dimensional array in proper format for later reference; this also places the information on the JTable.

♦ Lines 146-150: Positions and initializes the GUI component (JTable).

♦ Lines 155-161: This adds a scroll facility to the JTable by adding the JScrollPane component and the scrollbar to the frame.

♦ Lines 166-177: This Adds the window listener to the GUI window and adds the ActionListener to various buttons.

♦ Lines 184-290: This includes the code for actions performed on various buttons.

 • 184-290: This includes the code executed when the client selects a particular listing from the table and presses the Open button (file_open).

 • 196: The row selected by the client is identified and based on the selection the information is obtained whether the selection is a file or a folder.

 • 198-209: If a folder is selected and the Open button is pressed, an object of the class add_on is created (request). The object is responsible for sending the request to the listener by using a function of the class search_request. The parameters passed to the function are the folder
name, followed by the search criteria. In this case, the parameter is ".\.*", followed by the
ip_address of the listener, followed by the flag information of the folder searched for.

• 211-215: If a file is selected, a message box is displayed, as a file cannot be opened over a
network.

• 220-223: This includes the code executed when the client presses the Close button. Then the
current frame disappears.

• 224-241: This includes the code executed when the client selects an item from the table and
presses the Download button to download that option.

• 229-239: When the Download button is pressed, a class called **SwingWorker** provided by Sun
is called. This class helps in running a task in a separate thread and thus helps in multithreading
involving a GUI. A method **construct** is declared, which returns a class **download_file**
responsible for downloading the file. **SwingWorker** can be downloaded from
http://java.sun.com/docs/books/tutorial/uiswing/components/example-swing/SwingWorker.java.

• 242-263: This includes the code executed when the client presses the Upload button to upload
certain files from its system on to the listener’s machine if proper permissions are there.

• 248-258: When the Upload button is pressed, a class called **SwingWorker** provided by Sun is
called. This class helps in running a task in a separate thread and thus helps in multithreading
involving a GUI. A method **construct** is declared, which returns a class **upload_file**, which is
responsible for uploading the file.

• 264-287: This includes the code executed when the client presses the Search option to search for
a particular subset of files available. When the Search button is pressed by the client, root
information is checked. If root_information is "ROOT", a message box is shown indicating
that no search can be done if root_information is not "ROOT", a class search screen is called
that caters to the search.

• 272-275: If status_text is "ROOT", a message box is displayed; no search is possible on
root.

• 284-285: If search text is not root, create an object of the class **search_screen**; the parameters
passed to the constructor are: 1. Status_text. 2. The IP address of the listener. 3. The
flag_info of the folder on which the search is being performed. A GUI is also shown on top of
the existing frame; this GUI enables searching by specifying certain criteria.

♦ Lines 300-394: An important "formatting" function is called with the parameter a Vector value. This
helps in extracting information stored in the Vector and placing information in a string two-
dimensional array in proper format for later reference. This also places information on the JTable.

• 304-319: This checks whether the Vector has more than one value or not. If not, initialize a two-
dimensional array of the type String with the size of the Vector.

• 324-370: Loop through the Vector till the end, and extract the information and place it in the
information array as well as in the data array (used by the JTable).

• 383-393: If the size of the Vector is less than 1, simply put the values in both the arrays as
nonshareable.

♦ Lines 400-502: This includes the source code for the inner class **download_file**. This is used
when the user selects a file and presses the Download button. This inner class handles all the
functionality; that is, connecting to the listener’s machine, placing the file on the desired path
(JFileChooser), showing the progress on the monitor while downloading is in progress, and so
on.

• 407: The row selected by the client is identified and based on the row on which the attribute of
the selection is obtained (whether file/folder).

• 409-410: This creates a client_socket on the listener’s machine on port 7070.
Chapter 4: The P2P Application in Java 151

- 415-417: This attains the output as well as the input Streams on that socket.
- 421-424: If the selection made by the user turns out to be a folder, pop up a message box indicating that the a folder cannot be downloaded and that the user should try opening it instead.
- 427-430: If the request is that of a file, call a class XmlWriter to generate a request by using a function.
- 435-447: This calls an object of JFileChooser File dialog to place the file and to get the path where the file is being saved.
- 449: This creates an outputstream to that path. (the path obtained by the JFileChooser).
- 462-463: A method of the class add_on is used now (appropriatelenlength); this is used to make the request sent by the client to the listener 1024 bytes in length (for C# listeners).
- 465: The byte array is written on the output stream.
- 471-473: This generates a progress monitor to monitor the request.
- 475-482: The socket input stream is read and written to the file output stream, also updating the progress monitor in between.
- 484-500: This closes the file streams as well as the sockets and also handles the exceptions through the try... catch blocks.

◆ Lines 509-619: This includes the source code for the inner class upload_file. This is used when the user presses the Upload button. This inner class handles all the functionality (that is, checking for permissions, connecting to the listener’s machine, placing the file on the desired path (JFileChooser), showing the progress on the monitor while the uploading is in progress, and so on.
  - 517-518: This creates a client_socket on the listener’s machine on port 7070.
  - 523-525: This attains the output as well as the input streams on that socket.
  - 533-540: If the folder in which the user wants to upload the file turns out to be a read-only folder, pop up a message indicating the status. Also, pop up a message if the root folder turns out to be “ROOT”.
  - 543-555: This calls an object of JFileChooser File dialog to choose the file to upload and gets its path.
  - 557-560: This calls a class XMLWRITER to generate a request by using a function. Also, this appends the filename with the path on the listener’s machine.
  - 565: This creates an input stream to that path (path of the file to upload)
  - 581-582: A method of the class add_on is used now (appropriatelenlength). This is used to make the request sent by the client to the listener 1024 bytes in length (for C# listeners).
  - 584: The byte array is written on the output stream.
  - 590: This generates a progress monitor to monitor the request.
  - 595-600: The file input stream is read and written to the socket output stream, also updating the progress on the monitor in between.
  - 602-616: This closes the file streams as well as the sockets and also handles the exceptions through the try... catch blocks.

search_screen.java

This class is used when the client/user uses the search option from the previous GUI (see Listing 4-11). This search is different from the server search, as it searches the listener’s machine (a particular user) for files specified in the search criteria (see Figure 4-6).
After a search is completed, the class `file_gui.java` is called again to display the search results.

**Figure 4-6: search_screen**

**Listing 4-11: search_screen.java**

```
//© 2001 Dreamtech Software India Inc.
// All rights reserved

1. import javax.swing.*;
2. import javax.swing.event.*;
3. import java.awt.event.*;
4. import java.awt.*;
5. import java.util.*;
6. /* This class implements Searching for files on the Listener....*/
7. public class search_screen extends JFrame implements ActionListener
8. {
9.     JLabel label; //
10.    JTextField search_text; // GUI Components for various purposes..
11.    JButton search_ok, search_cancel; //
12.    Container contentpane; // For placing the GUI Components..
13.    String root_information;
14.    String ip_address;
15.    String flag_info = "";
16.    /* When this class is called from the "file_gui.class" by pressing the search button it is provided with three parameters...
17.    root ... The root/directory on which search is being performed. This cannot be the root directory...
18.    ip ... The ip address of the listener's machine...
19.    flag ... The flag information of the root/ directory...
20.    */
21.    public search_screen(String root, String ip, String flag )
22.    {
23.        super("Search");
```
36. setSize(270,110); // Size of the Search Screen Frame...
37. 
38. contentpane = getContentPane();
39. contentpane.setLayout(null); // Setting the Layout to Absolute Layout..
40. 
41. label = new JLabel("Search Criteria "); // Initializing the GUI Component.
42. label.setBounds(10,10,100,25); // Positioning the GUI Component.
43. 
44. search_text = new JTextField(30); // Initializing the GUI Component.
45. search_text.setBounds(110,10,150,25); // Positioning the GUI Component.
46. 
47. search_ok = new JButton("Search"); // Initializing the GUI Component.
48. search_ok.setMnemonic('S'); // Setting the Mnemonic.
49. search_ok.setBounds(45,50,80,25); // Positioning the GUI Component.
50. 
51. search_cancel = new JButton("Cancel"); // Initializing the GUI Component.
52. search_cancel.setMnemonic('C'); // Setting the Mnemonic.
53. search_cancel.setBounds(150,50,80,25); // Positioning the GUI Component.
54. 
55. search_ok.addActionListener(this); // Add action listener to all the
56. search_cancel.addActionListener(this); // Buttons...
57. 
58. // Adding the window Listener to the gui window
59. // ie. Code for the "cross"
60. 
61. addWindowListener (new java.awt.event.WindowAdapter () {
62.         public void windowClosing (java.awt.event.WindowEvent evt) {
63.           setVisible(false);
64.         }
65.       });
66. 
67. 
68. contentpane.add(label); // Adding the GUI Buttons.....
69. contentpane.add(search_text); // Adding the
70. contentpane.add(search_ok); // Adding the
71. contentpane.add(search_cancel); //
72. 
73. 
74. root_information = root; // Initializing the variables with parameters...
75. ip_address = ip; // Initializing the variables with
76. flag_info = flag; // parameters...
77. 
78. } 
79. 
public void actionPerformed(ActionEvent ae) {
if (ae.getSource() == search_ok) // When Ok Button Is pressed....
{
this.setVisible(false); // Hide this GUI screen
// another method of the class add_on is used now (search request)
this
// to send the listener the search criteria. For sending the search
// criteria the function search_request takes four parameters....
// 1. root_information .. On Which the search request is being
made...
// 2. search text ie. the text entered by the user on the text field
// provided.
// 3. ip_address of the listener's machine....
// 4. Flag information of the directory on which request is made...

add_on search_with_condition = new add_on();
search_with_condition.search_request(root_information,search_text.getText(),ip_address,flag_info);
}
else if (ae.getSource() == search_cancel) // When Cancel Button Is pressed....
{
this.setVisible(false); // Hide this GUI screen
}
}
• 82-98: This includes the code executed when the client, after entering certain search criteria in the JTextField, presses the Search button to commence the searching operation.

• 96-97: A method of the class add_on is used now (search request). This is used to send to the listener the search criteria. For sending the search criteria, the function search_request takes four parameters: root_information on which the search request is being made; search text, the text entered by the user on the text field provided; the ip_address of the listener's machine; and flag information of the directory on which the request is made.

• 99-102: If the Cancel button is pressed, the search window hides.

add_on.java

This is a multiutility class, as it has functions used by nearly all classes (see Listing 4-12). These functions are as follows:

♦ search_request(): Sends the search request to a particular listener. The search request can open/browse a folder, in which case “*.*” is passed or can be requested based on a specific criteria (“*.java”, “ja*.ja*”, and so on).

♦ appropriate_length(): Used to adjust the length of the request sent to the listener to 1024 bytes. This adjustment is made to balance request handling by the “C#” and java listeners.

♦ Sorting(): Used to implement sorting (the bubble sort technique is a commonly used technique of sorting records). In this the smallest or the largest record depending upon the condition bubbles comes out after each iteration so that the listeners’ names appear in alphabetical order.

♦ Start_connection(): Used to send requests to the server; hence, this method is used two times: when the request is sent to the server to provide a list of the users connected at a particular time and when the search_window class is called to implement server-level searches.

Listing 4-12: add_on.java

// © 2001 Dreamtech Software India Inc.
// All rights reserved

1. import java.net.*;
2. import java.io.*;
3. import java.util.*;
4. /* This class file is a multipurpose class file it contains many functions which are used
5. over a series of classes in the project...
6. */
7. class add_on
8. {
9.   Socket file_socket;
10.  String viewfiles[];
11.  static Vector filenames ; // A vector to maintain the user list (static)
12.  
13.  
14.  add_on() // Constructor...
15.  {
16.    
17.    }
18.  
19.  /* This function is used to adjust the length of the byte array and to make it
equal to 1024 bytes. This is done in order to make the size of the
request...
equal to the request accepted by the C# listener...

In the function appropriatelen...  

two parameters are passed
1. Byte array...
2. Length of byte array...

The rest of the bytes (1024 - length) are first converted into character and then made into char ’13’ and then converted into bytes again..

/**

public byte[] appropriatelen(byte[] file_bytes, int file_size)
{
    int count = 0;
    byte b[] = new byte[1024];
    int remaining = 1024 - file_size;
    for (int i = 0; i < file_bytes.length; i++)
    {
        b[i] = file_bytes[i];
    }
    char a[] = new char[remaining];
    for (int i = 0; i < remaining; i++)
    {
        a[i] = 13;
    }
    String tempw = new String(a);
    byte d[] = tempw.getBytes();
    for (int i = file_size; i < 1024; i++)
    {
        b[i] = d[(i - file_size)];
    }
    return (b);
} // End Appropriate length.....

/* This function is used to issue the search request...
It takes 4 parameters...
1. Directory information...
2. Search Criteria...
3. Ip_address of the listener...
4. Flag_information of the directory on which search is made...
*/

public void search_request(String directory, String condition, String ip_address, String flag_info)
{

try {
    InetAddress inet = InetAddress.getByName(ip_address);
    file_socket = new Socket(inet, 7070); // Create a client_socket on the
    // Listener's machine at port 7070.

    // Get The output as well as the input Streams on that socket...
    BufferedOutputStream out = new BufferedOutputStream(file_socket.getOutputStream());
    BufferedInputStream br_socket = new BufferedInputStream(file_socket.getInputStream());

    // Now a request is sent to the Listener to show all the shareable files
    // of the particular directory that satisfy the search criteria..selected
    // by the user..
    // To send a request a class of the name xmlwriter is used which has two
    // functions of interest. They are... requestFString(String, String) and
    // returnRequest(), this xmlwriter is a versatile class as it is used to
    // generate xml requests for various purposes...
    // and SHOWFILES... therefore accordingly the requestFString takes two
    // parameters in case of SHOWFILES the second parameters is not used.
    XmlWriter writer = new XmlWriter(); // Initializing a object of xmlwriter..
    writer.requestFString("SEARCH", directory+condition); // calling the
    // function...
    String file_data = writer.returnRequest(); // getting the request in a
    // temp variable file_data.

    byte file_bytes[] = file_data.getBytes(); // getting byte array of string
    // An output stream is also initialized this is used to store all the
    // response from the listener..
    BufferedOutputStream out_file = new BufferedOutputStream(new FileOutputStream("response.xml"));
int file_size = file_bytes.length;

byte b[] = new byte[1024];  // Initializing a new byte array of 1024.

// another method of the same class add_on is used now

// this is used so as to make the request sent by the client to the
// listener 1024 in length..(for C#) listeners...

b = appropriatelenyth(file_bytes, file_size);

out.write(b, 0, 1024);  // The byte array is written on the output stream

int y = 0;
byte f[] = new byte[32];

while ((y = br_socket.read(f, 0, 32)) > 0)  // the socket input stream is read
{
    out_file.write(f, 0, y);  // written on to the file output stream...
}

out.close();  // The filestream and socket streams are
br_socket.close();  // A class SParser is also used here this class has a
function/method of parse the xml file
// the name perform which calls the xml parser to
// generated by the response from the client socket...
// the function perform returns a Vector which has the
// along with their flag information and size in case of files....

SParser sp = new SParser();
Vector parameters = sp.perform("response.xml");
file_gui showfiles = new file_gui(parameters, ip_address, directory, flag_info);
showfiles.show();

catch (Exception e)
{
    System.out.println( "Exception in search_request " + e );
}

try
{
    file_socket.close(); // Close the Socket...
}

} // End Search request....

/* This is a simple sorting function which implements bubble sort method to sort the Listeners’ names lexicographically...
It takes as its argument a 2-D array of the listeners’ names and a flag to call */

public String [][] sorting(String info[][], boolean cond)
{
    int k = 0;
    while (k < (info.length-1))
    {
        int i = 0;
        while (i < (info.length - k - 1))
        {
            if((info[i][0] != null) &&(info[i+1][0] != null))
            {
                if (info[i][0].compareToIgnoreCase(info[i+1][0]) > 0)
                {
                    String temp = info[i][0];
                    info[i][0] = info[i+1][0]; // Swapping operation..
                    info[i+1][0] = temp;
                }
            }
            temp = info[i][1];
            info[i][1] = info[i+1][1]; // Swapping operation..
            info[i+1][1] = temp;
            if (cond)
            {
                temp = info[i][2];
                info[i][2] = info[i+1][2]; // Swapping operation..
                info[i+1][2] = temp;
            }
            i++;
        }
    }
}
Chapter 4: The P2P Application in Java

```java
202.     k++;
203. }
204. }
205. 
206. return info; // Returning the 2-D sorted array...
207. }
208. }
209. /* This function/method is used everytime an object of class Client is
210.     it is used to get Listeners’ List, Search List from the server...After
211.     the information, it arranges the information in proper format in a 2_D
212.     returns the array to the class which invoked the function...
213. 
214. The parameters this method takes are..
215. 1. param -- Criteria for calling the function (Search/Root).
216. 2. us -- if param equals Search then this will hold the Search
217. Criteria..
218. 3. fs -- if param equals Search then this will hold the Search
219. Criteria..
220. 4. present_users[][] -- Present users connected to the server...
221. */
222. public String[][] start_connection(String param, String us, String fs,
223. String present_users[][])
224. {
225. URLConnection urlconnection;
226. InputStream url_inputstream; // For Connecting and gaining the
227. information..
228. String information[][] = {{ " ", " "};
229. Vector values = new Vector();
230. boolean go_on = true;
231. boolean search_flag = false;
232. try
233. {
234. if (param.equalsIgnoreCase("search")) // if Search
235. {
236. // Call The ASP with proper format and parameters and
237. initialize a vector to
238. // store the information generated from this request..
239. urlconnection = (new URL("http://www.s-
240. cop.com/search.asp?us=\"+us+\"&fs=\"+fs\")).openConnection();
241. urlconnection.connect();
242. search_flag = true;
243. filenames = new Vector();
244. filenames.add(0,(Object)"filesearch");
245. }
246. else // if root
247. {
248. urlconnection = (new URL("http://www.s-
249. cop.com/userlist.asp")).openConnection();
250. urlconnection.connect();
```
```java
248.    search_flag= false;
249. }
250. StringBuffer sb = new StringBuffer();
251. try
252. {
253.    url_inputstream = urlconnection.getInputStream(); // get the
254.    // read the response from the request and store it in the
255.    // response.xml file...
256.    BufferedReader br = new BufferedReader(new
257.    InputStreamReader(url_inputstream));
258.    BufferedWriter file_output = new BufferedWriter(new
259.    OutputStreamWriter(new FileOutputStream("response.xml"),32));
260.    int i = 0;
261.    char[] b = new char[32];
262.    String string = "";
263.    while ((i = br.read(b,0,32)) > 0 )
264. {
265.        String temp = new String(b,0,i);
266.        string = string +temp;
267.    }
268.    string = string.trim();
269.    char d[] = string.toCharArray();
270.    file_output.write(d,0,d.length);
271.    br.close(); // Close the inputStream..
272.    file_output.close(); // Close the inputStream..
273. }
274. catch(Exception ef)
275. {
276. }
277. // A class SParser is also used here this class has a function/method
278. // of
279. // the name perform which calls the xml parser to parse the xml file
280. // generated by the response from the client soket..
281. // the function perform returns a Vector which has the
282. // files/directories,
283. // along with their flag information and size in case of files....
284. SParse sp = new SParse();
285. values = sp.perform("response.xml");
286. if (param.equalsIgnoreCase("search")) // if the param is search then
287. {
288.    // storing is done in the
289.    // following format..
290.
```
information = new String[present_users.length][2];
viewfiles = new String[values.size()][3];
int i = 1;
int g = 0;
int count = 0;

while(i<(values.size()))
{
    viewfiles[g][0] = (String)values.get(i);
i++;
    viewfiles[g][1] = (String)values.get(i);
i++;
    viewfiles[g][2] = (String)values.get(i);
i++;
    String temp= "";
    temp = viewfiles[g][0]+"?"+viewfiles[g][2]+"?";
    filenames.add((Object)temp);
    temp = "";
    g++;
}

for (int index = 0;index<present_users.length ;index++ )
{
    if (present_users[index][0] != null)
    {
        int inf  = present_users[index][0].indexOf("(");
        if (inf != -1)
        {
            present_users[index][0] = present_users[index][0].substring(0,inf);
        }
    }
}

for (int index = 0;index<present_users.length ;index++ )
{
    if (present_users[index][0] != null)
    {
        int inf  = present_users[index][0].indexOf("(");
        if (inf != -1)
        {present_users[index][0] = present_users[index][0].substring(0,inf);
        }
    }
}

for (int index = 0;index<present_users.length ;index++ )
{
    if (present_users[index][0] != null)
    {
        count = 0;
        for (int temp = 0;temp<g ;temp++ )
        {
            if (viewfiles[temp][1].length()>=present_users[index][0].length())
            {
                // Code continues here...
            }
        }
    }
}
if (present_users[index][0].equalsIgnoreCase(viewfiles[temp][1].substring(0,present_users[index][0].length())))
    {
        count++;
    }

present_users[index][0] = present_users[index][0] + "(" + count + " )";
}

information = present_users;

}

else {
    if (values.size()>1)
    {
        information = new String[values.size()][2];
        int i = 1;
        int g = 0;
        while(i<(values.size()))
        {
            information[g][0] = (String)values.get(i);
            i++;
            information[g][1] = (String)values.get(i);
            i++;
            g++;
        }
    }
    else
        go_on = false;
}

if (!go_on)
{
    information = new String[1][2];
    information[0][0] = "Not Connected...";
    information[0][1] = " - ";
}
else
{
    information = sorting(information,false);  // Sorting is performed,,
}

// End Try..
catch(Exception e)
{
  go_on = false;
}
// End Else....

turn (information);
// return the sorted list...

public Vector returnfilenames()
{
  return filenames;
}

Code description

- Lines 1-3: This includes the basic packages used by the various classes used to build this application. The packages include java.net (sockets), java.io (file streams), and java.util (utility).
- Lines 8-12: This declares a public class (client) that represents the base class for this file. This declares the user-defined variables, which are used for various purposes during the course of the program. This includes variables for socket, string, and a static-vector connection.
- Lines 32-59: This includes the source code for appropriatelenhth(). This function is used to adjust the length of the byte array and to make it equal to 1024 bytes. This is done to make the size of the request equal to the request accepted by the C# listener. In the function appropriatelenhth(), two parameters are passed: byte array and length of byte array. The rest of the bytes (1024-length) are first converted into character and then made into char 13 and then converted into bytes again. This char 13 is used for the compatibility issue, so that the Java client can be used with C# listener. Char 13 is treated as blank white space. Any other character could create a problem while parsing the data.
- Lines 71-162: This includes the source code for search_request(). This function is used to issue the search request. This function takes four parameters: Directory information, Search Criteria, Ip_address of the listener, and Flag_information of the directory on which the search is made.
  - 76-77: This creates a client_socket on the listener’s machine on the port 7070.
  - 81-83: This gets the output as well as the input streams on that socket.
  - 96-97: Now a request is sent to the listener to show all the shareable files.
  - 108: An output stream is also initialized. This is used to store the response from the listener.
  - 119: A method of the same class add_on is used now (appropriatelenhth) to make the request sent by the client to the listener, 1024 bytes in length (for C# listeners).
  - 121: The byte array is written on the output stream.
  - 126-129: The socket input stream is read and written to the file output stream.
  - 131-133: Closing the file streams as well as the sockets and also handling the exceptions through the try...catch blocks.
• 142-143: The class $\text{SParser}$ is also used here. This class has a function/method of the name $\text{perform}$ that calls the XML parser to parse the XML file generated by the response from the client socket. The function returns a Vector, which is stored in the Temp Vector.

• 144-145: The Vector value returned by the XML parser is then passed as one of the parameters to a class named $\text{file\_gui}$. This class is responsible for displaying a GUI consisting of a table and some buttons, along with the root information and flag.

♦ Lines 171-207: This includes the code for the sorting function. This is a simple sorting function that implements a bubble sort method to sort the listeners' names in alphabetical order. The $\text{Sorting}$ function takes as its argument a two-dimensional array of the listeners' names and a flag and returns an array of the same type as that of the input parameter.

♦ Lines 221-410: This includes the code for the $\text{start\_connection}$ function. This function/method is used every time an object of class $\text{Client}$ is invoked. It is used to get a listener's list, search list from the server. After retrieving the information, the function arranges the information in a two-dimensional array and returns the array to the class that invoked the function. The parameters this method takes are: $\text{param}$ (criteria for calling the function Search/Root); $\text{us}$ (if $\text{param}$ equals $\text{search}$, then this holds the search criteria for a user search); $\text{fs}$ (if $\text{param}$ equals $\text{search}$, this holds the search criteria for a file search); and $\text{present\_users}[][]$ (present users connected to the server).

• 223-228: This declares the user-defined variables used for various purposes during the course of the program; this includes variables for $\text{URLConnection}$, $\text{inputstream}$, and variables of the type $\text{boolean}$ for various checks during the program.

• 233-243: If the $\text{param}$ value is equal to “$\text{SEARCH}$,” an ASP ($\text{search.asp}$) is called. This also initializes a Vector to store the information generated from this request.

• 244-249: Otherwise, an ASP ($\text{userlist.asp}$) is called.

• 253: This gets an inputstream to the URL connection created either in line no. 233-243 or 244-249.

• 258: An output stream is also initialized, which is used to store the response from the listener.

• 263-270: Read from the socket and write to file.

• 272-273: Close the streams.

• 288-289: Class $\text{SParser}$ is also used here. This class has a function/method of the name $\text{perform}$, which calls the XML parser to parse the XML file generated by the response from the client socket. The function returns a Vector, which is stored in the Temp Vector.

• 292-364: If the parameter passed is “$\text{Search}$”, arrange the Vector received in proper format by applying the proper delimiters “?”. If the param is not “$\text{Search}$”, arrange the values in the Vector in the array $\text{information}[][]$.

• 388-393: If no value is in the Vector, indicate in the array that no files are shareable.

• 394-397: If information has some values, function sorting is called to sort the values.

• 408: Information is returned.

♦ Lines 412-416: A function $\text{returnfilenames}$ is called here. This just returns the file names to the calling program; this function is called when the client presses the View Files button in the class $\text{client.java}$.

Summary
This chapter explained how to create a full-fledged peer-to-peer application using Java 2 Standard Edition and XML.
We have explained the concept of Peer to Peer Application by discussing the role of XML in our application, along with how to parse the XML (SParser.java). The class parses the XML response returned from the server and returns a Vector to the calling program. Another class has been discussed (XMLWriter.java). This class is used by both the listener program and the browser program as, it generates the XML requests, thus illustrating the fact that the XML forms the basis of client/server communication.

The listener program was explained. This program forms the basis of the peer-to-peer application. It makes use of a class Login.java, which registers the listener onto the server and then listens for requests on a specified port (7070, in our case). The listener program shares the files/folders to be viewed by the browser by using a class shareddilog.java. For implementing the listener-level search facility, the program uses the check_directory.java class. It has various features to implement any kind of search using wildcards.

After the listener program, the browser program was explained. The browser browses the list of all connected users and then connects to a user; the browser browses for files and folders shared on the user machine and uploads and downloads the files on his or her machine. The browser uses various classes. The most important class is client.java, which searches for various listeners connected. The file_gui, on the other hand, provides the detail of files/folders shared by a listener. The search_screen.java and the search_window.java implement listener-level and server-level searches, respectively.

In Chapter 5, we discuss the implementation of the peer-to-peer application by using C#.
Chapter 5
The P2P Application (C#)

This chapter introduces you to the task of cracking code and illustrates all aspects of detailed programming. In the C# version of this P2P application, the implementation is explained in detail so that you gain a sound understanding of the concepts. The code in this chapter has been written in Microsoft Visual Studio .NET Beta 2. As soon as the final version of Microsoft Visual Studio .NET is released, you can download the executables for this application from www.dreamtechsoftware.com/download.

The code explanation starts with a discussion on the ServerCommunication class (Listing 5-1), which is in the WorkingWithXML namespace. This class carries functions that are very handy to use, such as functions for communicating with the server, getting the IP address for a particular machine, and so on. Further, WorkingWithXML describes the class used to create XML requests/responses, the XMLCreater class, which also resides in the WorkingWithXML namespace. The functions of this class take some parameters, and depending upon the parameters, they either generate a request or a response. Finally, one more class is in the same namespace: XMLParser. This class is used for the parsing of the XML request/response.

While going through the chapter, you may find some lines of code in the WorkingWithXML class that are required by the add-ons (discussed in Chapters 6 and 7). You may skip these lines of code now, but you may need to come back and refer to them while going through Chapters 6 and 7.

After discussing the building blocks of our application, we discuss the application itself. This comprises a listener software and a browser software (client). These two interact with each other to give final shape to this P2P application. The listener, as the name suggests, listens to the requests sent by the browser and also sends a response to listeners. The browser is responsible for showing the response in a user-friendly manner so that the end user can interpret it.

ServerCommunication Class

namespace: WorkingWithXML

The name of the class is ServerCommunication, which is responsible for communication with the server.

Listing 5-1: ServerCommunication Class

// © 2001 Dreamtech Software India Inc.
// All rights reserved

```
1 namespace WorkingWithXML
2 {
3     using System; // Provides the basic functionality of .NET
4     using System.Net; // Provides the net related functionality
5     using System.IO; // Provides the I/O functionality
6     using System.Text; // Provides text based manipulations
7     using System.Windows.Forms; // Provides the use of graphic interface
8     using System.Web;
```
/// <summary>
/// Summary description for ServerCommunication.
/// This class is responsible for all the communication with
/// the server as well as Listener
/// It has got some handy functions which can be very helpful
/// like: GetIPAddress, FileDelete etc
/// </summary>
public class ServerCommunication
{
    /// <summary>
    /// Default constructor of the class
    /// This constructor of the class is generated automatically
    /// by the IDE
    /// </summary>
    public ServerCommunication()
    {
    
    }

    /// <summary>
    /// Get the response data from server represented by WebAddress
    /// When the request is made to the server it opens a stream
    /// to the response and the function reads bytes from that
    /// response stream and converts them to string and returns the value
    /// </summary>
    /// <param name="WebAddress"> </param>
    public string GetDataFromServer(string WebAddress)
    {
        // Declares a local variable webRequest of type HttpWebRequest
        // which is a part of System.Net package. It is used to form
        // an HttpRequest and sends it to the server
        HttpWebRequest webRequest;

        // Declares a local variable webResponse of type HttpWebResponse
        // which is a part of System.Net package. It is used to get the
        // response from the server against the HttpWebRequest
        HttpWebResponse webResponse;

        // Declares a variable responseStream of Stream type which is
        // used to get the response from the server's end
        Stream responseStream;

        // streamBuffer variable is declared here of type Byte array
        // this is used to read the actual data from the responseStream
        Byte[] streamBuffer;

        // Declares a variable ReturnData of type string which
        // at last stores the data which is to be returned
        string ReturnData;

        // initializes the ReturnData to null
        ReturnData = null;

        // Creates and initializes a webRequest by calling Create
        // function of the WebRequestFactory and type cast it to
        // HttpWebRequest type
webRequest = (HttpWebRequest)WebRequest.Create(WebAddress);

// After requesting the server for HttpWebRequest
// it will open a response for the clients end to read
// this response is catched by the following line of code
// and assigns it to webResponse
webResponse = (HttpWebResponse)webRequest.GetResponse();

// GetResponseStream method of webResponse actually gets
// the response stream and assigns it to the responseStream
responseStream = webResponse.GetResponseStream();

// initialize streamBuffer so that it can read 16 bytes of data
// at a time
streamBuffer = new Byte[16];

// Declares int variable iBytesRead which keeps the
// records of how many bytes have been read from the
// stream
int iBytesRead;

// Reads 16 bytes from the stream until the stream gets
// empty and the value assigned to iBytesRead is zero
while( 0 != (iBytesRead = responseStream.Read(streamBuffer, 0, 16)) )

    // This will convert the bytes data that is read from the
    // stream and stored in streamBuffer to string and
    // concatenates
    // it to ReturnData
    ReturnData += Encoding.ASCII.GetString(streamBuffer, 0, iBytesRead);

// Removes the leading and trailing spaces from the Data
if( ReturnData != null )
    ReturnData = ReturnData.Trim();

// Returns the value of ReturnData variable
return ReturnData;

/// <summary>
/// Get the response data from server represented by WebAddress
/// using Proxy server. When the request is made to the server it
/// opens a stream to the response and the function reads bytes
/// from that response stream and converts them to string and
/// returns the value
/// </summary>
/// <param name="WebAddress"> </param>
public string GetDataFromServerUsingProxy(string WebAddress, string ProxyAddress, int Port)
{
    // Declares a local variable webRequest of type HttpWebRequest
    // which is a part of System.Net package. It is used to form
    // an HttpRequest and sends it to the server
    HttpWebRequest webRequest;

    // Declares a local variable webResponse of type HttpWebResponse
    // which is a part of System.Net package. It is used to get the
    // response from the server against the HttpWebRequest
    HttpWebResponse webResponse;

    // Declares a variable responseStream of Stream type which is
    // used to get the response from the server's end
    Stream responseStream;

    // streamBuffer variable is declared here of type Byte array
    // this is used to read the actual data from the responseStream
    Byte[] streamBuffer;

    // Declares a variable ReturnData of type string which
    // at last stores the data which is to be returned
    string ReturnData;

    // initializes the ReturnData to null
    ReturnData = null;

    System.Net.IWebProxy ProxyData = new System.Net.WebProxy(ProxyAddress, Port);

    // Creates and initializes a webRequest by calling Create
    // function of the WebRequestFactory and type cast it to
    // HttpWebRequest type
    webRequest = (HttpWebRequest)WebRequest.Create(WebAddress);
    webRequest.Proxy = ProxyData;

    // After requesting the server for HttpWebRequest
    // it will open a response for the clients end to read
    // this response is catched by the following line of code
    // and assigns it to webResponse
    webResponse = (HttpWebResponse)webRequest.GetResponse();

    // GetResponseStream method of webResponse actually gets
    // the response stream and assigns it to the responseStream
    responseStream = webResponse.GetResponseStream();

    // initialize streamBuffer so that it can read 16 bytes of data
    // at a time
    streamBuffer = new Byte[16];
Chapter 5: The P2P Application (C#) 171

171 // Declares int variable iBytesRead which keeps the
172 // records of how many bytes have been read from the
173 // stream
174 int iBytesRead;
175
176 // Reads 16 bytes from the stream until the stream gets
177 // empty and the value assigned to iBytesRead is zero
178 while( 0 != (iBytesRead = responseStream.Read(streamBuffer, 0, 16)) )
179
180 // This will convert the bytes data that is read from the
181 // stream and stored in streamBuffer to string and
182 // concatenates
183 ReturnData += Encoding.ASCII.GetString(streamBuffer, 0, iBytesRead);
184
185 // Removes the leading and trailing spaces from the Data
186 // that is stored is returnData variable
187 if( ReturnData != null )
188 {
189     ReturnData = ReturnData.Trim();
190     // ReturnData =
191     ReturnData = ReturnData.Substring(0, ReturnData.LastIndexOf("</p2p_lng>") +
192         "+".Length );
193 }
194
195 // Flushes the responseStream
196 responseStream.Flush();
197
198 // Closes the responseStream
199 responseStream.Close();
200
201 // Returns the value of ReturnData variable
202 return ReturnData;
203
204 /// <summary>
205 /// Writes the DataToWrite to Filename
206 /// This function writes the string data which is stored in
207 /// DataToWrite variable to the file pointed by Filename
208 /// Usually we write XML file
209 /// </summary>
210 /// <param name="Filename"> </param>
211 /// <param name="DataToWrite"> </param>
212 public void WriteDataToFile(string Filename, string DataToWrite)
213 {
214     // Declares and initializes the FileToCreate variable of
215     // type File and passes Filename to its constructor to
216     // associate it with the File
217     //File FileToCreate = new File(Filename);
218     // Added for Beta 2
219     FileStream WriteStream = new FileStream(Filename, FileMode.Create);
220     // Declares and creates a WriteStream object, used
221     // to write the data to the stream which is present
// in DataToWrite variable
//Stream WriteStream = FileToCreate.OpenWrite();

// Writes the data to the file by converting data
to byte format
WriteStream.Write(Encoding.ASCII.GetBytes(DataToWrite), 0, DataToWrite.Length);

// Closes the written file
WriteStream.Close();

/// <summary>
/// Determine the type of request/response received in XML
/// This is done by extracting the attribute value from the
/// first node value of first child element of the document
/// <summary>
/// <param name="XMLFilename"> </param>
public string TypeOfXMLRecieved(string XMLFilename)
{
    // Declares and initializes a local variable document
    // of type IXMLDOMDocument present in MSXML2 class. This
    // variable is used to point to the XML filename or document
    MSXML2.IXMLDOMDocument document = new MSXML2.DOMDocument();

    // Declares a local variable element of type IXMLDOMElement
    // This is used to point to the elements present in the XML
    MSXML2.IXMLDOMElement element;

    // Declares a local variable node of type IXMLDOMNode
    // This is used to point to the nodes present in the XML
    MSXML2.IXMLDOMNode node;

    // A local variable NodeValue is declared of type string
    // it is used to store the retrieved value from the XML and
    // returns it from the function
    string NodeValue;

    // Read the XML document synchronously
    document.async = false;

    // Initializes NodeValue to null
    NodeValue = null;

    // Loads the XML document for reading
    if( document.load(XMLFilename) )
    {
        // Extract the first element of the XML
        element = document.documentElement;

        // Extract the first child node from the element
        // and stores it to the node
        node = element.firstChild;
    }
Chapter 5: The P2P Application (C#) 173

```csharp
278     // now extract the first node value from the attributes
279     // present in the XML and saves it to NodeValue
280     NodeValue = node.attributes.nextNode().nodeValue.ToString();
281 
282     // Simply returns the NodeValue variable
283     return NodeValue;
284 
285 
286     /// <summary>
287     /// This function is used to delete a file represented by filename
288     /// </summary>
289     /// <param name="Filename"> </param>
290     public void FileDelete(string Filename)
291     {
292         // Declares and initializes an object f of type File which is
293         // present in System.IO package and assigns Filename to it
294         // File f = new File(Filename);
295         // Added for Beta 2
296         File.Delete(Filename);
297         // Calls the Delete function of File Class to delete the
298         // file represented by Filename
299         // f.Delete();
300     }
301 
302     /// <summary>
303     /// Retrieve the IP Address of the machine represented by
304     /// Hostname. This function makes the use of the DNS class
305     /// for extracting the IP address and returns the first entry
306     /// from the IP list obtained
307     /// </summary>
308     /// <param name="Hostname"> </param>
309     public string GetIPAddress(string Hostname)
310     {
311         // Creates a new local variable named LocalHost of type
312         // IPHostEntry which is present in the System.Net package
313         // It then calls the GetHostByName function of the DNS class
314         // and passes the Hostname to it
315         // To retrieve my computer's IP
316         IPHostEntry LocalHost = Dns.GetHostByName(Hostname); // To
317         // Now the LocalHost has got the list of IPs corresponding
318         // to the hostname and it will return the first entry from the
319         // list
320         return LocalHost.AddressList[0].ToString();
321     }
322 
323 }
Code description of ServerCommunication class (clsServerCommunication.cs)

- Lines 3-8: This includes the basic packages needed for the various classes used to build this application. This also includes packages used for communicating with network, streaming facility, file I/O, text manipulations, and so on.
- Line 17: This line declares a public class ServerCommunication.
- Lines 24-26: This is the default constructor of the ServerCommunication class and is not used anywhere in our application for any purpose.
- Lines 35-112: The GetDataFromServer function retrieves the response from the server’s end, that is, the result of calling the ASP pages on the server. The address of the ASP page is passed to this function in the WebAddress parameter of this function. It uses this WebAddress parameter to get the response from the server’s end. This response is saved in string format in ReturnData for later use.
  - 40: A webRequest variable of the HttpWebRequest class is declared. This variable is used to create a Web request by calling the ASP page.
  - 45: A webResponse variable of the HttpWebResponse class is used to get the response to the requested ASP page from the server.
  - 49: A responseStream variable of the Stream class type is declared to get the response stream of the server.
  - 53: This streamBuffer variable reads the response from the server that is captured by using the responseStream variable.
  - 57: The ReturnData variable of String type is declared here to store the final data, which is to be returned by this function.
  - 65: Here we call the Create() function of the WebRequest class to call the given ASP page from the server; the created request is then type cast to the HttpWebRequest class and gets stored in the webRequest variable.
  - 71: After creating the Web request, we now need to catch the response from the server. This line of code does this by calling the GetResponse() function of the HttpWebRequest class and assigns the response to the webResponse variable.
  - 75: This line gets the stream where the response data is stored on the server.
  - 79: Variable streamBuffer of Byte type is initialized here to read 16 bytes of data at a time from the response stream that it stores in the responseStream variable.
  - 84: This variable is used to store the number of bytes read from the response stream of the server.
  - 88-111: Read 16 bytes of data from stream until stream gets empty, and convert the bytes data into string; read from stream and store it into ReturnData variable while concatenating it. All leading and trailing spaces are removed, and finally stream is cleaned up and closed. The data stored in ReturnData is returned by the function.
- 122-201: The GetDataFromServerUsingProxy() function retrieves the response from the server’s end. If you are behind a proxy, you have to give the proxy address and port number to this function, that is, the result of calling the ASP pages on the server. The address of the ASP page is passed to this function in the WebAddress parameter of this function. It uses this WebAddress parameter to get the response from the server’s end. This response is saved in string format in ReturnData variable for later use.
- Lines 211-232: This function writes the string data returned by the GetDataFromServerUsingProxy() function in an XML file. The name of the file is passed in the Filename variable, and the data to write is passed in the DataToWrite variable.
  - 219: This creates a stream for writing to the file.
228: This writes data to the file by converting it to Byte type.
231: The written file is closed here.

− Lines 240-285: This function determines the type of the request/response received in XML format. This is done by extracting the type attribute of the XML from the first child node of this XML.
245: The document object declared here is of type IXMLDOMDocument. This interface is present inside the MSXML2 class and is created as a reference to this application.
250: The element object of the IXMLDOMElement interface is used to point to the elements present in the XML document.
255: The node object of the IXMLDOMNode interface is used to point to the nodes present in the XML document.
260: The NodeValue variable is used to store the node value of the type attribute in the XML document.
263: This line allows the application to read the XML document synchronously.
266: This initializes the NodeValue variable to null.
269-281: First, the XML document is loaded into memory. Then the first element is extracted from the document, which in turn is used to get the first child node from it, and the actual data is extracted from the first attribute of this child node. This value is then stored in NodeValue and returned by the function.

− Lines 291-302: This function is used to delete the file represented by the Filename variable that is passed to this function.

− Lines 311-323: This retrieves the IP address of the machine whose name is passed in the Hostname parameter of this function. The function uses this parameter to extract the IP address from the DNS lookup and returns the first-found entry from the list retrieved.

XMLCreater Class

namespace WorkingWithXML

The name of the class is XMLCreater and holds the responsibility of creating different kinds of requests. The entire coding has been numbered for better clarity and wider legibility (see Listing 5-2).

Listing 5-2: XML CreaterClass

// © 2001 Dreamtech Software India Inc.
// All rights reserved

```csharp
namespace WorkingWithXML
{
    // basic package includes
    using System;
    using System.Windows.Forms;
    using System.IO;

    /// Creates requests and appropriate responses
    public class XMLCreater
    {
        // member variables for storing Application path of
        // application that's using this class
        string ResponsPath;
        string SharedResourceInfoPath;
```
// constructor to initialize both the paths
public XMLCreater(string responsePath, string sharedResourceInfoPath)
{
    ResponsePath = responsePath;
    SharedResourceInfoPath = sharedResourceInfoPath;
}

// creates an XML document
private MSXML2.IXMLDOMDocument CreateDocument()
{
// create a document object and return that
MSXML2.IXMLDOMDocument document = new MSXML2.DOMDocument();
return document;
}

// saves and closes XML document
private void SaveAndCloseDocument(MSXML2.IXMLDOMElement responseElem, MSXML2.IXMLDOMDocument document)
{
    document.async = false;
    // create Processing Instruction for XML document
    MSXML2.IXMLDOMProcessingInstruction procsInstruct =
        document.createProcessingInstruction("xml",
            "version=" + "1.0" + "encoding=" + "utf-8");
    // create primary element
    MSXML2.IXMLDOMElement elem = document.createElement("p2p_lng");
    // add response object passed to the primary element as child node
    elem.appendChild(responseElem);
    // add Processing Instruction object and primary object to document
    // object passed as child nodes and save the document
    document.appendChild(procsInstruct);
    document.appendChild(elem);
    document.save(ResponsePath);
}

// determines the request type passed in the given XML document
public string DetermineRequestType(string path, out int UploadDownloadPrint, out string[] chatInfo)
{
    int b = 0;
    string scopeVal = "";
    bool flag = false;
    string[] st = new string[3];
    try
    {
        // load the XML document
        MSXML2.IXMLDOMDocument document = new MSXML2.DOMDocument();
        if (!document.load(path))
            throw new Exception("XML request found corrupted.");
        // retrieve the request element
        MSXML2.IXMLDOMElement element = document.documentElement;
        MSXML2.IXMLDOMNode node = element.firstChild;
69  MSXML2.IXMLDOMNamedNodeMap nodemap = node.attributes;
70  // retrieve its attributes
71  MSXML2.IXMLDOMNode childNode = nodemap.nextNode();
72  
73  if(0 == node.nodeName.CompareTo("request"))
74  {
75    // see what value does the element request holds
76    // and react appropriately
77    switch(childNode.nodeValue.ToString())
78    {
79        case "CHAT":
80        {
81            b = 4;
82            MSXML2.IXMLDOMNode scope = node.firstChild;
83            MSXML2.IXMLDOMNamedNodeMap nodemap2 = scope.attributes;
84            MSXML2.IXMLDOMNode childNode2 = nodemap2.nextNode();
85            MSXML2.IXMLDOMNode childNode3 = nodemap2.nextNode();
86            MSXML2.IXMLDOMNode childNode4 = nodemap2.nextNode();
87            // set file name to upload to "path" parameter
88            st.Initialize();
89            st.SetValue(childNode2.nodeValue.ToString(),0);
90            st.SetValue(childNode3.nodeValue.ToString(),1);
91            st.SetValue(childNode4.nodeValue.ToString(),2);
92            break;
93        }
94        case "SEARCH":
95        {
96            WriteSearchResponse(node);
97            break;
98        }
99        case "SHOWFILES":
100        {
101            WriteShowfileResponse("SHOWFILES");
102            break;
103        }
104        case "DOWNLOAD":
105        {
106            // set flag that it’s download request
107            b = 2;
108            flag = true;
109            break;
110        }
111        case "UPLOAD":
112        {
113            // set flag that its upload request
114            b = 1;
115            flag = true;
116            break;
117        }
118        case "PRINT":
119        {
120            // set flag that its print request
121            b = 3;
122            flag = true;
123            break;
124        }
case "STREAMING":
{
  // set flag that its Streaming request
  b = 5;
  flag = true;
  break;
}
default:
  throw new Exception("Request type could not be resolved.");

if(flag)
{
  MSXML2.IXMLDOMNode scope = node.firstChild;
  MSXML2.IXMLDOMNamedNodeMap nodemap2 = scope.attributes;
  MSXML2.IXMLDOMNode childNode2 = nodemap2.nextNode();
  // set file name to upload to "path" parameter
  scopeVal = childNode2.nodeValue.ToString();
}

try
{
  chatInfo = st;
  UploadDownloadPrint = b;
  return scopeVal;
}

catch(Exception e)
{
  WriteErrorResponse(e.Message);
}

public void WriteErrorResponse(string error)
{
  // create a document object
  MSXML2.IXMLDOMDocument document = CreateDocument();
  // create response and error info elements
  MSXML2.IXMLDOMElement responseElem =
    document.createElement("response");
  MSXML2.IXMLDOMElement errorInfoElem =
    document.createElement("errorinfo");
  responseElem.setAttribute( "type", "ERROR");
  errorInfoElem.setAttribute( "errorcode", "1");
  errorInfoElem.setAttribute( "severity", "Error" );
  errorInfoElem.setAttribute( "description", error);
  // add errorinfo element to response object as a child
  responseElem.appendChild(errorInfoElem);
  // save the document
  SaveAndCloseDocument(responseElem, document);
}
public void WriteRequest(string type, string searchValue, string mask) {
    // create a document object
    MSXML2.IXMLDOMDocument document = CreateDocument();
    // create request element
    MSXML2.IXMLDOMElement requestElem = document.createElement("request");
    // set attributes of request element
    requestElem.setAttribute("type", type);
    // if one of these kinds of request is to be made
    // specify the filename and pertaining info too.
    if (type.CompareTo("SHOWFILES") != 0) {
        string ReqType = ";
        if (type.CompareTo("CHAT") == 0)
            ReqType = "message";
        else
            ReqType = "scope";
        MSXML2.IXMLDOMElement file_infoElem = document.createElement(ReqType.ToString());
        if (type.CompareTo("CHAT") == 0) {
            file_infoElem.setAttribute("sendername", searchValue.Substring(0, searchValue.IndexOf("(")));
            file_infoElem.setAttribute("chatmsg", mask);
        } else {
            file_infoElem.setAttribute("type", searchValue);
            file_infoElem.setAttribute("mask", mask);
        }
        requestElem.appendChild(file_infoElem);
    }
    // close and save the document
    SaveAndCloseDocument(requestElem, document);
}

private void WriteShowfileResponse(string reqType) {
    // create a document object
    MSXML2.IXMLDOMDocument document = CreateDocument();
    // create response and error info elements
    MSXML2.IXMLDOMElement responseElem = document.createElement("response");
    // set attribute of response element

responseElem.setAttribute("type", reqType);

// open share.ini for reading
StreamReader readfile = new StreamReader(SharedResourceInfoPath);
string readData;

// read entire file
while((readData = readfile.ReadLine()) != null)
{
try
{
// for each entry in share.ini create a fileinfo element
// and fill it with required information
MSXML2.IXMLDOMElement file_infoElem =
document.createElement("fileinfo");
int index = readData.IndexOf("=",0);
file_infoElem.setAttribute("filename", readData.Substring(0,index));
file_infoElem.setAttribute("mask", readData.Substring(index+1,1));

int secindex = -1;
if(-1 != (secindex = readData.IndexOf("=",index+1)))
file_infoElem.setAttribute("filesize", readData.Substring(secindex+1));

// add this element to response element as child
responseElem.appendChild(file_infoElem);

} catch(Exception e)
{
MessageBox.Show("Problem faced while responding : " + e.Message);
}

// close and save the document
SaveAndCloseDocument(responseElem,document);
}

// responds for search requests
private void WriteSearchResponse(MSXML2.IXMLDOMNode node)
{
try
{
MSXML2.IXMLDOMNode scope = node.firstChild;
MSXML2.IXMLDOMNode scopeNode = node.firstChild;
string scopeVal = childNode.nodeValue.ToString();
string maskVal = childNode2.nodeValue.ToString();

// make sure that search request has criteria specified in it
if(0 != scope.nodeName.CompareTo("scope"))
return;

// validated that directory's existing
if(!Directory.Exists(scopeVal.Substring(0,
```csharp
throw new Exception("Directory does not exist any more");

MSXML2.IXMLDOMDocument document = CreateDocument();
MSXML2.IXMLDOMElement responseElem =
document.createElement("response");
responseElem.setAttribute( "type", "SHOWFILES" );

int i = 0;
// get files in the specified directory satisfying the
// given criteria
string[] files = Directory.GetFiles( scopeVal.Substring(0, scopeVal.LastIndexOf("\") + 1), scopeVal.Substring( scopeVal.LastIndexOf("\") + 1 ));
files.Initialize();

while (i < files.Length)
{
    // make fileinfo elements and fill them up with
    // required
    MSXML2.IXMLDOMElement file_infoElem =
document.createElement("fileinfo");
    file_infoElem.setAttribute( "filename", files[i] );
    file_infoElem.setAttribute( "mask", maskVal );
    file_infoElem.setAttribute( "filesize", Convert.ToString(files[i].Length) );
    i++;

    // add them to response element as children;
    responseElem.appendChild(file_infoElem);
}

// get files in the specified directory satisfying the
// given criteria
string[] dirs = Directory.GetDirectories( scopeVal.Substring(0, scopeVal.LastIndexOf("\") + 1), scopeVal.Substring( scopeVal.LastIndexOf("\") + 1 ));
dirs.Initialize();

while (i < dirs.Length)
{
    // make fileinfo elements and fill them up with
    // required
    MSXML2.IXMLDOMElement file_infoElem =
document.createElement("fileinfo");
    file_infoElem.setAttribute( "filename", dirs[i] + "\" ");
    file_infoElem.setAttribute( "mask", maskVal );
    i++;

    // add them to response element as children;
    responseElem.appendChild(file_infoElem);
}

// close and save the document
SaveAndCloseDocument(responseElem, document);
```
catch(Exception e) { WriteErrorResponse(e.Message); }
}
}

Code description of XMLCreater class (clsXMLCreater.cs)

♦ Lines 4-6: This includes necessary packages to build the application.

♦ Lines 9-15: This public class XMLCreater has been declared with two strings for storing the path of the application using this class and for storing the share.ini file path, respectively.

♦ Lines 18-22: This is the implementation of the constructor XMLCreater() of the XMLCreater class. It initializes the two previously mentioned specified paths with values passed.


♦ Lines 33-48: The SaveAndCloseDocument() function saves and closes the passed XML document after putting in it other required elements and the passed response element.
  • 37: Process Instructions are created for this XML document.
  • 39: The primary element of the XML document (p2p_lng) is created and stored in the MSXML2.IXMLDOMElements element object.
  • 42: This adds a response object passed to the primary element as its child node.
  • 45: Here, process-instructions are added to the document and appended to it as a child node.
  • 46: The primary element is added to the document.
  • 47: The document is saved on the application path.

♦ Lines 52-156: The DetermineRequestType() function loads the XML request document sent by the browser, determines the kind of request sent, and reacts appropriately.
  • 62-64: XML document is loaded by calling IXMLDOMDocument's load.
  • 67-69: Primary elements are retrieved through, which its child elements (request) and then its attributes are retrieved and stored in a IXMLDOMNodeMap element;
  • 71: Here, the first attribute of the child element is received and stored in an object of the MSXML2.IXMLDOMNode class named childNode.
  • 79-93: Here we are checking what the request element holds. If the request type is chat, information pertaining the user who has initiated the chat process is retrieved and stored in variables for returning to the listener at the end of this function.
  • 94-103: This determines what kind of request it is and reacts appropriately by calling appropriate functions.
  • 104-132: Here we are extracting the child node of the request (scope element) element and storing it into a scope object. In the next line, the attributes of this element are taken in a IXMLDOMNodeMap element, and the value of the first attribute is extracted and stored in a string variable for returning at the end of the function.
  • 134-136: In case the request made is exceptional and not understood by these functions, an error message is thrown to the end user by calling WriteErrorResponse in a catch block.
  • 138-145: Here we extract the child node of the request (scope element) element and store it into a scope object. In the next line, the attributes of this element are taken in a IXMLDOMNodeMap element, and the value of the first attribute is extracted and stored in a string variable for returning at the end of the function to the caller function; upload, print, download, and streaming processes are signaled by setting the flag value according to the type of request.
• 152-156: The values extracted previously are assigned to the outgoing parameters, and the information is returned to the caller.

♦ Lines 159-177: The WriteErrorResponse() function holds the responsibility of writing error XML responses. An XML document is created. A response element is created, and its type attribute is set to error. In the following lines error XML attributes are set to indicate the error message, sources, and so on.

• 165: An element called errorinfo is created.

• 167-172: In these lines, we set the request type to ERROR, and errorinfo’s attributes, such as errorcode, severity, and description, are set to their appropriate values.

• 174-176: The errorinfo element, bundled with its attributes, is added to the response object as a child; the document is saved and closed.

♦ Lines 180-219: The WriteRequest() function is used generally by the browser to create XML requests to send to the listener.

• 183: An XML document object is created.

• 186: A request element is created.

• 189: An attribute called type is associated to XML document object.

• 193-213: If request type to be created is SEARCH, PRINT, CHAT, STREAMING, DOWNLOAD, or UPLOAD, a scope element is created to specify the search criteria and folder to carry out a search operation on or files to download or upload or print or streaming and their masks if required. If request is found to be CHAT, unlike other types of requests with which the scope element is assigned type and mask attributes, sendername, senderIP and chatmsg are associated to it.

• 218: The document is saved and closed.

♦ Lines 222-258: This function writes the response on an XML document to show files.

• 224-230: Here we are making an object of XMLDOMDocument and an object of XMLDOMElement called response. Next, we are associating an attribute to a response object called type with the value SHOWFILES.

• 231-232: We are making an object of the StreamReader class, passing it the path and name of share.ini file.

• 235-255: This while loop keeps reading the file, and, for each file or folder found, it creates a fileinfo element and sets its attributes such as filename, mask and filesize. The filesize attribute is not set if the found resource is a folder.

• 241-242: In each iteration, the fileinfo element created is added to the request element, which is sent to SaveAndCloseDocument() for saving and closing the document.


• 265: The Scope element, with its list of attributes, is retrieved.

• 266: The First attribute of the scope element is retrieved here.

• 268: The Second attribute of scope element is retrieved here.

• 269-271: Value of the First and Second attribute is received.

• 275: Search criteria is extracted from first attribute of the scope element and is passed to the Directory class’s object.

• 276-278: Here we are checking the physical existence of the directory on the hard disk of the computer.
• 280-283: An XmlDocument and an XMLDOMElement object are created. The XMLDOMElement represents the response element, and we have coupled it with an attribute called type with that has the value SHOWFILE.

• 287-288: The list of files in the directory specified in the XML search request is obtained by using the Directory object’s GetFiles() function, passing the path and criteria as arguments.

• 291-303: We are filling up information of each file as attributes to an XMLDOM element called fileinfo, created for each file found. Attributes for this element are filename, the value of which is the full name of the file; mask, whose value is the mask of the directory specified in the search request; and filesize, the file size in bytes of the file found. In iterations, we add this element to response element as child node.

• 305-309: We are calling the Directory class object’s other function called GetDirectories(), passing it the path and criteria. This function returns an array of Directory objects for directories satisfying the given search criteria.

• 310-321: We are iterating through the retrieved array. In iterations, we are creating an XML element called fileinfo and setting its attributes. Attributes are filename and mask values of which will be the directory name of the directory found and mask value of the directory specified in the search criteria, respectively.

• 324: Here the response element is passed to SaveAndCloseDocument() to write this response XML element to a response XMLDocument.

• 325: If any exception is caught, it is passed to a WriteErrorResponse() function, which makes an XML response document with a response-type error that specifies the error message in the errorinfo element’s attributes.

**XMLParser Class**

The entire coding in Listing 5-3 has been numbered and explained.

**Listing 5-3: XML Parser Class**

```csharp
// © 2001 Dreamtech Software India Inc.
// All rights reserved

namespace WorkingWithXML
{
    using System; // Provides the basic functionality of .NET
    using System.Net; // Provides the net related functionality
    using System.IO; // Provides the I/O functionality
    using System.Text; // Provides text based manipulations

    /// <summary>
    ///  Generic structure used for parsing the XML data
    ///  This structure composes of various sub structures
    ///  Each sub structure represents and XML request in whole
    ///  Every XML is parsed into its corresponding structure to
    ///  fill its values TypeofXMLRecieved() function of
    ///  ServerCommunication class will determine that which structure
    ///  has to be filled
    /// </summary>
    public struct XMLSTRUCT
    {
        /// <summary>
        /// This structure is used to store the parsed
        ```
/// values of the AUTH XML which is returned
/// after login process
/// </summary>
public struct __AUTHENTICATION
{
    /// <summary>
    /// Stores the Code value in it 0(successful)
    /// or 1(some error occured)
    /// </summary>
    public int iCode;

    /// <summary>
    /// This will stores the status of the login process
    /// and any error message if occured while login
    /// </summary>
    public string sStatus;

    /// <summary>
    /// This is used for cross checking the IP address
    /// which is sent to the server that login is successful
    /// or not
    /// </summary>
    public string sIPAddress;
}

/// <summary>
/// This structure is used to store the List of all
/// the Listeners from the server that are currently
/// The values are returned in the USERLIST response XML
/// running
/// </summary>
public struct __USERLIST
{
    /// <summary>
    /// Name by which the Listener has logged in
    /// </summary>
    public string sUsername;

    /// <summary>
    /// IP Address of that Listener
    /// </summary>
    public string sIPAddress;
}

/// <summary>
/// This is used to store the values which are parsed
/// from the SHOWFILES response XML from the Listener
/// It contains the Files and Folders which are to
/// be shown to the user
/// </summary>
public struct __SHOWFILES
{
    /// <summary>
    /// Stores the Filename or Folder name
    /// </summary>
    public string sFilename;
/// <summary>
/// Stores the FileSize, 0 in case of folders
/// </summary>
public int iFileSize;

/// <summary>
/// Mask stores the mask value of a file or folder
/// 0(readonly file/folder) 1(read/write access)
/// </summary>
public int iMask;

/// <summary>
/// In case of any Error an ERROR response XML is
/// thrown from the Listener. The values are parsed into
/// this structure
/// </summary>
public struct __ERROR
{
    /// <summary>
    /// Stores the error code
    /// </summary>
    public int iErrCode;

    /// <summary>
    /// Stores the severity of the error
    /// Message or Warning or Error
    /// </summary>
    public string sSeverity;

    /// <summary>
    /// The actual error description is stored in this
    /// variable
    /// </summary>
    public string sDescription;
}

/// <summary>
/// no XML parser has been made for this structure,
/// since it is not used in this version
/// </summary>
public struct __UPDNLOAD
{
    public string sFilename;
    public int iMask;
}

/// <summary>
/// no XML parser has been made for this structure,
/// since it is not used in this version
/// </summary>
public struct __MESSAGE
{
    public string sSenderName;
    public string sMessage;
Chapter 5: The P2P Application (C#) 187

133 public string sIPAddress;
134 }
135
136 /// <summary>
137 /// this structure stores the values from the
138 /// SERVERSEARCH XML that is returned by the Server
139 /// as the result of search
140 /// </summary>
141 public struct __SERVERSEARCH
142 {
143     /// <summary>
144     /// IP address of the machine where the file or folder
145     /// is found
146     /// </summary>
147     public string sIPAddress;
148
149     /// <summary>
150     /// Username i.e login name of the machine
151     /// </summary>
152     public string sUsername;
153
154     /// <summary>
155     /// Name of the file found for search criteria is
156     /// in this variable
157     /// </summary>
158     public string sFilename;
159 }
160
161 /// <summary>
162 /// Global variables which are used
163 /// in different parts of the code
164 /// for their specific structures
165 /// </summary>
166 public __AUTHENTICATION AUTH;
167 public __USERLIST[] USERLIST;
168 public __SHOWFILES[] SHOWFILES;
169 public __SHOWFILES[] SEARCH;
170 public __SERVERSEARCH[] SERVERSEARCH;
171 public __ERROR ERROR;
172 public __MESSAGE MESSAGE;
173 }
174
175 /// <summary>
176 /// Summary description for clsXMLParser.
177 /// This class is used to parse any XML that is received
178 /// by the Listener of Browser(Client)
179 /// and stores the values to their corresponding
180 /// structures so that the application could use them
181 /// </summary>
182 public class XMLParser
183 {
184     /// <summary>
185     /// Stores the Filename to write when login response
186     /// when has arrived to the Browser
187     /// </summary>
188     public string LOGINXML;
/// <summary>
/// Stores the Filename to write when USERLIST response has arrived to the Browser
/// </summary>
public string USERLISTXML;

/// <summary>
/// Stores the Filename to write when SERVERSEARCH response has arrived to the Browser
/// </summary>
public string SERVERSEARCHRESULTXML;

/// <summary>
/// stores the number of tags that are found in the response XML
/// </summary>
protected int iTags;

/// <summary>
/// Used to store the counter that is how many time a loop is running
/// </summary>
protected int iCounter;

/// <summary>
/// This document variable points to the XML document
/// </summary>
protected MSXML2.IXMLDOMDocument document;

/// <summary>
/// Points to the element of the XML document
/// </summary>
protected MSXML2.IXMLDOMElement element;

/// <summary>
/// Points to the node of the XML
/// </summary>
protected MSXML2.IXMLDOMNode node, ChildNode;

/// <summary>
/// points to the node list of the XML document
/// </summary>
protected MSXML2.IXMLDOMNodeList nodeList;

/// <summary>
/// Stores the node list of the XML
/// </summary>
protected MSXML2.IXMLDOMNamedNodeMap nodeMap;

/// <summary>
/// <summary>
/// Default constructor
/// </summary>
public XMLParser()
Chapter 5: The P2P Application (C#) 189

```csharp
244         {  
245             
246         }  
247         /// <summary>  
248         /// Initialize some important variables  
249         /// </summary>  
250         protected void InitVariables()  
251         {  
252             iTags = 0;  
253             iCounter = 0;  
254             document = new MSXML2.DOMDocument();  
255         }  
256         
257         /// <summary>  
258         /// This function is responsible for parsing the XML  
259         /// Actually this function will call the exact parse function  
260         /// depending upon the type of XML Received  
261         /// </summary>  
262         /// <param name="XMLFilename"> </param>  
263         /// <param name="outStruct"> </param>  
264         /// <param name="TagName"> </param>  
265         public int ParseXML(string XMLFilename, out XMLSTRUCT outStruct, string TagName)  
266         {  
267             // Declare and initializes the iElements to 0  
268             int iElements = 0;  
269             
270             // Initializes the outStruct variable of this function  
271             // this structure is used to store the values of parsed XML  
272             outStruct = new XMLSTRUCT();  
273             
274             // The following 12 lines of code checks the Type of XML recieved  
275             // and calls are made to the corresponding parser function  
276             // which actually are responsible for parsing the XML  
277             // all the parse functions are user defined functions  
278             // the Number of Parsed records are stores in the iElements  
279             // variable which is returned by the function  
280             if( 0 == TagName.CompareTo("AUTH") )  
281                 iElements = ParseAUTHXML(XMLFilename, out outStruct);  
282             else if( 0 == TagName.CompareTo("USERLIST") )  
283                 iElements = ParseUSERLISTXML(XMLFilename, out outStruct);  
284             else if( 0 == TagName.CompareTo("SHOWFILES") )  
285                 iElements = ParseSHOWFILESXML(XMLFilename, out outStruct);  
286             else if( 0 == TagName.CompareTo("SEARCH") )  
287                 iElements = ParseERRORXML(XMLFilename, out outStruct);  
288             else if( 0 == TagName.CompareTo("SERVERSEARCH") )  
289                 iElements = ParseSERVERSEARCHXML(XMLFilename, out outStruct);  
290             else if( 0 == TagName.CompareTo("CHAT") )  
291                 iElements = ParseCHATXML(XMLFilename, out outStruct);  
292             
293             // Returns the iElements variable to the calling function  
294             return iElements;  
295         }
```
protected int ParseCHATXML(string Filename, out XMLSTRUCT outStruct) {
    // initializes all the required variables
    InitVariables();
    // Initialize outStruct variable of this function
    outStruct = new XMLSTRUCT();
    // Process the XML document synchronously
    document.async = false;
    // load the xml document in memory for parsing
    if (document.load(Filename)) {
        // get the first element of the XML
        element = document.documentElement;
        // get the first child of the element
        node = element.firstChild;
        // extracts the node list present under the node
        nodeList = node.childNodes;
        // iTags will assigns to the number of nodes present
        // in node list
        iTags = nodeList.length;
        // Initialize the AUTH sructure of the outStruct
        // variable
        outStruct.MESSAGE = new XMLSTRUCT.__MESSAGE();
        // move the node to the next node of the nodelist
        node = nodeList.nextNode();
        // Extract each value from its specific node
        for (iCounter = 0; iCounter < iTags; iCounter++) {
            // gets the attribute map that is how many attributes
            // are present in the node
            nodeMap = node.attributes;
            // extract the next node from the node map
            ChildNode = nodeMap.nextNode();
            // The following 9 lines of code will extract the
            // various attribute values from the XML node
            // and fills it to the outStruct's corresponding
            // structure
            do {
                if (0 == ChildNode.nodeName.CompareTo("sendername") )
                    outStruct.MESSAGE.sSenderName =
                        ChildNode.nodeValue.ToString();
            } while (ChildNode.nodeValue != null && ChildNode.nodeName != "sendername");
        }
    }
}
```csharp
else if (0 == ChildNode.nodeName.CompareTo("chatmsg"))
    outStruct.MESSAGE.sMessage = ChildNode.nodeValue.ToString();
else if (0 == ChildNode.nodeName.CompareTo("ip"))
    outStruct.MESSAGE.sIPAddress = ChildNode.nodeValue.ToString();
while (null != (ChildNode = nodeMap.nextNode()));

// now move to next node
node = nodeList.nextNode();

// Return the number of nodes parsed for the values
return iCounter==iTags?iCounter:0;
}

/// <summary>
/// Actual Parsing of AUTHENTICATION XML
/// </summary>
/// <param name="Filename"> </param>
/// <param name="outStruct"> </param>
protected int ParseAUTHXML(string Filename, out XMLSTRUCT outStruct)
{
    // initializes all the required variables
    InitVariables();

    // Initialize outStruct variable of this function
    outStruct = new XMLSTRUCT();

    // Process the XML document synchronously
    document.async = false;

    // load the xml document in memory for parsing
    if(document.load(Filename))
    {
        // get the first element of the XML
        element = document.documentElement;

        // get the first child of the element
        node = element.firstChild;

        // extracts the node list present under the node
        nodeList = node.childNodes;

        // iTags will assigns to the number of nodes present
        // in node list
        iTags = nodeList.length;

        // Initialize the AUTH structure of the outStruct
        // variable
        outStruct.AUTH = new XMLSTRUCT.AUTHENTICATION();

        // move the node to the next node of the nodelist
    }
```

 node = nodeList.nextNode();

  // Extract each value from its specific node
  for (iCounter = 0; iCounter < iTags; iCounter++)
  {
      
      // gets the attribute map that is how many attributes
      // are present in the node
      nodeMap = node.attributes;

      // extract the next node from the node map
      ChildNode = nodeMap.nextNode();

      // The following 9 lines of code will extract the
      // various attribute values from the XML node
      // and fills it to the outStruct's corresponding
      // structure
      do
      {
          if (0 == ChildNode.nodeName.CompareTo("code"))
              outStruct.AUTH.iCode = Convert.ToInt32(ChildNode.nodeValue);
          else if (0 == ChildNode.nodeName.CompareTo("status"))
              outStruct.AUTH.sStatus = ChildNode.nodeValue.ToString();
          else if (0 == ChildNode.nodeName.CompareTo("ip"))
              outStruct.AUTH.sIPAddress = ChildNode.nodeValue.ToString();
      } while (null != (ChildNode = nodeMap.nextNode()));

      // now move to next node
      node = nodeList.nextNode();
  }

  // Return the number of nodes parsed for the values
  return iCounter==iTags?iCounter:0;

/// <summary>
/// Actual Parsing of USERLIST XML
/// </summary>
/// <param name="Filename"> </param>
/// <param name="outStruct"> </param>
protected int ParseUSERLISTXML(string Filename, out XMLSTRUCT outStruct)
{
    // initializes all the required variables
    InitVariables();

    // Initialize outStruct variable of this function
    outStruct = new XMLSTRUCT();

    // Process the XML document synchronously
    document.async = false;
456   // load the xml document in memory for parsing
457   if(document.load(Filename))
458     {
459       // get the first element of the XML
460       element = document.documentElement;
461
462       // get the first child of the element
463       node = element.firstChild;
464
465       // extracts the node list present under the node
466       nodeList = node.childNodes;
467
468       // iTags will assigns to the number of nodes present
469       // in node list
470       iTags = nodeList.length;
471
472       // Initialize the USERLIST sructure of the outStruct
473       // variable
474       outStruct.USERLIST = new XMLSTRUCT.__USERLIST[iTags];
475
476       // move the node to the next node of the nodelist
477       node = nodeList.nextNode();
478
479       // Extract each value from its specific node
480       for(iCounter = 0; iCounter < iTags; iCounter++)
481         {
482           // gets the attribute map that is how many attributes
483           // are present in the node
484           nodeMap = node.attributes;
485
486           // extract the next node from the node map
487           ChildNode = nodeMap.nextNode();
488
489           // The following 9 lines of code will extract the
490           // various attribute values from the XML node
491           // and fills it to the outStruct's corresponding
492           // structure
493           do
494             {  // the attribute name is "username"
495               if( 0 ==
496                 ChildNode.nodeName.CompareTo("username") )
497                 outStruct.USERLIST[iCounter].sUsername =
498                   ChildNode.nodeValue.ToString();
499             } else if( 0 ==
500               ChildNode.nodeName.CompareTo("ip") )
501                 outStruct.USERLIST[iCounter].sIPAddress =
502                   ChildNode.nodeValue.ToString();
503           } while( null != (ChildNode = nodeMap.nextNode()) );
504
505         }
506
507       // Return the number of nodes parsed for the values
return iCounter==iTags?iCounter:0;
}

/// <summary>
/// Actual Parsing of SERVERSEARCH XML
/// </summary>
/// <param name="Filename"> </param>
/// <param name="outStruct"> </param>
protected int ParseSERVERSEARCHXML(string Filename, out XMLSTRUCT outStruct)
{
    // initializes all the required variables
    InitVariables();
    // Initialize outStruct variable of this function
    outStruct = new XMLSTRUCT();
    // Process the XML document synchronously
    document.async = false;
    // load the xml document in memory for parsing
    if(document.load(Filename))
    {
        // get the first element of the XML
        element = document.documentElement;
        // get the first child of the element
        node = element.firstChild;
        // extracts the node list present under the node
        nodeList = node.childNodes;
        // iTags will assigns to the number of nodes present
        iTags = nodeList.length;
        // Initialize the SERVERSEARCH sructure of the outStruct variable
        outStruct.SERVERSEARCH = new XMLSTRUCT.__SERVERSEARCH[iTags];
        // move the node to the next node of the nodelist
        node = nodeList.nextNode();
        // Extract each value from its specific node
        for(iCounter = 0; iCounter < iTags; iCounter++ )
        {
            // gets the attribute map that is how many attributes
            // are present in the node
            nodeMap = node.attributes;
            // extract the next node from the node map
            ChildNode = nodeMap.nextNode();
            // The following 9 lines of code will extract the
            // various attribute values from the XML node
            // and fills it to the outStruct's corresponding
Chapter 5: The P2P Application (C#) 195

563 // structure
564 do
565 {
566     if( 0 == ChildNode.nodeName.CompareTo("ip") )
567         outStruct.SERVERSEARCH[iCounter].sIPAddress = ChildNode.nodeValue.ToString();
568     else if( 0 ==
569         ChildNode.nodeName.CompareTo("username") )
570         outStruct.SERVERSEARCH[iCounter].sUsername = ChildNode.nodeValue.ToString();
571     else if( 0 ==
572         ChildNode.nodeName.CompareTo("filename") )
573         outStruct.SERVERSEARCH[iCounter].sFilename = ChildNode.nodeValue.ToString();
574     while( null != (ChildNode = nodeMap.nextNode()) );
575 }
576
577 // Return the number of nodes parsed for the values
578 return iCounter==iTags?iCounter:0;
579
580 /// <summary>
581 /// Actual Parsing of SHOWFILES XML
582 /// </summary>
583 /// <param name="Filename"> </param>
584 /// <param name="outStruct"> </param>
585 protected int ParseSHOWFILESXML(string Filename, out XMLSTRUCT outStruct)
586 {
587     // initializes all the required variables
588     InitVariables();
589
590     // Initialize outStruct variable of this function
591     outStruct = new XMLSTRUCT();
592
593     // Process the XML document synchronously
594     document.async = false;
595
596     // load the xml document in memory for parsing
597     if(document.load(Filename))
598     {
599         // get the first element of the XML
600         element = document.documentElement;
601         
602         // get the first child of the element
603         node = element.firstChild;
604
605         // extracts the node list present under the node
606         nodeList = node.childNodes;
607
608         // iTags will assigns to the number of nodes present
609         iTags = nodeList.length;
// Initialize the SHOWFILES structure of the outStruct
outStruct.SHOWFILES = new XMLSTRUCT.__SHOWFILES[iTags];

// move the node to the next node of the nodeList
node = nodeList.nextNode();

// Extract each value from its specific node
for(iCounter = 0; iCounter < iTags; iCounter++)
{
    // gets the attribute map that is how many attributes
    // are present in the node
    nodeMap = node.attributes;

    // extract the next node from the node map
    ChildNode = nodeMap.nextNode();

    // The following 9 lines of code will extract the
    // various attribute values from the XML node
    // and fills it to the outStruct's corresponding
    // structure
    do
    {
        if( 0 ==
            ChildNode.nodeName.CompareTo("filename") )
            outStruct.SHOWFILES[iCounter].sFilename
                = ChildNode.nodeValue.ToString();
        else if( 0 ==
            ChildNode.nodeName.CompareTo("mask") )
            outStruct.SHOWFILES[iCounter].iMask
                = Convert.ToInt32(ChildNode.nodeValue);
        else if( 0 ==
            ChildNode.nodeName.CompareTo("filesize") )
            outStruct.SHOWFILES[iCounter].iFileSize
                = Convert.ToInt32(ChildNode.nodeValue);
    } while( null != (ChildNode = nodeMap.nextNode()) );

    // now move to next node
    node = nodeList.nextNode();
}

// Return the number of nodes parsed for the values
return iCounter==iTags?iCounter:0;

/// <summary>
/// Actual Parsing of ERROR XML
/// </summary>
protected int ParseERRORXML(string Filename, out XMLSTRUCT outStruct)
{
    // initializes all the required variables
    InitVariables();
664 // Initialize outStruct variable of this function
665 outStruct = new XMLSTRUCT();

667 // Process the XML document syncronously
668 document.async = false;

670 // load the xml document in memory for parsing
672 if(document.load(Filename))
673 {
674     // get the first element of the XML
675     element = document.documentElement;

676     // get the first child of the element
677     node = element.firstChild;

678     // extracts the node list present under the node
679     nodeList = node.childNodes;

680     // iTags will assigns to the number of nodes present
681     // in node list
682     iTags = nodeList.length;

683     // Initialize the ERROR sructure of the outStruct
684     // variable
685     outStruct.ERROR = new XMLSTRUCT.__ERROR();

686     // move the node to the next node of the nodelist
687     node = nodeList.nextNode();

688     // Extract each value from its specific node
689     for(iCounter = 0; iCounter < iTags; iCounter++ )
690     {
691         // gets the attribute map that is how many attributes
692         // are present in the node
693         nodeMap = node.attributes;

694         // extract the next node from the node map
695         ChildNode = nodeMap.nextNode();

696         // The following 9 lines of code will extract the
697         // various attribute values from the XML node
698         // and fills it to the outStruct's corresponding
699         // structure
700         do
701             if( 0 == ChildNode.nodeName.CompareTo("code") )
702                 outStruct.ERROR.iErrCode =
703                        Convert.ToInt32(ChildNode.nodeValue);
704             else if( 0 ==
705                         ChildNode.nodeName.CompareTo("severity") )
706                 outStruct.ERROR.sSeverity =
707                     ChildNode.nodeValue.ToString();
708             else if( 0 ==
709                         ChildNode.nodeName.CompareTo("description") )
outStruct.ERROR.sDescription = ChildNode.nodeValue.ToString();
} while( null != (ChildNode = nodeMap.nextNode()) );
// now move to next node
node = nodeList.nextNode();

// Return the number of nodes parsed for the values
return iCounter==iTags?iCounter:0;

Code description of XML Parser class (clsXMLParser.cs)

♦ Lines 3-6: This includes the basic packages needed for the various classes used to build this application. This includes packages used for communicating with the network, streaming facility, file I/O, text manipulations, and so on.

♦ Lines 17-173: This declares the structure of the public type named XMLSTRUCT. This structure is used while parsing XML data. This structure is composed of various substructures and represents an XML request in whole. Every XML is parsed into its corresponding structure to fill its value. The TypeOfXMLReceived() function of the ServerCommunication class, determines which structures have to be filled by the response XML returned by the server or listener.

• 24-44: This __AUTHENTICATION structure is used to store the parsed XML values of the AUTH XML, which is returned after login.

• 52-62: This __USERLIST structure is used to store the parsed XML values of the USERLIST XML, which is returned after requesting for the list of the currently running listeners.

• 71-88: This __SHOWFILES structure is used to store the parsed XML values of the XML, which is returned after the SHOWFILES request has been sent to the listener.

• 95-113: This __ERROR structure is used to store the parsed XML values of the error XML, which is returned if any error has occurred at the listener end while making a response to the request.

• 119-134: These two structures are not used in this application. They are kept here for future expandability. You may use these structures as needed. The __MESSAGE structure here can be used for integrating an instant-messaging client to this application. No XML parser function is associated with any of these functions.

• 140-159: This __SERVERSEARCH structure is used to have the parsed values, which result from the server after calling the search.asp on the server. This structure contains server-search results.

• 166-173: Variables for these structures are declared here. These variables are globally accessible in this application.

♦ Lines 182: This is the declaration of the XMLParser class.

♦ Lines 188-238: Some global variables used in this application are declared here.

• 188: LOGINXML stores the name of the response file, which is generated after login.

• 194: USERLISTXML stores the name of the response file, which is after requesting for the list of the currently running listeners.

• 200: SERVERSEARCHXML stores the name of the response file, which generated after the search process is completed on the server and response data has been sent.

• 206: iTags stores the number of tags present in any XML response.
• 212: `iCounter` is used for bookkeeping purposes here.
• 217: The `MSXML2.IXMLDOMDocument` document is used to point toward the XML document reference.
• 222: The `MSXML2.IXMLDOMElement` element is used to point toward the single XML element reference.
• 227: The `MSXML2.IXMLDOMNode` node, `ChildNode`, is used to point toward the XML document's main node and its child nodes.
• 233: The `MSXML2.IXMLDOMNodeList` nodeList is used to point toward the number of nodes in an XML document.
• 238: The `MSXML2.IXMLDOMNodeMap` nodeMap is used to point toward the map of the nodes present inside an XML document.

♦ Lines 243-245: This is for the implementation of the default constructor of this `XMLParser` class.
♦ Lines 250-255: This function is used to initialize the three variables every time it is called.
♦ Lines 265-297: This function is called many times in this application. It is used to parse the XML document. Actually for the reason that parsing becomes simple for the user to understand only a single parse function is made to handle every parse request. This function determines the type of XML received from the server or listener and then calls the appropriate function for parsing that XML. It only acts as a router. This function takes three parameters: the name of the XML file to parse, the reference to the structure passed to the actual parse function, and the tag name so that the actual parse function associated with that tag is called.
  • 272: The `outStruct` variable is initialized here.
  • 280-297 In these lines, the type of XML received is checked. The XML is then passed to the corresponding user-defined parse functions for parsing. The number of parsed records is stored in `iElements` variable, which is returned by the function after filling the `outStruct` variable.
♦ Lines 299-366: This function processes the parsing of `CHATXML`.
  • 302-311: In these lines, required variables are initialized. In line 309, `document.async` property is set to false to process the document synchronously.
  • 312-325: Here the XML document is loaded into memory for parsing. Then the first element of XML is retrieved, followed by retrieval of the first child of the element. Once the first element and its child element are extracted, the node list present under the node is fetched, and the number of fetches is assigned to the `iTags` variable.
  • 329-332: The `MESSAGE` structure of the `outStruct` variable is initialized. The next is to the control to the next node of the node list, meanwhile storing it into the `node` variable, which points toward the current node.
  • 355-343: Each value is extracted from specific node while getting the attribute map, which indicates how many attributes are present in the node. This process goes on by moving to the next node map.
  • 349-358: Various attributes' values from the XML node are extracted, filled into the `outStruct` structure, and move to next node.
  • 365: The number of nodes parsed for the values are returned by the `iCounter` variable, which counts how many times the loop has run and has been returned by the function.

The following functions carry out the same things. Every line in these functions is the same as the preceding lines. The only difference is that the functions fill their corresponding structures after parsing the values. The parsing logic of these functions is same as described previously.
Login Form

namespace: Listener

The name of the following class is frmLogin, which performs the entire task of listening for what browsers connecting to it are demanding and responding appropriately. This class also handles the life cycles of the login form (described in following paragraphs) and the listener's existence on the tray icon.

Figure 5-1 shows the first form that opens when you start the listener application to log to the server. After the logging process is successful, this form disappears, and only the tray icon remains to indicate that the listener is running.

![Figure 5-1: The dialog box that opens when the user starts the listener application](image)

- **Login:** After typing in the LoginID in the text box given, clicking this button logs in to the server.
- **Share Files/Folder:** This button opens a Share Files/Folders dialog (explained later) for you to share files or folders with other peers.
- **Quit:** Quit closes the application.

The entire coding in Listing 5-4 has been numbered and explained in this regard.
namespace Listener
{
    // Library includes
    using System;
    using System.Drawing;
    using System.Collections;
    using System.ComponentModel;
    using System.Windows.Forms;
    using System.Net; // added for httprequest and responses and hostname resolving
    using System.IO; // added for file streaming
    using System.Text; // added for text encodings
    using WorkingWithXML;
    using System.Net.Sockets;
    using System.Threading;
    using System.Runtime.InteropServices;
    using System.Windows.Forms.Design;

    public class frmLogin : System.Windows.Forms.Form
    {
        Socket newSock;
        [DllImport("Shell32.dll")]
        public static extern int ShellExecute(int hwnd,
            string lpVerb,
            string lpFile,
            string lpParameters,
            string lpDirectory,
            int nShowCmd );

        // State maintainig variables;
        private bool bLoggedIn;
        private int IconNo;
        private bool bShareDialogOpend;
        private bool bDownLoading;

        private Byte[] buffer;
        private string LoginName;
        private int ChatWindowCount;
        // Variable required in communication with server
        private ServerCommunication xmlServerComm;

        // Communication facilitators on sockets.
        private Socket servSock;

        //Xml requests and responses are created using these objects
        private XMLCreater xmlCreater;
        private XMLParser xmlParser;

        // Threads of this application's being.
private Thread AcceptThread, ThreadIcon, RespondingThread;
private System.ComponentModel.IContainer components;
private System.Windows.Forms.Label lbCopyright;
private System.Windows.Forms.ToolTip toolTip1;
private System.Windows.Forms.MenuItem ctxMenuShare;
private System.Windows.Forms.MenuItem ctxMenuQuit;
private System.Windows.Forms.TextBox textLoginID;
private System.Windows.Forms.Label labelLoginID;
private System.Windows.Forms.CheckBox chkRemeberID;
private System.Windows.Forms.Button btnLogin;
private System.Windows.Forms.MenuItem menuItem1;
private System.Windows.Forms.MenuItem ListenCntxtMenu;
private System.Windows.Forms.TextBox textBox1;
private System.Windows.Forms.NotifyIcon ListenIcon;
private System.Windows.Forms.Button button1;

// First functions that gets called in this form's life cycle
// Initialises all the GUI controls and reads username last
// used, if any and puts that on the form.
public frmLogin()
{
    InitializeComponent();
    // initialise objects getting used while communication
    // and parsing
    xmlServerComm = new WorkingWithXML.ServerCommunication();
    xmlParser = new WorkingWithXML.XMLParser();
    xmlParser.LOGINXML = Application.StartupPath + "\Login.xml";
    textLoginID.Text = ReadUsername();
    bLoggedIn = false;
    bShareDialogOpend = false;
    bDownLoading = false;
    IconNo = 1;
    ChatWindowCount = 0;
    LoginName = "";
    AcceptThread = null;
    ThreadIcon = null;
    RespondingThread = null;
}

// Reads user name and returns that to the caller
public string ReadUsername()
{
    string Username, sTemp;
    Username = null;
    if ( File.Exists(Application.StartupPath + "\UserInfo.ini") )
    {
        // open userinof.ini file and read user name
        Stream fstr = File.OpenRead(Application.StartupPath + "\UserInfo.ini");
    
}
Byte[] buffer = new Byte[Convert.ToInt32(fstr.Length)];
fstr.Read(buffer, 0, Convert.ToInt32(fstr.Length));
sTemp = Encoding.ASCII.GetString(buffer, 0,
    Convert.ToInt32(buffer.Length));
Username = sTemp.Substring(sTemp.IndexOf("=")+1);
chkRemeberID.Checked = true;
fstr.Close();
}
else

    chkRemeberID.Checked = false;

return Username;
}

// Writes user name to file for next instantiation
public void WriteUsername()
{
    string buffer = "username=";
    Stream fstr = File.OpenWrite(Application.StartupPath + 
    "\UserInfo.ini");
    buffer += textLoginID.Text;
    fstr.Write(Encoding.ASCII.GetBytes(buffer), 0, buffer.Length);
    fstr.Close();
}

        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            System.Resources.ResourceManager resources = new
            System.Resources.ResourceManager(typeof(frmLogin));
            this.contextMenu = new System.Windows.Forms.ContextMenu();
            this.ctxMenuQuit = new System.Windows.Forms.MenuItem();
            this.ctxMenuShare = new System.Windows.Forms.MenuItem();
            this.ListenIcon = new
            System.Windows.Forms.NotifyIcon(this.components);
            this.textBox1 = new System.Windows.Forms.TextBox();
            this.button1 = new System.Windows.Forms.Button();
            this.menuItem1 = new System.Windows.Forms.MenuItem();
            this.btnQuit = new System.Windows.Forms.Button();
            this.btnLogin = new System.Windows.Forms.Button();
            this.labelLoginID = new System.Windows.Forms.Label();
            this.toolTip1 = new System.Windows.Forms.ToolTip(this.components);
            this.btnShareFileFolder = new System.Windows.Forms.Button();
            this.textLoginID = new System.Windows.Forms.TextBox();
            this.chkRemeberID = new System.Windows.Forms.CheckBox();
            this.ListenCntxtMenu = new System.Windows.Forms.MenuItem();
            this.lbCopyright = new System.Windows.Forms.Label();
            this.SuspendLayout();
// contextMenu
this.contextMenu.MenuItems.AddRange(new System.Windows.Forms.MenuItem[] {
    this.ctxMenuQuit,
    this.ctxMenuShare});

// ctxMenuQuit
this.ctxMenuQuit.Index = 0;
this.ctxMenuQuit.Text = "Quit";
this.ctxMenuQuit.Click += new System.EventHandler(this.ctxMenuQuit_Click);

// ctxMenuShare
this.ctxMenuShare.Index = 1;
this.ctxMenuShare.Text = "Share File/Folders";
this.ctxMenuShare.Click += new System.EventHandler(this.ctxMenuShare_Click);

// ListenIcon
this.ListenIcon.ContextMenu = this.contextMenu;
this.ListenIcon.Icon = ((System.Drawing.Icon)(resources.GetObject("ListenIcon.Icon")));
this.ListenIcon.Text = "Peering Peers";
this.ListenIcon.Visible = true;

// textBox1
this.textBox1.Location = new System.Drawing.Point(895, 211);
this.textBox1.Name = "textBox1";
this.textBox1.TabIndex = 6;
this.textBox1.Text = "";
this.textBox1.Visible = false;
this.textBox1.WordWrap = false;

// button1
this.button1.Location = new System.Drawing.Point(450, 72);
this.button1.Name = "button1";
this.button1.TabIndex = 7;
this.button1.Visible = false;

// menuItem1
this.menuItem1.Index = -1;
this.menuItem1.Text = "";

// btnQuit
Chapter 5: The P2P Application (C#) 205

```csharp
this.btnExit.Location = new System.Drawing.Point(349, 145);
this.btnExit.Name = "btnExit";
this.btnExit.Size = new System.Drawing.Size(75, 24);
this.btnExit.TabIndex = 5;
this.btnExit.Text = "Quit";
this.toolTip1.SetToolTip(this.btnExit, "Quit and Close");
this.btnExit.Click += new System.EventHandler(this.btnExit_Click);

((System.Byte)(192)), ((System.Byte)(0)));
this.btnLogin.DialogResult = System.Windows.Forms.DialogResult.OK;
this.btnLogin.ForeColor = System.Drawing.Color.LemonChiffon;
this.btnLogin.Location = new System.Drawing.Point(74, 76);
this.btnLogin.Name = "btnLogin";
this.btnLogin.Size = new System.Drawing.Size(83, 29);
this.btnLogin.TabIndex = 4;
this.btnLogin.Text = "Login";
this.toolTip1.SetToolTip(this.btnLogin, "Log-in with this LoginID.");
this.btnLogin.Click += new System.EventHandler(this.btnLogin_Click);

this.labelLoginID.Location = new System.Drawing.Point(18, 30);
this.labelLoginID.Name = "labelLoginID";
this.labelLoginID.Size = new System.Drawing.Size(54, 14);
this.labelLoginID.TabIndex = 2;
this.labelLoginID.Text = "Login ID :");

this.btnExit.Click += new System.EventHandler(this.btnExit_Click);

this.btnExit.Click += new System.EventHandler(this.btnExit_Click);
this.btnExit.Click += new System.EventHandler(this.btnExit_Click);
this.btnExit.Click += new System.EventHandler(this.btnExit_Click);
```
```csharp
this.textLoginID.TabIndex = 0;
this.textLoginID.Text = "";
this.toolTip1.SetToolTip(this.textLoginID,
"Write your Login name here.");

// chkRemeberID
//
this.chkRemeberID.Location = new System.Drawing.Point(74, 54);
this.chkRemeberID.Name = "chkRemeberID";
this.chkRemeberID.Size = new System.Drawing.Size(134, 18);
this.chkRemeberID.TabIndex = 3;
this.chkRemeberID.Text = "Remember My Login ID";
this.toolTip1.SetToolTip(this.chkRemeberID,
"System will remember LoginID.");

// ListenCntxtMenu
//
this.ListenCntxtMenu.Index = -1;
this.ListenCntxtMenu.Text = "";

// lbCopyright
//
this.lbCopyright.Location = new System.Drawing.Point(226, 6);
this.lbCopyright.Name = "lbCopyright";
this.lbCopyright.Size = new System.Drawing.Size(196, 12);
this.lbCopyright.TabIndex = 8;
this.lbCopyright.Text = "© 2001 www.dreamtechsoftware.com";

// frmLogin
//
this.AcceptButton = this.btnLogin;
this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
this.CancelButton = this.btnQuit;
this.ClientSize = new System.Drawing.Size(438, 181);
this.ContextMenu = this.contextMenu;
this.Controls.AddRange(new System.Windows.Forms.Control[] {
    this.lbCopyright,
    this.btnShareFileFolder,
    this.labelLoginID,
    this.chkRemeberID,
    this.btnLogin,
    this.btnQuit,
    this.textBox1,
    this.button1});
this.Font = new System.Drawing.Font("Microsoft Sans Serif", 8F,
System.Drawing.FontStyle.Bold);
this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedDialog;
this.Icon = ((System.Drawing.Icon)(resources.GetObject("$this.Icon")));
this.MaximizeBox = false;
this.Name = "frmLogin";
```
Chapter 5: The P2P Application (C#) 207

```csharp
this.Text = "Peering Peers";
this.TransparencyKey = System.Drawing.Color.Olive;
this.Closing += new
System.ComponentModel.CancelEventHandler(
    this.frmLogin_Closing);
this.ResumeLayout(false);
}

// Quit clicked pack up, clean up and leave, and yes
// inform server calling "Logout()".
protected void ctxMenuQuit_Click (object sender, System.EventArgs e)
{
    Logout();
    this.Dispose();
    this.Close();
    Application.Exit();
}

// Time to be kind to share resources with others
// through "Share Files/Folders" dialog box
protected void ctxMenuShare_Click (object sender, System.EventArgs e)
{
    if(!bShareDialogOpend)
    {
        // open this dialog box only if it is not opened already
        bShareDialogOpend = true;
        // make an object of frmSelection form and show it
        frmSelection formSelection = new frmSelection();
        formSelection.ShowDialog();
        bShareDialogOpend = false;
    }
}

// another way of popping-up the "Share Files/Folders" dialog box just
// in case u didn’t like the first one
protected void btnShareFileFolder_Click (object sender, System.EventArgs e)
{
    frmSelection formSelection = new frmSelection();
    formSelection.ShowDialog();
}

// Quit clicked pack up, clean up and leave.
protected void btnQuit_Click (object sender, System.EventArgs e)
{
    Logout();
    this.Dispose();
    this.Close();
    Application.Exit();
}

// Unless u log-in the log book on server u don’t exist for browser
protected void btnLogin_Click (object sender, System.EventArgs e)
{
    string ServerAddress;
```
ServerAddress = 'http://webaddress/login.asp?USERID=" +
textLoginID.Text + 
"&IP=" + xmlServerComm.GetIPAddress("");

XMLSTRUCT xmlStruct;
LoginName = textLoginID.Text;

try {
    if( 0 != textLoginID.Text.Trim().Length )
    {
        string Share = "";

        // if remember user name check box's checked write username to
        // userinfo.ini
        if( chkRemeberID.Checked ) WriteUsername();

        // else delete that file as it is no longer needed
        else xmlServerComm.FileDelete(Application.StartupPath +"\Userinfo.ini");

        // if u had never shared anything set SHARE parameter
to ASP to empty
        if( !
            File.Exists(Application.StartupPath+"\Share.ini") )
            Share = "";
        else
        {
            // else read share.ini and read all the files and folders
            // u had shared and set them to SHARE parameter
            StreamReader stReader = new
                StreamReader(Application.StartupPath +"\Share.ini");

            string readData;
            while((readData = stReader.ReadLine()) != null)
            {
                if(!readData.Substring(0,
                    readData.IndexOf("=")).EndsWith("\"\")
                    Share += readData.Substring(0,
                        readData.IndexOf("=")).Substring(
                    readData.Substring(0,readData.IndexOf(
                        ")-"())).LastIndexOf("\")+1) + "*";
                else
                    Share +=
                        readData.Substring(0,readData.IndexOf(
                        ")-"())) + "*";

            }
            stReader.Close();

            ServerAddress += 
                
                ServerAddress += 
                "&SHARE=" + Share;

            // make a call to loging.asp and recieve the response in an XML file
            xmlServerComm.WriteDataToFile(
                xmlParser.LOGINXML,
                xmlServerComm.GetDataFromServer(
                    ServerAddress));
Chapter 5: The P2P Application (C#) 209

```
398  // parse that file and store in XMLSTRUCT
399  xmlParser.ParseXML(xmlParser.LOGINXML, out xmlStruct,
400      xmlServerComm.TypeOfXMLRecieved(xmlParser.LOGINXML));
401  // delete XML file
402  xmlServerComm.FileDelete(xmlParser.LOGINXML);
403  // now check whether login was successful
404  if (0 == xmlStruct.AUTH.iCode)
405  {
406      //
407      buffer = new Byte[256];
408      servSock = new Socket(
409          AddressFamily.InterNetwork,
410          SocketType.Stream,
411          ProtocolType.Tcp);
412      // if successful make a socket giving the IP and port
413      IPAddress localIP = IPAddress.Parse((
414          xmlServerComm.GetIPAddress("")));
415      IPEndPoint localEP = new IPEndPoint(
416          localIP, 7070);
417      servSock.Bind(localEP);
418      servSock.Listen(40);
419      
420      AcceptThread = new Thread(new ThreadStart(AcceptFunction));
421      
422      AcceptThread.Start();
423      
424      // make a new thread to keep updating the icons on sys tray
425      ThreadIcon = new Thread(new ThreadStart(IconUpdate));
426      ThreadIcon.Priority = ThreadPriority.BelowNormal;
427      ThreadIcon.Start();
428      
429      // hide the login form
430      this.Hide();
431  }
432  else
433  {
434      // if unsuccessful show user the error occurred
435      MessageBox.Show(xmlStruct.AUTH.sStatus,
436          "Peering Peers", MessageBoxButtons.OK,
437          MessageBoxIcon.Information);
438      textLoginID.Focus();
439      textLoginID.SelectAll();
440  }
441  }
442  else
443  {
444      // before actually writing your username u cannot loggin
445      MessageBox.Show("Blank LoginID detected",
```
"Peering Peers", MessageBoxButtons.OK,
    MessageBoxIcon.Error);
        textLoginID.Focus();
        textLoginID.SelectAll();
    }
    catch( Exception err )
    {
        // if logging fails tell user why it did
        // "Could'nt Login."
        MessageBox.Show(err.Message,
            "Peering Peers", MessageBoxButtons.OK,
            MessageBoxIcon.Error);
        // and close application
        this.Dispose();
        this.Close();
        Application.Exit();
    }
    // declare that logging process's successful
    bLoggedIn = true;
}

// keep updating ur looks to avoid being overlooked
protected void IconUpdate()
{
    try
    {
        while(true)
        {
            IconNo += 1;
            string Filename;
            // if something's getting downloaded or uploaded
            // pick different icons to put on sys tray.
            if(bDownLoading)
            {
                if((IconNo) == 5)
                    IconNo = 1;
                Filename = "\Trans" + IconNo.ToString() + ".ico";
                ListenIcon.Text = "Downloading or Uploading in Progress";
            }
            else
            {
                // otherwise keep cycling these four icons on sys tray.
                if((IconNo) == 5)
                {
                    IconNo = 1;
                    Filename = "\Listener" + 
                        IconNo.ToString() + ".ico";
                    ListenIcon.Text = "Peering Peers";
                }
                System.Drawing.Icon icon = new System.Drawing.Icon(
                    Application.StartupPath + Filename);
                ListenIcon.Icon = icon;
                // make the process a bit slow for user to be able to
493     // notice it
494     System.Threading.Thread.Sleep(500);
495     
496     }
497     catch(Exception e) { WriteErrorLog(  
498         "Updating Sys-tray icon.",e.Message);  }
499     }
500     
501     // Keep a log of all the exceptions and errors.
502     protected void WriteErrorLog(string Origin, string Message)
503     {
504         StreamWriter stWriter = File.AppendText(Application.StartupPath +  
505         "\Error.log");
506         // write the time of occurrence of this error
507         stWriter.WriteLine("Origin = " + Origin +  
508         " Date & Time (ms-s-m-h dd/mm/yy/) = " +  
509         DateTime.Now.Millisecond.ToString() +  
510         "-" +  
511         DateTime.Now.Second.ToString() +  
512         "-" +  
513         DateTime.Now.Minute.ToString() +  
514         "-" +  
515         DateTime.Now.Hour.ToString() +  
516         "/" +  
517         DateTime.Now.Month.ToString() +  
518         "/" +  
519         DateTime.Now.Year.ToString() +  
520         " Error Message = " + Message);
521         
522         stWriter.Close();
523     }
524     
525     // Form closing, dispose all the resources used and Exit the Application
526     protected void frmLogin_Closing (object sender,
527         System.ComponentModel.CancelEventArgs e)
528     {
529         Logout();
530         this.Dispose();
531         this.Close();
532         Application.Exit();
533     }
534     
535     // Keeps accepting connections
536     protected void AcceptFunction()  
537     {
538         try
539         {
540             while(true)  
541             {
542                 newSock = servSock.Accept();
543             
544             
545             RespondingThread = new Thread(new
ThreadStart(RequestResponse));
RespondingThread.Priority = ThreadPriority.Highest;
RespondingThread.Start();
}
}
}
}catch( Exception e ) { WriteErrorLog("Client's attempt of establishing connection failed.",e.Message); }
}

// Respond to the requests made
protected void RequestResponse()
{
    Encoding ASCII = Encoding.ASCII;
    NetworkStream DataStream = new NetworkStream(newSock);
    try
    {
        Byte[] readBytes = new Byte[1024],read = new Byte[1024];
        string[] ChatInfo;
        int bytes = 0,UploadDownloadPrint;
        bool responseWritten = false;
        FileStream fs1 = null;
        try
        {
            Byte[] readBytes = new Byte[1024],read = new Byte[1024];
            string[] ChatInfo;
            int bytes = 0,UploadDownloadPrint;
            bool responseWritten = false;
            FileStream fs1 = null;
            // read what the browser wrote on its stream and
            bytes = DataStream.Read(read, 0, read.Length);
            // make a temporary file
            FileStream fs = new FileStream(Application.StartupPath + "\Temp.xml", FileMode.OpenOrCreate);
            // then read file and determine the kind of request
            xmlCreater = new XMLCreater(Application.StartupPath + "\Temp1.xml",Application.StartupPath + "\Share.ini");
            string DnLoadFile = xmlCreater.DetermineRequestType(Application.StartupPath + "\Temp.xml",out UploadDownloadPrint,out ChatInfo);
            // if it's download/upload request make file object for
            // the file browser has requested to upload to
            // or download from his comp
            if(DnLoadFile.Length != 0)
            {
                if(UploadDownloadPrint == 1)
                {
                    fs1 = new FileStream(DnLoadFile,FileMode.CreateNew);
                }
                else if(UploadDownloadPrint == 2)
                {
                    fs1 = new FileStream(DnLoadFile,FileMode.Open);
                }
                bDownLoading = true;
            }
            else
            {
Chapter 5: The P2P Application (C#) 213

595 // else make a file object for responding in xml formate
596 fs1 = new FileStream(Application.StartupPath + "\Temp1.xml", FileMode.Open);
597 responseWritten = true;
598 }

599 // write response
600 if(UploadDownloadPrint == 1)
601 {
602 // if it was upload request
603 BinaryWriter wr = new BinaryWriter(fs1);
604 int nreadBytes;
605 // read network stream and write data to a loacal file
606 while( (nreadBytes = DataStream.Read(readBytes, 0, readBytes.Length)) > 0)
607 wr.Write(readBytes, 0, nreadBytes);
608 wr.Close();
609 fs1.Close();
610 }
611 else
612 {
613 // write the response or file requested for download
614 // in browsers stream
615 BinaryReader r = new BinaryReader(fs1);
616 int nreadBytes;
617 // write upon the network stream
618 while( (nreadBytes = r.Read( readBytes, 0, readBytes.Length)) > 0)
619 DataStream.Write(readBytes, 0, nreadBytes);
620 r.Close();
621 fs1.Close();
622 }

623 // delete all temporary files
624 if(responseWritten)
625 File.Delete(Application.StartupPath + "\Temp1.xml");
626 File.Delete(Application.StartupPath + "\Temp.xml");
627 DataStream.Flush();
628 newSock.Close();
629 bDownLoading = false;
630 }
631 catch(SocketException e) {
632 WriteErrorLog("Responding Browser",e.Message);
633 }
634 /// The main entry point for the application.
635 public static void Main(string[] args)
636 {
637 Application.Run(new frmLogin());
638 }

639 // Listener's through, it will inform server that it is making an exit.
public void Logout ()
{
    try
    {
        if(bLoggedIn)
            
            MessageBox.Show("Listener Logged-out.", "Process Successfull", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}

//release threads
if(ThreadIcon!=null)
      if(ThreadIcon.IsAlive)
        ThreadIcon.Abort();

if(AcceptThread!=null)
      if(AcceptThread.IsAlive)
        AcceptThread.Abort();

if(RespondingThread!=null)
      if(RespondingThread.IsAlive)
        RespondingThread.Abort();

    }
}

    }

    }

    }

    }

    }

    }

    }

    }

    catch( Exception err ) { MessageBox.Show(err.Message, "Could not Logout properly.", MessageBoxButtons.OK, MessageBoxIcon.Error);}
♦ Lines 118-126: The `WriteUserName` function writes the user name to the `userinfo.ini` file with the help of the stream class’s `OpenWrite()` function for use in next instantiation of this application.

♦ Lines 128-132: `Dispose()` disposes all resources of the application at its termination.

♦ Lines 135-310: `InitializeComponent()` initializes all GUI components. This part of coding is IDE generated, and it is recommended that programmers do not modify this code.

The following functions are related with various buttons placed on forms. These functions provide required functionality and instructions to perform specified tasks. This section incorporates detailed functionality with complete references regarding various options related to the context or shortcut menu. First, we give you the functionality of the context menu or shortcut menu. Some of the functions implemented are detailed later.

♦ Lines 314-320: The `ctxMenuQuit_Click()` function controls the Quit option of context menu. This function is invoked when you click the Quit option of the shortcut menu. To log out and free the threads, it calls the `Logout()` function, and the application exists.

♦ Lines 324-335: The `ctxMenuShare_Click` function is invoked when you click the Share Resource option of context menu. It opens the Share Files/Folders dialog box.

♦ Lines 338-344: `btnShareFileFolder_Click()` holds the functionality of the Share button located on the application’s form. Once you click this button, it opens the Share File/Folder dialog box.

♦ Lines 335-352: `btnQuit_Click()` comes into action when you click the Quit button located on the application’s form. To log out and free the threads, it calls the `Logout()` function, and the application exists.

♦ Lines 355-461: This section of code holds for the entire breath of the application. In the lines that follow, you are logging on to the server and communicating with it. (Unless you log in to the log book on the server, you don’t exist for the browser.) The role of the Login button holds for the entire working to facilitate logging on to the server. `btnLogin_Click()` is invoked when you click the Login button placed on the application’s form.

- 359: This line is making use of the `ServerCommunication` (defined in the `WorkingWithXML` assembly) class object’s `GetIPAddress` function to get the IP address of the computer this application is running on. A list of all the shared files and folders is passed to server, along with the listener’s login name during connection to server.

- 370: This line of code checks whether you have checked the Remember My Login ID checkbox or not. If this checkbox is checked, your user name is written to `userinfo.ini` for use in the next instantiation of this application by a call to `WriteUserName()`.

- 373: Just in case you haven't checked the checkbox, the `userinfo.ini` file is deleted, as it is no longer required.

- 376-377: This checks the existence of `share.ini` to make sure you have shared some resources.

- 378-391: If this file is found, it is opened and all the shared files/folders are read into a string variable, with delimiters after every file/folder name.

- 393: This information is then concatenated as another parameter, along with `USERID` and IP to the URL of login.asp to make it look like the following:

  `http://webaddress/login.asp?USERID=keith&IP=192.192.192.192&SHARE=smith.txt*cris.exe*C:\Shared\Temp\`*  

- 397: This calls login.asp by using the `ServerCommunication` class’s `GetDataFromServer()` method, passing three parameters: `USERID`, `IP`, and `SHARE` (shared resources) and stores the response as an XML document by using the `ServerSubscription` class object’s `WriteDataToFile()` function.
- 400: The XML response document is parsed by XMLParser’s ParseXML() function. The XMLParser class is defined in the WorkingWithXML assembly. ParseXML() returns the output to the XMLSTRUCT variable (defined in the WorkingWithXML assembly).
- 405-439: Now the type of response is determined by checking output and is returned to XMLSTRUCT’s variable. Here we are checking the XML response, indicating that authentication is successful, after which the listener begins the process of listening for connections.
- 410-417: A socket is made, giving it the IP address and port number to listen on.
- 421-422: The AcceptFunction() function is then called in a thread to keep on listening for connections. This function is detailed later.
- 423-429: Another thread is created to keep updating icons on the system tray by passing the IconUpdate() function in it, which is coming up next.
- 436-438: If the login is unsuccessful, the message is sent to the user.
- 442-446: If the login name is detected as blank, the user is taken back to the login name box to write the login name.
- 447-457: If the login is unsuccessful, the reason given to the user and the application is closed after releasing all resources and threads.
- 459: A flag is set to indicate that logging is successful.

- Lines 462-498: The IconUpdate() function is used by this application to show its presence on the machine. It keeps updating the icons on the system tray once the login form is closed, and no other window belonging to this application is open.
- 474-489: Check if anything is being downloaded, then change the tray icons accordingly.
- 490: The file name generated previously is loaded.
- 491: This loaded icon is set to the system tray object’s Icon property of this application.
- 493: A thread is made to sleep for 500 milliseconds before the next update is made to make the process noticeable to reader.

- Lines 500-523: WriteErrorLog() is called by other functions as they encounter any internal problem. This function stores errors, their type in error.log, and their time of occurrence as it runs in unattended mode. You can check it any time to see exceptions the application has encountered.
- Lines 526-534: frmLogin_Closing() is called when the application’s Login form is closed by the user either by clicking the Quit button or by clicking the Cross button, in both cases without logging in. It enables the user to dispose all resources the application has been holding and to close the form and finally to terminate the application.
- Lines 536-550: The AcceptFunction() function keeps accepting connections; that is why use of the while(true) looping structure is put into code. The while(true) statement runs this process until the thread in which this function is running is aborted. As soon as a connection is detected, the RequestResponse() function is called to serve the requests made on this connection.
- Lines 552-632: The RequestResponse() function appropriately responds to requests made on a connection.
  - 556: This opens a network stream passing the socket to the calling function.
  - 560-576: The user (listener) is reading via a network stream whatever the user at other end has written to it and is storing the information to a temporary XML file.
  - 579: An object of the XMLCreater class is created and initialized. The XMLCreater class is defined in the WorkingWithXML assembly.
• 580: The `DetermineRequestType()` function of the XMLCreator class is called to determine the request type and to retrieve the information accordingly.

• 584-591: If the request type is uploading, an empty file is created with the same name as specified in the uploaded request to store the downloaded file. Otherwise, a file is opened for uploading.

• 593-597: If none of these requests is found, an empty XML document is created to store responses for other request types.

• 600-610: If the request is of uploading type, the file stream for writing is opened, and the network stream is read for the file being written by the browser on it in the form of bytes; read bytes are written in an empty document created earlier.

• 611-622: If it is a download request, search request or merely a show all files request. The listener makes a reader stream either for reading the file to be downloaded or for reading the response file. Either of these read files are written on a network stream for the browser to read on his end.

• 623-630: All temporary files are deleted while closing the connection.

♦ Lines 634-637: `Main()` starts the application and instantiates `frmLogin`.

♦ Lines 639-660: Once the listener completes its work and its processing, it disposes all resources, releases all threads, and exits the application.

### Share Files/Folders Form

**namespace**: Listener

The `frmSelection` class provides the user at the listener’s end with an intuitive interface, facilitating the user with the process of sharing resources (see Listing 5-5).

The dialog box in Figure 5-4 is displayed when you click the Share Files/Folders button on the login form or choose the same option through the tray icon short-cut menu.

![Share Dialog Window](ShareDialog.png)

**Figure 5-4: The Share Dialog window**

♦ **Share File Tab**: This tab is devoted to file sharing.

♦ **Browse**: This opens the File Open dialog box for you to select any file to share.

♦ **Share Folder Tab**: This tab is devoted to file sharing.

♦ **Browse**: This opens the Folder Selection dialog box for you to select any folder to share.
♦ **Share It:** This shares the file or folder selected under either of the two preceding tabs.

♦ **Rights:** This group box offers two credential options to choose from to assign to the selected file or folder.

♦ **Add this entry only:** This checkbox is given for you to indicate whether the dialog box should be closed or not after choosing one file or folder.

♦ **Close:** This closes the dialog box.

### Listing 5-5: frmSelection

```csharp
// © 2001 Dreamtech Software India Inc.
// All rights reserved

namespace Listener
{
    // Library includes
    using System;
    using System.Drawing;
    using System.Collections;
    using System.ComponentModel;
    using System.Windows.Forms;
    using System.Windows.Forms.Design;
    using System.Text;
    using System.IO;

    ///    Share Files/Folders form class.
    public class frmSelection : System.Windows.Forms.Form
    {
        private System.ComponentModel.IContainer components;
        private System.Windows.Forms.Label lbCopyright;
        private System.Windows.Forms.ToolTip toolTipText;
        private System.Windows.Forms.CheckBox chkEntry;
        private System.Windows.Forms.GroupBox grpRights;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.TextBox txtFoldername;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.TextBox txtFilename;
        private System.Windows.Forms.TabPage tpFolder;
        private System.Windows.Forms.TabPage tpShare;
        private System.Windows.Forms.TabControl tabShare;

        public frmSelection() { InitializeComponent(); } }

        // Free resources it was using.
        public override void Dispose()
        {
            base.Dispose();
            components.Dispose();
        }
```
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
private void InitializeComponent()
{
    this.components = new System.ComponentModel.Container();
    this.grpRights = new System.Windows.Forms.GroupBox();
    this.rbWrite = new System.Windows.Forms.RadioButton();
    this.rbRead = new System.Windows.Forms.RadioButton();
    this.chkEntry = new System.Windows.Forms.CheckBox();
    this.toolTipText = new System.Windows.Forms.ToolTip(this.components);
    this.openFileDialog1 = new System.Windows.Forms.OpenFileDialog();
    this.label1 = new System.Windows.Forms.Label();
    this.label2 = new System.Windows.Forms.Label();
    this.tpShare = new System.Windows.Forms.TabPage();
    this.btnBrowse = new System.Windows.Forms.Button();
    this.txtFilename = new System.Windows.Forms.TextBox();
    this.tabShare = new System.Windows.Forms.TabControl();
    this.tpFolder = new System.Windows.Forms.TabPage();
    this.btnBrowseFileFolder = new System.Windows.Forms.Button();
    this.txtFoldername = new System.Windows.Forms.TextBox();
    this.btnCancel = new System.Windows.Forms.Button();
    this.grpRights.SuspendLayout();
    this.tpShare.SuspendLayout();
    this.tabShare.SuspendLayout();
    this.tpFolder.SuspendLayout();
    this.SuspendLayout();
    //
    // grpRights
    //
    this.grpRights.Controls.AddRange(new System.Windows.Forms.Control[] { this.rbWrite, this.rbRead });
    this.grpRights.Location = new System.Drawing.Point(16, 128);
    this.grpRights.Name = "grpRights";
    this.grpRights.Size = new System.Drawing.Size(124, 56);
    this.grpRights.TabIndex = 3;
    this.grpRights.TabStop = false;
    this.grpRights.Text = "Rights";
    //
    // rbWrite
    //
    this.rbWrite.Location = new System.Drawing.Point(6, 36);
    this.rbWrite.Name = "rbWrite";
    this.rbWrite.Size = new System.Drawing.Size(96, 14);
    this.rbWrite.TabIndex = 3;
    this.rbWrite.Text = "Read/Write";
    this.toolTipText.SetToolTip(this.rbWrite, "Allows the user to upload to this file or folder");
    //
    // rbRead
    //
    this.rbRead.Checked = true;
    this.rbRead.Location = new System.Drawing.Point(6, 54);
    this.rbRead.Name = "rbRead";
    this.rbRead.Size = new System.Drawing.Size(96, 14);
    this.rbRead.TabIndex = 4;
    this.rbRead.Text = "Read Only";
    this.toolTipText.SetToolTip(this.rbRead, "Prevents the user from uploading to this file or folder");
    //
    // chkEntry
    //
    this.chkEntry.Location = new System.Drawing.Point(6, 71);
    this.chkEntry.Name = "chkEntry";
    this.chkEntry.Size = new System.Drawing.Size(96, 17);
    this.chkEntry.TabIndex = 5;
    this.chkEntry.Text = "Entry Only";
    this.toolTipText.SetToolTip(this.chkEntry, "Prevents the user from uploading to this file or folder");
    //
    // tabPageShare
    //
    this.tpShare.Controls.AddRange(new System.Windows.Forms.Control[] { this.lblShare, this.txtShare, this.btnShare });
    this.tpShare.Location = new System.Drawing.Point(4, 27);
    this.tpShare.Name = "tpShare";
    this.tpShare.Padding = new System.Windows.Forms.Padding(3);
    this.tpShare.Size = new System.Drawing.Size(181, 145);
    this.tpShare.TabIndex = 2;
    this.tpShare.Text = "Share";
    //
    // labelShare
    //
    this.lblShare.Location = new System.Drawing.Point(6, 9);
    this.lblShare.Name = "lblShare";
    this.lblShare.Size = new System.Drawing.Size(210, 14);
    this.lblShare.TabIndex = 10;
    this.lblShare.Text = "Enter the path of the file or folder you wish to share:
    //";
    //
    // txtShare
    //
    this.txtShare.Location = new System.Drawing.Point(6, 22);
    this.txtShare.Name = "txtShare";
    this.txtShare.Size = new System.Drawing.Size(210, 20);
    this.txtShare.TabIndex = 11;
    this.txtShare.Text = "C:\MyShare\";
    //
    // btnShare
    //
    this.btnShare.Location = new System.Drawing.Point(6, 42);
    this.btnShare.Name = "btnShare";
    this.btnShare.Size = new System.Drawing.Size(96, 25);
    this.btnShare.TabIndex = 12;
    this.btnShare.Text = "Share";
    this.toolTipText.SetToolTip(this.btnShare, "Click to share the selected path
    //");
    //
    // tabPageFolder
    //
    this.tpFolder.Controls.AddRange(new System.Windows.Forms.Control[] {this.txtFoldername, this.btnBrowseFileFolder, this.btnCancel, this.lbCopyright, this.grpRights });
    this.tpFolder.Location = new System.Drawing.Point(4, 27);
    this.tpFolder.Name = "tpFolder";
    this.tpFolder.Padding = new System.Windows.Forms.Padding(3);
    this.tpFolder.Size = new System.Drawing.Size(181, 145);
    this.tpFolder.TabIndex = 4;
    this.tpFolder.Text = "Folder";
    //
    // txtFoldername
    //
    this.txtFoldername.Location = new System.Drawing.Point(6, 9);
    this.txtFoldername.Name = "txtFoldername";
    this.txtFoldername.Size = new System.Drawing.Size(210, 20);
    this.txtFoldername.TabIndex = 11;
    this.txtFoldername.Text = "C:\MyFolder\";
    //
    // btnBrowseFileFolder
    //
    this.btnBrowseFileFolder.Location = new System.Drawing.Point(6, 42);
    this.btnBrowseFileFolder.Name = "btnBrowseFileFolder";
    this.btnBrowseFileFolder.Size = new System.Drawing.Size(96, 25);
    this.btnBrowseFileFolder.TabIndex = 12;
    this.btnBrowseFileFolder.Text = "Choose Folder";
    this.toolTipText.SetToolTip(this.btnBrowseFileFolder, "Click to browse for the
    //path of the file or folder you wish to share");
    //
    // btnCancel
    //
    this.btnCancel.Location = new System.Drawing.Point(108, 42);
    this.btnCancel.Name = "btnCancel";
    this.btnCancel.Size = new System.Drawing.Size(75, 25);
    this.btnCancel.TabIndex = 14;
    this.btnCancel.Text = "Cancel";
    this.toolTipText.SetToolTip(this.btnCancel, "Cancels the current operation
    //and returns to the previous window");
    //
    // lbCopyright
    //
    this.lbCopyright.Location = new System.Drawing.Point(6, 102);
    this.lbCopyright.Name = "lbCopyright";
    this.lbCopyright.Size = new System.Drawing.Size(210, 20);
    this.lbCopyright.TabIndex = 16;
    this.lbCopyright.Text = "Copyright © 2019 by [Your Name]. All Rights Reserved.
    //";
    //
    // tabPageFileOpenDialog
    //
    this.tpFileOpenDialog.Controls.AddRange(new System.Windows.Forms.Control[] { this.txtFilename, this.btnOK, this.FileOpenDialog });
    this.tpFileOpenDialog.Location = new System.Drawing.Point(4, 27);
    this.tpFileOpenDialog.Name = "tpFileOpenDialog";
    this.tpFileOpenDialog.Padding = new System.Windows.Forms.Padding(6);
    this.tpFileOpenDialog.Size = new System.Drawing.Size(181, 145);
    this.tpFileOpenDialog.TabIndex = 5;
    this.tpFileOpenDialog.Text = "File Open Dialog";
    //
    // txtFilename
    //
    this.txtFilename.Location = new System.Drawing.Point(6, 9);
    this.txtFilename.Name = "txtFilename";
    this.txtFilename.Size = new System.Drawing.Size(210, 20);
    this.txtFilename.TabIndex = 11;
    this.txtFilename.Text = "C:\MyFile.txt";
    //
    // btnOK
    //
    this.btnOK.Location = new System.Drawing.Point(6, 42);
    this.btnOK.Name = "btnOK";
    this.btnOK.Size = new System.Drawing.Size(96, 25);
    this.btnOK.TabIndex = 12;
    this.btnOK.Text = "Open File";
    this.toolTipText.SetToolTip(this.btnOK, "Opens the selected file or folder
    //and displays it in the main window");
    //
    // FileOpenDialog
    //
    this.FileOpenDialog.Filter = "*.*";
    this.FileOpenDialog.RestoreDirectory = false;
    this.FileOpenDialog.Title = "Open File or Folder";
    //
    // Form1_Load
    //
    this.AutoScaleDimensions = new System.Drawing.SizeF(6f, 13f);
    this.ClientSize = new System.Drawing.Size(210, 251);
    this.ControlBox = false;
    this.Controls.Add(this.tabShare);
    this.Controls.Add(this.grpRights);
    this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedSingle;
    this.MaximizeBox = false;
    this.Name = "Form1";
    this.Text = "Rights Management Screen";
    //
    // Form1_Load
    //
    this.grpRights.ResumeLayout(false);
    this.tpShare.ResumeLayout(false);
    this.tabShare.ResumeLayout(false);
    this.tpFolder.ResumeLayout(false);
    this.ResumeLayout(false);
}
this.rbRead.Location = new System.Drawing.Point(6, 16);
this.rbRead.Name = "rbRead";
this.rbRead.Size = new System.Drawing.Size(95, 14);
this.rbRead.TabIndex = 2;
this.rbRead.TabStop = true;
this.rbRead.Text = "Read only";
this.toolTipText.SetToolTip(this.rbRead, "Allows the user to read");

//
// chkEntry
//
this.chkEntry.Location = new System.Drawing.Point(296, 128);
this.chkEntry.Name = "chkEntry";
this.chkEntry.Size = new System.Drawing.Size(120, 16);
this.chkEntry.TabIndex = 4;
this.chkEntry.Text = "Add this entry only";
this.toolTipText.SetToolTip(this.chkEntry, "Quits after adding the entry, if checked");

//
// label1
//
this.label1.Location = new System.Drawing.Point(8, 8);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(248, 14);
this.label1.TabIndex = 2;
this.label1.Text = "Type a filename here or click browse to select:";

//
// label2
//
this.label2.Location = new System.Drawing.Point(8, 8);
this.label2.Name = "label2";
this.label2.Size = new System.Drawing.Size(232, 14);
this.label2.TabIndex = 1;
this.label2.Text = "Type a folder name here:";

//
// tpShare
//
this.tpShare.Controls.AddRange(new System.Windows.Forms.Control[] {
    this.btnBrowse,
    this.label1,
    this.label2,
    this.txtFilename});
this.tpShare.Location = new System.Drawing.Point(4, 25);
this.tpShare.Name = "tpShare";
this.tpShare.Size = new System.Drawing.Size(388, 89);
this.tpShare.TabIndex = 0;
this.tpShare.Text = "Share File";

//
// btnBrowse
//
this.btnBrowse.Location = new System.Drawing.Point(296, 56);
this.btnBrowse.Name = "btnBrowse";
this.btnBrowse.Size = new System.Drawing.Size(80, 24);
this.btnBrowse.TabIndex = 1;
this.btnBrowse.Text = "&Browse";
this.btnBrowse.Click += new System.EventHandler(this.btnBrowseFile_Click);
Chapter 5: The P2P Application (C#) 221

```csharp
154 // txtFilename
155 //
156 this.txtFilename.Location = new System.Drawing.Point(13, 26);
157 this.txtFilename.Name = "txtFilename";
158 this.txtFilename.Size = new System.Drawing.Size(363, 20);
159 this.txtFilename.TabIndex = 0;
160 this.txtFilename.Text = "";
161 //
162 // tabShare
163 //
164 this.tabShare.Appearance = System.Windows.Forms.TabAppearance.FlatButtons;
165 this.tabShare.Controls.AddRange(new System.Windows.Forms.Control[] {
    this.tpShare,
    this.tpFolder});
166 this.tabShare.Location = new System.Drawing.Point(16, 8);
167 this.tabShare.Name = "tabShare";
168 this.tabShare.SelectedIndex = 0;
169 this.tabShare.Size = new System.Drawing.Size(396, 118);
170 this.tabShare.TabIndex = 2;
171 //
172 // tpFolder
173 //
174 this.tpFolder.Controls.AddRange(new System.Windows.Forms.Control[] {
    this.btnBrowseFileFolder,
    this.label2,
    this.txtFoldername});
175 this.tpFolder.Location = new System.Drawing.Point(4, 25);
176 this.tpFolder.Name = "tpFolder";
177 this.tpFolder.Size = new System.Drawing.Size(388, 89);
178 this.tpFolder.TabIndex = 1;
179 this.tpFolder.Text = "Share Folder";
180 //
181 // btnBrowseFileFolder
182 //
183 this.btnBrowseFileFolder.Location = new System.Drawing.Point(296, 56);
184 this.btnBrowseFileFolder.Name = "btnBrowseFolder";
185 this.btnBrowseFileFolder.Size = new System.Drawing.Size(80, 24);
186 this.btnBrowseFileFolder.TabIndex = 2;
187 this.btnBrowseFileFolder.Text = "Browse";
188 this.btnBrowseFileFolder.Click += new System.EventHandler(this.btnBrowseFolder_Click);
189 //
190 // txtFoldername
191 //
192 this.txtFoldername.Location = new System.Drawing.Point(13, 26);
193 this.txtFoldername.Name = "txtFoldername";
194 this.txtFoldername.Size = new System.Drawing.Size(363, 20);
195 this.txtFoldername.TabIndex = 0;
196 this.txtFoldername.Text = "";
197 //
198 // btnOK
199 //
```
this.btnOK.DialogResult = System.Windows.Forms.DialogResult.OK;
this.btnOK.ForeColor = System.Drawing.Color.LemonChiffon;
this.btnOK.Location = new System.Drawing.Point(240, 168);
this.btnOK.Name = "btnOK";
this.btnOK.Size = new System.Drawing.Size(85, 25);
this.btnOK.TabIndex = 0;
this.btnOK.Text = "&Share it";
this.btnOK.Click += new System.EventHandler(this.btnOK_Click);

//
// lbCopyright
//
this.lbCopyright.Location = new System.Drawing.Point(208, 8);
this.lbCopyright.Name = "lbCopyright";
this.lbCopyright.Size = new System.Drawing.Size(200, 12);
this.lbCopyright.TabIndex = 5;
this.lbCopyright.Text = "© 2001 www.dreamtechsoftware.com";

//
// btnCancel
//
this.btnCancel.Location = new System.Drawing.Point(336, 168);
this.btnCancel.Name = "btnCancel";
this.btnCancel.Size = new System.Drawing.Size(85, 24);
this.btnCancel.TabIndex = 1;
this.btnCancel.Text = "&Close";

//
// frmSelection
//
this.AcceptButton = this.btnOK;
this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
this.CancelButton = this.btnCancel;
this.ClientSize = new System.Drawing.Size(432, 203);
this.Controls.AddRange(new System.Windows.Forms.Control[] { this.lbCopyright,
this.chkEntry,
this.grpRights,
this.btnCancel,
this.btnOK,
this.tabShare });
this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedToolWindow;
this.HelpButton = true;
this.Icon = null;
this.Name = "frmSelection";
this.ShowInTaskbar = false;
this.Text = "Share Dialog";
this.grpRights.ResumeLayout(false);
this.tpShare.ResumeLayout(false);
Chapter 5: The P2P Application (C#) 223

```csharp
            this.tabShare.ResumeLayout(false);
            this.tpFolder.ResumeLayout(false);
            this.ResumeLayout(false);
        }

        //Browse folder Button clicked, open folder open dialog
        protected void btnBrowseFolder_Click (object sender, System.EventArgs e)
        {
            BrowseFolder Folder = new BrowseFolder();
            txtFoldername.Text = Folder.SelectFolder();
        }

        // Checkes whether file or folder user's trying to share
        // is already shared
        private bool DoesEntryExists(string Filename, string Data)
        {
            bool bFound = false;
            if( ! File.Exists(Filename) )
                return bFound;
            else
            {
                StreamReader stReader = new StreamReader(Filename);
                string readData;
                while((readData = stReader.ReadLine()) != null)
                    // what u read's equal or shorter in length than what ur intend to
                    // share
                    // that means if i've shared "C:\" it'll allow u to share "C:\Shared"
                    // finding substring alone wouldn't have worked here.
                    if(readData.Length <= Data.Length)
                        if( -1 !=
                            readData.Substring(0,readData.IndexOf(="")).Trim().IndexOf(Data.Substring(0,Data.IndexOf(=""))) )
                            {
                                bFound = true;
                                break;
                            }
                stReader.Close();
            }
            return bFound;
        }

        // checks the file of folder passed to it is not shared already.
        // checks it exists and writes that to the share.ini file.
        private void WriteDataToFile(string Filename, string Data)
        {
            try
            {
                // if it's a folder
                if( 0 == Data.Substring(Data.Length-3,1).CompareTo("\"))
                {
                    // check if this entry already exists
                    if( !DoesEntryExists(Filename, Data) )
                    {
                        // if not append this entry to share.ini with its mask
                        StreamWriter stWriter = File.AppendText(Filename);
```
stWriter.WriteLine(Data);
MessageBox.Show("'" + Data.Substring(0,Data.IndexOf('=')) + "' has
been successfully shared.","Information",MessageBoxButtons.OK,
 MessageBoxIcon.Information);
stWriter.Close();
else throw new Exception("Entry already exists");
}
else if( File.Exists(Data.Substring(0,Data.IndexOf("=",0))) )
{
// else if it's a file and it exists
if( !DoesEntryExists(Filename, Data) )
{
// and if it doesn't exist already append this entry
// to share.ini with its size and mask
StreamWriter stWriter = File.AppendText(Filename);
stWriter.WriteLine(Data);
MessageBox.Show("'" + Data.Substring(0,Data.IndexOf('=')) + "' is
successfully shared.","Information",MessageBoxButtons.OK,
MessageBoxIcon.Information);
stWriter.Close();
} else throw new Exception("Entry already exists");
else throw new Exception("File/Folder does not exists");
}
catch( Exception err )
{
MessageBox.Show(err.Message,"Warning",MessageBoxButtons.OK,
MessageBoxIcon.Warning); }
}

// Share file or folder selected
protected void btnOK_Click (object sender, System.EventArgs e)
{
string ResourceName = null;
long fileSize = 0;
bool bIsFile = false;
try
{
System.Windows.Forms.TabPage tp = tabShare.SelectedTab;
// find the file or folder selected
if( 0 == tp.Text.Trim().CompareTo("Share File") )
{
ResourceName = txtFilename.Text;
Stream s = File.Open(txtFilename.Text,FileMode.Open);
fileSize = s.Length;
bIsFile = true;
}
else if( 0 == tp.Text.Trim().CompareTo("Share Folder") )
{
ResourceName = txtFoldername.Text;
if( !Directory.Exists(ResourceName) )
{
}
throw new Exception("Directory does not exist");

if(!ResourceName.Trim().EndsWith("\")
    ResourceName += "\";
}

// add append mask to it
if( 0 < ResourceName.Trim().Length )
{
    if( rbRead.Checked )
        ResourceName += "=0";
    else if( rbWrite.Checked )
        ResourceName += "=1";
    else
    {
        throw new Exception("Rights are missing");
    }

    if(bIsFile)
    ResourceName += "=" + fileSize.ToString();

    // and write this entry to share.ini
    WriteDataToFile(Application.StartupPath + "\Share.ini",ResourceName);
    if( !chkEntry.Checked )
    else
    {
        throw new Exception("Cannot add a blank entry");
    }
}

catch( Exception err )
{
    MessageBox.Show(err.Message,"Warning",MessageBoxButtons.OK,
MessageBoxIcon.Warning);
}

// Browse file button clicked, open file open dialog
protected void btnBrowseFile_Click (object sender, System.EventArgs e)
{
    FileOpenDialog.Title = "Select a file to share";
    if( System.Windows.Forms.DialogResult.OK ==
FileOpenDialog.ShowDialog() )
    txtFilename.Text = FileOpenDialog.FileName;
}
Lines 37: `frmSelection()` the constructor is implemented.

Lines 38-43: `Dispose()` is called at the time of termination of the application to dispose of all the resources used by the application.

Lines 47-261: `InitializeComponent()` is initializing all the GUI components. Because this part of the code is IDE generated, users are recommended not to alter the code.

Lines 263-269: `btnBrowseFolder_Click()` is a click-event handler for the Browse button placed on the application’s form. Whenever you click it, it makes an object of the BrowseFolder class and calls its `SelectFolder()` function to display the Folder Selection dialog box and puts the name of the folder selected in Folder Name textbox.

Lines 271-294: `DoesEntryExist()` checks whether entry (file/folder name) passed to it has already been written to `share.ini`, besides checking for the physical existence of that file/folder on hard disk.

275-276: This checks whether the file or folder name selected physically exists on machine or not.
279: A reading stream for `share.ini` is opened.
297-207: This file is searched for the given resource. If found, the caller function is notified by setting a flag.
281: This flag is returned to the caller at the end of this function.

Lines 297-334: `WriteDataToFile()` makes sure that the file name or folder name passed to it isn’t shared already and exists on the machine. If both these conditions are satisfied, the file/folder name is written to `share.ini`.

306-314: This function checks whether information passed to it is a file name or a folder name and then makes sure that it exists on the machine. If it is not already in `share.ini`, it is written to it.
316-333: If it is found to be a file and its entry does not exist in `share.ini`, it is written in `share.ini`, along with information pertaining to it (mask and size).

Lines 337-395: `btnOK_click()` function is called when you click the Share It button on the File/Folder Selection form. It checks what has been selected, whether it exists on the machine, and makes sure that the resource selected is not shared already and writes that to `share.ini`. These validations are carried out by using various functions implemented in the same class.

344-365: This checks whether the selected tab is a folder tab or a file tab through its title. Once decided, the name of the selected file/folder is stored in a variable.
367-378: This checks what credentials you have decided to be given to the selected file or folder, and those credentials are appended to a variable used for storing the file or folder name.
379-380: If the selected content is a file, its size is calculated and is appended to the same variable used for storing the file or folder name with its respective credentials.
383: The `WriteDataToFile()` function is called to append this resource to `share.ini`.

Lines 396-402: `btnBrowseDialog_Click()` is called when the Browse button placed on the form’s File Share tab is clicked to open the File Selection dialog box and puts the selected file in file name textbox.

**Client Form**

```csharp
namespace Client
```

The name of the class is `frmClient`, which performs the task of interacting with users (see Listing 5-6).
This window shown in Figure 5-5 is displayed on the startup of the browser application. The main browser window contains the computer name, its IP address, and the search results found in the list. It has five buttons; the function of each button is described in the following list:

- **Open button**: This button is used to open a selected computer. The contents of the selected computer are shown to you in the browser shared window when you click the Open button.

- **Refresh button**: This button updates the list view with the most recent entries from the server. When you click this button, the new user list is requested from the server and displayed to the user.

- **Search button**: This button allows you to search for a particular file or group of files on the computer. The search window is displayed to enter the search criteria.

- **ViewFiles button**: After the search is complete, the number of searches found are shown in front of their respective computer names. You can select the computer and click the View File button to see the searched files.

- **Quit button**: This button forces the application to terminate. The application closes when you click this button.

![Figure 5-5: Main browser window](image)

**Listing 5-6: frmClient**

// © 2001 Dreamtech Software India Inc.
// All rights reserved

```csharp
1 namespace Client // Copyright 2001 Dreamtech Software India Inc.
2 { // All rights reserved
3     using System; // Provides the basic functionality of .NET
4     using System.Collections; // Provides the different type of class collections
5     using System.ComponentModel; // Provides the facility of using components
6     using System.Drawing; // Provides the Drawing features, Used for cursors
7     using System.Net; // Provides the net related functionality
8     using System.Text; // Provides the text manipulation functions
9     using System.IO; // Provides I/O features
10    using WorkingWithXML; // Custom Class
11
12    /// <summary>
13 ```
This structure is used for book keeping purpose
The currently running listener list will be requested
by the server and server sends that list
Each record in this structure represents the data for the single running listener

/// This structure is used for book keeping purpose
/// The currently running listener list will be requested
/// by the server and server sends that list
/// Each record in this structure represents the data for the single running listener
/// </summary>

public struct __CONNECTEDCOMPUTERS
{
/// <summary>
/// Declares the string used to store the name of the listener
/// This name is in human readable format
/// </summary>
public string sComputername;

/// <summary>
/// sIPAddress variable is used to store the IP address of the listener
/// </summary>
public string sIPAddress;

/// <summary>
/// This will be 0 in starting and consequently filled by every search operation
/// </summary>
public int iFilesFound;
}

/// <summary>
/// Summary description for frmClient class.
/// </summary>
public class frmClient : System.Windows.Forms.Form
{
private System.ComponentModel.IContainer components;
private System.Windows.Forms.Button btnViewFiles;
private System.Windows.Forms.ToolTip ToolTipText;

/// <summary>
/// The below declared variables are the user defined variables
/// used within this class
/// </summary>
private int iConnectedComputers;

/// <summary>
/// Stores the total number of listeners found
/// </summary>
private int iSearchResult;

/// <summary>
/// Stores the number of search results found for the matching criteria
/// </summary>
private int iSearchResult;
/// <summary>
/// Declares a xmlParser variable of type XMLParser (User defined class)
/// </summary>
private XMLParser xmlParser;

/// <summary>
/// Declares a xmlServerComm variable of type ServerCommunication
/// </summary>
private ServerCommunication xmlServerComm;

/// <summary>
/// Declares an object xmlStruct of type XMLSTRUCT (User defined class)
/// </summary>
private XMLSTRUCT xmlStruct;

/// <summary>
/// Declares an array of ConnectedComputers of type __CONNECTEDCOMPUTERS structures
/// </summary>
private __CONNECTEDCOMPUTERS[] ConnectedComputers;

private System.Windows.Forms.ColumnHeader clhComputername;
private System.Windows.Forms.ColumnHeader clhIPAddress;
private System.Windows.Forms.ColumnHeader clhObjects;
private System.Windows.Forms.Label lblCopyright;

/// <summary>
/// declares sSubItems variable as an array of string
/// </summary>
private System.Windows.Forms.ListViewItem lvItems;

/// <summary>
/// This is the default constructor of the class
/// </summary>
public frmClient()
{
    // Required for Windows Form Designer support
    // Auto generated function by the IDE
    InitializeComponent();

    this.Icon = new System.Drawing.Icon(Application.StartupPath + "\Computer.ico");

    // Creates a new instance of XMLParser class
    xmlParser = new WorkingWithXML.XMLParser();

    // Creates a new instance of XMLSTRUCT structure
    xmlStruct = new WorkingWithXML.XMLSTRUCT();

    // Creates a new instance for ServerCommunication class
    xmlServerComm = new WorkingWithXML.ServerCommunication();
125 // Assigns the global value for USERLISTXML
126 xmlParser.USERLISTXML = Application.StartupPath + "\userlist.xml";
127
128 // Assigns the global value for SERVERSEARCHXML
129 xmlParser.SERVERSEARCHRESULTXML = Application.StartupPath + "\search.xml";
130 try
131 {
132 // Fills the List view with the values
133 // these values are the response from the server
134 if( 0 == PopulateList() )
135     // Displays a message if no computer is connected to the network
136     throw new Exception("No computer is connected to the network. The list will be empty");
137 }
138
139 // Handles every exception that is thrown
140 catch( Exception err ) {
141     MessageBox.Show(err.Message,"Warning",MessageBoxButtons.OK,
142         MessageBoxIcon.Warning); }
143 }
144
145 /// <summary>
146 /// Clean up any resources being used.
147 /// This is auto generated by the IDE
148 /// </summary>
149 public override void Dispose()
150 {
151     base.Dispose();
152     components.Dispose();
153 }
154
155 /// <summary>
156 /// Required method for Designer support - do not modify
157 /// the contents of this method with the code editor.
158 /// this code is generated automatically by the IDE
159 /// </summary>
160 private void InitializeComponent()
161 {
162     this.components = new System.ComponentModel.Container();
163     this.ToolTipText = new System.Windows.Forms.ToolTip(this.components);
164     this.btnSearch = new System.Windows.Forms.Button();
165     this.btnQuit = new System.Windows.Forms.Button();
166     this.btnViewFiles = new System.Windows.Forms.Button();
167     this.btnRefresh = new System.Windows.Forms.Button();
168     this.btnOpen = new System.Windows.Forms.Button();
169     this.lvComputers = new System.Windows.Forms.ListView();
170     this.clhComputername = new System.Windows.Forms.ColumnHeader();
171     this.clhIPAddress = new System.Windows.Forms.ColumnHeader();
172     this.clhObjects = new System.Windows.Forms.ColumnHeader();
173     this.lblCopyright = new System.Windows.Forms.Label();
174     // btnSearch
175     //
177  this.btnSearch.DialogResult =
178  this.btnSearch.FlatStyle = System.Windows.Forms.FlatStyle.Popup;
180  this.btnSearch.Location = new System.Drawing.Point(137, 1);
181  this.btnSearch.Name = "btnSearch";
182  this.btnSearch.Size = new System.Drawing.Size(67, 35);
183  this.btnSearch.TabIndex = 2;
184  this.btnSearch.Text = "&Search";
185  this.ToolTipText.SetToolTip(this.btnSearch, "Search on computers for
filenames");
186  this.btnSearch.Click += new
System.EventHandler(this.btnSearch_Click);
187  //
188  // btnQuit
189  //
192  this.btnQuit.FlatStyle = System.Windows.Forms.FlatStyle.Popup;
194  this.btnQuit.Location = new System.Drawing.Point(270, 1);
195  this.btnQuit.Name = "btnQuit";
196  this.btnQuit.Size = new System.Drawing.Size(67, 35);
197  this.btnQuit.TabIndex = 3;
198  this.btnQuit.Text = "&Quit";
199  this.ToolTipText.SetToolTip(this.btnQuit, "Quit this application");
200  this.btnQuit.Click += new System.EventHandler(this.btnQuit_Click);
201  //
202  // btnViewFiles
203  //
204  this.btnViewFiles.BackColor = System.Drawing.Color.Chocolate;
205  this.btnViewFiles.DialogResult =
206  this.btnViewFiles.Enabled = false;
207  this.btnViewFiles.FlatStyle = System.Windows.Forms.FlatStyle.Popup;
208  this.btnViewFiles.ForeColor = System.Drawing.Color.White;
209  this.btnViewFiles.Location = new System.Drawing.Point(203, 1);
210  this.btnViewFiles.Name = "btnViewFiles";
211  this.btnViewFiles.Size = new System.Drawing.Size(67, 35);
212  this.btnViewFiles.TabIndex = 5;
213  this.btnViewFiles.Text = "&View Files";
214  this.ToolTipText.SetToolTip(this.btnViewFiles, "View the searched
files");
215  this.btnViewFiles.Click += new
System.EventHandler(this.btnViewFiles_Click);
216  //
217  // btnRefresh
218  //
220  this.btnRefresh.FlatStyle = System.Windows.Forms.FlatStyle.Popup;
221  this.btnRefresh.ForeColor = System.Drawing.Color.White;
222  this.btnRefresh.Location = new System.Drawing.Point(70, 1);
223  this.btnRefresh.Name = "btnRefresh";
224  this.btnRefresh.Size = new System.Drawing.Size(67, 35);
225  this.btnRefresh.TabIndex = 1;
232  Chapter 5: The P2P Application (C#)

```csharp
226       this.btnRefresh.Text = "&Refresh";
227       this.ToolTipText.SetToolTip(this.btnRefresh, "Refresh the computer list");
228       this.btnRefresh.Click += new System.EventHandler(this.btnRefresh_Click);
229 //
230 // btnOpen
231 //
233       this.btnOpen.FlatStyle = System.Windows.Forms.FlatStyle.Popup;
235       this.btnOpen.Location = new System.Drawing.Point(3, 1);
236       this.btnOpen.Name = "btnOpen";
237       this.btnOpen.Size = new System.Drawing.Size(67, 35);
238       this.btnOpen.TabIndex = 0;
239       this.btnOpen.Text = "&Open";
240       this.ToolTipText.SetToolTip(this.btnOpen, "Connect to the selected computer");
241       this.btnOpen.Click += new System.EventHandler(this.btnOpen_Click);
242 //
243 // lvComputers
244 //
246           this.clhComputername,
247           this.clhIPAddress,
248           this.clhObjects});
249       this.lvComputers.Location = new System.Drawing.Point(3, 41);
250       this.lvComputers.Name = "lvComputers";
251       this.lvComputers.Size = new System.Drawing.Size(530, 146);
252       this.lvComputers.TabIndex = 7;
253       this.ToolTipText.SetToolTip(this.lvComputers, "Select a computer to connect");
255       this.lvComputers.KeyPress += new System.Windows.Forms.KeyPressEventHandler(this.lvComputers_KeyPress);
256       this.lvComputers.DoubleClick += new System.EventHandler(this.lvComputers_DoubleClick);
257       this.lvComputers.SelectedIndexChanged += new System.EventHandler(this.lvComputers_SelectedIndexChanged);
258 //
259 // clhComputername
260 //
261       this.clhComputername.Text = "Computername";
262       this.clhComputername.Width = 222;
263 //
264 // clhIPAddress
265 //
266       this.clhIPAddress.Text = "IP Address";
267       this.clhIPAddress.Width = 142;
268 //
269 // clhObjects
270 //
```
this.clhObjects.Text = "No. of Objects found";
this.clhObjects.Width = 112;
//
// lblCopyright
//
this.lblCopyright.Location = new System.Drawing.Point(348, 10);
this.lblCopyright.Name = "lblCopyright";
this.lblCopyright.Size = new System.Drawing.Size(192, 14);
this.lblCopyright.TabIndex = 6;
this.lblCopyright.Text = "© 2001 www.dreamtechsoftware.com";
//
// frmClient
//
this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
this.CancelButton = this.btnQuit;
this.ClientSize = new System.Drawing.Size(536, 190);
this.Controls.AddRange(new System.Windows.Forms.Control[] {
this.lvComputers,
this.lblCopyright,
this.btnCloseFiles,
this.btnCloseSearch,
this.btnCloseQuit,
this.btnCloseRefresh,
this.btnCloseOpen});
this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedDialog;
this.MaximizeBox = false;
this.Name = "frmClient";
this.Text = "Peer to Peer Browser";
this.ResumeLayout(false);

/// <summary>
/// Fills the List view with initial values
/// This function is called from the constructor of this class
/// </summary>
private int PopulateList()
{
try
{
// This line of code gets the response from the server
// by calling an asp page and writes it in a file
// represented by xmlParser.USERLISTXML
xmlServerComm.WriteDataToFile(xmlParser.USERLISTXML,xmlServerComm.GetDataFromServer("http://webaddress/userlist.asp"));
Chapter 5: The P2P Application (C#)

```csharp
316 // This line will parse the returned XML by the server
317 // and saves it in the xmlStruct variable
318 // iConnectedComputers will have value for total
319 // number of connected listeners
320 iConnectedComputers =
321 xmlParser.ParseXML(xmlParser.USERLISTXML, out xmlStruct,
322 xmlServerComm.TypeOfXMLRecieved(xmlParser.USERLISTXML));
323
324 // Clears the every item of the list view
325 lvComputers.Items.Clear();
326
327 // Deletes the File represented by xmlParser.USERLISTXML
328 xmlServerComm.FileDelete(xmlParser.USERLISTXML);
329
330 // initialize the ConnectedComputers array to
331 // the number of listeners found connected
332 ConnectedComputers = new
333 __CONNECTEDCOMPUTERS[iConnectedComputers];
334
335 // Initializes lvItems object
336 lvItems = new System.Windows.Forms.ListViewItem();
337
338 /// The below 17 lines of code is used for sorting the
339 /// USERLIST alphabetically
340 for( int i = 0; i < iConnectedComputers; i++ )
341 {
342     for( int j = i+1; j < iConnectedComputers; j++ )
343     {
344         if(
345             xmlStruct.USERLIST[i].sUsername.GetHashCode() <
346             xmlStruct.USERLIST[j].sUsername.GetHashCode() )
347             {
348                 string sTemp;
349                 sTemp = xmlStruct.USERLIST[j].sUsername;
350                 xmlStruct.USERLIST[j].sUsername =
351                 xmlStruct.USERLIST[i].sUsername;
352                 xmlStruct.USERLIST[i].sUsername = sTemp;
353             }
354         }
355     }
356     // Fills the ConnectedComputer array with user list values
357     for( int i = 0; i < iConnectedComputers; i++ )
358     {
359         ImageList imgList = new ImageList();
```
Chapter 5: The P2P Application (C#) 235

```csharp
364 lvComputers.SmallImageList = imgList;
365 ConnectedComputers[i].sComputername = xmlStruct.USERLIST[i].sUsername.Trim();
366 ConnectedComputers[i].sIPAddress = xmlStruct.USERLIST[i].sIPAddress.Trim();
367 ConnectedComputers[i].iFilesFound = 0;
368 // Insert the records one by one in the list view
369 lvItems = lvComputers.Items.Insert(i, ConnectedComputers[i].sComputername);
370 lvItems.SubItems.Add(ConnectedComputers[i].sIPAddress);
371 lvItems.SubItems.Add("");
372 lvItems.ImageIndex = 0;
373 }
374 }
375 // Catches any exception that is thrown by the application
376 // and display message
377 catch (Exception err) {
378 MessageBox.Show(err.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
379 }
380 // Return the number of connected computers
381 return iConnectedComputers;
382 }
383 /// <summary>
384 /// Fills the Listview with the entries as well as with
385 /// the search criteria if performed any
386 /// </summary>
387 private void PopulateWithSearchResults()
388 {
389     // Decrales and initializes the iCounter value to zero
390     int iCounter = 0;
391     try
392         {
393             // Parse the XML Request returned by the server and store
394             // the number of total searches found in iSearchResult
395             iSearchResult = xmlParser.ParseXML(xmlParser.SERVERSEARCHRESULTXML, out xmlStruct,
396             xmlServerComm.TypeOfXMLRecieved(xmlParser.SERVERSEARCHRESULTXML));
397             // Delete the SEARVERSEARCHXML file after parsing
398             xmlServerComm.FileDelete(xmlParser.SERVERSEARCHRESULTXML);
399             // The 8 lines of code will find in the search results
400             // and counts the number of matched files found on each
401             // computer
402             // as the result of the search i.e for example
403             // on computer A 10 files are found , on B 3 files are found
404             // etc
```
// This code will do the individual level breakup for the search results
for( int i = 0; i < ConnectedComputers.Length; i++)
{
    for( int j = 0; j < iSearchResult; j++ )
        if( 0 == xmlStruct.SERVERSEARCH[j].sIPAddress.Trim().CompareTo(ConnectedComputers[i].sIPAddress.Trim()) )
            iCounter++;
    ConnectedComputers[i].iFilesFound = iCounter;
    iCounter = 0;
}

/////////////////////////////////////////////////////////////////////
///
/// initialize lvItems
lvItems = new System.Windows.Forms.ListViewItem();
// Clears the list view items
lvComputers.Items.Clear();
for( int i = 0; i < ConnectedComputers.Length; i++)
{
    ImageList imgList = new ImageList();
    lvComputers.SmallImageList = imgList;
    // Insert the items in the list view
    lvItems = lvComputers.Items.Insert(i,ConnectedComputers[i].sComputername);
    // Assigns the IP Address at first subscript
    lvItems.SubItems.Add(ConnectedComputers[i].sIPAddress.Trim());
    // Assigns the no of search results found
    // at second subscript
    lvItems.SubItems.Add(ConnectedComputers[i].iFilesFound.ToString().Trim());
    lvItems.ImageIndex = 0;
}

// catches any exception that is thrown by the application
catch( Exception err ) {
    MessageBox.Show(err.Message,"Error",MessageBoxButtons.OK, MessageBoxIcon.Error);
}

/// <summary>
/// Handles the key press events on the list view
/// </summary>
    if( 13 == e.KeyChar || 32 == e.KeyChar )
        btnOpen_Click(null,System.EventArgs.Empty);
}

/// <summary>
/// Invoked when the user double clicks on the list view
/// </summary>
/// <param name="sender"> </param>
/// <param name="e"> </param>
protected void lvComputers_DoubleClick (object sender, System.EventArgs e) {
    btnOpen_Click(null,System.EventArgs.Empty);
}

/// <summary>
/// Invoked, when the user clicks the refresh button
/// This will refresh the contents of the listview
/// by getting the USERLIST from the server again and
/// shows it in the list view
/// </summary>
/// <param name="sender"> </param>
/// <param name="e"> </param>
protected void btnRefresh_Click (object sender, System.EventArgs e) {
    try
        { 
            if( 0 == PopulateList() )
                throw new Exception("No computer is connected to the network. The list will be empty");
        }
    catch( Exception err ) {
        MessageBox.Show(err.Message,"Warning",MessageBoxButtons.OK,MessageBoxIcon.Warning);
    }
}

/// <summary>
/// Invoked when the user clicks on the Open button
/// </summary>
/// <param name="sender"> </param>
/// <param name="e"> </param>
protected void btnOpen_Click (object sender, System.EventArgs e) {
    try
        { 
            // Get the currently selected item from the list view
            if( 0 < items.Count )
            { 
            
        }
// Makes the string that is required to pass to the constructor of the frmShare class
// Appends the "(" and ")" at the begin and end of the IPAddress
string SelectedIP = "(" + items[0].SubItems[1].Text + ")" ;

// gets the name of the computer from Computername column
string Computername = items[0].Text;

// Concatenates the Computername to IP address
Computername = Computername + SelectedIP;

// declares a variable ShareForm of type frmShare class
// and passes the Computername to it
frmShare ShareForm = new frmShare(Computername);

// Shows the frmShare dialog window
ShareForm.Show();

// Throws the exception
else throw new Exception("No selected computer found");
}

// cathces the exception and shows it in a message box
catch( Exception err ) {
    MessageBox.Show(err.Message,"Warning",MessageBoxButtons.OK,
    MessageBoxIcon.Warning);} }

/// <summary>
/// Performs a quit operation when the user clicks the quit button
/// </summary>
/// <param name="sender"> </param>
/// <param name="e"> </param>
protected void btnQuit_Click (object sender, System.EventArgs e) {
    Application.Exit();
}

/// <summary>
/// This function is called when you click the View files button
/// </summary>
/// <param name="sender"> </param>
protected void btnViewFiles_Click (object sender, System.EventArgs e) {
    // Creates an ShareForm object from frmShare class and initializes it
    frmShare ShareForm = new frmShare();
// Get the currently selected item from the list view

// Get the IPAddress of the Selected item from IP Address
string SelectedIP = items[0].SubItems[1].Text.Trim();

// The followin 12 lines of code Scans through the
// SERVERSEARCH list and insert the values in the List view
// of the frmShare class Folder and file wise
for( int i = 0; i < iSearchResult; i++ )
{
    if( 0 == xmlStruct.SERVERSEARCH[i].sIPAddress.Trim().CompareTo(SelectedIP) )
    {
        ImageList imgList = new ImageList();
        ShareForm.lvFiles.SmallImageList = imgList;
        if( xmlStruct.SERVERSEARCH[i].sFilename.EndsWith("\") )
        {
            lvItems = ShareForm.lvFiles.Items.Insert(i,xmlStruct.SERVERSEARCH[i].sFilename);
            lvItems.ImageIndex = 0;
        }
        else
        {
            lvItems = ShareForm.lvFiles.Items.Insert(i,xmlStruct.SERVERSEARCH[i].sFilename);
            lvItems.ImageIndex = 1;
        }
    }
}

// Shows the ShareForm window
ShareForm.ShowDialog();

/// <summary>
/// Whenever the user changes the selection in the List view
/// This function is called
/// </summary>
protected void lvComputers_SelectedIndexChanged(object sender, System.EventArgs e)
{
int iTemp;

GetNumberOfObjectsFound(out iTemp);

if( iTemp > 0 )
    btnViewFiles.Enabled = true;
else
    btnViewFiles.Enabled = false;

/// <summary>
/// This function returns the total object found in search
/// for a particular selected entry so the the state of
/// View Files button can be toggled
/// </summary>
/// <param name="iReturn"> </param>
private int GetNumberOfObjectsFound( out int iReturn )
{
    // get the currently selected item from the list view

    int iIndex = -1;

    // initialize the iReturn valiable to -1
    iReturn = -1;

    // get the index of the selected item
    if( 0 < items.Count )
    {
        iIndex = items[0].Index;

        // get the corresponding iFilesFound value from the
        // List that we have maintained by supplying the
        // iIndex value to it
        iReturn = ConnectedComputers[iIndex].iFilesFound;
    }

    // Also returns the iIndex number which is selected,
    // iIndex contains -1 if nothing is selected
    return iIndex;
}

/// <summary>
/// Invoked when the search button is clicked
/// Shows the search form to enter the search criteria
/// <param name="sender"> </param>
/// <param name="e"> </param>
protected void btnSearch_Click (object sender, System.EventArgs e)
{
    try
    {
        // Declare and initialize a new variable SearchForm
        // of type frmSearch class
        frmSearch SearchForm = new frmSearch();

        // enables the text box control of the search Form
        SearchForm.txtSearchOn.Enabled = true;

        // enables the label control of the search Form
        SearchForm.lblSearchOn.Enabled = true;

        // Sets the bFlag of search form to true
        SearchForm.bFlag = true;

        // if Search is pressed from within the search form
        if( System.Windows.Forms.DialogResult.OK ==
SearchForm.ShowDialog() )
        {
            // get the search response from the server by calling
            // an asp file and getting the data returned by that
            // file
            // which is in XML format. Later save that data in the
            // SERVERSEARCHRESULTXML for parsing
            xmlServerComm.WriteDataToFile(xmlParser.SERVERSEARCHRESULTXML,xmlServerComm.Get
+ SearchForm.SearchFor));

            // calls the PopulateWithSearchResults function
            PopulateWithSearchResults();
        }

        // Catches any exception thrown by the application and shows it in
        // message box
        catch( Exception err ) { MessageBox.Show(err.Message,
"Error",MessageBoxButtons.OK, MessageBoxIcon.Error); }
    }
}

/// The main entry point for the application.
/// This is auto generated by the IDE
public static void Main(string[] args) { Application.Run(new
frmClient()); }
}
**Code description of 'frmClient' class (frmClient.cs)**

- Lines 3-11: This includes the basic packages needed for the various classes used to build this application. This include packages used for communicating with network, streaming facility, file I/O, text manipulations, and so on.
  - 10: This includes a user-defined class called WorkingWithXML; this class is used to communicate with the server, parse XML data, and create XML requests.

- Lines 20-39: A structure is declared here with public access. This structure is used to store the information about each running listener. This information is received from the server by calling an ASP page. Three variables are declared in this structure. The variable sComputername is used to store the human-readable name for the connected listener. The second variable, sIPAddress, is used to store the IP address of the computer represented by sComputername. The last variable, iFilesFound, is declared here to store the information about the number of searches found on a particular computer.

- Lines 44 -52: These lines of code are generated automatically by the C# environment.
  - 45: This declares a public class frmClient, the default class created for this file. This class is inhereted from System.Windows.Forms.
  - 52: Some designer variables are declared. These variables represent the various controls created at the design time of this form.

- Lines 62-91: These lines declare some important user-defined variables. These variables are used at various places in this class.
  - 62: The iConnectedComputers variable stores the total number of currently running listeners that are successfully connected with the server.
  - 68: The iSearchResult variable is used to store the number of results found for a specific search criteria.
  - 73: This line declares an object xmlParser for the XMLParser class present inside the WorkingWithXML assembly created for this application.
  - 79: The xmlServerComm object of the ServerCommunication class is declared here to access the functions present in ServerCommunication class. This class is user defined and has its source in the WorkingWithXML assembly.
  - 84: The xmlStruct variable of the XMLSTRUCT structure is declared here to have the parsed data to store in their respective places.
  - 90: This Makes an array of the _CONNECTEDCOMPUTERS structure represented by the ConnectedComputers variable.

- Lines 100: This declares a ListViewItem variable lvItems.

- Lines 105-141: These lines of code implement the default constructor for this class. Some extra coding has been done in this constructor for the purpose of retrieving the list of the listeners from the server. This is furnished by calling the PopulateList() function of this class.
  - 117 -123: New instances of user-defined classes (XMLParser, XMLSTRUCT, and ServerCommunication) are created.
  - 126-129: Here we are assigning the filename to the USERLISTXML variable of XMLParser class. The userlist XML returned by the server is stored by this filename on the local machine. The next variable SERVERSEARCHRESULTXML does the same thing. It stores the name of the file in which the search result XML response is to be stored.
  - 134 -141: With the help of the PopulateList() function in this class, the list view is filled with values, which are retrieved as the response from the server. In case no computer is connected to the network, an error message is displayed, and the list is empty.
Lines 147-151: The main purpose of the function is to clear and release all resources held up by the application. This coding is autogenerated by IDE and is called on the closing event of this form.

Lines 158-302: This method is used to initialize and assign the user-interface variables. This is generated automatically by the C# environment. It is advisable not to manipulate these lines of code unless you know where the impact is going to be shown.

Lines 308-384: This PopulateList() function is used to populate the list view with the entries retrieved from the userlist XML.

- 315: The response from the server is received by calling an ASP page on the server. This response is then saved in the file represented by the USERLIST variable of XMLParser class.
- 321: The xmlStruct variable stores the parsed data from the XML returned by the server, in response to calling the ASP page. After parsing, the iConnectedComputer variable stores the total number of listener entries found on the server.
- 324-354: Each time this function is called, the contents of list view are first cleared and then updated with the recent list. The parsed XML file is then deleted from the system, as we have data saved in the XMLSTRUCT structure. The next step is to sort the list by Computername.
- 360-376: The array represented by the iConnectedComputers variable is being filled here by the values present in the USERLIST structure of XMLSTRUCT.
- 380: In case the application encounters an error, the message is displayed.
- 383: Lastly, this function returns the number of entries populated in the list view.

Lines 390-449: This function does the same thing described for PopulateList() and also populates the number of objects found column of the list view with the search results.

Lines 456-461: This function handles the key-press events of the Enter and SpaceBar keys when pressed while some item is selected on the list view.

Lines 467: Whenever you double click the list view, this function comes into action. It calls the click function associated with the Open button, as double click indicates that you want to open or connect to the selected computer from the list view.

Lines 477-485: When you click the Refresh button placed on the application’s form, this function is invoked. This function refreshes the contents of the list view by retrieving the fresh USERLIST from server again and shows it in the list view. The error message is displayed if no listener is connected to the network. As in constructor, the PopulateList() function is called.

Lines 492-529: This function comes into action when you click the Open button. Here the currently selected computer from the computer list view is extracted separately with its IP address and a string is made out of that (for example, COMPUTERNAME (IP ADDRESS)). This string is then passed to the contractor of the frmShare class by making its object ShareForm; after that, the form is displayed on the screen (line 481).

Lines 536: This performs the quit operation when you click the Quit button placed on the application's form.

Lines 544-586: This function is invoked when you click the View button. This function is used to view the searched files.

- 548: An object of the frmShare class is initialized as ShareForm.
- 560-580: These lines scan through the SERVERSEARCH list and insert the values in the list view of the frmShare class. Values are inserted in Folder/File as a hierarchy. Finally the results are displayed.

Lines 594-613: Whenever you change the selection in the list view, this function is activated. This function is responsible for enabling or disabling the ViewFiles button. If no searches are found on the computer, the ViewFiles button becomes disabled when you selects that computer from the list view.
599: The local variable \textit{iTemp} of integer type is declared to store the number of individual search results returned by the \texttt{GetNumberOfObjectsFound()} function.

605: The \texttt{GetNumberOfObjectsFound()} function is called, and its values are stored in the local variable \textit{iTemp}.

609: If \textit{iTemp} is 0, the ViewFiles button is disabled, if the \textit{iTemp} variable is not zero than keep the ViewFiles button enabled.

Lines 621-647: This function returns the total objects found in the search for a selected entry so that the state of view file can be toggled.

624: This retrieves the currently selected item from the list view.

627-630: Variables \textit{iIndex} and \textit{iReturn} are initialized to -1.

633-646: After verifying that items variables have something to read, the index of selected item is extracted and placed into \textit{iIndex} variable. Then the corresponding value for the \textit{iFilesFound} variable is extracted from the ConnectedComputer array. Finally, the index and the value of the \textit{iFilesFound} variable are returned.

Lines 655-689: This function is invoked when the Search button is clicked. A form is displayed at this point to enter the search criteria. Search criteria are then returned to the caller of the form and passed to an ASP page on the server. The server handles the request and fetches the matching records, making an XML and returning the response to the request. This XML is then parsed in the \texttt{PopulateWithSearchResults()} function and used to display the values in the list view.

Lines 695: This is the main entry point of this application. This function has the same task as the \texttt{main()} function in C or C++. Execution starts here.

\textbf{frmShare Class}

The window shown in Figure 5-6 allows you to see the shared contents of a selected computer. It shows the files/folders, their size, and the type of files/folders on the remote computer.

\begin{itemize}
\item **Upload button**: This button allows you to select any file from your local machine and to transfer that file on the connected machine. For example, if you want to send a file to the listener, you can use this button.
\item **Download Button**: To download any file from the remote machine, select a file, and click the Download button to save that file on the local machine over the network.
\item **Search Button**: This button allows you to search for a specific file in the current folder being viewed.
\item **Close Button**: This button does not close the application; instead, it closes this window and brings you back to the main application window.
\end{itemize}
Figure 5-6: Shared Content window

Listing 5-7 describes the code for frmShare.

**Listing 5-7: frmShare**

//© 2001 Dreamtech Software India Inc.  
// All rights reserved

```csharp
1 namespace Client // Copyright 2001 Dreamtech Software India Inc. 
2 { // All rights reserved 
3     using System; // Provides the basic functionality of .NET 
4     using System.Drawing; // Provides the Drawing features, Used for 
5         cursors 
6         using System.Windows.Forms;// Provides the drawing of buttons, listviews 
7         etc 
8         using System.Net; // Provides the net related functionality 
9         using System.Net.Sockets; // Provides the functionality of sockets 
10         using System.Text; // Provides the text manipulation functions 
11         using System.IO; // Provides I/O features 
12         using WorkingWithXML; // Custom class 
13         using System.Collections; // Provides the different type of class 
14         collections 
15         using System.ComponentModel;// Provides the facility of using components 
16         { 
17         // <summary> 
18         /// Summary description for frmShare. 
19         /// </summary> 
20         public class frmShare : System.Windows.Forms.Form 
21         { 
22         private System.ComponentModel.IContainer components; 
26         private System.Windows.Forms.ToolTip toolTipText; 
```
public System.Windows.Forms.ListView lvFiles;
private System.Windows.Forms.ColumnHeader clhFilename;
private System.Windows.Forms.ColumnHeader clhType;

/// <summary>
/// User defined variables.
/// </summary>

/// <summary>
/// stores the number of bytes written or read from any stream
/// </summary>
private int iBytes;

/// <summary>
/// These variables are used to store the name of the computer
/// to which you have connected and the parent folder name.
/// Parent folder name is the name of the folder of the contents
/// which you are viewing in the window
/// </summary>
private string COMPUTERNAME,PARENTFOLDER;

/// <summary>
/// Stores a new created socket of type TCPClient(System defined class)
/// used to communicate with the listener
/// </summary>
private TcpClient ClientSocket;

/// <summary>
/// creates a variable for XMLCreater(User defined class) to
/// create XML requests for the listener
/// </summary>
private XMLCreater xmlCreate;

/// <summary>
/// StreamTCP points to the NetworkStream(System defined class)
/// which is used for transfer data over the Socket connection
/// </summary>
private NetworkStream StreamTCP;

/// <summary>
/// fileStream is an object of type FileStream(System defined class)
/// used to have I/O capabilities for files which are used
/// in this program
/// </summary>
private FileStream fileStream;

/// <summary>
/// ReadBuffer and WriteBuffer Byte arrays used for
/// Reading and Writing any file.
/// </summary>
private byte[] ReadBuffer,WriteBuffer;
Chapter 5: The P2P Application (C#) 247

```csharp
/// <summary>
/// xmlParser is of type XMLParser(User defined class)
/// It is used to have the access for Parsing the XML file
/// This class is present in WorkingWithXML
/// </summary>
private XMLParser xmlParser;

/// <summary>
/// xmlStruct is of type XMLSTRUCT(User defined structure)
/// It is used to store the different records obtained
/// from parsing the XML. This structure is present in WorkingWithXML
/// </summary>
private XMLSTRUCT xmlStruct;

/// <summary>
/// strArray if of Type SHOWFILES of XMLSTRUCT Structure and
/// used to save the corresponding Files/Folder which are seen in the
/// the List view at run time
/// </summary>
private XMLSTRUCT.__SHOWFILES[] strArray;

/// <summary>
/// The below variables are readonly variables that u cannot
/// assign a new value to these variables again. They are
/// constant type variables
/// </summary>

/// <summary>
/// REQUESTFILE has the name of the file which has to be created
/// for a particular request
/// </summary>
public readonly string REQUESTFILE = Application.StartupPath + "\Request.xml";

/// <summary>
/// RESPONSEFILE has the name of the file which has to be created
/// for a particular request
/// </summary>
public readonly string RESPONSEFILE = Application.StartupPath + "\Response.xml";

/// <summary>
/// MAX_SIZE defines the maximum size of the read or write buffer
/// which is used for reading or writing a file
/// </summary>
public readonly int MAX_SIZE = 512;

/// <summary>
/// This is the default constructor of the this class
/// This is called from the View files button
/// </summary>
public frmShare()
{
    //
```
136     // Required for Windows Form Designer support
137     ///
138     /// Auto generated code line by the IDE
139     InitializeComponent();
140
141     /// Puts the Computer.ico as the form icon
142     this.Icon = new System.Drawing.Icon(Application.StartupPath + "\Computer.ico");
143
144     /// Changes the Caption of this dialog box
145     this.Text = "Search Result";
146     sBar.Text = "Root";
147     COMPUTERNAME = null;
148     PARENTFOLDER = null;
149     }
150
151     /// <summary>
152     /// This is a user defined constructor called from the
153     /// Open button in the frmClient
154     /// Computername is passed to this function, which you
155     /// have selected from the Main window
156     /// </summary>
157     /// <param name="Computername"> </param>
158     public frmShare(string Computername)
159     {
160         
161         // Required for Windows Form Designer support
162         // This line is not auto generated, instead it has been copied from
163         // default constructor
164         InitializeComponent();
165
166         /// Puts the Computer.ico as the form icon
167         this.Icon = new System.Drawing.Icon(Application.StartupPath + "\Computer.ico");
168
169         /// COMPUTERNAME is a global variable used to store the name
170         /// and IP Address of the computer to which you are currently
171         /// connected
172         COMPUTERNAME = Computername;
173
174         /// Open connection is a user defined function responsible for
175         /// opening a socket connection for listener. This function
176         /// returns a bool value
177         if( OpenConnection(COMPUTERNAME) )
178             {
179                 
180                 // This will creates a SHOWFILE request XML for seding it
181                 // to the listener
182                 CreateRequest("SHOWFILES","","");
183
184                 // This will actually send the REQUESTFILE to the listener
185                 SendDataToListener(REQUESTFILE);
186
187                 // This will get the response of the above request from
188                 // listener and stores that response in a RESPONSEFILE
189                 GetDataFromListener(RESPONSEFILE);
190
191                 // Change the Caption of this dialog box
192                 this.Text = "Search Result";
193                 sBar.Text = "Root";
194                 COMPUTERNAME = null;
195                 PARENTFOLDER = null;
196             }
Chapter 5: The P2P Application (C#) 249

190 // This will Parse that response XML File and results are shown to the user
191 Parsing(RESPONSEFILE);
192
193 // Closes any Opened socket or stream connection
194 CloseConnection();
195
196 // Since this constructor is called at the root level therefore no parent folder is associated with it
197 PARENTFOLDER = null;
198
199 // Changes the caption of this dialog box
200 this.Text = "Shared contents on: " + Computername.ToUpper();
201
202 // Sets the text for the Status bar
203 sBar.Text = "Root";
204 }
205 }
206
207 /// <summary>
208 /// This is also a user defined constructor called from the Search button to view the search results
209 /// This constructor is called from within this code only it is used to pass any type of request if needed at the time of its construction
210 /// </summary>
211 /// <param name="Computername stores the name of the computer to connect"> </param>
212 /// <param name="Request stores what type of request you want to send to the listener"> </param>
213 /// <param name="Scope scope value needed, if any"> </param>
214 /// <param name="Mask mask value needed if any"> </param>
215 public frmShare(string Computername, string Request, string Scope, string Mask)
216 {
217             // Required for Windows Form Designer support
218 // This line is not auto generated, instead it has been copied from default constructor
219             InitializeComponent();
220     
221     // Puts the Computer.ico as the form icon
222     this.Icon = new System.Drawing.Icon(Application.StartupPath + "\Computer.ico");
223 
224     // COMPUTERNAME is a global variable used to store the name
225     // and IP Address of the computer to which you are currently connected
226     COMPUTERNAME = Computername;
227 
228     // Open connection is a user defined function responsible for opening a socket connection for listener. This function returns a bool value
229     if( OpenConnection(COMPUTERNAME) )
230     {
231         // Creates an XML request with scope and mask
232         CreateRequest(Request,Scope,Mask);
// Sends this request to listener
SendDataToListener(REQUESTFILE);

// get the response from listener
GetDataFromListener(RESPONSEFILE);

// Parse that response for records and show those records
Parsing(RESPONSEFILE);

// Closes the active connection
CloseConnection();

// Assign scope to sText local variable
string sText = Scope;

// Extract the name of the PARENTFOLDER
sText = sText.Substring(0, sText.LastIndexOf("\") + 1);

// changes the caption of this dialog box
this.Text = sText + " on " + Computername + "";

// Assign the value for PARENTFOLDER
PARENTFOLDER = sText;

// Changes the text of the status bar
sBar.Text = sText;

/// <summary>
///    Clean up any resources being used.
///    auto generated function by IDE
/// </summary>
public override void Dispose()
{
    base.Dispose();
    components.Dispose();
}

/// <summary>
///    Required method for Designer support - do not modify
///    the contents of this method with the code editor.
///    These are also auto generated lines of code
/// </summary>
private void InitializeComponent()
{
    this.components = new System.ComponentModel.Container();
    this.clhType = new System.Windows.Forms.ColumnHeader();
    this.toolTipText = new System.Windows.Forms.ToolTip(this.components);
    this.btnClose = new System.Windows.Forms.Button();
    this.btnSearch = new System.Windows.Forms.Button();
    this.btnUpload = new System.Windows.Forms.Button();
    this.btnDownload = new System.Windows.Forms.Button();
    this.lvFiles = new System.Windows.Forms.ListView();
    this.clhFilename = new System.Windows.Forms.ColumnHeader();
this.clhFileSize = new System.Windows.Forms.ColumnHeader();
this.sBar = new System.Windows.Forms.StatusBar();
this.FileSaveDialog = new System.Windows.Forms.SaveFileDialog();
this.lblCopyright = new System.Windows.Forms.Label();
this.SuspendLayout();

//
// clhType
//
this.clhType.Text = "Type";
this.clhType.Width = 108;
//
// btnClose
//
this.btnClose.FlatStyle = System.Windows.Forms.FlatStyle.Popup;
this.btnClose.Location = new System.Drawing.Point(217, 5);
this.btnClose.Name = "btnClose"
this.btnClose.Size = new System.Drawing.Size(71, 31);
this.btnClose.TabIndex = 3;
this.btnClose.Text = "&Close";
this.toolTipText.SetToolTip(this.btnClose, "Exist from this window");
this.btnClose.Click += new System.EventHandler(this.btnClose_Click);
//
// btnSearch
//
this.btnSearch.Enabled = false;
this.btnSearch.FlatStyle = System.Windows.Forms.FlatStyle.Popup;
this.btnSearch.Location = new System.Drawing.Point(146, 5);
this.btnSearch.Name = "btnSearch"
this.btnSearch.Size = new System.Drawing.Size(71, 31);
this.btnSearch.TabIndex = 2;
this.btnSearch.Text = "&Search";
this.toolTipText.SetToolTip(this.btnSearch, "Search for file(s) or folder(s)");
this.btnSearch.Click += new System.EventHandler(this.btnSearch_Click);
//
// btnUpload
//
this.btnUpload.Enabled = false;
this.btnUpload.FlatStyle = System.Windows.Forms.FlatStyle.Popup;
this.btnUpload.Location = new System.Drawing.Point(4, 5);
this.btnUpload.Name = "btnUpload"
this.btnUpload.Size = new System.Drawing.Size(71, 31);
this.btnUpload.TabIndex = 0;
this.btnUpload.Text = "&Upload";
this.toolTipText.SetToolTip(this.btnUpload, "Writes the file to current location");
```csharp
350     this.btnUpload.Click += new System.EventHandler(this.btnUpload_Click);
351     //
352     // btnDownload
353     //
355     this.btnDownload.Enabled = false;
356     this.btnDownload.FlatStyle = System.Windows.Forms.FlatStyle.Popup;
357     this.btnDownload.ForeColor = System.Drawing.Color.White;
358     this.btnDownload.Location = new System.Drawing.Point(75, 5);
359     this.btnDownload.Name = "btnDownload";
360     this.btnDownload.Size = new System.Drawing.Size(71, 31);
361     this.btnDownload.TabIndex = 1;
362     this.btnDownload.Text = "&Download";
363     this.toolTipText.SetToolTip(this.btnDownload, "Download file to this computer");
364     this.btnDownload.Click += new System.EventHandler(this.btnDownload_Click);
365     //
366     // lvFiles
367     //
368     this.lvFiles.Columns.AddRange(new System.Windows.Forms.ColumnHeader[]
369     {
370         this.clhFilename,
371         this.clhFileSize,
372         this.clhType
373     });
374     this.lvFiles.ForeColor = System.Drawing.SystemColors.WindowText;
375     this.lvFiles.FullRowSelect = true;
376     this.lvFiles.HideSelection = false;
377     this.lvFiles.Location = new System.Drawing.Point(3, 41);
378     this.lvFiles.MultiSelect = false;
379     this.lvFiles.Name = "lvFiles";
380     this.lvFiles.Size = new System.Drawing.Size(502, 211);
381     this.lvFiles.TabIndex = 4;
382     this.toolTipText.SetToolTip(this.lvFiles, "Double click an entry to open");
383     this.lvFiles.View = System.Windows.Forms.View.Details;
384     this.lvFiles.KeyPress += new System.Windows.Forms.KeyPressEventHandler(this.lvFiles_KeyPress);
385     this.lvFiles.DoubleClick += new System.EventHandler(this.lvFiles_DoubleClick);
386     //
387     // clhFilename
388     //
389     this.clhFilename.Text = "File / Folder";
390     this.clhFilename.Width = 303;
391     //
392     // clhFileSize
393     //
394     this.clhFileSize.Text = "Size";
396     this.clhFileSize.Width = 69;
```
Chapter 5: The P2P Application (C#) 253

395  //
396  // sBar
397  //
398  this.sBar.Location = new System.Drawing.Point(0, 259);
399  this.sBar.Name = "sBar";
400  this.sBar.Size = new System.Drawing.Size(508, 16);
401  this.sBar.TabIndex = 5;
402  //
403  // FileOpenDialog
404  //
405  this.FileOpenDialog.Filter = "*.* (All files)|";
406  this.FileOpenDialog.Title = "Select a file to upload";
407  //
408  // lblCopyright
409  //
410  this.lblCopyright.Location = new System.Drawing.Point(314, 14);
411  this.lblCopyright.Name = "lblCopyright";
412  this.lblCopyright.Size = new System.Drawing.Size(192, 14);
413  this.lblCopyright.TabIndex = 6;
414  this.lblCopyright.Text = "© 2001 www.dreamtechsoftware.com";
415  //
416  // frmShare
417  //
418  this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
419  this.CancelButton = this.btnClose;
420  this.ClientSize = new System.Drawing.Size(508, 275);
421  this.Controls.AddRange(new System.Windows.Forms.Control[] { this.lblCopyright,
422  this.btnClose, this.sBar, this.lvFiles, this.btnSearch, this.btnDownload, this.btnUpload });
423  this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedDialog;
424  this.MaximizeBox = false;
425  this.Name = "frmShare";
426  this.Text = "Shared Contents on : COMPUTERNAME";
427  this.ResumeLayout(false);
428  //
429  
430  /// <summary>
431  /// This function is used to extract the mask value for
432  /// a given filename
433  /// </summary>
434  private int GetMask(string Filename)
435  {
436    // assigns -1 to a local variable. If no matching file is found then return -1
437    int iReturn = -1;
438  
439    // <summary>
440    // This function is used to extract the mask value for
441    // a given filename
442    // </summary>
443    // <param name="Filename"> </param>
444    private int GetMask(string Filename)
445    {
446      // assigns -1 to a local variable. If no matching file is found then return -1
447      int iReturn = -1;
448    }
// take one by one entry and check it for the matching value
for(int i = 0; i < strArray.Length; i++)
    // if match is found
    if( 0 == strArray[i].sFilename.Substring(0,Filename.Length).CompareTo(Filename) )
        // assign the actual mask value of the match to iReturn
        iReturn = strArray[i].iMask;
    //returns the value for iReturn
    return iReturn;

/// <summary>
/// Invoked when Upload button is clicked
/// </summary>
/// <param name="sender"> </param>
/// <param name="e"> </param>
protected void btnUpload_Click (object sender, System.EventArgs e)
{
    try
    {
        //Since you cannot upload at root level
        // This line check that if you are at root level or not
        if( null != PARENTFOLDER )
        {
            // if Mask is 0 or less than 0 then you cannot upload a file
            if( 0 < GetMask(PARENTFOLDER) )
            {
                // Since we have used OpenFileDialog to select the
                // filename to save. This initialises InitialDirectory
                // property of the FileOpenDialog to The path from where
                // the application is running
                FileOpenDialog.InitialDirectory = Application.StartupPath;
                // If use press OK in the FileOpenDialog box
                if( System.Windows.Forms.DialogResult.OK == FileOpenDialog.ShowDialog() )
                {
                    // Assigns a local variable to the name of the
                    // to upload
                    string LocalFilename = FileOpenDialog.FileName;
                    // Assigns a local variable to the name of the
                    // to upload which is to be sent to the listener
                    string RemoteFilename = LocalFilename;
// Extract the filename from the Full Qualified name RemoteFilename = RemoteFilename.Substring(RemoteFilename.LastIndexOf("\") + 1);

// Open a connection if( OpenConnection(COMPUTERNAME) ) {

// Temporary variable used to store the current value of the status bar text and to replace it further
string sTemp = sBar.Text;

// Creates an UPLOAD request with the filename and mask
CreateRequest("UPLOAD", PARENTFOLDER + RemoteFilename, GetMask(PARENTFOLDER).ToString());

// sends this request to the listener
SendDataToListener(REQUESTFILE);

// Show wait cursor while uploading the file
Cursor = Cursors.WaitCursor;

// changes the status bar text
sBar.Text = "Uploading file. Please wait...";

// first process all the pending events from the message queue
Application.DoEvents();

// Assign fileStream object to the local file which is to be uploaded
FileStream(LocalFilename, FileMode.Open, FileAccess.Read); fileStream = new

// bReader is used to read data from the file in binary mode. BinaryReader is a System defined class reader link it created above
BinaryReader(fileStream);
// Initialized the ReadBuffer variable here
// to read only 512 bytes at a time
ReadBuffer = new Byte[MAX_SIZE];

// Read only 512 bytes at a time from the file
// and writes to the socket stream.
// This read continues until the control reaches the
// end of file
while( 0 != (iBytes = bReader.Read(ReadBuffer,0,ReadBuffer.Length)) )
StreamTCP.Write(ReadBuffer,0,iBytes);

// now close the binary reader since it is no longer needed
bReader.Close();

// closes the fileStream object
fileStream.Close();

// close the socket connection
CloseConnection();

// Restore the status bar text
sBar.Text = sTemp;

// restore the cursor
Cursor = Cursors.Default;

// If mask is 0 or less than 0 then throw the exception
else throw new Exception("Read Only folder detected. Access Denied");

// Throw the exception os root level it detected
// since you cannot upload at root level
else throw new Exception("Cannot upload at Root level");

// Catches any thrown exception
catch( Exception err ) {
MessageBox.Show(err.Message,"Error",MessageBoxButtons.OK, MessageBoxIcon.Error);
}

/// <summary>
/// Handles any key press events in the list view
/// </summary>
protected void lvFiles_KeyPress(object sender, System.Windows.Forms.KeyPressEventArgs e) {
    if (13 == e.KeyChar || 32 == e.KeyChar) {
        lvFiles_DoubleClick(null, System.EventArgs.Empty);
    }
}

/// <summary>
/// Called when close button is clicked
/// </summary>
/// <param name="sender"> </param>
/// <param name="e"> </param>
protected void btnClose_Click(object sender, System.EventArgs e) {
    this.Close(); /* closes this dialog box*/
}

/// <summary>
/// This Function creates all XML requests
/// by given Request Scope and Mask
/// </summary>
/// <param name="Request"> </param>
/// <param name="Scope"> </param>
/// <param name="Mask"> </param>
public void CreateRequest(string Request, string Scope, string Mask) {
    xmlCreate = new XMLCreater(REQUESTFILE, "");
    xmlCreate.WriteRequest(Request, Scope, Mask);
}

/// <summary>
/// Opens a Socket connection for every transaction
/// </summary>
/// <param name="Computername"> </param>
public bool OpenConnection(string Computername) {
    bool IsConnected = false;
    try {
        // Initialize the local bool variable to false
        // value of this variable will be returned by the function
        string AddressIP = Computername.Substring(Computername.IndexOf("(")+1);
        AddressIP = AddressIP.Substring(0, AddressIP.Length - 1);
        // defines the remote end point. That is where to connect
        // and at which port to connect to the listener
        // IPAddress is a system define class
    }
IPEndPoint RemoteEP = new IPEndPoint(IPAddress.Parse(AddressIP), 7070);

// initializes the ClientSocket variable
ClientSocket = new TcpClient();

// Performs a remote connection operation
ClientSocket.Connect(RemoteEP);

// Enable the various buttons
btnDownload.Enabled = true;
btnSearch.Enabled = true;
btnUpload.Enabled = true;

// gets the stream for the currently connected socket
// This stream is used to send and recieve data
// from and to the listener
StreamTCP = ClientSocket.GetStream();

// assigns the true value to IsConnected variable
IsConnected = true;

// return the value for IsConnected variable
}

catch (Exception err)
{
    MessageBox.Show(err.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}

return IsConnected;

/// <summary>
/// This function sends any type of request to the listener
/// which is present in a file represented by filename
/// </summary>
/// <param name="Filename"> </param>
public void SendDataToListener(string Filename)
{
    // Creates a new object for fileStream with File open mode
    fileStream = new FileStream(Filename, FileMode.Open);

    // initializes and assigns the ReadBuffer with the
    // length of the file
    ReadBuffer = new Byte[Convert.ToInt32(fileStream.Length)];

    // read whole file in one shot in ReadBuffer variable
    fileStream.Read(ReadBuffer, 0, ReadBuffer.Length);

    // close the fileStream
    fileStream.Close();

    // Delete the RequestFile
    new ServerCommunication().FileDelete(REQUESTFILE);

    // Write the read data to the Socket stream...This data is read by the
Chapter 5: The P2P Application (C#) 259

```csharp
675     // listener
676     StreamTCP.Write(ReadBuffer, 0, ReadBuffer.Length);
677 }
678
679     /// <summary>
680     /// After sending the request, the response will be handled
681     /// by this function. The response will be written in a file
682     /// represented by the filename. This is used to read the response
683     /// that is sent by the listener
684     /// </summary>
685     /// <param name="Filename"> </param>
686     public void GetDataFromListener(string Filename)
687     {
688         // Initializes the WriteBuffer variable to hold
689         // 512 character at a time
690         WriteBuffer = new byte[MAX_SIZE];
691
692         // Creates a File stream to store the response data.
693         fileStream = new FileStream(Filename, FileMode.Create);
694
695         // Creates a binary write by which we can write to the file
696         BinaryWriter bWriter = new BinaryWriter(fileStream);
697
698         // Read only 512 bytes at a time from the socket
699         // stream and writes to the File.
700         // This read continues until the control reaches the
701         // finds no more bytes to read
702         while( 0 != (iBytes =
703             StreamTCP.Read(WriteBuffer, 0, WriteBuffer.Length)) )
704             bWriter.Write(WriteBuffer, 0, iBytes);
705
706         // Closed the binary writer
707         bWriter.Close();
708
709         // Closed the fileStream Object
710         fileStream.Close();
711     }
712
713     /// <summary>
714     /// Invoked when user double clicks on the list view
715     /// </summary>
716     /// <param name="sender"> </param>
717     /// <param name="e"> </param>
718     protected void lvFiles_DoubleClick (object sender, EventArgs e)
719     {
720         // local variable to store index value
721         int index;
722
723         // local variable to store the name of the entry at which user
724         // double clicks
725         string Filename;
726
727         try
728             // Get the selected entry from the list view,
729             // its name and its index at which it is present
```
index = GetSelectedItemFromListView(out Filename);

// Index must be greater than -1
if( -1 != index )
{
   // if Filename is null then directory is selected
   if( null == Filename )
   {
      // This will send the request to listener with
      // the name of the folder
      frmShare ShareForm = new
      frmShare(COMPUTERNAME,"SEARCH",strArray[index].sFilename + 
      "*.*",strArray[index].iMask.ToString());

      //Now show the contents here
      ShareForm.Show();
   }
   else DownloadFile(index,Filename,false);
}

// if index is -1 then displays an appropriate error message
else throw new Exception("Nothing Selected");

// Catch and show any system error message here
catch( Exception err ) {
   MessageBox.Show(err.Message,"Warning",MessageBoxButtons.OK,
   MessageBoxIcon.Warning);
}

/// <summary>
/// This function gets the value coressponding to the array
/// which is currently selected in a list view
/// It returns the value in sReturn and function returns the
/// index of this value in the array
/// </summary>
/// <param name="sReturn"> </param>
private int GetSelectedItemFromListView( out string sReturn )
{
   // This will get the selected items from the list
   // in our case only one item is selected

   // initialize ant assign iIndex to -1
   int iIndex = -1;

   // This value will be returned by the function in out string sReturn
   sReturn = null;

   // if nothing is selected dont go inside this iteration
   if( 0 < items.Count )
   {
// Get the index of the selected item
iIndex = items[0].Index;

// finds the actual filename by the given index from the list
// which we have maintained
string Filename = strArray[iIndex].sFilename;

// Checks whether the selected entry is a Folder or File
// if Folder is selected then return the name of the folder
// else return null
if( !Filename.EndsWith("\\") )
    sReturn = Filename.Substring(Filename.LastIndexOf("\\") + 1);
else sReturn = null;

return iIndex;

/// <summary>
/// This function parses any XML response and displays it in the window
/// The Response is in a file denoted by Filename
/// </summary>
/// <param name="Filename"> </param>
public bool Parsing(string Filename)
{
    // This variable is returned by the function, if the value
    // of this variable is true then Parsing is successfull else
    // Parsing Failed
    bool bReturn = false;

    // This line of code will create a new instance of
    // XMLParser class and assigns it to the xmlParser variable
    xmlParser = new WorkingWithXML.XMLParser();

    // This will create a new instance of the XMLSTRUCT structure
    // and assigns it to the xmlStruct variable
    xmlStruct = new WorkingWithXML.XMLSTRUCT();

    // This will check whether the correct XML is received by the browser
    // or not if correct XML is received then only parse that XML
    // The XML response must me of type "SHOWFILES"
    if( 0 == new ServerCommunication().TypeOfXMLRecieved(RESPONSEFILE).CompareTo("SHOWFILES") )
    {
        // The ParseXML function is present in the XMLParser class
        // which is represented by xmlParser variable, and it will
        // return the number of records parsed by the parser
        int iEntries = xmlParser.ParseXML(RESPONSEFILE, out xmlStruct,new ServerCommunication().TypeOfXMLRecieved(RESPONSEFILE));

        // If number of records greater than zero then continue
        // further, else dont go inside
        if( 0 < iEntries )
        {
            /// This block of code is used to sort the records
/// first the Folders should come and then the files
/// are to be sorted by their sizes
for( int i = 0; i < iEntries; i++ )
{
    for( int k = i + 1; k < iEntries; k++ )
        if(!xmlStruct.SHOWFILES[i].sFilename.EndsWith("\") && xmlStruct.SHOWFILES[k].sFilename.EndsWith("\") || xmlStruct.SHOWFILES[i].iFileSize > xmlStruct.SHOWFILES[k].iFileSize )
            string Temp;
            Temp = xmlStruct.SHOWFILES[i].sFilename;
            xmlStruct.SHOWFILES[i].sFilename = xmlStruct.SHOWFILES[k].sFilename;
            xmlStruct.SHOWFILES[k].sFilename = Temp;
            Temp = xmlStruct.SHOWFILES[i].iFileSize.ToString();
            xmlStruct.SHOWFILES[i].iFileSize = xmlStruct.SHOWFILES[k].iFileSize;
            xmlStruct.SHOWFILES[k].iFileSize = Convert.ToInt32(Temp);
            Temp = xmlStruct.SHOWFILES[i].iMask.ToString();
            xmlStruct.SHOWFILES[i].iMask = xmlStruct.SHOWFILES[k].iMask;
            xmlStruct.SHOWFILES[k].iMask = Convert.ToInt32(Temp);
}
Chapter 5: The P2P Application (C#) 263

876     string fName;
877
878     // This will take one by one record from the SHOWFILES
879     // structure of the XMLSTRUCT
880     for( int i = 0; i < iEntries; i++ )
881     {
882         ImageList imgList = new ImageList();
885         lvFiles.SmallImageList = imgList;
886
887     // assigns the filename present in the
888     // SHOWFILES struct
889     fName = xmlStruct.SHOWFILES[i].sFilename;
890     // Fills the strArray structure with its values
891     strArray[i].sFilename = fName;
892     strArray[i].iMask = xmlStruct.SHOWFILES[i].iMask;
893     strArray[i].iFileSize = xmlStruct.SHOWFILES[i].iFileSize;
894
895     // not used in this version, but kept for
896     future
897     // Checks whether the value present in fName is
898     // Folder
899     // or file, because folder name ends with
900     // backslash
901     if( fName.EndsWith("\\") )
902     {
903         // This will remove the trailing
904         // fName
905         fName = fName.Substring(0,fName.Length - 1);
906         // This will insert a single record in
907         // the listview
908         // represented by fName and sSubItems
909         // size and type. 0 Here displays the
910         // the image list which is the image of
911         // the folder
912         lvItems = lvFiles.Items.Insert(i,fName);
// Item 0 represents the size, since there is no size for the directory so this item, since
lvItems.SubItems.Add("");
// Assigns the folder value to the 1 item, since
lvItems.SubItems.Add("Folder");

lvItems.ImageIndex = 0;
}
else
{
    // The control will come into this code only if the record is a
    // File not folder
    else
    {
        // Extracts only filename from Full qualified path
        fName = fName.Substring(fName.LastIndexOf("\") + 1);

        // This will insert a single record in the listview
        // represented by fName and sSubItems which includes
        // size and type. 1 Here displays the 1st image from
        // the file
        lvItems = lvFiles.Items.Insert(i, fName);

        // Assigns the file size
        lvItems.SubItems.Add(xmlStruct.SHOWFILES[i].iFileSize.ToString());

        // extract the extension of the file and assigns it to the
        // last element of sSubItem
        lvItems.SubItems.Add(fName.Substring(fName.LastIndexOf\(\."\)) + 1) + " File");

        lvItems.ImageIndex = 1;
    }
    // assigns a true value to bReturn variable to indicate that
    // parsing and displaying of the records are successful
    bReturn = true;
}
else
953 
954     // Calls the DisableUI function
955     DisableUI();
956 
957     // Displays the Message Box
958     MessageBox.Show("No result found for this request","Warning",MessageBoxButtons.OK, MessageBoxIcon.Warning);
959 
960 
961     // The case if any error has occurred at the listener's side
962     // is informed by the Error XML. If SHOWFILES XML is not returned then parse the ERROR XML and Show the error to the user
963     else
964     {
965         // Parses the error XML returned by the listener
966         xmlParser.ParseXML(RESPONSEFILE, out xmlStruct,new ServerCommunication().TypeOfXMLRecieved(RESPONSEFILE));
967         // calls DisableUI to disable various the Userinterface controls
968         DisableUI();
969 
970         // Popup the message box and displays the error message
971         MessageBox.Show(xmlStruct.ERROR.sDescription, xmlStruct.ERROR.sSeverity,MessageBoxButtons.OK, MessageBoxIcon.Error);
972 
973         // Delete the File containing the XML Response from the listener
974         new ServerCommunication().FileDelete(RESPONSEFILE); 
975         // returns the value for bReturn Variable
976         return bReturn;
977     }
978 
979     /// <summary>
980     /// Disable the buttons when not needed
981     /// </summary>
982     private void DisableUI()
983     {
984         btnUpload.Enabled = false;
985         btnDownload.Enabled = false;
986         btnSearch.Enabled = false;
987         lvFiles.Enabled = false;
988     }
989 
990     /// <summary>
991     /// Closes the current TCP and Stream Connections
992     /// </summary>
993     public void CloseConnection() { StreamTCP.Close(); ClientSocket.Close();
994     }
995 
996     /// <summary>
997     /// This function performs a download operation of a file
998     /// from the listener's end. This function returns the actual
999     /// download path from the remote machine. It takes the index
private string DownloadFile(int index, string Filename, bool bDelete) {
    // Change the title of the File savedialog box
    FileSaveDialog.Title = "Download As";
    // Pops up the File save dialog box with the default filename
    FileSaveDialog.FileName = Filename;
    // Assigns the Initial directory of the File save dialog box to the application's startup path
    FileSaveDialog.InitialDirectory = Application.StartupPath;
    // Declares an initialize the variable sReturn which is used to return the value by function. This stores the name of the file which is to be downloaded from the remote end
    string sReturn = null;
    // If user chooses OK from the File Save dialog box then only download can begin
    if (System.Windows.Forms.DialogResult.OK == FileSaveDialog.ShowDialog()) {
        // Stores the current text of the status bar is temporary variable sTemp
        string sTemp = sBar.Text;
        // Writes new text on the status bar
        sBar.Text = "Downloading File. Please wait...";
        // Show the hour glass cursor while downloading the file
        Cursor = Cursors.WaitCursor;
        // process all the pending events first from the message loop
        Application.DoEvents();
        // Opens socket connection to the listener and checks whether connected or not
        if (OpenConnection(COMPUTERNAME)) {
            // Creates a DOWNLOAD request for a particular file to download.
            CreateRequest("DOWNLOAD", strArray[index].sFilename, "");
            // Sends this request to the listener
            SendDataToListener(REQUESTFILE);
        }
    }
}
1058 // Assigns the sReturn variable to the filename
1059 chosen for
1060 sReturn = FileSaveDialog.FileName;
1061
1062 // Get the download data from the listener and save it
1063 // in the filename represented by the Filename property
1064 // of File save dialog box
1065 GetDataFromListener(FileSaveDialog.FileName);
1066
1067 // Close the currently opened socket connection
1068 CloseConnection();
1069
1070 // Delete the file if bDelete is true
1071 if( bDelete ) new
1072 ServerCommunication().FileDelete(Filename);
1073
1074 // Restore the previous text of the status bar
1075 sBar.Text = sTemp;
1076 }
1077 }
1078 // restore the default cursor state
1079 Cursor = Cursors.Default;
1080
1081 // returns the filename with Full qualified path
1082 return sReturn;
1083 }
1084
1085 /// <summary>
1086 /// Invoked when the download button is clicked
1087 /// </summary>
1088 /// <param name="sender"> </param>
1089 /// <param name="e"> </param>
1090 protected void btnDownload_Click (object sender, System.EventArgs e)
1091 {
1092  // Declares a local Filename variable
1093  string Filename;
1094  // Get the currently selected item from the List view
1095  int index = GetSelectedItemFromListView(out Filename);
1096  try
1097  {
1098   // Checks whether any entry is selected or not
1099   if( -1 != index)
1100    // Checks whether a filename is selected for
1101    // downloading
1102    // or not
1103    if( null != Filename )
1104     // If filename is selected then download it
1105     DownloadFile(index,Filename,false);
// else throw an exception, Folders cannot be downloaded
else throw new Exception("Cannot download folder");

// If nothing is selected than displays an error message
else throw new Exception("Nothing Selected");

// catches any system generated error message and displays it to the user
catch( Exception err ) {
    MessageBox.Show(err.Message,"Warning",MessageBoxButtons.OK,
                    MessageBoxIcon.Warning); }

/// <summary>
/// Invoked when the search button is clicked
/// </summary>
/// <param name="sender"> </param>
/// <param name="e"> </param>
protected void btnSearch_Click(object sender, System.EventArgs e) {
    try
    {
        // Check that if search is performed at root level or not,
        // since root level search is not possible from here
        if( null != PARENTFOLDER )
        {
            // Creates a new object for frmSearch and initializes it.
            frmSearch SearchForm = new frmSearch();

            // This statement restricts the search form from
            // Showing Search on: criteria Textbox
            SearchForm.bFlag = false;

            // If user presses OK in the search form then go inside
            if( System.Windows.Forms.DialogResult.OK ==
                SearchForm.ShowDialog() )
            {
                // Get the search criteria from the SearchForm
                // variable
                // of the frmSearch class and passes it to the
                // constructor
                // of this class with some more details
                frmShare ShareForm = new
                frmShare(COMPUTERNAME,"SEARCH",PARENTFOLDER +
                SearchForm.SearchFor,GetMask(PARENTFOLDER).ToString());

                // Again show this dialog with the search results
                ShareForm.ShowDialog();
            }
        }
    }
}
Chapter 5: The P2P Application (C#)  269

1154 // Displays an error message
1155 else throw new Exception("Cannot search at root level here");
1156 }
1157
1158 // Catches any system generated error and displays it
1159 try{
1160     catch( Exception err ) {
1161         MessageBox.Show(err.Message,"Error",MessageBoxButtons.OK, MessageBoxIcon.Error);
1162     }
1163 }

Code description of ‘frmShare’ class (frmShare.cs)

♦ Lines 4-13: This includes the basic packages needed for the various classes used to build this application. This includes packages used for communicating with network, streaming facility, file I/O, text manipulations, and so on. Line 11 includes a user-defined class called WorkingWithXML. This class is used to communicate with the server, to parse XML data, and to create XML requests.

♦ Line 18: This declares a public class frmShare that represents the base class for this file. This class is derived from System.Windows.Forms. All the variables defined here represent their respective controls that are pasted on the form. C# automatically generates these lines of code.

♦ Lines 20-32: In these lines of code, the designer variables for the form are declared. These variables are declared automatically by the C# editor and represent the various controls that are drawn at the design time window on the form.

♦ Lines 41-127: These are the user-defined global variables. They are used in various situations in this application; each of these either represents a class or a data type.
  • 41: This variable, iBytes, is used to store the number of bytes that are either read or written to the stream on the socket or in file using the file stream.
  • 49: The variables COMPUTERNAME and PARENTFOLDER declared here are used to store the name of the computer to which you are currently connected, along with its IP address, and PARENTFOLDER stores the name of the parent folder, the contents of which you are viewing.
  • 55: The ClientSocket object of TcpClient class declared here is used to establish a socket connection between the listener and the browser (client). This object is also used to get the stream of the listener and to send the request over that socket stream.
  • 61: XMLCreater class’s object xmlCreater is declared here to have the functionality of creating the XML requests to be sent over the socket to the listener. This is a user-defined class and is defined in the WorkingWithXML assembly.
  • 67: This object, StreamTCP, of the NetworkStream class is used to get the stream object for the current TCP socket connection. By using this stream object, we have the ability to transfer the data between the listener and browser (client) or vice-versa.
  • 74: This global object, fileStream, of the FileStream class is generally used to have the I/O capabilities over the files. Any I/O performed in a file in this application uses this object.
  • 80: ReadBuffer and WriteBuffer are byte arrays used to read or write data to the file in byte format. These variables are initialized later in this application where they are needed and are initialized to store 512 bytes at a time.
  • 87: The xmlParser object of the user-defined class XMLParser present in the ‘WorkingWithXML’ assembly is used to acquire the capabilities for parsing an XML file being generated in this application.
  • 94: The xmlStruct variable of the XMLSTRUCT structure is used to store the parsed values from the XML. These parsed values are the output from the preceding XMLParser class.
270  Chapter 5: The P2P Application (C#)

• 101: The variable `strArray` declared here is used to store the corresponding files/folders, which are currently being shown in the window. This array is used only for internal purposes by this application, not for user-interaction purposes.

• 114-120: These two string-type variables are used to store the names of the request and response files, respectively. These names are used to store the contents of the XML.

♦ Lines 133-149: This is the default constructor for the `frmShare` class. This is used to show the results generated when you search for a file on a particular computer by using the Search button present in the main window, where the computer-name list is shown.

♦ Lines 158-207: This is not a default constructor and is not generated automatically; rather, it is a user-defined constructor created to show all the shared files when you select a particular remote computer you want to connect. This is called from the Open button situated on the main window where the computer name list is shown. The name of the computer along with its IP address is passed to this constructor in the `Computername` parameter. This constructor makes use of this parameter and extracts the IP address from it, which is later used to make a connection to the computer represented by that IP address. This constructor by default creates a `SHOWFILES` request, which is sent to the listener.

♦ Lines 220-270: Once again, implementation of the user-defined constructor is associated in this application. But this time the constructor is invoked when you click the Search button when viewing files/folders, for searching a particular file on the connected computer. In this constructor implementation, four strings are passed for storing specific target information about computer name. The strings are: Type of Request user wants to send to listener, scope and mask values if there are any, along with the computername. This information is passed to this constructor.

♦ Lines 276-280: The `Dispose()` function declared here is automatically generated by the C# environment. It is used to free all the resources acquired by this class.

♦ Lines 287-435: This initializes various designer variables required for the smooth running of the application. This part of the code is autogenerated by IDE, and users are advised not to modify the contents.

♦ Lines 442-456: This function is used to retrieve the mask for a particular file/folder from the `strArray` array. This mask is used to distinguish between read-only files and read-write access files. A selected name of the file/folder is passed as a parameter to this function in the `Filename` variable. The function then looks up the name in the list and returns the associated mask value for that file/folder. If no file/folder is presented by the `Filename` variable, the function returns -1.

♦ Lines 463-566: This function is invoked when you click the Upload button in the view files window. This function is responsible for sending a particular selected file to the listener's end.

• 469: Uploading cannot be done at the root level as well as in situations in which the mask is ‘0’ (read-only access). These lines are for checking these conditions.

• 478: This initializes the `InitialDirectory` property of the `FileOpenDialog`, which is set to the default path from where the application is running.

• 481: If you click the Save button in the `FileOpenDialog` box, the filename from the fully qualified name is extracted and is stored in the `RemoteFilename` variable.

• 495: Here the connection to the remote listener is opened via the `OpenConnection()` function of this class. The computer name with its IP address is passed to this function.

• 502-511: An upload request is created with the `filename` and `mask`. The request is sent to the listener via the `SendDataToListener()` function, and, in the mean time, the cursor is changed into ‘wait cursor’ along with the change in the text of the status bar.

• 515: All pending events are processed from the message queue so that the application might not block.
• 519-535: A file stream object is assigned to the local file, which is to be uploaded. The BinaryReader class’s object bReader is used to read 512 bytes at a time from the file and transmits over the socket stream. This reading continues until control reaches the end of the file.
• 536: Data is then written on the network stream by using the Write() function of the StreamTCP object.
• 539-551: After completion of their tasks, BinaryReader and FileStream objects are closed, as they are no longer needed. Connection to the socket is closed, too, and text on the status bar and cursor is restored.
• 557-566: Error messages are trapped and shown to you if uploading has failed anywhere.
• Lines 573-579: This function handles any key-press events in the list view of the frmShare class.
  ♦ Line 586: The function is invoked when you press the Close button to exit from this form. It is incorporated by using the Close() function of this class.
  ♦ Lines 595-602: This function is responsible for creating the XML request queries for the listener. These requests are created on the basis of the request type, the scope of that request, and the access mask. The requests are created in the filename represented by the REQUESTFILE variable used in the second line of the function.
  ♦ Lines 608-649: This function opens a socket connection to the remote computer and returns true in case of a successful connection and false if not connected successfully. Apart from this, it enables all the buttons on a successful connection. Line 597 of this function gets the network stream of this socket, which is used to transfer data between two applications. This function is called every time you need to send a request or get a response from the listener.
• 618-619: The IP address is extracted from the Computername variable.
• 623: A remote endpoint is mentioned in this line, which states where and at which port to connect to the listener.
• 629: The remote connection operation is performed here.
  ♦ Lines 656-677: The SendDataToListener() function sends any type of request in a file represented by Filename to the listener. This has been incorporated for reading the file byte by byte and sending it over the network stream of the listener.
  ♦ 672: The requestfile is deleted, as data has been read from it and stored in the ReadBuffer variable.
• 686-710: The main purpose of this function, after the request is sent, is to read the response sent by the listener. The response is written to a file, which is later parsed for its data.
• 690: The WriteBuffer variable is initialized to hold 512 Bytes at a time.
• 693: A FileStream is created to store data, which is being read from the network stream over the socket.
• 702-703: Only 512 bytes of data are read at a time and written to the file. This reading and writing of data continues to operate until the stream becomes empty and no data is left to read.
  ♦ 706-709: Finally, the BinaryWriter and FileStream are closed.
  ♦ Lines 717-757: This function is invoked when you double click any item in the list view box. First, the selected item is retrieved from the list. It is checked whether it is a file or a folder. If a file is selected, the application opens a dialog box and asks you where to download this selected file. If a folder name is selected, the application opens a new window in which the contents of the selected folder are shown.
• 720-724: This is a locally declared variable index store zero-based index position of the selected item from the list view. The Filename variable stores the name of the file/folder that is selected when you double click it.
733-736: If the Filename variable stores a null value and the index is greater than -1, a folder is selected or else a filename.

Lines 766-796: This function retrieves the original name of the selected item from the list view, corresponding to the array strArray.

Lines 803-982: This function is responsible for parsing the XML response sent by the listener. When the parsing of the XML document is successful, this function shows the parsed data in tabular form in a list view.

808: The value of this bool-type variable is returned by the function. If the parsing is successful and no error response is returned, the value of this variable bReturn is true; otherwise, it is false.

812: A new instance of XMLParser class is created and assigned to the xmlParser object.

816: A new instance of XMLSTRUCT is created here. It is used to store the parsed values from the XML document.

821: This line checks the type of XML response received by the browser.

826: This is responsible for parsing the response XML received by the browser. To this function, we pass the name of the file to parse and the reference to the structure where the parsed results are stored and the type of XML received, so that the parsed results are stored in their respective structures. This function returns the number of records parsed from the XML. This number is then used to extract the exact records and fills them in the list view.

835-855: The code given in these lines is used to sort the records present in xmlSruct, first folder-wise and then file size-wise.

866-946: In these lines of code, the strArray is initialized to hold the number of records that are parsed previously. These records are then manipulated accordingly. A check has been made for the files and folder, as a folder name always ends with a backslash(‘\’). While inserting the items in the list view, if a folder is found, a folder icon is being used and a file icon in the case of files.

952-959: These lines are executed only if no record is parsed from the XML.

965-975: If the SHOWFILES response XML is not returned here check for the ERROR XML. If this error XML is returned, it is being parsed for the Error and error descriptions. After parsing, the error description is popped up to the user, the popup shows the exact error information that has occurred on the istener's end.

998: This CloseConnection() function does very little work. It closes the socket connection and the network stream associated with this socket connection. After opening every connection, you find an associated CloseConnection() function being called.

1013-1083: This function holds the responsibility of handling download operations of a file from the listener. This function performs a download operation and, after downloading successfully it, returns the path along with the name of the downloaded file. When downloading, double click a file, or click the Download button after selecting a file. The Download As window appears; choose the location on your local computer indicating where to download that file, and press the Enter button. While downloading, the text of the status bar is changed, and the wait cursor is displayed. If download is successful, these two are restored.

1019: This assigns the default filename to the Download dialog box, which you have selected.

1023: This assigns the initial directory of the File Save dialog box to the application’s startup path.

1032: This checks whether you have selected to download the file or not.

1049-1082: A new socket connection is opened toward the listener and checks the connectivity. Then the download request is made for the same file and sent to the listener. A file selected for
downloading is then being saved into the `sReturn` variable. Once the file is downloaded via `GetDataFromListener()` function, it is saved on the local machine wherever you choose. Lastly, the connection is closed. The text and the cursor are restored. In the case that a connection with the listener cannot be established, an error message is displayed.

♦ Lines 1090-1119: This function is invoked when the Download button is clicked, which is placed on the application's form. The purpose of this function is to gather the required information about the file selected for downloading and to pass that information to the `DownloadFile()` function discussed previously.

• 1096: This retrieves the currently selected item from the list view, which contains many files or a folder. It is then checked that only files are valid for downloading.

• 1107: Here the actual download function is called to download the selected file from the listener. The `index`, `Filename`, and a flag (false) is passed to the `DownloadFile()` function. This flag determined whether to delete the file after downloading or not. The appropriate error messages are displayed wherever necessary.

♦ Lines 1126-1160: This function is invoked when you click the Search button while viewing the files/folders. A search form is displayed. Enter the search criteria in this form. These criteria are passed to this function and are used to make an XML request. This request is then sent to the listener by calling the constructor of the `frmShare` class, and all the matched files to that criteria are shown in a new window.

• 1132: Here we are checking whether the search is happening on the root-level, as such a search cannot be done from this point. You cannot search if there is no parent folder for the items that you are currently viewing.

• 1135: This creates a new object for the `frmSearch` class and initializes it.

• 1139: Setting the variable to false restricts the Search Form from showing Search on: criteria box.

• 1142-1151: When you click the OK button on the Search dialog box, search criteria are retrieved from the `SearchFor` variable of the `frmSearch` class and it are passed to the constructor of this class (`frmShare`) with some details.

• 1155-1159 If you perform a search at root level, an appropriate error message is displayed along with any system-generated error.

### Search Window

The window in Figure 5-7 allows you to search for a specific file or folder on a single computer or multiple computers. In the Search on box, enter the computer name (ABC), or use wild cards (A*). In the Search for box, enter the file/folder name, and wildcards are also supported here.

![Figure 5-7: The Search dialog box](image-url)
Search Form

namespace: Client

frmSearch provides you with an intuitive interface to perform searches at various levels and of different types. The entire coding in Listing 5-8 has been numbered for better legibility.

Listing 5-8: frmSearch

//© 2001 Dreamtech Software India Inc.
// All rights reserved

```csharp
namespace Client // Copyright 2001 Dreamtech Software India Inc.
{
    using System; // Provides the basic functionality of .NET
    using System.Drawing; // Provides the Drawing features, Used for cursors
    using System.Windows.Forms; // Provides the drawing of buttons,
    //listviews etc
    using System.Collections; // Provides the different type of class
    //collections
    using System.ComponentModel; // Provides the facility of using
    //components

    /// <summary>
    /// Summary description for frmSearch.
    /// </summary>
    public class frmSearch : System.Windows.Forms.Form
    {
        private System.ComponentModel.IContainer components;
        private System.Windows.Forms.Label label1;
        public System.Windows.Forms.TextBox txtSearchFor;
        public System.Windows.Forms.Label lblSearchFor;
        private System.Windows.Forms.ToolTip toolTipText;
        public System.Windows.Forms.TextBox txtSearchOn;
        public System.Windows.Forms.Label lblSearchOn;

        /// <summary>
        /// The below variables are user defined variables
        /// used to store various transient values.
        /// </summary>
        public string SearchOn; // Stores the value for first search scope
        public string SearchFor; // Stores the value for second search scope
        public bool bFlag; // Determines whether to perform SearchOn
        //validation

        /// <summary>
        /// Default constructor of the class
        /// </summary>
        public frmSearch()
        {
            // Required for Windows Form Designer support

            InitializeComponent(); // auto generated line by the IDE
        }
    }
```
public override void Dispose()
{
    base.Dispose();
    components.Dispose();
}

private void InitializeComponent()
{
    this.components = new System.ComponentModel.Container();
    System.Resources.ResourceManager resources =
        new System.Resources.ResourceManager(
            typeof(frmSearch));
    this.lblSearchFor = new System.Windows.Forms.Label();
    this.toolTipText = new System.Windows.Forms.ToolTip(this.components);
    this.txtSearchOn = new System.Windows.Forms.TextBox();
    this.txtSearchFor = new System.Windows.Forms.TextBox();
    this.btnSearch = new System.Windows.Forms.Button();
    this.btnCancel = new System.Windows.Forms.Button();
    this.lblSearchOn = new System.Windows.Forms.Label();
    this.label1 = new System.Windows.Forms.Label();
    this.SuspendLayout();
    //
    // lblSearchFor
    //
    this.lblSearchFor.Location = new System.Drawing.Point(13,
55);
    this.lblSearchFor.Name = "lblSearchFor";
    this.lblSearchFor.Size = new System.Drawing.Size(63, 13);
    this.lblSearchFor.TabIndex = 4;
    this.lblSearchFor.Text = "Search for :";
    //
    // txtSearchOn
    //
    this.txtSearchOn.Enabled = false;
    this.txtSearchOn.Location = new System.Drawing.Point(80, 26);
    this.txtSearchOn.Name = "txtSearchOn";
    this.txtSearchOn.Size = new System.Drawing.Size(199, 20);
    this.txtSearchOn.TabIndex = 0;
    this.txtSearchOn.Text = "Write the search criteria here
(For ex: *, A* or HAR* etc. for
matching computer" +
    "names");
}
// txtSearchFor
this.txtSearchFor.Location = new System.Drawing.Point(80, 53);
this.txtSearchFor.Name = "txtSearchFor";
this.txtSearchFor.Size = new System.Drawing.Size(199, 20);
this.txtSearchFor.TabIndex = 1;
this.txtSearchFor.Text = "";
this.toolTipText.SetToolTip(this.txtSearchFor, "Write the search criteria here
(For ex: *, A* or HAR* etc.
for matching Filename" + "s")");

// btnSearch
this.btnSearch.FlatStyle = System.Windows.Forms.FlatStyle.Popup;
this.btnSearch.Location = new System.Drawing.Point(79, 88);
this.btnSearch.Name = "btnSearch";
this.btnSearch.Size = new System.Drawing.Size(90, 32);
this.btnSearch.TabIndex = 2;
this.btnSearch.Text = "&Search"
this.toolTipText.SetToolTip(this.btnSearch, "Performs search
operation");
this.btnSearch.Click += new System.EventHandler(this.btnSearch_Click);

// btnCancel
this.btnCancel.FlatStyle = System.Windows.Forms.FlatStyle.Popup;
this.btnCancel.Location = new System.Drawing.Point(189, 88);
this.btnCancel.Name = "btnCancel";
this.btnCancel.Size = new System.Drawing.Size(90, 32);
this.btnCancel.TabIndex = 3;
this.btnCancel.Text = "&Cancel"
this.toolTipText.SetToolTip(this.btnCancel, "Cancel the search and
close this window");

// lblSearchOn
this.lblSearchOn.Enabled = false;
this.lblSearchOn.Location = new System.Drawing.Point(13, 28);
this.lblSearchOn.Name = "lblSearchOn";
this.lblSearchOn.Size = new System.Drawing.Size(63, 13);
this.lblSearchOn.TabIndex = 0;
this.lblSearchOn.Text = "Search on ":

```csharp
136     //
137     // label1
138     //
139     this.label1.Location = new System.Drawing.Point(93, 5);
140     this.label1.Name = "label1";
141     this.label1.Size = new System.Drawing.Size(192, 14);
142     this.label1.TabIndex = 5;
143     this.label1.Text = "© 2001 www.dreamtechsoftware.com";
144     //
145     // frmSearch
146     //
147     this.AcceptButton = this.btnSearch;
148     this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
149     this.CancelButton = this.btnCancel;
150     this.ClientSize = new System.Drawing.Size(288, 132);
151     this.Controls.AddRange(new System.Windows.Forms.Control[] { this.label1,
152     this.txtSearchFor,
153     this.lblSearchFor,
154     this.btnCancel,
155     this.btnSearch,
156     this.txtSearchOn,
157     this.lblSearchOn });
158     this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedToolWindow;
159     this.Name = "frmSearch";
161     this.Text = "Search Dialog";
162     
163     // Puts the Computer.ico as the form icon
164     this.Icon = new System.Drawing.Icon(Application.StartupPath + "\Computer.ico");
165     this.ResumeLayout(false);
166    }
167    /// <summary>
168    /// Invoked when click on the search button
169    /// </summary>
170    /// <param name="sender"> </param>
171    /// <param name="e"> </param>
172    protected void btnSearch_Click (object sender, System.EventArgs e)
173    {
174      // Start of the try block
175      try
176      {
177      // Validates for the correctness of the SearchFor Variables
178      // and check that the user has entered correct search scope
179      // or not.
180      if( 0 < txtSearchFor.Text.Trim().Length )
181        SearchFor = txtSearchFor.Text.Trim();  // Assigns
182        // the value of Textbox text to "SearchFor" variable
183      else
184          {
185            
```
```csharp
187     // This line stops the dialog from returning
188     throw new Exception("Cannot search for blank values");
189     // Displays an Error message in Messagebox
190     }
191     
192     if( bFlag )// Checks whether to validate SearchOn or not
193     {
194     // Validates for the correctness of the SearchOn
195     //Variables
196     // and check that the user has entered correct search
197     //scope or not.
198     if( 0 < txtSearchOn.Text.Trim().Length )
199     SearchOn = txtSearchOn.Text.Trim(); // Assigns the
200     //value of Textbox text to "SearchOn" variable
201     else
202     
204     // This line stops the dialog from returning
205     throw new Exception("Cannot search for blank values");
206     // Displays an Error message in Messagebox
207     }
208 }
209 
210 // This block catches any system generated error message and
211 // stops the program from crashing
212 catch( Exception err ) { MessageBox.Show(err.Message,"Warning",
213     MessageBoxButtons.OK,
214     MessageBoxIcon.Warning);}
215 
216 }
217 }
218 }
219 
220 
```

**Code description of ‘frmSearch’ class (frmSearch.cs)**

- **Lines 3-7:** This includes necessary packages used for the basic functionality of our application.
- **Line 12:** This is an automatically generated code line, which represents your base class for this file. It is inherited from System.Windows.Forms.Form.
- **Lines 14-22:** C# generates these lines of code automatically. The variables declared in these lines represent the user interface controls drawn on the form.
- **Lines 28-30:** This declares three user-defined variables. These are used to store the values of the two text boxes used for entering the search criteria. The third variable determines whether to show the search for computer text box or not.
- **Lines 35-41:** This is the default constructor for this class. It is implemented by C# automatically.
- **Lines 47-51:** This function is used to clean up all the memory and to free resources acquired by this class. This is automatically called on the closing of the form created in this class.
- **Lines 58-168:** This initializes various designer variables required for the smooth running of the application. This code is autogenerated by IDE, and users are advised not to modify the contents.
- **Lines 175-206:** This function is invoked when you click the Search button on this form.
  - **182-188:** This checks the correctness of the search criteria entered for searching for files/folders. This validates the search (that is, it cannot be empty and must not have special characters of unwanted signs). The search pattern entered here is stored in the SearchFor variable.
• 194-201: This checks the correctness of the search pattern entered for searching for files/folders on a specific computer or group of computers. This validates the search (that is, it cannot be empty and must not have special characters of unwanted signs). The search criteria entered is stored in the `SearchOn` variable.

Summary

In this chapter, we take up the advanced-level programming of the C# version of our P2P application. We have used the Microsoft XML parser 3.0 (you can download this parser from http://msdn.microsoft.com/msxml) for parsing the XML responses being generated as a result of each request. Several classes have been integrated to give our application the proper constitution. The `ServerCommunication` class is responsible for all the communication with the server. The `XMLParser` class is responsible for parsing the XML responses. Similarly, the `xmlCreator` class is responsible for creating the XML requests. The other classes are the individual parts of their forms (for example, the `frmShare` class, `frmLogin`, and so on). These classes have been crafted so that they can be perfectly integrated to build our P2P application.
Chapter 6

Streaming in the P2P Application

For quite some time, streaming has been the major tool for sharing digital audio/video data over computer networks. It has indeed revolutionized the very concept of availing audio/video data. Before the advent of streaming, users had to endure long hours of waiting for downloading a single file before they could actually play it. Worse, in case of an error occurring while downloading the file, the user could not play even that amount of data already downloaded after waiting patiently for hours. Thus, the conventional method was far from being time effective.

With the emergence of streaming, handling audio/video data became very easy — audio/video data is streamed for the user, which means that chunks of data are directed to the destination computer so that the user need not download the required audio/video file completely before beginning to use it. Thus, streaming is a process of downloading an audio/video file occurs in consignments. The amount of data that has been streamed can be played or displayed to the user even as the rest of the data is being streamed in the background. Thus, playing and downloading audio/video data occur simultaneously, which proves to be highly time effective for the user.

Because audio/video data is received in a compressed form, a media player is needed to uncompress it for display. Several media players are available on the market such as Windows Media Player and Real Player. The streaming add-on described in this chapter uses Windows Media Player. You can download the latest version of the Windows Media Player from http://www.msdn.Microsoft.com.

Streaming in the Application

The design of our application permits new concepts to be implemented. We may avail the opportunity for streaming the audio/video data between two remotely located peers over the Internet. In our application, Windows Media Player 6.0 or higher can be used as the default.

The Streaming Process

At the start up of the streaming process, a number of valid audio/video sources are specified that serve as the reservoir for unformatted media data. If any of the selected sources is not eligible for the streaming process, an error message is displayed to the user, and the streaming process stops immediately. On the other hand, if valid media sources are selected, an environment is set-up by adding the various sources of audio/video data to form a Source Group, which is subsequently added to a Source Collection Group.

Before the encoding session starts, attributes such as display information, title, author name, and so on, are assigned. Although the encoding session can proceed without assigning these attributes, it is advisable to assign these attributes, as they offer a complete reference to the session at the receiving end. Once you are through with selecting the media sources and assigning the attributes, set the protocol and the port number over which data is to be broadcasted.

After setting all the associated properties, the encoding engine is prepared to begin encoding. The task of encoding is done through the Windows Media Encoder, which converts data into Windows Media-based format. This filtered data is written on a network stream, which is received by the user at the receiving end. The process of converting data into Windows Media Format continues until no more media data remains to be encoded (see Figure 6-1).
Figure 6-1: The flow of streaming

1. Start
2. A set of valid audio/video files are requested for streaming?
   - Yes: Set up the environment by adding a source group to the source group collection.
   - No:Displays an error message.
3. Set attributes and display information or attributes to be associated?
   - No: Set the protocol and port number to broadcast the data on.
   - Yes: Set all the properties and prepare the encoding engine to begin encoding.
4. More uncoded data left?
   - Yes: Convert the file to windows media format using windows media encoder.
   - No: Write the data to the network stream.
5. Stop
Windows Media Encoder SDK

Encoding and broadcasting can be done using either Window Media SDK or DirectShow. Of these, Windows Media SDK gives developers better control of media through the Windows Media Encoder SDK, which provides:

♦ APIs to encode a range of media types in Windows Media Format
♦ Deeper access to the processes of encoding and streaming over the network to give applications a higher degree of control over media streams in Window Media Format
♦ COM-based components to support unicast or multicast delivery of encoded media data
♦ The following codecs for capturing and delivering Windows Media files and streams:
  • The Windows Media Audio version 2 codec, for high-fidelity music and mixed audio
  • The Sipro ACELP.net codec, for voice applications
  • The Microsoft MPEG-4 version 3 codec, for music and video
♦ Support for receiving media inputs from microphones, video cameras, or stored media files to encode them into Window Media Format to either store as a local file or to broadcast

We have chosen Windows Media Encoder for Encoding and Broadcasting the media data. The tools required apart from Visual C++ 6.0 are listed in the following section.

Windows Media Encoder SDK for Visual C++ 6.0

DirectX 8.0 SDK for supporting nonWindows Media Formats, as Windows Media SDK uses some DirectShow component to render and play back nonWindows Media Formats (like rendering services for the Windows Media Player are provided by DirectShow through the audio renderer, video renderer, and Overlay Mixer filters).

Because the client interface has already been designed using C#, it is more rational to develop this extension in C# only, but the two SDKs listed previously work specifically with VC++6.0 and VB 6.0. We have decided on VC++6.0 as the development tool for this extension, as it lets developers understand the technology better and provides for deeper access and improved control over the various tasks involved in encoding and broadcasting media data over Internet.

This extension is developed as a COM component used by our C# application to broadcast media data over the Internet. The following sections present the code that accomplishes the task of encoding the data into Windows Media Format and then broadcasting it. Every code file is followed by an explanation that clarifies the details.

Uses of Streaming in P2P

Although the basic form of this application provides users, the facility to share any kind of file including media files, this is not the only facility users seek. At times users may want to play a part of media files before actually downloading the entire file. Besides, there is always the probability that users might want to play the song or video online without letting anything be written on the hard drives. This is the case when a user is using wireless mobile devices to listen to a media file or is watching online news that includes audio or video data. There are also cases in which users have to economize on storage capacity and cannot afford to download huge media files on their storage devices. Streaming presents the solution to all such storage restrictions.

The incorporation of streaming in this application enables developers to extend this application for many more commercially useful purposes such as live chat, which can be done by combining this extension with the chat extension described later. Also, users can maintain a radio station, which is possible because this extension gives developers access to media devices installed on computers. By virtue of this
extension, developers can use video cameras and attached audio-media devices to broadcast news or presentations over the Internet.

Design Considerations

The incorporation of this extension in our application demands a few changes both on the listener and on the browser side. Because the communication media has always been XML files, this part of the structure does not change. Only a new request type called **STREAMING** has to be introduced.

Listeners have been made aware of this new request type, whereas the browser has to be modified a bit to be able to make such requests. To serve both these purposes, the shared assembly **WorkingWithXML** has been changed to make it capable of creating such requests upon the browser's demand and of parsing these requests for the listener.

Because this extension has been built on Windows Encoder SDK for VC++ 6.0, which is not available for C# yet, a component named **BroadcastIt** has been written in VC++ 6.0, which has been embedded in the listener. The listener makes an instance of this component to cater to streaming requests made by browsers.

The entire code of this component has been given in this chapter module-wise, and every module has been explained immediately after its code. The changes made on the browser and listener and the request XML design used for this purpose also have been explained.

Before beginning to explore the code for the streaming component (Listing 6-1), let’s learn a few facts about that:

<table>
<thead>
<tr>
<th>Component Name</th>
<th>BroadcastIt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executable file</td>
<td>BroadcastDll.dll</td>
</tr>
<tr>
<td>Interface</td>
<td>IBroadcastIt</td>
</tr>
<tr>
<td>Declaration file</td>
<td>BroadcastDll.idl</td>
</tr>
<tr>
<td>Implementing class</td>
<td>CBroadcastIt (Interface implemented apart from IBroadcastIt, ISupportsErrorInfo)</td>
</tr>
</tbody>
</table>

**Listing 6-1: Declaration of CBroadcastIt class**

```c++
1 // BroadcastIt.h : Declaration of the CBroadcastIt
2
3 #ifndef __BROADCASTIT_H_
4 #define __BROADCASTIT_H_
5
6 #include "resource.h"       // main symbols
7 #include <atlctl.h>
8 #include "BroadcastDllCP.h"
9
10 //******************************************************************************
11 // CBroadcastIt
12 class ATL_NO_VTABLE CBroadcastIt :
13 public CComObjectRootEx<CComSingleThreadModel>,
```

© 2001 Dreamtech Software India Inc.
All rights reserved
public CStockPropImpl<CBroadcastIt, IBroadcastIt, &IID_IBroadcastIt, &LIBID_BROADCASTDLLLib>,
public CComControl<CBroadcastIt>,
public IPersistStreamInitImpl<CBroadcastIt>,
public IOleControlImpl<CBroadcastIt>,
public IOleObjectImpl<CBroadcastIt>,
public IOleInPlaceActiveObjectImpl<CBroadcastIt>,
public IViewObjectExImpl<CBroadcastIt>,
public IOleInPlaceObjectWindowlessImpl<CBroadcastIt>,
public ISupportErrorInfo,
public IConnectionPointContainerImpl<CBroadcastIt>,
public IPersistStorageImpl<CBroadcastIt>,
public IPropertyNotifySinkCP<CBroadcastIt>,
public CComCoClass<CBroadcastIt, &CLSID_BroadcastIt>,
public CProxy_IBroadcastItEvents< CBroadcastIt >,
public CProxy_IConfigureEvents< CBroadcastIt >
{
public:
CBroadcastIt()
{
  m_bWindowOnly = TRUE;
  bSession = FALSE;
  video = FALSE;
  audio = FALSE;
  shUseDevice = 0;
  shUseScript = 0;
  bEncoder = FALSE;
}

DECLARE_REGISTRY_RESOURCEID(IDR_BROADCASTIT)

DECLARE_PROTECT_FINAL_CONSTRUCT()

BEGIN_COM_MAP(CBroadcastIt)
  COM_INTERFACE_ENTRY(IBroadcastIt)
  COM_INTERFACE_ENTRY(IDispatch)
  COM_INTERFACE_ENTRY(IViewObjectEx)
  COM_INTERFACE_ENTRY(IViewObject2)
  COM_INTERFACE_ENTRY(IOleInPlaceObjectWindowless)
  COM_INTERFACE_ENTRY(IOleInPlaceObject)
  COM_INTERFACE_ENTRY2(IOleWindow, IOleInPlaceObjectWindowless)
  COM_INTERFACE_ENTRY(IOleInPlaceActiveObject)
  COM_INTERFACE_ENTRY(IOleControl)
  COM_INTERFACE_ENTRY(IOleObject)
  COM_INTERFACE_ENTRY(IPersistStreamInit)
  COM_INTERFACE_ENTRY2(IPersist, IPersistStreamInit)
  COM_INTERFACE_ENTRY(ISupportErrorInfo)
  COM_INTERFACE_ENTRY(IConnectionPointContainer)
  COM_INTERFACE_ENTRY(IQuickActivateImpl<CBroadcastIt>,
    &IID__IBroadcastItEvents, &LIBID_BROADCASTDLLLib),
  public IPropertyNotifySinkCP<CBroadcastIt>,
  public CComCoClass<CBroadcastIt, &CLSID_BroadcastIt>,
  public CProxy_IBroadcastItEvents< CBroadcastIt >,
  public CProxy_IConfigureEvents< CBroadcastIt >

  DECLARE_REGISTRY_RESOURCEID(IDR_BROADCASTIT)

  DECLARE_PROTECT_FINAL_CONSTRUCT()

  BEGIN_COM_MAP(CBroadcastIt)
    COM_INTERFACE_ENTRY(IBroadcastIt)
    COM_INTERFACE_ENTRY(IDispatch)
    COM_INTERFACE_ENTRY(IViewObjectEx)
    COM_INTERFACE_ENTRY(IViewObject2)
    COM_INTERFACE_ENTRY(IOleInPlaceObjectWindowless)
    COM_INTERFACE_ENTRY(IOleInPlaceObject)
    COM_INTERFACE_ENTRY2(IOleWindow, IOleInPlaceObjectWindowless)
    COM_INTERFACE_ENTRY(IOleInPlaceActiveObject)
    COM_INTERFACE_ENTRY(IOleControl)
    COM_INTERFACE_ENTRY(IOleObject)
    COM_INTERFACE_ENTRY(IPersistStreamInit)
    COM_INTERFACE_ENTRY2(IPersist, IPersistStreamInit)
    COM_INTERFACE_ENTRY(ISupportErrorInfo)
    COM_INTERFACE_ENTRY(IConnectionPointContainer)
Chapter 6: Streaming in the P2P Application

COM_INTERFACE_ENTRY(IQuickActivate)
COM_INTERFACE_ENTRY(IPersistStorage)
COM_INTERFACE_ENTRY(IDataObject)
COM_INTERFACE_ENTRY(IPROVIDECCLASSINFO)
COM_INTERFACE_ENTRY(IPROVIDECCLASSINFO2)
COM_INTERFACE_ENTRY_IMPL(IConnectionPointContainer)
END_COM_MAP()

BEGIN_PROP_MAP(CBroadcastIt)
PROP_DATA_ENTRY("_cx", m_sizeExtent.cx, VT_UI4)
PROP_DATA_ENTRY("_cy", m_sizeExtent.cy, VT_UI4)
PROP_PAGE(CLSID_Configure)
END_PROP_MAP()

BEGIN_CONNECTION_POINT_MAP(CBroadcastIt)
CONNECTION_POINT_ENTRY(IID_IPropertyNotifySink)
CONNECTION_POINT_ENTRY(DIID_IBroadcastItEvents)
CONNECTION_POINT_ENTRY(DIID_IConfigureEvents)
END_CONNECTION_POINT_MAP()

BEGIN_MSG_MAP(CBroadcastIt)
CHAIN_MSG_MAP(CComControl<CBroadcastIt>)
DEFAULT_REFLECTION_HANDLER()
END_MSG_MAP()

// ISupportsErrorInfo
STDMETHOD(InterfaceSupportsErrorInfo)(REFIID riid)
{
  static const IID* arr[] =
  {
&IID_IBroadcastIt,
  };
  for (int i=0; i<sizeof(arr)/sizeof(arr[0]); i++)
  {
    if (InlineIsEqualGUID(*arr[i], riid))
      return S_OK;
  }
  return S_FALSE;
}

// IViewObjectEx
DECLARE_VIEW_STATUS(VIEWSTATUS_SOLIDBKGND | VIEWSTATUS_OPAQUE)

// IBroadcastIt
public:
STDMETHOD(InitializeBroadcaster)();
STDMETHOD(get_UseScript)(/*[out, retval]*/ short *pVal);
STDMETHOD(put_UseScript)(/*[in]*/ short newVal);
STDMETHOD(PrepareToEncode)();
STDMETHOD(SendURL)(/*[in]*/ BSTR bstrURL);
STDMETHOD(SendScript)(/*[in]*/ BSTR bstrScript);
STDMETHOD(get_AudioDevices)(/*[out, retval]*/ VARIANT *pVal);
STDMETHOD(get_VideoMedia)(/*[out, retval]*/ BSTR *pVal);
STDMETHOD(put_VideoMedia)(/*[in]*/ BSTR newVal);
STDMETHOD(get_AudioMedia)(/*[out, retval]*/ BSTR *pVal);
STDMETHOD(put_AudioMedia)(/*[in]*/ BSTR newVal);
Chapter 6: Streaming in the P2P Application

124 STDMETHODCALLTYPE(Status)();
125 STDMETHODCALLTYPE(CloseSession)();
126 STDMETHODCALLTYPE(MakeSession)();
127 STDMETHODCALLTYPE(get_ProfilesList)(/*[out, retval]*/ VARIANT *pVal);
128 STDMETHODCALLTYPE(HeraldThisMessage)(/*[in]*/ BSTR bstr);
129 STDMETHODCALLTYPE(get_Port)(/*[out, retval]*/ short *pVal);
130 STDMETHODCALLTYPE(put_Port)(/*[in]*/ short newVal);
131 STDMETHODCALLTYPE(get_Profile)(/*[out, retval]*/ short *pVal);
132 STDMETHODCALLTYPE(put_Profile)(/*[in]*/ short newVal);
133 STDMETHODCALLTYPE(Broadcast)();
134
135 HRESULT OnDraw(ATL_DRAWINFO& di)
136 {
137  RECT& rc = *(RECT*)di.prcBounds;
138  Rectangle(di.hdcDraw, rc.left, rc.top, rc.right, rc.bottom);
139  SetTextAlign(di.hdcDraw, TA_CENTER|TA_BASELINE);
140  LPCTSTR pszText = _T("BroadcastIt Dll © Dreamtech Softwares Inc., India
Developed By Ankur Verma");
141  TextOut(di.hdcDraw,
142    (rc.left + rc.right) / 2,
143    (rc.top + rc.bottom) / 2,
144    pszText,
145    lstrlen(pszText));
146  return S_OK;
147 }
148
149 short shUseDevice;
150 short shPortNo;
151 short shProfile;
152 short shUseScript;
153
154 bool video,audio,bEncoder;
155 CComBSTR bstrAudioMedia;
156 CComBSTR bstrVideoMedia;
157
158 IWMEncoder* pEncoder;
159 IWMEncSourceGroupCollection* pSrcGrpColl;
160 IWMEncSourceGroup* pSrcGrp;
161
162 BOOL bSession;
163
164 short m_nAppearance;
165 OLE_COLOR m_clrBackColor;
166 LONG m_nBackStyle;
167 OLE_COLOR m_clrBorderColor;
168 LONG m_nBorderStyle;
169 BOOL m_bBorderVisible;
170 LONG m_nBorderWidth;
171 CComBSTR m_bstrCaption;
172 LONG m_nDrawMode;
173 LONG m_nDrawStyle;
174 LONG m_nDrawWidth;
175 BOOL m_bEnabled;
176 OLE_COLOR m_clrFillColor;
Chapter 6: Streaming in the P2P Application 287

```c
179  LONG m_nFillStyle;
180  CComPtr<IFontDisp> m_pFont;
181  OLE_COLOR m_clrForeColor;
182  CComPtr<IPictureDisp> m_pMouseIcon;
183  LONG m_nMousePointer;
184  CComPtr<IPictureDisp> m_pPicture;
185  BOOL m_bTabStop;
186  CComBSTR m_bstrText;
187  BOOL m_bValid;
188 }
189 #endif //__BROADCASTIT_H_
190 #endif //__BROADCASTIT_H_
```

**Code description (CBroadcastIt class declaration)**

- Lines 1-34: CBroadcastIt class declaration. This class inherits itself from IDispatch interface, apart from others like ISupportErrorInfo that makes it more powerful.
- Lines 35-46: Constructor of the class initializing some variables.
- Lines 47-74: Com map specifying the interfaces supported by this component.
- Lines 75-80: Property map defining the properties and property pages supported by this component.
- Lines 81-86: Connection point map specifying the sink interface IDs through CONNECTION_POINT_ENTRY macro.
- Lines 88-106: Implementation of InterfaceSupportsErrorInfo function that the clients call to determine whether this component supports ISupportInfo or not.
- Lines 108-134: Method declaration for this component class.
- Lines 135-149: Painting tasks are handled through this function.
- Lines 150-158: Variable declarations to carry the property values.
- Lines 159-162: Interfaces used for accessing the capabilities of the encoder component.
- Line 164: Local variable to keep track of when the session is active.

A brief description of the functions exposed by this component follows:

- **MakeSession**: This function is called to make a session for encoding and broadcasting the specified media sources, in addition to associating proper attributes and displaying information to this session.
- **Broadcast**: This function is called to start the encoding session after the required attributes and media sources have been specified.
- **CloseSession**: This function is called to cease the session after realizing all the occupied resources and releasing the media sources being used.
- **Status**: This is an optional function and is not required to be called to get the encoder to initialize and begin encoding. But it can be used to get the statistics of running the encoding session.
- **SendScript**: This function can be called only if script-media source is supported in a given session to send actual messages along with audio/video media data.
- **SendURL**: Like SendScript, this function works only if a script-media source has been supported by the given encoding session; this function is used to send companion URLs that are opened in default browsers as soon as they are received on the other end.
PrepareToEncode: Having specified the attributes of audio and video-script media sources along with the port and the protocol to broadcast the media data on, this function is called for assigning all these properties to the encoding session.

InitializeBroadcaster: This function is called to initialize the encoder component to carry out any kind of media encoding and broadcasting.

put_Profile: This function is called to specify the index of the profile in the given list of available profiles for this session.

put_Port: This function is called to specify where to broadcast the data.

put_AudioMedia: This function is called to specify an audio-media source for this encoding session.

put_VideoMedia: This function is called to specify a video-media source for this encoding session.

put_UseScript: This function is called to instruct the encoding session to support script media. Script media enables the users to sent site links to the receiver end (that is, the receiver of the broadcast). These site links are opened at the receiver end as soon as they are received. This feature facilitates organization of presentation sessions which may include Web sites as well as voice and video media.

get_Profile, get_Port, get_AudioMedia, get_VideoMedia, get_UseScript: These are used to retrieve the values of the properties set previously.

get_AudioDevices: Before you decide whether to use an archive audio file or an audio device installed on your computer for audio input for this encoding session, you need to have the list of all the audio devices installed on your computer. This list can be obtained by calling this function.

get_ProfilesList: Based on the audio and video-script media you have chosen for this encoding session, the profiles available can be obtained by calling this function.

Now that we are through with the declarations of the class that implements the IBroadcastIt interface, let’s proceed to the code for implementation.

Listing 6-2 shows the implementation file for this class.

**Listing 6-2: Implementation for CBroadcastIt class**

Implementation file BroadcastIt.cpp

© 2001 Dreamtech Software India Inc.
All rights reserved

```
1 // BroadcastIt.cpp : Implementation of CBroadcastIt
2
3 #include "stdafx.h"
4 #include "BroadcastDll.h"
5 #include "BroadcastIt.h"
6 #include "Events.h"
7 #include <comdef.h>
8
9 // CBroadcastIt
10 11 STDMETHODIMP CBroadcastIt::HeraldThisMessage(BSTR bstr)
12 13 { Fire_BroadcastStatus(bstr);
14 15 return S_OK;
16 ```
STDMETHODIMP CBroadcastIt::MakeSession()
{
HRESULT hr;
if(!bEncoder)
return S_FALSE;
if(bSession)
return S_FALSE;
bSession = TRUE;

CComPtr<IWMEncAttributes> pAttr;
CComBSTR m_bstrName;
long cnt;
CComVariant m_varValue;
CComVariant m_varIndex;
VARIANT_BOOL* vbAutoStop = 0;
IWMEncDisplayInfo* pDispInfo;
CComBSTR m_bstrAuthor("Ankur Verma.");
CComBSTR m_bstrCopyright("© 2000 Dreamtech Softwares Inc., India.");
CComBSTR m_bstrDescription("Media Data.");
CComBSTR m_bstrRating("");
CComBSTR m_bstrTitle("The Dreamtech P2P Streaming Demo");

hr = pEncoder->get_Attributes(&pAttr);

_bstr_t bName[] =
{
_bstr_t("Title: "),
_bstr_t("Author: "),
_bstr_t("Copyright: "),
_bstr_t("Date Created: "),
_bstr_t("Time Created: "),
_bstr_t("File Content: ")
};

SYSTEMTIME sysTime;
GetLocalTime(&sysTime);
TCHAR SysDate[20];
TCHAR SysTime[20];

wsprintf(SysTime, "%d : %d : %d", sysTime.wHour, sysTime.wMinute, sysTime.wSecond);
wsprintf(SysDate, "%d : %d : %d", sysTime.wDay, sysTime.wMonth, sysTime.wYear);

USES_CONVERSION;
Fire_BroadcastStatus(T2OLE(SysTime));
Fire_BroadcastStatus(T2OLE(SysDate));

_variant_t vVal[] =
{
_variant_t("The P2P Media Streaming Demo"),
for(int i=0; i<6; i++)
{
    pAttr->Add(bName[i], vVal[i]);
}

for (i=0; i<cnt; i++)
{
    pAttr->Item(i, &m_bstrName, &m_varValue);
    wsprintf(dh, "%s : %s", OLE2T(m_bstrName), OLE2T(m_varValue.bstrVal));
    Fire_BroadcastStatus(T2OLE(dh));
    pEncoder->put_AutoStop(VARIANT_TRUE);
    pEncoder->get_AutoIndex(&vbAutoStop);
    hr = pDispInfo->put_Author(m_bstrAuthor);
    hr = pDispInfo->put_Copyright(m_bstrCopyright);
    hr = pDispInfo->put_Description(m_bstrDescription);
    hr = pDispInfo->put_Rating(m_bstrRating);
    hr = pDispInfo->put_Title(m_bstrTitle);
    if(shUseScript == 1)
    {
        CComPtr<IWMEncSource> pScriptSrc;
        hr = pSrcGrp->AddSource(WMENC_SCRIPT, &pScriptSrc);
        CComBSTR bstrScript(L"UserScript://");
        hr = pScriptSrc->SetInput(bstrScript);
    }
    IWMEncSource* pAudSrc;
    IWMEncSource* pVidSrc;
    if(audio)
    {
        hr = pSrcGrp->AddSource(WMENC_AUDIO, &pAudSrc);
        hr = pSrcGrp->AddSource(WMENC_VIDEO, &pVidSrc);
        if(FAILED(hr))
        {
            Fire_BroadcastStatus(L"Specifing Audio Source Failed");
            return S_FALSE;
        }
    }
Chapter 6: Streaming in the P2P Application

127  
128  hr = pAudSrc->SetInput(bstrAudioMedia.Copy());
129  if(FAILED(hr))
130  {
131      Fire_BroadcastStatus(L"Setting Audio Media Source Failed");
132      return S_FALSE;
133  }
134  Fire_BroadcastStatus(bstrAudioMedia.Copy());
135 
136
137
138  if(video)
139  {
140    hr = pSrcGrp->AddSource(WMENC_VIDEO, &pVidSrc);
141    if(FAILED(hr))
142    {
143      Fire_BroadcastStatus(L"Specifying Video Source Failed");
144      return S_FALSE;
145    }
146    hr = pVidSrc->SetInput(bstrVideoMedia.Copy());
147    if(FAILED(hr))
148    {
149      Fire_BroadcastStatus(L"Setting Video Media Source Failed");
150      return S_FALSE;
151    }
152    Fire_BroadcastStatus(bstrVideoMedia.Copy());
153  }
154  return S_OK;
155
156
157
158 STDMETHODIMP CBroadcastIt::Broadcast()  
159  {
160    if(!bEncoder)
161    return S_FALSE;
162 
163    HRESULT hr;
164
165    try
166    {
167      hr = pEncoder->Start();
168      if(FAILED(hr))
169      {
170        Fire_BroadcastStatus(L"Could not Start Encoder");
171        return S_FALSE;
172      }
173    }
174    catch(_com_error e)
175    {
176      Fire_BroadcastStatus(L"Error ............ ");
177      char no2[10];
178      itoa(e.Error(),no2,10);
179      Fire_BroadcastStatus(T2OLE(_T(no2)));
180      Fire_BroadcastStatus(T2OLE(e.ErrorMessage()));
181      Fire_BroadcastStatus(L"Error ............ ");
182  
183  
184  
185  
186  
187  
188  
189  
18
Chapter 6: Streaming in the P2P Application

183 }
184
185
186 return S_OK;
187 }
188
189
190 STDMETHODIMP CBroadcastIt::CloseSession()
191 {
192 if(!bEncoder)
193 return S_FALSE;
194
195 if(bSession)
196 {
197 HRESULT hr;
198
199 short shACount, shVCount, shSCount;
200 CComVariant varIndex;
201 varIndex.vt = VT_I2;
202 varIndex.iVal = 0;
203
204 hr = pSrcGrp->PrepareToEncode(VARIANT_FALSE);
205 if(FAILED(hr))
206     Fire_BroadcastStatus(L"Prepare encode Failed from Source Group");
207
208 hr = pSrcGrp->get_SourceCount(WMENC_AUDIO, &shACount);
209 if(FAILED(hr))
210     Fire_BroadcastStatus(L"Getting Audio Source count Failed");
211 else
212     if (shACount != 0)
213         hr = pSrcGrp->RemoveSource(WMENC_AUDIO, varIndex);
214
215 hr = pSrcGrp->get_SourceCount(WMENC_VIDEO, &shVCount);
216 if(FAILED(hr))
217     Fire_BroadcastStatus(L"Getting Video Source count Failed");
218 else
219     if (shVCount != 0)
220         hr = pSrcGrp->RemoveSource(WMENC_VIDEO, varIndex);
221
222 hr = pSrcGrp->get_SourceCount(WMENC_VIDEO, &shSCount);
223 if(FAILED(hr))
224     Fire_BroadcastStatus(L"Getting Script Source count Failed");
225 else
226     if (shSCount != 0)
227         hr = pSrcGrp->RemoveSource(WMENC_SCRIPT, varIndex);
228
229 hr = pSrcGrpColl->Remove(varIndex);
230 if(FAILED(hr))
231     Fire_BroadcastStatus(L"Remove Source Group Failed");
232
233 hr = pEncoder->PrepareToEncode(VARIANT_FALSE);
234 if(FAILED(hr))
235     Fire_BroadcastStatus(L"Prepare encode Failed");
hr = pEncoder->Stop();
if(FAILED(hr))
    Fire_BroadcastStatus(L"Could not Stop");

pEncoder->Release();
audio = FALSE;
video = FALSE;
bEncoder = FALSE;
}
Fire_BroadcastStatus(L"Session Closed");
bSession = false;
return S_OK;
}

STDMETHODIMP CBroadcastIt::Status()
{
    HRESULT hr;
    IWMEncStatistics* pStatistics;
    IWMEncOutputStats* pOutputStats;
    IDispatch* pDispOutputStats;

    short iStreamCount;
    long lAvgBitrate, lAvgSampleRate; long lCurrentBitRate,
    lCurrentSampleRate; long lExpectedBitRate,
    lExpectedSampleRate; CURRENCY qwByteCount, qwSampleCount; CURRENCY
    qwDroppedByteCount, qwDroppedSampleCount;

    // Initialize the COM library and retrieve a pointer
    // to an IWMEncoder interface.
    hr = CoInitialize(NULL);
    CoCreateInstance(CLSID_WMEncoder,
        NULL,
        CLSCTX_INPROC_SERVER,
        IID_IWMencoder,
        (void**)&pEncoder);

    // Retrieve an IWMEncStatistics interface pointer.
    hr = pEncoder->get_Statistics(&pStatistics);

    // Retrieve the number of multiple bit rate output streams.
    hr = pStatistics->get_StreamOutputCount(WMENC_VIDEO,
        0,
        &iStreamCount);

    // Retrieve an IDispatch pointer for the IWMEncOutputStats
    // interface.
    hr = pStatistics->get_StreamOutputStats(WMENC_VIDEO,
        0,
        0,
        &pDispOutputStats);
292 &pDispOutputStats);
293
294 // Call QueryInterface for the IWMEncNetConnectionStats
295 // interface pointer.
296
297 hr = pDispOutputStats->QueryInterface(IID_IWMEncOutputStats,
298 (void**)&pOutputStats);
299
300 // Manually configure the encoder engine or load
301 // a configuration from a file. For an example, see the
302 // IWMEncFile object.
303
304 // You can create a timer to retrieve the statistics
305 // after you start the encoder engine.
306
307 hr = pOutputStats->get_AverageBitrate(&lAvgBitrate);
308 hr = pOutputStats->get_AverageSampleRate(&lAvgSampleRate);
309 hr = pOutputStats->get_ByteCount(&qwByteCount);
310 hr = pOutputStats->get_CurrentBitrate(&lCurrentBitRate);
311 hr = pOutputStats->get_CurrentSampleRate(&lCurrentSampleRate);
312 hr = pOutputStats->get_DroppedByteCount(&qwDroppedByteCount);
313 hr = pOutputStats->get_DroppedSampleCount(&qwDroppedSampleCount);
314 hr = pOutputStats->get_ExpectedBitrate(&lExpectedBitRate);
315 hr = pOutputStats->get_ExpectedSampleRate(&lExpectedSampleRate);
316 hr = pOutputStats->get_SampleCount(&qwSampleCount);
317
318 CURRENCY  TimeElpsed;
319 pStatistics->get_EncodingTime(&TimeElpsed);
320 Fire_EncoderStatus(lAvgBitrate,lAvgSampleRate,lCurrentBitRate,
321 lCurrentSampleRate, lExpectedBitRate,
322 lExpectedSampleRate,qwByteCount,qwDroppedByteCount,
323 qwDroppedSampleCount,qwSampleCount
324 ,TimeElpsed);
325
326 return S_OK;
327 }
328
329
330
331 STDMETHODIMP CBroadcastIt::SendScript(BSTR bstrScript)
332 {
333     if(!bSession)
334         return S_FALSE;
335     CComBSTR bstrType(L"TEXT");
336     CComBSTR bstrData(bstrScript);
337     pEncoder->SendScript(0, bstrType, bstrData);
338     return S_OK;
339 }
340
341 STDMETHODIMP CBroadcastIt::SendURL(BSTR bstrURL)
342 {
343     if(!bSession)
344         return S_FALSE;
345     CComBSTR bstrType(L"URL");
Chapter 6: Streaming in the P2P Application

```cpp
347     CComBSTR bstrData(bstrURL);
348     pEncoder->SendScript(0, bstrType, bstrData);
349     return S_OK;
350 
351    
352    STDMETHODIMP CBroadcastIt::PrepareToEncode()
353    {
354       if(!bEncoder)
355          return S_FALSE;
356       if(!bSession)
357          return S_FALSE;
358        
359       USES_CONVERSION;
360       HRESULT hr;
361       
362       IWMEncProfileCollection* pProColl;
363       IWMEncProfile* pPro;
364       long lCount;
365       
366       hr = pEncoder->get_ProfileCollection(&pProColl);
367       if(FAILED(hr))
368          { 
369             Fire_BroadcastStatus(L"Get Profile Collection Failed");
370             return S_FALSE;
371          }
372       
373       CComVariant m_varProfile;
374       m_varProfile.vt = VT_DISPATCH;
375       
376       hr = pProColl->get_Count(&lCount);
377       if(FAILED(hr))
378          { 
379             Fire_BroadcastStatus(L"Get Profile Count Failed");
380             return S_FALSE;
381          }
382       
383       CComBSTR m_bstrProfName;
384       if(shProfile<0)
385          { 
386             Fire_BroadcastStatus(L"Profile Name ");
387          }
388       
389       hr = pProColl->Item(shProfile, &pPro);
390       if(FAILED(hr))
391          { 
392             Fire_BroadcastStatus(L"Get Profile Failed");
393             return S_FALSE;
394          }
395       
396       hr = pPro->get_Name(&m_bstrProfName);
397       
398       hr = pPro->get_Name(&m_bstrProfName);
```
if(FAILED(hr))
{
    Fire_BroadcastStatus(L"Get Profile Name Failed");
    return S_FALSE;
}
Fire_BroadcastStatus(m_bstrProfName);

m_varProfile.pdispVal = pPro;
hr = pSrcGrp->put_Profile(m_varProfile);
if(FAILED(hr))
{
    Fire_BroadcastStatus(L"Set Profile Failed");
    return S_FALSE;
}

IWMEncBroadcast* pBrdcst; long PortNum;
hr = pEncoder->get_Broadcast(&pBrdcst);
if(FAILED(hr))
{
    Fire_BroadcastStatus(L"Problem calling get_Broadcast");
    return S_FALSE;
}

hr = pBrdcst->get_PortNumber(WMENC_PROTOCOL_HTTP, &PortNum);
if(FAILED(hr))
{
    Fire_BroadcastStatus(L"Problem calling get_PortNumber");
    return S_FALSE;
}

PortNum = shPortNo;
hr = pBrdcst->put_PortNumber(WMENC_PROTOCOL_HTTP, PortNum);
if(FAILED(hr))
{
    Fire_BroadcastStatus(L"Problem calling put_PortNumber");
    return S_FALSE;
}

char number[10];
itoa(shProfile,number,10);
Fire_BroadcastStatus(T2OLE(_T(number)));

char number1[10];
itoa(shPortnumber,number1,10);
Fire_BroadcastStatus(T2OLE(_T(number1)));

hr = pEncoder->PrepareToEncode(VARIANT_TRUE);
if(FAILED(hr))
{
    Fire_BroadcastStatus(L"Problem calling PrepareToEncode");
    return S_FALSE;
}
return S_OK;
STDMETHODIMP CBroadcastIt::InitializeBroadcaster()
{
    HRESULT hr = CoCreateInstance(CLSID_WMEncoder,
    NULL,
    CLSCTX_INPROC_SERVER,
    IID_IWMEncoder,
    (void**)&pEncoder);

    if(FAILED(hr))
    {
        Fire_BroadcastStatus(L"Encoder Could not be Initialized");
        return S_FALSE;
    }

    hr = pEncoder->get_SourceGroupCollection(&pSrcGrpColl);
    if(FAILED(hr))
    {
        Fire_BroadcastStatus(L"Source Group collection retrieval Failed");
        return S_FALSE;
    }

    hr = pSrcGrpColl->Add(L"SG_2", &pSrcGrp);
    if(FAILED(hr))
    {
        Fire_BroadcastStatus(L"Source Group could not be added");
        return S_FALSE;
    }

    bEncoder = TRUE;
    Fire_BroadcastStatus(L"Encoder Initialized");
    return S_OK;
}

/****************** Properties ******************/

// Want to have a profile

STDMETHODIMP CBroadcastIt::put_Profile(short newVal)
{
    USES_CONVERSION;
    shProfile = newVal;
    char number[10];
    itoa(shProfile,number,10);
    HeraldThisMessage(T2OLE(_T(number)));
    return S_OK;
}

// Want to have a port

STDMETHODIMP CBroadcastIt::put_Port(short newVal)
{
Chapter 6: Streaming in the P2P Application

515 USES_CONVERSION;
516 shPortNo = newVal;
517 char number[10];
518 itoa(shPortNo, number, 10);
519 HeraldThisMessage(T2OLE(_T(number)));
520 return S_OK;
521 }
522
523 STDMETHODIMP CBroadcastIt::put_AudioMedia(BSTR newVal)
524 {
525 bstrAudioMedia = newVal;
526 HeraldThisMessage(bstrAudioMedia);
527 audio = TRUE;
528 return S_OK; 529 }
530
531 STDMETHODIMP CBroadcastIt::put_VideoMedia(BSTR newVal)
532 {
533 bstrVideoMedia = newVal;
534 HeraldThisMessage(bstrVideoMedia);
535 video = TRUE;
536 return S_OK;
537 }
538
539 STDMETHODIMP CBroadcastIt::put_UseScript(short newVal)
540 {
541 if((newVal == 0) || newVal == 1)
542 shUseScript = newVal;
543 else
544 Fire_BroadcastStatus(L"Use Script = 1 or Don’t use Script = 0");
545 return S_OK;
546 }
547
548 /*************************** Get Properties ****************************/
549
550 STDMETHODIMP CBroadcastIt::get_Profile(short *pVal) { *pVal = shProfile; return S_OK;}
551 STDMETHODIMP CBroadcastIt::get_Port(short *pVal) { *pVal = shPortNo; return S_OK;}
552 STDMETHODIMP CBroadcastIt::get_AudioMedia(BSTR *pVal) { *pVal = bstrAudioMedia.Copy(); return S_OK;}
553 STDMETHODIMP CBroadcastIt::get_VideoMedia(BSTR *pVal) { *pVal = bstrVideoMedia.Copy(); return S_OK;}
554 STDMETHODIMP CBroadcastIt::get_UseScript(short *pVal) { *pVal = shUseScript; return S_OK;}
555 STDMETHODIMP CBroadcastIt::get_AudioDevices(VARIANT *pVal)
556 { 557
HRESULT hr;
IWMEncSourcePluginInfoManager* pSrcPlugMgr;
IWMEncPluginInfo* pPlugInfo;

int j,i;
long lPlugCount, lResCount;
VARIANT_BOOL bResources;

hr = pEncoder->get_SourcePluginInfoManager(&pSrcPlugMgr);
hr = pSrcPlugMgr->get_Count(&lPlugCount);

for (i=0; i<lPlugCount; i++)
{
    hr = pSrcPlugMgr->Item(i, &pPlugInfo);
    CComBSTR bstrScheme;
    hr = pPlugInfo->get_SchemeType(&bstrScheme);

    if (_wcsicmp(bstrScheme, L"DEVICE") == 0 || _wcsicmp(bstrScheme, L"UserScript") == 0)
    {
        hr = pPlugInfo->get_Resources(&bResources);
        if (bResources == VARIANT_TRUE)
        {
            hr = pPlugInfo->get_Count(&lResCount);

            VariantInit(pVal);
pVal->vt = VT_ARRAY | VT_BSTR;
SAFEARRAY *pTheArray;
SAFEARRAYBOUND pBounds = {lResCount, 0};
pTheArray = SafeArrayCreate(VT_BSTR, 1, &pBounds);
bstrArray = (BSTR *)SafeArrayAccessData(pTheArray, reinterpret_cast<void**>(&bstrArray));

for (j=0; j<lResCount; j++)
{
    CComBSTR bstrResource;
    hr = pPlugInfo->Item(j, &bstrResource);
bstrArray[j] = bstrResource.Copy();
    //MessageBox(OLE2T(bstrResource),"adf",MB_OK);
}
SafeArrayUnaccessData(pTheArray);
pVal->parray = pTheArray;
break;
}
return S_OK;
}

**** Get Available Profiles For This Media Types ****/
STDMETHODIMP CBroadcastIt::get_ProfilesList(VARIANT *pVal)
{
if(!bEncoder)
  return S_FALSE;
  
HRESULT hr;
IWMEncProfileCollection* pProColl;
IWMEncProfile* pPro;

hr = pEncoder->get_ProfileCollection(&pProColl);
if(FAILED(hr))
  {
Fire_BroadcastStatus(L"Retrieving Collection of Profiles Failed");
return S_FALSE;
  }

long lCount;
hr = pProColl->get_Count(&lCount);
if(FAILED(hr))
  {
Fire_BroadcastStatus(L"Retrieving Count of Profiles Failed");
return S_FALSE;
  }

VariantInit(pVal);
pVal->vt = VT_ARRAY | VT_BSTR;
SAFEARRAY *pTheArray;
SAFEARRAYBOUND pBounds = {lCount,0};
pTheArray = SafeArrayCreate(VT_BSTR,1,&pBounds);
BSTR *bstrArray;
SafeArrayAccessData(pTheArray,reinterpret_cast<void**>(&bstrArray));

char ch[100];
USES_CONVERSION;
for (int i=0; i<lCount; i++)
  {
  CComBSTR m_bstrName;
  hr = pProColl->Item(i, &pPro);
  hr = pPro->get_Name(&m_bstrName);
bstrArray[i] = m_bstrName.Copy();
  }

SafeArrayUnaccessData(pTheArray);
pVal->parray = pTheArray;
return S_OK;
}

**Code description (CBroadcastIt implementation)**
- Lines 1-7: This provides the necessary packages required for building this component.
- Lines 9-17: Fire_BroadcastStatus is being used here to invoke a user-defined event to send messages back to the user during different stages of encoding.
Lines 18-26: This checks that the encoder has been initialized successfully and that the session has already been created.

Line 27: A variable is initialized positively to signify that the session is being created currently.

Lines 29-45: Some local variables are declared to be fed as the attribute values of the encoder session. A pointer of type IWMEncAttributes is retrieved, calling the get_Attributes method on IWMEncoder's pointer.

Lines 46-55: A structure of bstr type is initialized with the titles of attributes to be associated with the encoding session.

Lines 56-63: Current system date and time are obtained calling GetLocalTime() API, passing in a reference of SYSTEMTIME object. The obtained date and time are formatted in a legible manner.

Lines 64-77: A structure of type variant is initialized to store the values for attributes assigned to the current encoding session.

Lines 78-82: This session is then assigned all these attributes as name-value pairs, running a loop and making as many iterations as the number of attributes to be assigned.

Lines 83-91: All the attributes assigned previously are signaled back to the user as a confirmation that all the attributes have been assigned successfully.

Lines 92-95: Encoder session is instructed to stop automatically as soon as the media source is completely encoded.

Lines 96-105: A pointer to the IWMEncDisplayInfo interface is obtained by invoking the get_DisplayInfo() method on IWMEncoder pointer initialized previously in the InitializeBroadcaster() function. This pointer is used to fill up the information for a session to be displayed on the receiver’s end. Values for different information types, such as author name, copyright information, and so on, are fed.

Lines 106-113: If the component is instructed to support scripts, a script-media stream is added to the encoder by calling the AddSource() method on IWMEncSourceGroup interface's pointer retrieved previously in the InitializeBroadcaster() method.

Lines 114-136: If an audio source has been specified, it is added to the media-source group calling AddSource() method on the IWMEncSourceGroup interface's pointer retrieved previously in InitializeBroadcaster() function.

Lines 137-154: If a video source has been specified, it is added to the media-source group calling AddSource() method on IWMEncSourceGroup interfaces' pointer retrieved previously in InitializeBroadcaster().

Lines 161-162: This confirms that the encoder has been initialized successfully.

Lines 169-174: Start method is called on IWMEncoder interface's pointer to start the encoding and broadcasting session.

Lines 175-184: Any error encountered is conveyed back to the user.

Lines 190-194: This makes sure that the encoder has been initialized successfully.

Line 195: This ensures that the session is to be closed down.

Line 204: Media Source Group is instructed to pack up by passing FALSE as the parameter while calling PrepareToEncode on it.

Lines 207-214: Media Source Group is queried for the number of audio-media sources associated with it, and those sources are released.

Lines 215-221: Media Source Group is queried for the number of video-media sources associated with it, and those sources are released.
♦ Lines 215-228: Media Source Group is queried for the number of script-media sources associated with it, and those sources are released.
♦ Lines 230-234: The media source $SG \rightarrow 2$ that was added to the media source group collection in the function $InitializeBroadcaster$ is then removed.
♦ Lines 235-238: Media Encoder is instructed to cease any active encoding session and to close any active source stream, calling $PrepareToEncode()$ with the only parameter it takes set as $FALSE$.
♦ Lines 239-241: Media Encoder is instructed to stop the current encoding session.
♦ Lines 242-246: Media Encoder component is released, and the variables are assigned negative values to signify that no encoding session is active.
♦ Lines 247-249: The user receives the message that the session has been closed.
♦ Lines 250-333: This part of the code gives users an idea of the usage of the interfaces $IWMEncStatistics$ and $IWMEncOutputStats$ which may be availed for getting the status of the ongoing encoding session.
♦ Lines 334-335: This makes sure a session is active.
♦ Line 338: SendScript is called on $IWMEncoder$ pointer to broadcast the textual message along with running media specifying the type of script $TEXT$.
♦ Lines 343-345: This makes sure a session is active.
♦ Line 348: SendScript is called on $IWMEncoder$ pointer to broadcast the textual message along with running media specifying the type of script $URL$, which opens the site associated with this URL on the receiver's end.
♦ Lines 355-360: This ensures that the encoder is initialized and that a session is active at the moment.
♦ Lines 371-377: $IWMEncoder$'s $get\_ProfileCollection()$ method is invoked to retrieve the pointer to $IWMEncProfileCollection$ interface. This interface contains the enumerated profiles available to this encoder session. Any error encountered while doing this is conveyed back to the user.
♦ Lines 378-387: $IWMEncProfileCollection$ interface's method $get\_Count$ is invoked to retrieve the number of profiles available for this encoding session. Any error encountered is conveyed back to the user.
♦ Lines 395-400: $IWMEncProfileCollection$'s $Item$ method is invoked to retrieve the pointer to a $IWMEncProfile$ type pointer that represents the profile at the indices specified as the first argument.
♦ Lines 401-407: $IWMEncProfile$'s $get\_Name$ method is invoked to get the name of the profile retrieved.
♦ Lines 408-416: This profile is associated to the Media Source Group.
♦ Lines 421-427: $IWMEncoder$'s $get\_Broadcast$ is invoked to retrieve a pointer of type $IWMEncBroadcast$, which manages the port number and the protocol used for broadcasting the encoded media data.
♦ Lines 428-433: The port already being used for broadcasting is retrieved calling the $get\_PortNumber()$ method on this interface pointer.
♦ Lines 437-442: An appropriate port number is set to broadcast the media data.
♦ Lines 452-457: $PrepareToEncode$ is called on $IWMEncoder$ to set all the properties set previously and to prepare the encoder engine for starting.
♦ Lines 463-476: Calling $CoCreateInstance$ with the following parameters: the Class ID of the Windows Media Encoder; aggregating interface (which is null in this case); constant that specifies whether the present component will run as in-process and the interface ID of the Windows
Media Encoder. In case Windows Media Encoder could not be initialized, an error message is passed to the user. If initialization is successful, the pointer of type IWMEncoder which is passed as the fifth parameter is allocated memory.

♦ Lines 477-483: Invoke the get_SourceGroupCollection method through the pointer of IWMEncoder type to get the collection of source groups in a pointer of type IWMEncSourceGroupCollection. Any error encountered while retrieving Source-Group collection is passed to the user.

♦ Lines 484-494: To the collection retrieved, a new source group is added, and the reference to it is attained in a pointer of type IWMEncSourceGroup. An error is passed back to the user if the Source Group cannot be loaded successfully.

♦ Lines 503-512: For receiving the index number of the profile that the user selects from the list of profiles, the index number is stored in the local variable shProfile. The user is then notified that the value has been stored successfully, by sending him a message along with the number that he had passed. Here it is imperative to mention that profile in this discussion means encoding profile that is comprised of the information regarding the use of available bandwidth for voice and video streaming.

♦ Lines 513-522: This specifies the port number for broadcasting the data. The new value passed is stored in the local variable shPortNo.

♦ Lines 523-530: This is called to specify the audio source for this encoding session. The new value passed is stored in the local variable bstrAudioMedia, and the variable is initialized to a positive value to signify that the encoding session has audio data in it.

♦ Lines 531-538: This is called to specify the video source for this encoding session. The new value passed is stored in the local variable bstrVideoMedia, and the variable will be initialized to a positive value to signify that encoding session has video data.

♦ Lines 539-547: This is called to mention whether this encoding session supports Script Media. Only 1 and 0 are acceptable; 1 specifying support for Script Media and 0 e specifying no support.

♦ Line 551: The profile number set by the user calling put_Profile can be retrieved by calling this function.

♦ Line 552: The port specified by the user can be retrieved by calling this function.

♦ Line 553: The audio source specified by the user calling put_AudioMedia function can be retrieved by calling this function.

♦ Line 554: Calling this function can retrieve the video source specified by the user calling the put_VideoMedia() function.

♦ Line 555: Whether this encoding session supports script media or not can be known by calling this function.

♦ Lines 556-571: Method get_SourcePluginInfoManager is invoked on the interface pointer of type IWMEncoder to get a pointer to the IWMEncSourcePluginInfoManager interface. The get_Count method is called to get the count of installed source plug-ins.

♦ Lines 572-577: For each source plug-in installed, the plug-in information is obtained by calling the Item method on this interface's pointer, which gives a pointer to the IWMEncPluginInfo interface to retrieve registry information of plug-ins.

♦ 578-594: Each registry plug-in information retrieved is checked for whether the scheme type is DEVICE. When a plug-in information of this type is found, it is checked if it supports any resource by calling get_Resources method on the IVMEncPluginInfo pointer. If such resources are supported, the number of resources supported by the plug-in is retrieved by calling the get_Count() method. A SAFEARRAY pointer is made with SAFEARRAYBOUND, specifying the number of elements equal to the number of resources supported.
Lines 595-611: The IWMEncPluginInfo interface's item method is invoked to iterate through the list of supported resources and is stored in a BSTR type array returned by calling SafeArrayAccessData() API. Then SafeArrayUnaccessData is called to unlock the array. The array is then stored in the outgoing parameter.

Lines 617-619: This makes sure the encoder has been initialized successfully.

Lines 620-631: IWMEncoder's get_ProfileCollection method is invoked to retrieve the pointer to the IWMEncProfileCollection interface. This interface contains the enumerated profiles that are available to this encoder session. Any error encountered while doing so is conveyed back to the user.

Lines 632-639: IWMEncProfileCollection interface's method getCount is invoked to retrieve the number of profiles available for this encoding session. Any error encountered is conveyed back to the user.

Lines 640-647: A SAFEARRAY pointer is declared and created specifying the bounds as the number of elements being equal to the number of profiles found for the current session. This array is locked for access calling SafeArrayAccessData().

Lines 648-659: The enumerator is iterated throughout, and the list names of profiles are stored in the array.

Line 660: The array is unlocked calling SafeArrayUnaccessData.

Line 661: The array is assigned to the outgoing parameter.

The Windows Media Encoder we are using here to encode media data supports connection points you can use to your advantage to get the status of the active-encoding session at various stages. For this, implement the _IWMEncoderEvent interface. This interface is implemented by the CEvents class that follows and as shown in Listing 6-3:

Implementing class: CEvents (Interface implemented: _IWMEncoderEvent)

Listing 6-3: Declaration of CEvents class

Declaration and implementation file: Events.h

© 2001 Dreamtech Software India Inc.
All rights reserved

```c
1 #ifndef __EVENT_H__
2 #define __EVENT_H__
3
4 #include <atlbase.h>
5 #include <atlcmt.h>
6
7 #include "wmencode.h"
8 #include "wmsencid.h"
9 #include "BroadcastIt.h"
10
11 #define EVENT_ID        100
12 #define LIB_VERMAJOR    1
13 #define LIB_VERMINOR    0
14
15 class CEvents : public IDispEventImpl< EVENT_ID, 
16 CEvents, 
17 &DIID__IWMEncoderEvents, 
18 &LIBID_WMEncoderLib, 
19 LIB_VERMAJOR, 
20 LIB_VERMINOR >
```
Chapter 6: Streaming in the P2P Application

21 {
22
23 public:
24 int a;
25
26 static _ATL_FUNC_INFO StateChangeInfo;
27 static _ATL_FUNC_INFO ErrorInfo;
28 static _ATL_FUNC_INFO ArchiveStateChangeInfo;
29 static _ATL_FUNC_INFO ConfigChangeInfo;
30 static _ATL_FUNC_INFO ClientConnectInfo;
31 static _ATL_FUNC_INFO ClientDisconnectInfo;
32 static _ATL_FUNC_INFO SourceStateChangeInfo;
33 static _ATL_FUNC_INFO IndexerStateChangeInfo;
34
35 BEGIN_SINK_MAP(CEvents)
36 SINK_ENTRY_INFO( EVENT_ID,
37 DIID__IWMEncoderEvents,
38 DISPID_ENCODEREVENT_STATECHANGE,
39 OnStateChange, &StateChangeInfo )
40 SINK_ENTRY_INFO( EVENT_ID,
41 DIID__IWMEncoderEvents,
42 DISPID_ENCODEREVENT_ERROR,
43 OnError, &ErrorInfo )
44 SINK_ENTRY_INFO( EVENT_ID,
45 DIID__IWMEncoderEvents,
46 DISPID_ENCODEREVENT_ARCHIVESTATECHANGE,
47 OnArchiveStateChange, &ArchiveStateChangeInfo )
48 SINK_ENTRY_INFO( EVENT_ID,
49 DIID__IWMEncoderEvents,
50 DISPID_ENCODEREVENT_CONFIGCHANGE,
51 OnConfigChange, &ConfigChangeInfo )
52 SINK_ENTRY_INFO( EVENT_ID,
53 DIID__IWMEncoderEvents,
54 DISPID_ENCODEREVENT_CLIENTCONNECT,
55 OnClientConnect, &ClientConnectInfo )
56 SINK_ENTRY_INFO( EVENT_ID,
57 DIID__IWMEncoderEvents,
58 DISPID_ENCODEREVENT_CLIENTDISCONNECT,
59 OnClientDisconnect, &ClientDisconnectInfo )
60 SINK_ENTRY_INFO( EVENT_ID,
61 DIID__IWMEncoderEvents,
62 DISPID_ENCODEREVENT_SRCSSTATECHANGE,
63 OnSourceStateChange, &SourceStateChangeInfo )
64 SINK_ENTRY_INFO( EVENT_ID,
65 DIID__IWMEncoderEvents,
66 DIID__IWMEncoderEvents,
67 DISPID_ENCODEREVENT_INDEXERSTATECHANGE,
68 OnIndexerStateChange, &IndexerStateChangeInfo )
69 END_SINK_MAP()
306 Chapter 6: Streaming in the P2P Application

```cpp
public:

 STDMETHOD(OnStateChange)(/*[in]*/ WMENC_ENCODER_STATE enumState);
 STDMETHOD(OnSourceStateChange)(
     /*[in]*/WMENC_SOURCE_STATE enumState,
     /*[in]*/WMENC_SOURCE_TYPE enumType,
     /*[in]*/short iIndex,
     /*[in]*/BSTR bstrSourceGroup);
 STDMETHOD(OnError)(/*[in]*/ long hResult);
 STDMETHOD(OnArchiveStateChange)(
     /*[in]*/ WMENC_ARCHIVE_TYPE enumArchive,
     /*[in]*/ WMENC_ARCHIVE_STATE enumState );
 STDMETHOD(OnConfigChange)(/*[in]*/ long hResult, /*[in]*/ BSTR bstr);
 STDMETHOD(OnClientConnect)(
     /*[in]*/ WMENC_BROADCAST_PROTOCOL protocol,
     /*[in]*/ BSTR bstr);
 STDMETHOD(OnClientDisconnect)(
     /*[in]*/ WMENC_BROADCAST_PROTOCOL protocol,
     /*[in]*/ BSTR bstr);
 STDMETHOD(OnIndexerStateChange)(
     /*[in]*/ WMENC_INDEXER_STATE enumIndexerState,
     /*[in]*/ BSTR bstrFile );

 HRESULT CEvents::Init( IWMEncoder* pEncoder )
 {
     HRESULT hr = DispEventAdvise( pEncoder );
     if( FAILED( hr ) )
     {
         return hr;
     }

     HRESULT CEvents::ShutDown( IWMEncoder* pEncoder )
     {
         HRESULT hr = DispEventUnadvise( pEncoder );
         if( FAILED( hr ) )
         {
             return hr;
         }

         CEvents(CBroadcastIt* br);
     }

     CEvents::CEvents(CBroadcastIt* br);
     
     _ATL_FUNC_INFO CEvents::ArchiveStateChangeInfo= {CC_STDCALL,
     VT_ERROR, 2, { VT_I4, VT_I4 } };
```
133 ///////////////////////////////////////////////////
134 STDMETHODIMP CEvents::OnArchiveStateChange(
135     WMENC_ARCHIVE_TYPE enumArchive,
136     WMENC_ARCHIVE_STATE enumState )
137 {
138     switch ( enumArchive )
139     {
140         case WMENC_ARCHIVE_LOCAL:
141             broadcast->HeraldThisMessage(L"WMENC_ARCHIVE_LOCAL");
142             break;
143         default:
144             break;
145     }
146     switch ( enumState )
147     {
148         case WMENC_ARCHIVE_RUNNING:
149             // Process the case.
150             broadcast->HeraldThisMessage(L"WMENC_ARCHIVE_RUNNING");
151             break;
152         case WMENC_ARCHIVE_PAUSED:
153             // Process the case.
154             broadcast->HeraldThisMessage(L"WMENC_ARCHIVE_PAUSED");
155             break;
156         case WMENC_ARCHIVE_STOPPED:
157             // Process the case.
158             broadcast->HeraldThisMessage(L"WMENC_ARCHIVE_STOPPED");
159             break;
160         default:
161             broadcast->HeraldThisMessage(L"Archive");
162             break;
163     }
164     return E_NOTIMPL;
165 }
166
167 _ATL_FUNC_INFO CEvents::IndexerStateChangeInfo = {CC_STDCALL,
168     VT_ERROR, 2, { VT_I4, VT_BSTR } );
169 ///////////////////////////////////////////////////
170 STDMETHODIMP CEvents::OnIndexerStateChange(
171     WMENC_INDEXER_STATE enumIndexerState,
172     BSTR bstrFile )
173 {
174     broadcast->HeraldThisMessage(L"IndexerStateChange");
175     return E_NOTIMPL;
176 }
177
178 _ATL_FUNC_INFO CEvents::ClientDisconnectInfo = {CC_STDCALL,
Chapter 6: Streaming in the P2P Application

```cpp
VT_ERROR, 2, { VT_I4, VT_BSTR } );

STDMETHODIMP CEvents::OnClientDisconnect(
    WMENC_BROADCAST_PROTOCOL protocol,
    BSTR bstr)
{
    broadcast->HeraldThisMessage(L"Client Disconnected");
    return E_NOTIMPL;
}

_ATL_FUNC_INFO CEvents::ClientConnectInfo = {CC_STDCALL,
    VT_ERROR, 2, { VT_I4, VT_BSTR } );

STDMETHODIMP CEvents::OnClientConnect(
    WMENC_BROADCAST_PROTOCOL protocol,
    BSTR bstr)
{
    broadcast->HeraldThisMessage(L"Client Connected");
    return E_NOTIMPL;
}

_ATL_FUNC_INFO CEvents::ConfigChangeInfo
    = {CC_STDCALL, VT_ERROR, 2, { VT_I4, VT_BSTR } );

STDMETHODIMP CEvents::OnConfigChange(long hResult, BSTR bstr)
{
    broadcast->HeraldThisMessage(L"Configuration Changed");
    broadcast->HeraldThisMessage(bstr);
    return E_NOTIMPL;
}

_ATL_FUNC_INFO CEvents::ErrorInfo = {CC_STDCALL,
    VT_ERROR, 1, { VT_I4 } );

STDMETHODIMP CEvents::OnError(long hResult)
{
    broadcast->HeraldThisMessage(L"Started");
    return E_NOTIMPL;
}

_ATL_FUNC_INFO CEvents::SourceStateChangeInfo
    = {CC_STDCALL, VT_ERROR, 3, { VT_I4, VT_I4, VT_I2 } );
```
Chapter 6: Streaming in the P2P Application

245
246 STDMETHODIMP CEvents::OnSourceStateChange(
247     WMENC_SOURCE_STATE enumState,
248     WMENC_SOURCE_TYPE enumType,
249     short iIndex,BSTR i)
250 {
251     switch ( enumState )
252     {
253     case WMENC_SOURCE_START:
254         broadcast->HeraldThisMessage(L"Started");
255         break;
256     case WMENC_SOURCE_STOP:
257         broadcast->HeraldThisMessage(L"Stop");
258         break;
259     default:
260         broadcast->HeraldThisMessage(L"Encoding");
261         break;
262     }
263     switch ( enumType )
264     {
265     case WMENC_AUDIO:
266         broadcast->HeraldThisMessage(L"WMENC_AUDIO");
267         break;
268     case WMENC_VIDEO:
269         broadcast->HeraldThisMessage(L"WMENC_VIDEO");
270         break;
271     case WMENC_SCRIPT:
272         broadcast->HeraldThisMessage(L"WMENC_SCRIPT");
273         break;
274     default:
275         break;
276     }
277     return E_NOTIMPL;
278 }
279
280 _ATL_FUNC_INFO CEvents::StateChangeInfo = {CC_STDCALL,
281     VT_ERROR, 1, { VT_I4 } );
282
283 //////////////////////////////////////////////////////////////////////////////////
284
285 STDMETHODIMP CEvents::OnStateChange(WMENC_ENCODER_STATE enumState)
310 Chapter 6: Streaming in the P2P Application

```c
301 {
302 switch ( enumState )
303 {
304     case WMENC_ENCODER_STARTING:
305         broadcast->HeraldThisMessage(L"WMENC_ENCODER_STARTING");
306         break;
307     case WMENC_ENCODER_RUNNING:
308         broadcast->HeraldThisMessage(L"WMENC_ENCODER_RUNNING");
309         break;
310     case WMENC_ENCODER_PAUSED:
311         broadcast->HeraldThisMessage(L"WMENC_ENCODER_PAUSED");
312         break;
313     case WMENC_ENCODER_STOPPING:
314         broadcast->HeraldThisMessage(L"WMENC_ENCODER_STOPPING");
315         break;
316     case WMENC_ENCODER_STOPPED:
317         broadcast->HeraldThisMessage(L"WMENC_ENCODER_STOPPED");
318         break;
319     default:
320         break;
321     }
322 }
323 return E_NOTIMPL;
324 }///////////////////////////////////////////////
325 #endif // __EVENT_H__
326 #endif // __EVENT_H__
```

**Code description (CEvents declaration and implementation)**

- Lines 1-6: The ATL Support header file is included.
- Lines 7-8: The Encoder header file is included.
- Lines 15-22: Class CEvents inherited from IDispEventImpl implements events raised by the encoder.
- Lines 26-34: ATL_FUNC_INFO structure objects declarations for each event are described in the type library. These structures are filled later to define the calling conventions, the number of arguments, and the argument types with the return types.
- Lines 35-77: Sink Map defines and links all the functions declared in _IWMEncoderEvents interface to the function declared in this class. All the events fired by Encoder are routed through this map to invoke the implementations of event-handler functions.
- Lines 78-99: This part of code is a set of function declarations.
- Lines 100-107: This function helps Encoder, which is a source of events in this case, to establish a connection with the event sink (that is, the catcher of these events which implements all the events raised by the encoder which essentially means the implementation of the IWMEncoderEvents
Chapter 6: Streaming in the P2P Application 311

interface). Internally, this function calls DispEventAdvise, implemented in IDispEventImpl from which this class is derived, passing the IUnknown pointer to the source of events.

♦ Lines 108-116: This function helps Encoder close the connection with the event sink (that is, the implementation of IWMEncoderEvents interface). Once the connection is broken, events are no longer routed to the handler functions listed in the event Sink Map. Internally, this function calls DispEventUnadvise, implemented in IDispEventImpl from which this class is derived, passing the IUnknown pointer to the source of events.

♦ Lines 124-128: The Constructor receives here a pointer of CBroadcastIt type for later use when it will be needed to be set as the source of events.

♦ Lines 130-132: ArchiveStateChangeInfo object of ATL_FUNC_INFO type is filled to specify the calling convention, number of arguments, argument types, and return type of the handler for the ArchiveStateChange event.

♦ Lines 136-171: The OnArchiveStateChange receives the event number of notifications for changes while encoding the local (in most cases) media and for different stages such as stopped, running, and paused.

♦ Lines 173-176: The IndexerStateChangeInfo structure is filled with information such as calling conventions, argument types, the number of arguments, and the return type for the handler of IndexerStateChange event.

♦ Lines 177-185: This function gets indications when the indexing process begins or stops for a Windows Media file.

♦ Lines 186-189: The ClientDisconnectInfo structure is filled with information such as calling conventions, argument types, the number of arguments, and the return type for the handler of the ClientDisconnect event.

♦ Lines 190-197: As any client closes the connection, this handler is invoked to signal the event.

♦ Lines 198-200: ClientConnectInfo structure is filled with information such as calling conventions, argument types, the number of arguments, and the return type for the handler of the ClientConnect event.

♦ Lines 201-210: While broadcasting the media on a particular port, the connection made by any client is signaled by the Encoder by firing this event, thus calling this handler.

♦ Lines 212-214: The ConfigChangeInfo structure is filled with information such as calling conventions, argument types, the number of arguments, and the return type for the handler of the ConfigChange event.

♦ Lines 217-224: Any configuration change is notified invoking this event by the Encoder.

♦ Lines 227-228: The ErrorInfo structure is filled with information such as calling conventions, argument types, the number of arguments, and the return type for the handler of the Error event.

♦ Lines 232-238: Any Error encountered at any stage of the encoding session is signaled, invoking this event.

♦ Lines 241-242: The SourceStateChangeInfo structure is filled with information such as calling conventions, argument types, the number of arguments, and the return type for the handler of the SourceStateChange event.

♦ Lines 245-294: This handler is invoked to signal different states of the encoding engine, such as Stopped, Stopping, Start, Starting, and Running.

♦ Lines 295-296: The SourceStateChangeInfo structure is filled with information such as calling conventions, argument types, the number of arguments, and return type for the handler of the SourceStateChange event.

♦ Lines 300-333: This function is invoked by the encoder to signal when it starts or stops.
Changes on the Browser Side

Now that you are familiar with the changes made on the listener side of the application, we take up changing the browser module. Because every request is generated by the browser and sent to the listener, we have to request the browser for streaming. For this, some changes are needed on the browser side, such as adding a Stream button to facilitate streaming.

In Figure 6-2, you see the Stream button placed at the top, along with the other buttons. Now that you have placed the Stream button, you need the code that is invoked when you click this button.

```csharp
private void btnStream_Click(object sender, System.EventArgs e)
{
    // Declares a local Filename variable
    string Filename;

    // Extract the IP address from the COMPUTERNAME variable
    string IPAddress = COMPUTERNAME.Substring(COMPUTERNAME.IndexOf("\") + 1);
    IPAddress = IPAddress.Substring(0, IPAddress.Length - 1);

    // Get the currently selected item from the List view
    int index = GetSelectedItemAtListView(out Filename);
    try
    {
        // Checks whether any entry is selected or not
        if (-1 != index)
        {
            // Checks whether a filename is selected for Streaming
            // or not
            if (null != Filename)
            {
```
Chapter 6: Streaming in the P2P Application

```csharp
21    OpenConnection(COMPUTERNAME);
22    // Creates a stream request for a particular
23    // file to stream.
24    CreateRequest("STREAMING",strArray[index].sFilename,"");
25    // Sends this request to the listener
26    SendDataToListener(REQUESTFILE);
27    // Get the response from the listener in lieu of the
28    // above request
29    GetDataFromListener(RESPONSEFILE);
30    xmlParser = new WorkingWithXML.XMLParser();
31    xmlStruct = new XMLSTRUCT();
32    xmlParser.ParseXML(RESPONSEFILE,out xmlStruct,new
33    ServerCommunication().TypeOfXMLRecieved(RESPONSEFILE));
34    if( 0 == xmlStruct.ERROR.sDescription.CompareTo("No Error") )
35    {
36        frmStreamer StreamerForm = new frmStreamer(IPAddress);
37        StreamerForm.ShowDialog();
38    }
39    else throw new Exception(xmlStruct.ERROR.sDescription);
40    else throw new Exception("Cannot stream a folder");
41    // If nothing is selected then displays an error message
42    else throw new Exception("Nothing Selected");
43    }
44    // catches any system generated error message and displays it
45    // to the user
46    catch( Exception err ) {
47        MessageBox.Show(err.Message,"Warning",MessageBoxButtons.OK,
48        MessageBoxIcon.Warning); }
49    finally { new ServerCommunication().FileDelete(RESPONSEFILE); }
```

The preceding function simulates the Stream button click event. When the button is clicked while selecting a file for streaming, the IP address of the connected listener is extracted. The currently selected file is retrieved by calling the `GetSelectedItemFromListView()` function. The control is then passed to the `OpenConnection()` function, which opens the connection to the listener for sending the streaming request. The streaming request is created as follows:

```xml
<?xml version="1.0" encoding="utf-8"?>
<p2p_lng>
<request type="STREAMING">
<scope type="C:\WINNT\Media\ringout.WAV" mask=""/>
</request>
</p2p_lng>
```

The preceding request is then sent to the listener by calling the `SendDataToListener()` function and passing the response filename to it. After sending the request, the listener reads it and responds to the browser. This response is then read by calling the `GetDataFromListener()` function. Actually, this response is the `ERROR` response in which `No Error` is written in the description part if the listener
Chapter 6: Streaming in the P2P Application

successfully listens to the request. Otherwise, the error is returned. This XML is then parsed. This check is performed at line 38 in the preceding code. If the streaming request is successful, a new instance of the frmStreamer form is created and shown to the user. This window contains the Windows Media Player component to play the selected file via streaming.

For streaming, you have to add a new form and name it frmStreamer. Add a Windows Media Player component to it, as in Figure 6-3.

![Streaming Window](image)

Figure 6-3: The streaming interface for browser

In the default constructor of this form, write the following lines of code below the InitializeComponent():

```csharp
try { axMediaPlayer.FileName = "http://" + IPAddress + ":9090"; }
catch( Exception err ) { MessageBox.Show(err.Message,"Error",MessageBoxButtons.OK,MessageBoxIcon.Error); }

public frmStreamer(string IPAddress)
{
    // Required for Windows Form Designer support
    InitializeComponent();
    try { axMediaPlayer.FileName = "http://" + IPAddress + ":9090"; }
catch( Exception err ) { MessageBox.Show(err.Message,"Error",MessageBoxButtons.OK,MessageBoxIcon.Error); }
}
```
Chapter 6: Streaming in the P2P Application

Because streaming is performed over a network connection, exceptions are likely to happen on either side. To handle these exceptions, we need to catch them. The following lines of code represent the Error event of the Windows Media Player, which is automatically invoked in case of an exception. For example, if you are in the middle of a streaming session and the listener closes without notification, or some error occurs on the listener side that stops the listener’s streaming session, this event handler warns the user on the browser side and then closes the streaming window.

```csharp
private void axMediaPlayer_Error(object sender, System.EventArgs e)
{
    MessageBox.Show("Some error has been occured while streaming. Cannot continue further","Error",MessageBoxButtons.OK,MessageBoxIcon.Error);
    axMediaPlayer.Dispose();
    this.Dispose();
    this.Close();
}
```

After the streaming session is completed successfully and the user is through with the audio/video file, he or she has to close the streaming window to return to the browser window. At this stage, the following event handler is invoked, and it clears all the memory resources that are in use by calling each of the object's Dispose functions.

```csharp
private void frmStreamer_Closing(object sender, System.ComponentModel.CancelEventArgs e)
{
    axMediaPlayer.Dispose();
    this.Dispose();
}
```

### Changes on the Listener Side

As soon as the request sent by the browser is received at the listener’s end, the RequestResponse() function that determines the request types and responds to the browser takes control and makes an object of class StreamIt, which is written specifically to handle streaming request types.

```csharp
protected void RequestResponse()
{
    if(UploadDownloadPrint == 5) // this represents a streaming request
    {
        StreamIt streamDlg = new StreamIt(DnLoadFile,new StreamingCallBackDeligates(StreamingCallBack));
        intStreamingThreadIndex = ThreadCount;
        bDownLoading = false;
        return;
    }
}
```

The name of the file requested to be streamed is passed to the object of the StreamIt class, as the argument of its constructor. The rest of the job is done by this class.

```csharp
public StreamIt(string MediaFile,StreamingCallBackDeligates callback)
{
    
```
The constructor of the `StreamIt` class stores the filename passed and the `CallBack()` function reference to the respective local-member variables. The `CallBack()` function here is of type `StreamingCallBackDeligates`, which has the following signature:

```csharp
public delegate void StreamingCallBackDeligates(string e, StreamIt str);
```

Also, for our purposes, it has the following implementation:

```csharp
public void StreamingCallBack(string e, StreamIt source)
{
    Application.DoEvents();
    if (e.CompareTo("Stopped") == 0)
    {
        if (bStreaming)
            new XMLCreater(Application.StartupPath + "\Temp1.xml","" ).WriteErrorResponse("A streaming session is already active on this peer");
        else
        {
            new XMLCreater(Application.StartupPath + "\Temp1.xml","" ).WriteErrorResponse("Problem encountered while initiating the Streaming Session");
            bStreaming = false;
        }
    }
    else if(e.CompareTo("Encoder Stopped") == 0)
    bStreaming = false;
    else if(e.CompareTo("Copy Complete") == 0)
    {
    bStreaming = true;
    new XMLCreater(Application.StartupPath + "\Temp1.xml","" ).WriteErrorResponse("No Error");
    }
    else
    return;
}
```

The `StreamingCallBack()` function handles various states of the ongoing encoding session and responds accordingly. If the encoder session is found to have experienced a problem, the session is closed, and an error response with the message about the failure is created to be sent to the calling browser later. The information is passed by using the `StreamIt` classes, as explained shortly.

The constructor, after initializing the local variables, calls the `btnApply_Click()` function:

```csharp
private void btnApply_Click(object sender, System.EventArgs e)
{
    axBroadcastIt1.InitializeBroadcaster();
    axBroadcastIt1.Port = 9090;
}
```
Chapter 6: Streaming in the P2P Application

```csharp
axBroadcastIt1.AudioMedia = strMediaSource;
axBroadcastIt1.VideoMedia = strMediaSource;
axBroadcastIt1.MakeSession();
axBroadcastIt1.PrepareToEncode();
axBroadcastIt1.Broadcast();
}
```

This function initializes the encoder session calling `InitializeBroadcaster()` function on the object of the `BroadcastIt` COM class (explained in Listing 6-1) and equips it with information such as the port number and the file to stream over the Internet before it calls the `MakeSession()` function to make the encoder session. The `PrepareToEncode()` and `Broadcast()` functions are then called to start the session.

The code that follows defines the `BroadcastStatus()` function that is called by the `BroadcastIt`'s COM object. It is the event handler of `BroadcastStatus` event that the `BroadcastIt` component fires as and when it senses any change in the encoder session’s state. The calling of this function ensures that any problem faced during the encoding process leads to the disposal of all the resources being used and that the encoding session terminates gracefully. Any significant change in the encoder session is fired to the login form through the callback function `StreamingCallBack()` to enable it to handle the change accordingly, as described previously.

```csharp
protected int axBroadcastIt1_BroadcastStatus(object sender,
AxBROADCASTDLLLib._IBroadcastItEvents_BroadcastStatusEvent e)
{
    Application.DoEvents();
    if((e.status.ToString().CompareTo(
        "Second attempt of calling PrepareToEncode failed") == 0)
    || (e.status.ToString().CompareTo("Couldn't Start Encoder") == 0))
    {
        callbackFunc("Stopped", this);
        this.btnCloseSession_Click(null, null);
        this.Dispose();
        this.Close();
    }
    if((e.status.ToString().CompareTo("Encoder Stopped") == 0))
    {
        callbackFunc("Encoder Stopped", this);
        this.btnCloseSession_Click(null, null);
        this.Dispose();
        this.Close();
    }
    if((e.status.ToString().CompareTo("Copy Complete") == 0) ||
    (e.status.ToString().CompareTo("Started") == 0))
        callbackFunc("Copy Complete", this);
    return 1;
}
```

Running the Application

Now that we have seen the coding part needed to incorporate streaming in the current design of our P2P application, we are about to see the code in action. Compile and build the application as described in Microsoft Visual Studio.NET, and double-click the executable file that is generated. Upon executing this file successfully, you are able to see the first window, as shown in Figure 6-4.
Choose the appropriate media you want to play. In this case, we have shared the Media folder in the window’s directory (see Figure 6-5).

Select any media file from the window and click the **Stream** button. If the streaming initiates successfully, you see the text **Now Streaming. Please wait...** in the status bar of the browser window (see Figure 6-6).
Chapter 6: Streaming in the P2P Application

Figure 6-6: User gets the notification that the streaming process has begun.

You can stream any of the files that are compatible with Windows Media Player; just select that file and click the Stream button. You can create only one streaming session at a time. If some error is encountered while streaming, the application is able to handle that error and to display the error message to the user while stopping the streaming process.

Summary

So far, we have discussed a modest implementation, considering the immense potential of the streaming technology. This technology has virtually unlimited applications in the fast-developing networking environment of today. Now that all software has to work on a network of some kind or other, this technology is of prime significance. Handheld and other wireless mobile devices are so constrained by storage restrictions that carrying media files on them is a far cry from the present. The technology of streaming is endowed with the capacity to bring forth solutions to such problems that have surfaced, along with novel applications. The strength of this technology is that it obviates the need for storing data of any kind before one can actually start availing the data.

This is first time a concerted endeavor has been made to reveal the concept of P2P with the objective of making it of practical value to people. Every effort has been made to discuss all the viable and conceivable extensions to the basic structure of this concept, on which this concept actually stands and works. Streaming and broadcasting (multicast delivery of media data), no new concept to developers, has been tailored in the basic structure of the P2P concept to facilitate carrying this concept beyond conventions and exploiting it to its limits.

The scope for extension this technology harbors is endless. The best of our efforts have been asserted to implement the maximum extensions to this P2P concept and to effect its most pragmatic commercial usage in an open-source learning methodology. Now it is up to you to explore your skills and expertise, which you are bound to have enhanced by perusing this book.
Chapter 7

Add-Ons to the P2P Application

In this chapter, we discuss the enhancements that can be made to the P2P application. These enhancements provide additional features to the users of the application. Enhancements are the following:

♦ Chat client
♦ Globalized printing
♦ P2P Wireless search (for Java version only)

Since the application explained in this book has an open design, it offers opportunities we can try. Readers of this book are free to develop any of their own applications that fit into this model. The add-ons for this application are not part of the application itself and are not included in the accompanying CD-ROM. They are explained only with short snippets of code. The reader can use these code-snippets to build add-ons as needed. The add-ons discussed in this chapter serve as new ideas that can be implemented in the current design.

Chat Client

This P2P application allows its users to share documents. In addition, as described in the previous section, you can send hard copies to other peers by means of printing extensions. Because any communication between two peers begins once they are connected, a chat application can make additional use of this connection.

Chat client is software that allows you to exchange online text messages with other peers. It is commonly used for chatting with other peers over a network. Chatting applications can easily be downloaded for free from the Web sites of software giants such as Yahoo, AOL, and Microsoft. But implementing chatting in a P2P application is open to our readers; it is as efficient and reliable as any other chatting application besides it gives users an opportunity to extend the application’s usability to make it work on devices, to add advanced conferencing and presentation features in it and much more, which is possible as it is open source. By following the steps in the code snippets discussed later, you can design your own chat client.

Why is chat client implemented in our application? One simple answer is that we are equipping the application with all possible technologies and thereby extending its approach. But there are several other reasons closely linked with the advantages of chatting, and we can't ignore these:

♦ Chatting allows users to communicate with each other in real time over the network.
♦ Because messages are directly delivered, chat client saves a lot of time in sending and receiving messages compared with sending e-mails.
♦ Because messages can be conveyed to any remotely located peer, chat client is an ideal business tool for handling overseas business operations.

Requirements for implementing chat client in this P2P application are the following:

♦ **Source peer (IP address and name):** The user sending the message.
♦ **Destination peer (IP address and name):** To whom the message should be delivered.

♦ **Message to be delivered:** Contents of the message.

♦ **XML for chat message:** Fixed XML format in which the message is converted for delivering to the destination.

♦ **XML parser structure:** Values of the chatting message are parsed into this XML Structure; the user receives the values.

Because this application is designed to distribute the workload between two components (the listener and browser), the listener is assigned the responsibility of responding to all kind of requests coming to its way. The listener is responsible for handling multiple chat requests. The listener handles more than one chatting process by opening a window for each chatting request initiated by browsers running on remote peers.

Notice that browsers initiate chatting requests. This expandability takes advantage of the XML format that has been used throughout the development of this application and throughout the addition of extensions.

Establishing a chat session between two peers is depicted in Figure 7-1.

### XML Format for Messaging

This XML format has been used for sending and receiving chat messages. This is very much like the other request/response XML formats we have seen in this book. As mentioned earlier, with slight alteration in XML formats, we can add any number of new add-ons to this application in its existing design.

```xml
<p2p_lng>
  <request type="CHAT">
    <message sendername="SMITH" senderip="192.192.169.11" chatmsg="hello"/>
  </request>
</p2p_lng>
```

Here, the request type is "CHAT"; the other information helps the peer receiving this request discover the name and IP of the peer making the request and sending the message. From both sides, message is sent in XML formats only, making it possible for a similar module written in any contemporary language to send and receive messages through this application.

As the browser initiates chatting, it sends a chat request to the concerned listener. Upon receiving such a request, the listener opens a window to indicate that the request has been successfully processed to begin the chatting session.

```csharp
if(UploadDownloadPrint == 4) // this represents a chat request
{
    frmChat oChatFrm = new frmChat(ChatInfo,newSock,LoginName,
    Application.StartupPath + "\ChatLog.log",++ChatWindowCount);
    oChatFrm.ShowDialog();
}
```

This window is passed the information such as the name of the peer, who initiated the chat process, and the IP address and the count of chat windows (as there can be more than one chat window running simultaneously on the listener's side).
Chapter 7: Add-Ons to the P2P Application

Figure 7-1: Establishing a chat session between two peers

On the browser’s end, the user clicks the Chat button (see Figure 7-2) to initiate chatting with an already connected peer.
Chapter 7: Add-Ons to the P2P Application

Figure 7-2: Clicking this button initializes the chat session.

A chat window is opened (see Figure 7-3).

Figure 7-3: The Chat window

These lines are executed when the user clicks the Chat button on the Client's application window.

// © 2001 Dreamtech Software India Inc.
// All rights reserved.

protected void btnChat_Click (object sender, System.EventArgs e)
{
    frmChat chat = new frmChat(COMPUTERNAME);
    chat.ShowDialog();
}
Implementing the Messaging Mechanism

Implementation of chatting is always linked to messaging techniques. To implement chatting in our P2P application, you must have a thorough understanding of the mechanism for sending and receiving messages.

Sending messages

As soon as this window is displayed, it writes an XML file specifying the request type as chat and information such as the IP address and the name of the peer it is going to connect to. The code snippet that follows shows how this task is accomplished.

```csharp
1 if(!Connected)
2 {
3    if( null != Computername )
4    {
5        string AddressIP =
6            Computername.Substring(
7                Computername.IndexOf("\") + 1);
8
9        AddressIP = AddressIP.Substring(0,
10            AddressIP.Length - 1);
11
12        IPEndPoint EPhost = new IPEndPoint(
13            new IPAddress(AddressIP), 7070);
14
15        //Create the Socket for sending data over TCP
16        ClientSocket = new TcpClient();
17
18        // Connect to host using IPEndPoint
19        if( ClientSocket.Connect(EPhost) != 0 )
20        {
21            MessageBox.Show("Unable to connect to host");
22            return ;
23        }
24
25        new frmShare().CreateRequest(
26            "CHAT", Computername, txtMessage.Text);
27
28        FileStream fileStream = new FileStream(
29            new frmShare().REQUESTFILE, FileMode.Open);
30
31        Byte[] ReadBuffer = new Byte[1024];
32        fileStream.Read(ReadBuffer, 0,
33            fileStream.Length.ToInt32());
34
35        ClientStream = ClientSocket.GetStream();
36        ClientStream.Write(
37            ReadBuffer, 0, fileStream.Length.ToInt32());
38        ClientStream.Flush();
39        fileStream.Close();
```
The preceding code snippet is executed each time the user clicks the Send button to send messages to the peer with whom he or she is connected.

**Code description**

- ♦ Lines 1-23: In this part of the code, the connection is established when the listener specifies the IP address and the port to communicate by using the `TcpClient` class.
- ♦ Lines 24-25: A new XML file is created specifying the request type as `Chat` and other information such as the IP address and the name of the remote listener by using the `XMLCreator` class's `CreateRequest()` function. The `XMLCreator` class is defined in the `WorkingWithXML` namespace.
- ♦ Lines 27-33: A new file stream is opened for the XML request written and is read in a buffer.
- ♦ Lines 34-42: A network is opened, and the XML request file read is written on it for other peers to read and is flushed and closed thereafter. The file is deleted, and variables are initialized to indicate a connection has been successfully established.

The chat session opens a new socket after closing the previous one through which it has been connected and sends a request to the listener. The listener keeps this particular socket open unlike other sockets that it closes as soon as it has processed requests received on them. This socket is kept open unless the connection is closed either on the listener side or browser side. A new connection is always opened to keep the normal request/response process running behind the normal chat session.

Once you know how to send messages, we need to address how messages are received. Because the processes of sending and receiving data work in unison, implementing only one of them has no meaning.

**Receiving messages**

To receive messages sent by other peers, sockets are checked for data availability every 0.2 seconds. This is done by initializing a timer with the time-elapse setting equal to 200 milliseconds (Listing 7-1).

**Listing 7-1**

```csharp
// © 2001 Dreamtech Software India Inc.
// All rights reserved.

1 if( Sock.Available>0 )
2 {
3   Byte[] RecvBytes = new Byte[Sock.Available];
4   Sock.Receive(RecvBytes, Sock.Available, 0);
5   if(Encoding.ASCII.GetString(RecvBytes,0,RecvBytes.Length).CompareTo(6       "heistyping") == 0)
7     { ClientTyping = true;
8       lChatSttsBar.Text = ChatUserName + " is typing a message.";
9       return;
```
326 Chapter 7: Add-Ons to the P2P Application

12 )
13
tMessages.ForeColor = Color.DarkGray;
14 Font f = new Font("Palatino Linotype",8,FontStyle.Bold);
15 rtMessages.Font = f;
16 rtMessages.ForeColor = Color.Aquamarine;
17 rtMessages.Text += ChatUserName + " says : ";
18
19 Font f2 = new Font("Palatino Linotype",8,FontStyle.Bold);
20 rtMessages.Font = f2;
21 rtMessages.ForeColor = Color.Black;
22 rtMessages.Text += Encoding.ASCII.GetString(RecvBytes, 0,
23 RecvBytes.Length) + "\r\n";
24 lChatSttsBar.Text = ChatUserName + " is connected";
25 )
♦ Line 1: This line checks whether data is available on the socket or not.
♦ Lines 3-5: A byte buffer is made of a size equal to the amount of data received.
♦ Lines 6-13: If this message is just to signify that the other peer is typing a message, information is
displayed on the status bar, and the rest of the code is skipped.
♦ Lines 14-24: The color and the font settings are set for the box in which incoming messages are
displayed and a message is displayed in the box.
♦ Line 25: The status bar is rewritten, indicating that the peer is connected.

The Final Touch

To give a professional look to the chat-client extension, a feature has been incorporated through which
peers receive notifications when other peers they are connected to are writing messages. The user is
notified when the peer he is connected with writes a message (see Figure 7-4). The code for this feature is
discussed in the preceding code listing as we discuss the code in lines 6-13.

Figure 7-4: The user is notified that the peer he is chatting with is writing a message.
Chapter 7: Add-Ons to the P2P Application

As the first letter is typed, the Typing variable is initialized with a positive value to indicate that something has been written, and a message is sent to the listener. This is to indicate that the peer is not away and is writing a message, making this chat extension more interactive.

To accomplish this task, a counter has been maintained to see that the message is sent only after 0.1 seconds of writing. The time counter is kept incrementing to keep track of time intervals.

It is only after not writing anything for 2 seconds that Typing variable is initialized to a negative value and the time counter is set back to 0 to indicate that typing has been temporarily halted. From here on, writing anything after a time elapse of 2 seconds causes the message to be sent to the peer.

If the user writes anything, the counter is initialized back to 0; only 2 seconds after the last letter is written is the Typing variable initialized to a negative value.

protected void txtMessage_KeyPress (object sender, System.WinForms.KeyPressEventArgs e)
{
    Typing = true;
    TimeElapse = 0;
}

The changes required to achieve the chat mechanism in our P2P application design are shown in the preceding code. The code is only a small snippet and does not give you the whole picture. We give the user a chance to develop a full-fledged chat application, based on the knowledge that he has acquired so far from the previous code snippets.

Implementing Globalize Printing

Before understanding Globalize Printing, you should be familiar with Network Printing. Network Printing means sharing a single printer among various users. Such printing is implemented in small networks such as LAN, which allows you to print your own document on a network printer by simply providing a print command from your desktop without requiring you to have your own desktop printer. Apart from escaping the cost of using multiple printers, a single printer easily handles the requirements of all networked users. We take Network Printing one step further in Globalize Printing. Globalize Printing is not restricted to any LAN or Intranet; with Globalize Printing, you can print your desired documents in different locations provided that the person in that location is a registered member with our P2P system. Any remotely located printer can be used for printing.
As mentioned at the outset, one of the features that account for the versatility of the P2P application we have developed is that it provides an ample scope for enhancements to be performed very easily. Globalize Printing is one such feature that many users may seek to incorporate in this application. With Globalize Printing, you can print a remote file that is present on a remote peer to which you are connected. This feature is significant when you are connected to a computer and are browsing its shared contents. To avail this feature, select a file and click the Print button in the browser window. In the same way requests for uploading and downloading files are generated, the request for printing is generated via the XML creator component of this application. The detailed design of the XML request needed for the purpose of printing is the following:

```xml
<?xml version="1.0" encoding="utf-8"?>
<p2p_lng>
<request type="PRINT">
  <scope type=Filename mask=""/>
</request>
</p2p_lng>
```

Note that here you have to specify the type of the request as PRINT as indicated in the third line of the preceding XML code. The Filename denotes the file that is to be printed.

You have to make some additions to the existing code:

1. Add a Print button in the browser window.
2. Add code for the Print button when you click the button.

The Print button can be added just as you add a simple button to a form. Just open the frmShare form and add a button to it. Make sure to give the name btnPrint to the button you have inserted. Figure 7-5 shows you where to insert the button.

![Figure 7-5](image-url)

**Figure 7-5:** Clicking the Print button initiates the Global Printing process on the selected document.

In Figure 7-5, you can see the Print button.

After adding the Print button to the frmShare dialog, you have to add code to it.

**Coding for the Print Button (Browser)**

The following the lines of code are for the click event of the Print button you have just added.
```csharp
protected void btnPrint_Click (object sender, System.EventArgs e)
{
    string Filename;

    // Get the currently selected item from the List view
    int index = GetSelectedItemFromListView(out Filename);
    try
    {
        // Checks whether any entry is selected or not
        if ( -1 != index)
        {
            // Checks whether a filename is selected for
            // printing
            if ( null != Filename )
            {
                // If filename is selected then print it
                if ( OpenConnection(COMPUTERNAME) )
                {
                    // Creates a PRINT request for a
                    // particular
                    // file to PRINT.
                    CreateRequest("PRINT",strArray[index].sFilename,"*");
                    // Sends this request to the
                    // listener
                    SendDataToListener(REQUESTFILE);
                }
            }
            else throw new Exception("Cannot print
            // folders cannot be
            // printed
            else throw new Exception("Cannot print
            // folder");
        }
        else throw new Exception("Nothing Selected");
    }
    catch( Exception err ) { MessageBox.Show(err.Message,
        "Error",MessageBox.IconError); }
}
```
**Code description**

Starting from line 3, we go into a detailed explanation of the preceding code. We have declared a `string` type object called `Filename`. This object `Filename` is used to store the name of the selected entry from the list view of the browser window. Thereafter, a call is made to the user-defined function `GetSelectedItemFromListView`, and the `Filename` is passed as an out parameter to this function. The function returns the index of the selected element in the `index` variable, and the name of the actual entry is returned to the `Filename`. If any folder is selected, the index value returned by the function is greater than -1, and the out parameter `Filename` is `NULL`. An exception is thrown, and an error message is generated and shown to the user (as shown in Figure 7-6). The function is terminated without executing further.

![Error](error.jpg)

Figure 7-6: Any exception faced in printing is indicated by an error message.

If, on the other hand, nothing is selected, this function returns -1 as the index value and `NULL` value in `Filename`. In this case, too, an exception is thrown, and the error message box is displayed (see Figure 7-7) to the user; the function is terminated without executing further.

![Error](error.jpg)

Figure 7-7: If the printing process is initiated without selecting any document, and error message is displayed.

After checking for the preceding validations, at line 17 we open a socket connection by using the `OpenConnection` function and passing `COMPUTERNAME`. This function returns either true or false, which is checked by the `if` statement. At line 21, the actual print request is being generated by using `CreateRequest`, a user-defined function, and passing arguments to it. The first argument of this function is the request type; in this case, the request type is `PRINT`. The second argument is the name of the file to be printed. Here the fully qualified filename is extracted from `strArray[index].sFilename` array by giving its index. The third parameter is blank.

At this moment, your request is successfully generated in the folder from which your application is running. Usually, an XML file containing the request data is created for the request. You can run the program to this location by adding a break point on a line below the `CreateRequest` function to check whether the request has been properly generated or not. Once the request is properly generated, a call to the `SendDataToListener(REQUESTFILE)` function is made, and the name of the request file is passed to the function. This function sends your request to the remote peer to whom you are connected; if the peer receives the request successfully, the document is printed on his or her printer.
Coding for the Print Button (Listener)

In the preceding discussion, we have considered the part of coding that is relevant to the browser. The initiation of printing is performed from the browser side, and the work of the listener is less in comparison with that of the browser. The following are the changes you have to make at the listener’s end. These changes are to be made in the `frmLogin` class.

```
namespace Listener
{

using System.Runtime.InteropServices;

public class frmLogin : System.WinForms.Form
{

[DllImport("Shell32.dll")]
public static extern int ShellExecute( int hwnd,
    string lpVerb,
    string lpFile,
    string lpParameters,
    string lpDirectory,
    int nShowCmd );

```  

**Code description**

In the beginning of the `frmLogin` class, where all the necessary namespaces are included, you have to include the `System.Runtime.InteropServices` namespace in order to achieve the functionality of a `Shell32.dll`. This is needed for the `ShellExecute` Win32 SDK function.

Now you need to import the `ShellExecute` function from the `Shell32.dll`. For this, the following statements are to be included:

```
[DllImport("Shell32.dll")]
public static extern int ShellExecute( int hwnd,
    string lpVerb,
    string lpFile,
    string lpParameters,
```
string lpDirectory,
    int nShowCmd );

The first line serves the same purpose as the `LoadLibrary` function in VC++. It loads the `Shell32.dll` file into the application-process space in memory. We have declared the prototype of the `ShellExecute` function that we intend to use in our application. This function is exported to our application from the `Shell32.dll` file. You can use any Win32 SDK function in C# provided that you know the file in which the definition of that function is present.

The code for handling the print request is the following:

```csharp
protected void RequestResponse()
{
    .
    .
    .
    .
    if(DnLoadFile.Length != 0)
    {
        if(UploadDownloadPrint == 1)
        {
            fs1 = new FileStream(DnLoadFile,FileMode.CreateNew);
            DnLoadFile.EndsWith("mp3");
        }
        else if(UploadDownloadPrint == 2)
        {
            new File(Application.StartupPath + "\Temp.xml").Delete();
            DataStream.Flush();
            newSock.Close();
            bDownLoading = false;
            return;
        }
        bDownLoading = true;
    }
    .
    .
    .
    .
}
```

You have to add only the part of the code shown in bold. Here the `ShellExecute` function discussed previously is used to print the document. The first parameter to this function is to handle the window, which can be 0; the second parameter represents the action you intend to perform; and the third parameter reveals the name of the file to be printed. The fourth, fifth, and sixth parameters should be exactly as shown in the preceding code. After you send the file to the printer, the next line deletes the temporary file, flushes the stream by using the stream object, closes the socket setting the `bDownload` to false, and returns from the function. These are the necessary changes on the listener side coding of the `frmLogin` form to achieve Globalize Printing.
In the same way, you can print your file on the remote printer or can print a file to multiple printers that are connected to different peers. You will find exploring these possibilities an interesting exercise. Implementations of these are not detailed here, as we have to proceed to some of the other interesting and useful features this project offers.

P2P Searching on Palm Using Java 2 Micro Edition

P2P applications are available for conventional programming environments. This means you can run such applications on your desktop system. But the Internet has gone far beyond PCs and servers to devices such as Personal Desktop Assistants (PDAs) and cell phones. We consider these in this chapter. Can P2P work on these devices?

Before we discuss the feasibility of P2P Searching on Palm PDAs or cell phones, we should understand the programming practices and limitations of such devices. PDAs and cell phones have the following limitations:

♦ **Limited processor power:** Unlike desktop systems, PDAs and cell phones are not exactly bursting with processor power. Because PDAs and cell phones are battery operated, batteries consume more power as processor speed increases.

♦ **Types of input and output methods:** Unlike desktop systems, PDAs and cell phones do not have a standard “QWERTY” keyboard for input or a monitor for output. Depending on devices, input and output methods vary. Some devices, such as two-way pagers or cell phones, have numeric keypads with alphabet support, yet other devices have a stylus-pads using which one can enter data. Unlike desktop systems, PDAs and cell phones do not boast of a 15-inch or 17-inch screen. Their screen size is limited to a maximum of 160 by 160 pixels.

♦ **Memory constraints:** Unlike desktop systems, PDAs and cell phones do not boast of gigabytes of memory; they have memory limitations as well as processor limitations. The memory in these devices is divided into heaps, static heaps and dynamic heaps, and is limited.

Taking these limitations into consideration, the programming techniques for these devices are to be modified (that is, because the seemingly unlimited available memory of the desktop is replaced by the limited memory of the PDAs, you must adapt to the memory model of the PDAs). Some techniques for efficient memory utilization are the following:

♦ Increased use of local variables

♦ Initializing objects only when they are required

♦ Smaller applications; that is, only features essential to an application should be created and not those which are only used rarely.

♦ Deinitializing objects whenever possible, thereby helping the garbage collector

Programming using the J2ME is not the same as programming using the J2SE because of the memory constraints involved in the Micro Edition. Therefore, the application in this book cannot be simply ported onto the small devices. It requires certain changes; for example, changes in GUI. We cannot use tables now to display information, and only one screen can be shown at a particular time, as opposed to the many screens displayed in the larger version. Also, changes arise in some classes and their methods. The most important change is the way XML parsing is handled. Because of small devices such as cell phones, PDAs, and so on, which have less memory, the XML parser should also be small in light of the memory requirements involved.

The J2ME version does not offer the same functionality as the J2SE version of the application. For instance, it does not allow you to download files. This is understandable, as a device such as a mobile phone cannot be expected or required to store downloaded files. Using this application, you can connect to any user and view shared files, but you cannot download them. For that, you have to use a PC. The idea is that you are at least able to find the required files even while you are on the move.
The explanation of the project follows. First, the user interface is explained, followed by the working of the project and a flow diagram. For the source code of this application, refer to *Wireless Projects Using J2ME* published by Hungry Minds, Inc.

**The User Interface**

When you start the application, the starting screen shows just two buttons: Start and Exit (see Figure 7-8). When you press the Start button, the next screen shows a scrollable list of users who are currently connected to the same site and using this application. Being connected means that they have logged on to the Internet with this application. Files can be searched in two ways: through a server-level search and a user-level search. A server-level search is useful when you know what files you want, or at least some characters in the names of those files, so that you can use wildcards to conduct the search. For example, if you know that the name of the song you want has the word *rock* in it, you can use this information as the search criteria and can use wildcards (*) for searching it. The other option is to conduct a user-level search. This is useful if you know on whose system the required files are located.

![Peer to Peer System.](image)

**Figure 7-8:** Screen 1: Starting screen

At its top, the next screen has two buttons, labeled Refresh and Search, respectively. Pressing Refresh updates the list. If you want to search files, you have to press the Search button. Two more buttons, labeled Browse and Exit, respectively, appear at the bottom of the screen. If you decide to see what files are available with a particular user, you can enter your choice in a `TextField` in the form of the user number. Then the user presses the Browse button. This takes you to a screen showing the listing of shared folders and files on the root directory of that user. If necessary, you may select a folder and again press the Browse button. This can go on until you find the folder of your choice. At any stage of this process, you can press the Search button after entering the directory number in the `TextField`. This gives you a chance to enter your search criterion with any wildcard to locate the required files.

If you do not have an idea where the files are, you may directly press the Search button without pressing the Browse button. With this, you see a screen with two `TextField`s: one field for entering the search criteria for the filename and one field for entering the search criteria for the computer name. After entering these, the Search button is pressed, to show the files found in the search. The search can be cancelled by pressing the Cancel button to go back to the starting screen.

**How It Works**

This application consists of five class files: `peer_application`, `userlist_screen`, `serversearch`, `searchfiles`, and `showfiles`. In addition, the application uses an external XML parser available in an `org.kxml.parser` package. Executing the first class file (`peer_application`) starts the application. This class performs the following tasks:
Chapter 7: Add-Ons to the P2P Application 335

♦ Imports the packages required, such as those for handling XML and parsing it (org.kxml.*, org.kxml.io.*, and org.kxml.parser).
♦ Declares variables for the XML parser and parse event, IP address, user list, and the text to be displayed in ScrollTextBox.
♦ Uses the Graphics class to clear the screen and to display the text welcoming the user through the clearScreen and drawString methods, respectively.
♦ Paints Start and Exit buttons on the screen by using the paint method of the Button class.
♦ Defines event handling for the buttons via the penDown method. If the Start button is pressed, the current spotlet is unregistered, and a startReading method is called. The callbacks generated by the XML parser in the method parseData() are stored in three variables: ip_address, users_connected, and text_for_display.
♦ Calls a method named startReading, which is responsible for sending a request to the server for a list of users connected at any time.
♦ Instantiates and calls the class userlist_screen, which displays the list of users. It takes its parameters as the three variables mentioned previously.
♦ Closes the application in case the user does not want to go on further and presses the Exit button.
♦ Defines the method startReading. This method instantiates the preceding three variables and opens an InputStream on the ASP file named userlist.asp. It also shows an error if an IOException occurs.
♦ Defines the method parseData. It uses the XML parser mentioned previously and finds the userinfo tag with the help of the do…while loop and an if block. The attributes of this tag are obtained by using the getValue method and are then added to their appropriate position. The vector ip_address is then returned to the calling class.
♦ The parseData method is available in the parser; it has only to be overridden. But we also need to return two more variables. For this reason, two more methods are defined. One is to return the text_for_display variable, and the other is to return the users_connected variable.

The class userlist_screen (shown in Figure 7-9) is called to display the list of users connected at a particular time. It performs the tasks that follow:

♦ Declares variables for buttons, ScrollTextBox; TextField; StreamConnection; XML parser and parse event; String variables; folder_data; file_data; vectors for the previously mentioned variables; and two more variables (folder_vector, record_of_users).
♦ Defines constructor of the class to take text; ip_address; users_connected; viewfile_flag; and record_of_users as the parameters. The parameters are then initialized by the variables earlier defined. The Methods of Graphics class are then used such as drawRectangle(), and so on, to display the user interface elements.
♦ Defines event handling via the keyDown and penDown methods. The former handles the input if the TextField is in focus, and the latter handles the event of buttons being pressed. If necessary, control is transferred to a relevant class.
♦ Defines a method called appropriatelen, which is called to make the application compatible with C# listeners. The method is required because the C# listener cannot read less than 1024 bytes at a time. This method takes an array called file_bytes and integer file_size as parameters and returns a byte array.
Chapter 7: Add-Ons to the P2P Application

The class `showfiles` (shown in Figure 7-10) is called when you have selected a computer to whose files/folders you want to view and have pressed the Browse button. It performs the following tasks:

- Creates the user interface for the screen showing shared files and folders.
- Provides the user interface for the buttons, `TextField`, and `ScrollTextBox`.
- Creates an object of the class `peer_application` and calls the method `startReading` to send a request to the server.
- Defines a method `browseDirectory` called when the Browse button is pressed. The screen that follows shows the shared files in that directory.

The class `serversearch` is called when you want to search all users (see Figures 7-11 and 7-12). The class `serversearch` takes care of the following tasks:

- Declares variables, buttons, `TextField`, `DataInputStream`, and vectors `ip_address` and `users_connected`.
- Displays the user interface elements by using the `Graphics` class.
- Defines event handling through the `keyDown` method. The code serves the purpose of shifting focus when the Tab key is pressed.
- Defines event handling through the `penDown` method. This method handles the events generated by pressing the buttons, such as open, view, exit etc. If the Search button is pressed, the results satisfying the search criterion are displayed.
- Defines the method `parseData` that holds the callbacks generated when XML parsing is performed.
The class `searchfiles` is called when the user wants to conduct a search (shown in Figure 7-13). The class `search_files` performs the tasks that follows:

- Declares buttons, `TextField`, `Strings` (`parent_information`, `text_for_display`, `folder_data`, `file_data`, `host_address`), `XML parser` and `parse event`, `vector` `folder_vector`, `StreamConnection`, `InputStream`, and `OutputStream`.
- Defines the constructor to take `address` and `parent_information` as parameters. The parameters are initialized with the variables already declared.
- Draws the user-interface elements, including `buttons` and `TextField`.
- Defines event handling through `penDown` and `keyDown`. The former simply calls the latter if an option has the focus. The `penDown` method responds to events generated by pressing the buttons such as `exit`, `view`, `open`, and so on.
- The method `appropriatelength` is called to provide compatibility with C# listeners.
Figure 7-14: The basic flow of the program is shown in the flow chart.
Chapter 7: Add-Ons to the P2P Application

Figure 7-15: Extension of Figure 7-14
When the application starts, a welcome screen appears. After you press the OK button, an asp of the name `userlist.asp` is called, which displays all the users connected at a particular time on the screen. Along with this information, several buttons are presented to you: Refresh (to refresh the list of users currently connected, so as to check whether any new user has joined or not); Search (to implement a server-level search, meaning that the user may search for a file or a group of files on various users connected at a particular time); Exit (to close the application); and Browse. The Browse button is by far the most important. The button is of use when you enter the user number of a particular user and press the button. When you press the button, a socket connection is established between you and the listener, and the listener shows you the list of all shared files and folders. This screen also provides several buttons: Search (when this button is pressed, you can search for files in the list displayed); Browse (this button is used to go deeper into the directory hierarchy); and Close (this button is used to close the socket connection to a particular user) (Figures 7-14 and 7-15).

**Summary**

In this chapter, you have explored the knowledge you have acquired during your journey through this book. Based on the code presented in this book, you can develop full-fledged commercial applications using P2P communication technology. This chapter shows you how to accomplish the concept of extending the basic P2P functionality by incorporating new coherent add-ons. However, if you are eager to see the full-featured P2P application with add-ons that are discussed previously, you can find them on the accompanying CD-ROM disk.
Appendix A

About the CD-ROM

This appendix provides you with information on the contents of the CD that accompanies this book. For the latest information, please refer to the ReadMe file located at the root of the CD. Here is what you will find:

♦ What’s on the CD
♦ Hardware/Software Requirements
♦ Using the CD
♦ Troubleshooting

What’s on the CD

The following sections provide a summary of the software and other materials you’ll find on the CD.

Author-Created Materials

All author-created material from the book, including code listings and samples, are on the CD in the folder named "P2P Source Code."

The source code is further categorized into three subfolders; namely, JavaVersion, C#Version (the two versions in which this application has been built), and P2PServer. A brief description of the various files in each folder is provided here as an aid to handling the CD-ROM effectively.

JavaVersion Folder

In this folder you will find two more subfolders called Listener and Browser in which you will find their respective source codes.

Listener folder

♦ org: This package handles all operations necessary for XML Parsing.
♦ Login.java: This class comes into action when the Listener logs on to the server for the first time. This class also invokes login.asp and creates the userinfo.ini file that stores the name of the user.
♦ Multiserver.java: This class is responsible for initiating the server on a specific port number to handle the incoming requests and returning the responses generated by the server as per the request made.
♦ Shared_dialog.java: This class stores information of all the shared resources in the share.ini file.
♦ Checkdirectory.java: This represents the search operation to handle the task of client side searching when the client passes some search criteria.
♦ XmlWriter.java: This class is responsible for generating requests and responses in XML format.
♦ Sparser.java: This class is responsible for parsing the XML request and responses.
Appendix A: About the CD-ROM

Browser folder
- **org:** This package handles all the operations necessary for XML Parsing.
- **client.java:** This class is of immense significance since it provides the entry point for the Browser. This class is also responsible for displaying the list of the already logged in users by making the appropriate call `userlist.asp`. It also caters to the GUI components.
- **file_gui.java:** This class displays the contents of the selected user. It is also responsible for handling uploading and downloading of files and folders.
- **search_screen.java:** This class contains the coding that enables client-side searching.
- **add_on.java:** This is a multi-utility class that houses various methods to handle the various tasks. This class is used interchangeably in the lifecycle of this application. The tasks it handles include alphabetically sorting the user names obtained from the server, constructing and sending XML requests to the server, catching responses returned by the server as well as by the Listener. This class is also responsible for making calls to functions which provides compatibility with the Listener that has the C# version.
- **search_window.java:** This class is responsible for managing server-side search operations.
- **XmlWriter.java:** This class is responsible for generating requests and responses in XML format.
- **Sparser.java:** This class is responsible for parsing the XML request and responses.

How to run the application
To run the JAVA version of the application, copy all the folders under the folder JavaVersion on to a folder on the hard disk. To run the Browser, go to the folder Browser and double click the file `client.jar`. Similarly, to run the Listener, go to the folder Listener and double click the file `login.jar`. The process has been made easier for the users by providing the installation option in the `setup` application for the CD which copies the required files to the location specified by the user.

C#Version Folder
This part describes the contents that are present for the C# version of the P2P application on this CD-ROM:
- **P2P folder**
- **P2PwithAddons folder**

P2P folder
In the P2P folder you will find two folders: Browser and Listener.

This folder contains the full source code for the Browser part of the application. This source code is the basic source code of our application, and includes coding for uploading and downloading files.
- **frmClient.cs:** This file has the definition for class `frmClient` which contains the basic coding for the client side.
- **frmSearch.cs:** This file contains `frmSearch`. The main task of this class is to handle various search operations performed on the Browser side.
- **frmShare.cs:** This file has the definition for the class `frmShare`. This class holds immense responsibility as it caters to all uploading and downloading tasks. Another important function of this class is to display all shared resources of the Listener within the boundary of the Browser.
- **frmLogin.cs:** This file contains the definition of the `frmLogin` class which contains the code that governs the overall functionality of the Listener module.
- **frmSelection.cs:** This file contains the definition of the `frmSelection` class which helps users to share files and folders.
Appendix A: About the CD-ROM

♦ **BrowseFolder.cs**: This file contains the definition of the class `BrowseFolder` that is written to provide the folder selection dialog box used by `frmSelection`.

This source code on compiling will generate a class library file named `WorkingWithXML.dll`. You have to add the reference of this dll file in your main project, that is, the P2P project. This dll will contain three classes that are described in the ensuing lines.

♦ **clsServerCommunication.cs**: This class is responsible for handling the communication with server whenever it is required in the lifecycle of the application.

♦ **clsXMLCreator.cs**: The primary task of this class is to generate request and response XMLs.

♦ **clsXMLParser.cs**: This class actually parses the request and response XMLs generated by the `clsXMLCreator.cs` class.

**P2PwithAddons folder**

The application in this folder contains the full source code of the P2P application with the add-ons described in Chapters 6 and 7.

♦ **frmClient.cs**: This class incorporates all coding necessary for the client side.

♦ **frmSearch.cs**: This class is responsible for handling all the search operations performed on the Browser side.

♦ **frmShare.cs**: Apart from handling the regular tasks (uploading, downloading) of this P2P application, this class handles chatting, global printing and streaming.

♦ **frmStreamer.cs**: This class contains all coding required for streaming.

♦ **frmChat.cs**: This class is responsible for enabling chatting. All the code required for chatting is inside this class.

♦ **clsChatSocket.cs**: This class is inherited from the `TcpClient` class. It is used to access the protected functions of the `TcpClient` Class. This class is used only in `frmChat.cs` class.

♦ **frmLogin.cs**: This file contains the definition of the `frmLogin` class which contains the code that governs the overall functionality of the Listener module.

♦ **frmSelection.cs**: This file contains the definition of the `frmSelection` class which helps the user to share files and folders.

♦ **BrowseFolder.cs**: This file contains the definition of the `BrowseFolder` class that is written to provide the folder selection dialog box used by the `frmSelection` class.

♦ **ThreadIndex.cs**: This file contains the definition of the class that helps the `frmLogin` class to effectively manage threads.

♦ **FrmChat.cs**: This file holds the code for `frmChat` class, which handles the chat extension to the original P2P application.

♦ **StreamIt.cs**: This file defines the class `StreamIt`, which holds the code for the streaming extension.

This folder is very similar to the one with the same name in the P2P folder. Since the files names and number of files in the two folders are not different and only the code in the files differ, a description is not needed here.

**How to run the application**

To build the C# version of this application, copy all the contents of the P2P folder or `P2PwithAddons` folder to a folder on your hard disk and compile the project files or solution files of the Browser, the Listener and the `WorkingWithXML` projects using Microsoft Visual Studio .NET beta 2. In Browser and in the Listener projects, add a reference to the `WorkingWithXML.dll` which is generated by Microsoft Visual Studio .NET in the Debug folder under the bin folder of this project. Now just double click on the
exe files of each, that is, the Client.exe and Listener.exe that have been generated by the Microsoft Visual Studio .NET environment.

The process has been made easier for the users by providing the installation option in the setup application for the CD which copies the required files on the location specified by the user.

**P2PServer Folder**

Here the contents of the server part on this CD-ROM are discussed.

- **Login.asp**: The Listener while logging the server calls this page.
- **Logout.asp**: This page is invoked when the Listener logs out from the server.
- **Search.asp**: The Browser while performing search operations on the server invokes this page.
- **Userlist.asp**: The main task of this page is to provide the list of users already logged on the server or which are available when the Browser logs on the server for the first time.

*Shareware programs* are fully functional, trial versions of copyrighted programs. If you like particular programs, register with their authors for a nominal fee and receive licenses, enhanced versions, and technical support. *Freeware programs* are copyrighted games, applications, and utilities that are free for personal use. Unlike shareware, these programs do not require a fee or provide technical support. *GNU software* is governed by its own license, which is included inside the folder of the GNU product. See the GNU license for more details.

*Trial, demo, or evaluation versions* are usually limited either by time or functionality (such as being unable to save projects). Some trial versions are very sensitive to system date changes. If you alter your computer’s date, the programs will "time out" and will no longer be functional.

**eBook version of Peer-to-Peer Application Development**

The complete text of this book is on the CD in Adobe’s Portable Document Format (PDF). You can read and search through the file with the Adobe Acrobat Reader (also included on the CD).

This is the first book to cover the entire code behind a Napster-style, file-sharing model. It contains unique coverage of Windows Media Technology development for making your P2P application multimedia aware. The book is loaded with code, keeping theory to a minimum. The applications, for which the source code is given, are 100 percent tested and working at Dreamtech Software Research Lab. The source code provided in the book is based on commercial applications, which are developed by the software company. Each program of the application is explained in detail so that you get insight into the implementation of the technology in a real-world situation. At the end of the book, some add-ons to this application are given so that you can further explore new developments that are taking place.

This book deals with the design, implementation, and coding of the latest form of the client/server model, the P2P model. The book serves to equip you with enough know-how on the subject so as to enable you to design a P2P model of your own.

The book begins with the history of the P2P model and goes on to explain the various types of P2P models, with detailed diagrams to elucidate the subject. After equipping you with basic concepts, it goes on to develop, step by step, a full-fledged application, which has the scope of being extended with add-ons.

This book is *not* meant for beginners. It teaches you the basics of specific technologies only. The *Cracking the Code* series is meant for software developers/programmers who wish to upgrade their skills and understand the secrets behind professional-quality applications. This book starts where other tutorial books end. It enhances your skills and takes them to the next level as you learn a particular technology. A thorough knowledge of the Java or C# programming languages is the prerequisite for benefiting the most from this book. Experience in network programming is an added advantage. For developing streaming
applications, knowledge of Visual C++ is a must. At least a nodding acquaintance with the XML mark-up language is desirable, although the book incorporates a section on XML. Instructions for embedding existing chat-client and audio/video components have been included. You can craft this application in such a way that you are able to send your files to be printed to any part of the globe. Besides Globalize Printing, you can make the application run on wireless models, too. The opportunity is open for you to assess your networking skills and to improve them.

The pivotal feature of the book is that it offers a complete, ready-to-deploy application with source code. The purpose of this book is to acquaint programmers with the subject thoroughly so that they are in a position to write their own codes to build P2P applications. Detailed explanations of the steps involved in writing your own code to build a P2P application in Java as well as in C# have been furnished.

Hardware/Software Requirements

This section lets you know the different S/W and hardware requirements that your system needs to meet to be able to host the applications on the CD.

Hardware Requirements

Make sure that your computer meets the minimum system requirements listed in this section. If your computer doesn’t match up to most of these requirements, you may have a problem using the contents of the CD.

For Windows 9x, Windows 2000, Windows NT4 (with SP 4 or later), Windows Me, or Windows XP:

♦ PC with a Pentium processor running at 120Mhz or faster
♦ If you are running the Java version of the P2P application, a minimum of 32MB of total RAM installed on your computer; for best performance, we recommend at least 64MB
♦ To run the C# version of the P2P application, you need to have a minimum of 128MB of total RAM installed on your computer, and for better performance, we recommend 256MB or more. This is because the Microsoft Visual Studio .NET (which has been used to develop the C# version of this application) has not come up with the provision to produce self sustained applications as yet. Thus you need to have Microsoft Visual Studio .NET installed on your computer which requires this much RAM, to be able to run the C# version of the P2P application.
♦ Ethernet network interface card (NIC) or modem with a speed of at least 28,800 bps
♦ A CD-ROM drive


Software Requirements

This section discusses the software you need to have on your computer to be able to run the two versions of the P2P application on the CD followed by the S/W requirements for setting up the P2P server.

The Java version

Following is the software the user needs to have on his system to run the Java version of this software.

This is the Software Development Kit for developing standard Java applications. The Java version of our application has been developed using this kit. You can download it from: [http://java.sun.com/j2se/1.3/](http://java.sun.com/j2se/1.3/). It is available on the CD as well.

Forte for Java release

The Forte for Java release software is an integrated development environment used for developing the Java application. You can use this to develop the Java version of our application. It is an IDE provided by
Sun Microsystems. You can download it from: http://www.sun.com/forte/ffj/. It is available on the CD as well.

The XML Parser for parsing the XML data. The Java version of the application internally uses this XML Parser. You can get this XML Parser from: http://xml.apache.org/dist/xerces-j/. It is available on the CD as well.

**The C# version**

Following is the software the user needs to have on his system to run the C# version of this software.

The XML Parser class for parsing the XML data. The C# version of the application internally uses this XML Parser. You can get this XML Parser from: http://msdn.microsoft.com/msxml.

This SDK is used for developing the streaming extensions of our application. You can download it from: http://msdn.microsoft.com/windowsmedia/.

Windows Encoder SDK, which has been used for developing the streaming extensions of our application, internally uses some DirectShow components to render and index some non-streaming media formats. Since the streaming extension could have been developed using this SDK, you may want to explore it. This SDK is available for downloading at: http://msdn.microsoft.com/directx/. It is available on the CD as well.

This contains the Window Media Player component that is embedded in the Browser and used to show the audio/video to the user. You can download it from: http://msdn.microsoft.com/windows/windowsmedia/.

**The P2P server**

Following are the software that users need to have on the server to set up the P2P server.

The P2P server can be set up using any of the servers of Windows NT series - 4 or above. P2P server uses the services of Windows NT series server to host its site and to manage database. You can learn more about this O/S at: http://www.microsoft.com/windows2000/.

P2P server uses this RDBMS (Relational Data Base Management System) to manage the P2P database. You can learn more about this SQL Server 2000 at: http://www.microsoft.com/sql/default.asp.

**IIS 5.0**

Internet Information Services 5.0 (IIS) is the Windows 2000 Web service that has been used to host the site for P2P. You can learn more about IIS 5.0 at: http://www.microsoft.com/windows2000/en/server/iis/.

Apart from the previously discussed software, you will need to have the following.

Software for viewing Adobe PDF files. You will need this software to read the eBook version of *Peer-to-Peer Application Development*. You can download this software from: http://www.adobe.com/products/acrobat/readermain.html. It is available on the CD as well.

**Using the CD**

To install the items from the CD to your hard drive, follow these steps:

1. Insert the CD into your computer’s CD-ROM drive.
2. A window will appear with the following options: Install, Explore, eBook, Links and Exit.
Appendix A: About the CD-ROM

• **Install:** Gives you the option to install the supplied software and/or the author-created samples on the CD-ROM.
• **Explore:** Allows you to view the contents of the CD-ROM in its directory structure.
• **eBook:** Allows you to view an electronic version of the book.
• **Links:** Opens a hyperlinked page of web sites.
• **Exit:** Closes the autorun window.

If you do not have autorun enabled or if the autorun window does not appear, follow the steps below to access the CD.

1. Click Start ➔ Run.
2. In the dialog box that appears, type `d:\setup.exe`, where `d:` is the letter of your CD-ROM drive. This will bring up the autorun window described above.
3. Choose the Install, Explore, eBook, Links, or Exit option from the menu. (See Step 2 in the preceding list for a description of these options.)

**Troubleshooting**

If you have difficulty installing or using any of the materials on the companion CD, try the following solutions:

♦ **Turn off any anti-virus software that you may have running.** Installers sometimes mimic virus activity and can make your computer incorrectly believe that it is being infected by a virus. (Be sure to turn the anti-virus software back on later.)

♦ **Close all running programs.** The more programs you’re running, the less memory is available to other programs. Installers also typically update files and programs; if you keep other programs running, installation may not work properly.

♦ **Reference the ReadMe:** Please refer to the ReadMe file located at the root of the CD-ROM for the latest product information at the time of publication.

♦ **Make sure that you are online:** The applications (that is, the P2P applications) on this CD are developed assuming the user has an internet/intranet connection. These applications will not work if the user is not connected to internet or intranet.

If you still have trouble with the CD, please call the Hungry Minds Customer Care phone number: (800) 762-2974. Outside the United States, call 1 (317) 572-3994. You can also contact Hungry Minds Customer Service by e-mail at techsupdum@hungryminds.com. Hungry Minds will provide technical support only for installation and other general quality control items; for technical support on the applications themselves, consult the program’s vendor or author.
Index

A
AcceptFunction( ) function, 216
Add this entry only checkbox, 87
add-on class, source code, 155-165
add-ons
  chat client, 320-321
    implementing messaging mechanism, 324-326
    message writing notification, 326-327
    XML messaging format, 321-324
  communication issues, 21-22
  Globalize Printing, 327-332
  addresses, IP
    proxy servers, 19
    retrieving, 175
    storing, 86
application (P2P)
  C# source code
    frmClient class, 227-244
    frmLogin class, 184, 200-217
    frmSearch class, 274-279
    frmSelection class, 217-227
    frmShare class, 244-273
    ServerCommunication class, 167-175
    XMLCreater class, 175-184
    XMLParser class, 184-200
  client
    design considerations, 28
    operation flow, 43-51
  communication with add-ons, 21, 22
  functions, 19
Java classes, 72
Java source code
  add-on class, 155-165
  check_directory class, 105-112
  client class, 118-130
  file_gui class, 134-151
  Login class, 76-86
  MultiServer class, 95-105
  search_screen class, 151-155
  search_window class, 130-134
  Shareddilog class, 87-94
  SParser class, 112-118
  XMLParserServer class, 73-76
  listener, 29-31
    multiple connections, 32-33
  ServerSocket class (Java), 31
  TCPLlistener class (C#), 32
    listening for requests, 32-33
  logging out, 22
  operation, 20-21
  structure, 19-21
  XML module, 29
  XML parsers, 29
    MS XML 3.0, 29
    Xerces, 29
  application example, 12
    browser, 13-14
    components’ purpose, 17
    design, achieving, 15
    listener, 13
    relationships of components, 14-15
    server, 13
  applications
    Gnutella, 6-7
    Napster, 5-6
    streaming, running, 317-319
  appropriate_length( ) function, 155
  ASP (Active Server Pages)
    browser to server communications, 22
    Login ASP, 22
  asp files
    flowcharts, 55, 60-71
    login.asp, 55-61
    logout.asp, 61-63
    search.asp, 67-71
    userlist.asp, 63-66
  asterisk (*), select statement (SQL), 54
  attributes, streaming, 280
  audio, streaming, 280
  AUTHENTICATION structure, 198

B
Broadcast function, CBroadcastIt class, 287
broadcasting
  Gnutella, 7
    streaming capability and, 283
  BroadcastIt component (streaming), 283-288
  BroadcastStatus( ) function, 317
  Browse button, 87
  browsers. See also client (P2P application), 112
    application example, 13-14
Index

- listener relationship, 14
- server relationship, 15
- C# source code on CD-ROM with book, 342
- chat client add-on, 321
- Java source code on CD-ROM with book, 342
- P2P application
  - communication considerations, 22
  - operation, 21
  - streaming, implementing, 312-315
- user interface form, 227
- btnQuit Click( ) function, 215
- btnShareFileFolder Click( ) function, 215
- buttons
  - browser, user interface, 227
  - Login class, 76
- Print
  - adding, 328
  - coding, 328-330
  - coding for listener, 331-333
- Shareddilog class, 87

C

- C#
  - connecting to server, 37
  - disadvantages, 72
  - NetworkStream class, 40
  - source code on CD-ROM (with book), 342-344
  - TCPClient class, 38
  - TCPLi.stener class, 32
  - XML parser, MS XML 3.0, 29
- Cancel button, 76
- CD-ROM (with book)
  - C# source code, 342-344
  - eBook description, 344-345
  - hardware requirements, 345
  - installation, 347
  - Java source code, 341-342
  - P2PServer folder contents, 344
  - software requirements, 345-347
  - troubleshooting, 347-348
- CEvents class, declaring, 304-311
- chat live, streaming and, 282
- chat client add-on, 320-321
  - implementing messaging mechanism, 324-326
  - message writing notification, 326-327
  - XML messaging format, 321-324
- CHAT request type, 321
- chatting, advantages, 320
- check_directory class source code, 105-112
- classes
  - add-on, source code, 155-165
- check_directory source code, 105-112
- client, source code, 118-130
- DNS, 38
- file_gui source code, 134-151
- frmClient source code, 227-244
- frmLogin source code, 184, 200-217
- frmSearch source code, 274-279
- frmSelection source code, 217-227
- frmShare source code, 244-273
- InputStream, 40
- Java, P2P application, 72
- JDialog, setting dialog titles, 94
- Login source code, 76-86
- MultiServer source code, 95-105
- NetworkStream, 40
- OutputStream, 40
- peer application, 334
- search_screen source code, 151-155
- search_window source code, 130-134
- searchfiles, 337
- ServerCommunication source code, 167-175
- serversearch, 336
- ServerSocket, 31
- Shareddilog source code, 87-94
- showfiles, 336
- Socket, 38
- Sparser, 72
- SParser source code, 112-115
- StreamIt, 315
- TCPClient, 38
- TCPLi.stener, 32
- URL, connecting to servers, 37
- userlist screen, 335
- Vector, 72
- XMLCreater source code, 175-184
- XMLParser source code, 184-200
- XMLParserServer source code, 73-76
- xmlwriter, 72
- XMLWriter source code, 115-118
- client (P2P application)
  - design considerations, 28
  - operation flow, 43-51
  - source code
    - add-on class, 155-165
    - client class, 118-130
    - file_gui class, 134-151
    - search_screen class, 151-155
    - search_window class, 130-134
    - SParser class, 112-115
    - XMLWriter class, 115-118
  - client class source code, 118-130
client request handling MultiServer class (Java), 95-105
client/server model, P2P-based model and, 2
client-listener connections file_gui class (Java), 134-151
clients, chat add-on, 320-321
implementing messaging mechanism, 324-326
message writing notification, 326-327
XML messaging format, 321-324
client-side searches, 11
Close button, 87, 245
CloseSession function, CBroadcastIt class, 287
CLR (Common Language Runtime), 72
columns, SQL select statement, 54
components, 19
connected user lists, retrieving, 36-37
connections, listener design considerations, 32-33
create table keyword, SQL, 53
CreateDocument( ) function, 182
ctxMenuQuit( ) function, 215

D
database (P2P application) design issues, 26-27
databases, tables, 52
date and time, obtaining, 301
declarations
CBroadcastIt class, 287
CEvents class, 304-311
client class, 129
earch_window class, 134
file_gui class, 149
frmSelection class, 226
frmShare class (C#), 269
GUI objects, 86
MyContentHandler class, 75
search_screen class (Java), 154
ServerCommunication class (C#), 174
SParser class, 115
XMLCreator class (C#), 182
XMLParser class (C#), 198
XMLParserServer class, 75
XMLSTRUCT, 198
definitions, wild_card method, 112
design issues, 18
client, 28
communication with add-ons, 21-22
connections
DNS class, 38
HTTP, 37
peers, 37
retrieving connected user lists, 36-37
Socket class (Java), 38
TCPClient class (C#), 38
URL class, 37
data transfer, 19
database, 26-27
downloading procedure flow, 38-40
listener, 29-31
listening for requests, 32-33
multiple connections, 32-33
ServerSocket class (Java), 31
TCPLInnster class (C#), 32
proxy IP resolution, 19
searching functionality, 35, 40
global searches, 42
local searches, 42
security, 19
server, 25
SQL Server 2000, 25-26
streaming uploads/downloads, 40
uploading procedure flow, 40
XML, implementing in SQL Server 2000, 27-28
XML module, 29
XML parser, 29
MS XML 3.0, 29
Xerces, 29
XML request and response, 24-25
DetermineRequestType( ) function, 182
discovery and lookup server, 4
content searching, 12
querying peers for content, 9
resource sharing, 10
tracing out peers, 8
discovery/lookup/content server, 4-5
content searches, 12
querying peers for content, 9
resource sharing, 10
tracing out peers, 8
Dispose function, 315
Dispose( ) function, 215
DNS class, 38
Download button, 244
DOWNLOAD request type, 105
downloading
design considerations, 19
procedure flow, 38, 40
dropped packets, Gnutella, 7

E
coding, audio/video for streaming, 280
environments, design considerations, 18
error messages, audio/video streaming, 280
352 Index

ERROR structure, 198
errorinfo element, 183
exceptions, 315
    handling
        parsing XML, 115
        XMLParserServer class, 75
Print button, 330
streaming, 315
XML, WriteErrorRequest( ) function, 183

F
file_gui class source code, 134-151
files
    search functionality, design considerations, 35
    sharing
        file_gui class (Java), 134-151
        Shareddialog class (Java), 87-94
firewalls
    application design considerations, 19
    Gnutella, 7
flowcharts, asp files, 55, 60-71
folders
    search functionality, design considerations, 35
    sharing, Shareddialog class (Java), 87-94
FOR XML clause, 28
foreign keys, SQL, 54
forms
    Login source code, 200-217
    search interface source code, 274-279
    shared resource contents source code, 244-273
    sharing resources source code, 217-227
    user interface source code, 227-244
    Windows Media Player, displaying, 314
frmClient class source code, 227-244
frmLogin class source code, 184, 200-217
frmSearch class source code, 274-279
frmSelection class source code, 217-227
frmShare class source code, 244-273
functions
    AcceptFunction( ), 216
    add-on class, 155
    BroadcastStatus, 317
    btnQuit Click( ), 215
    btnShareFileFolder Click( ), 215
    CBroadcastIt class, 287, 288
    connecting to servers, 37
    CreateDocument( ), 182
    ctxMenuQuit( ), 215
    DetermineRequestType( ), 182
    Dispose( ), 215
    GetDataFromListener( ), 313
    GetDataFromServer, 174
    GetDataFromServerUsingProxy( ), 174
    GetLocalTime( ), 301
    GetRequest( ), 174
    GetSelectedItemFromListView( ), 313
    IconUpdate( ), 216
    InitializeComponent( ), 215
    OpenConnection, 330
    parse, 115
    ReadUserName( ), 215
    requests/responses, 115
    SaveAndCloseDocument( ), 182
    SendDataToListener( ), 313
    WriteErrorLog( ), 216
    WriteErrorRequest( ), 183
    WriteRequest( ), 183
    WriteSearchResponse( ), 183
    WriteUserName( ), 215

G
get_AudioDevices function, CBroadcastIt class, 288
get_AudioMedia function, CBroadcastIt class, 288
get_Port function, CBroadcastIt class, 288
get_Profile function, CBroadcastIt class, 288
get_ProfilesList function, CBroadcastIt class, 288
get_UseScript function, CBroadcastIt class, 288
get_VideoMedia function, CBroadcastIt class, 288
GetDataFromListener( ) function, 313
GetDataFromServer function, 174
GetDataFromServerUsingProxy( ) function, 174
GetLocalTime( ) function, 301
GetRequest( ) function, 174
GetResponse( ) function, 37
GetResponseStream( ) function, 37
GetSelectedItemFromListView( ) function, 313
global requests, 13
global searches, 42
global variables, XMLParser class, 198
Globalize Printing add-on, 327-332
Gnutella, 6-7
GUI (graphical user interface)
    buttons
        Login class, 76
        Shareddialog class, 87
    components, initializing, 129
    Java classes, Shareddialog, 87-94

H
| hardware requirements, 345 |
| Horizons, Gnutella, 6 |
| HTTP servers, connecting to, 37 |

**I**

| I/O limitations, PDAs, 333 |
| IANA port assignment list, Web site, 19 |
| IconUpdate() function, 216 |
| iCounter variable, 199 |

| implementation |
| chat client add-on, 320 |
| message writing notification, 326-327 |
| messaging mechanism, 324-326 |
| streaming |
| BroadcastIt component, 288-304 |
| CEvents class, 304-311 |
| browser, 312-315 |
| XML in SQL Server 2000, 27-28 |
| InitializeBroadcaster function, CBroadcastIt class, 288 |
| InitializeComponent() function, 215 |
| InputStream class, 40 |
| installing CD-ROM with book, 347 |
| IP addresses |
| proxy servers, 19 |
| retrieving, 175 |
| storing, 86 |
| iTags variable, 199 |

**J-K**

| J2ME programming considerations, 333-334 |
| Java |
| advantages, 72 |
| connecting to server, 37 |
| InputStream class, 40 |
| OutputStream class, 40 |
| P2P application, classes, 72 |
| ServerSocket class, 31 |
| Socket class, 38 |
| source code on CD-ROM (with book), 341-342 |
| XML parser, Xerces, 29 |
| Jdialog class, dialog titles, 94 |

| keys, SQL, 54 |

**L**

| listener, 13 |
| application example, 14 |
| C# source code on CD-ROM with book, 343 |
| chat client add-on, 321 |

| Java source code on CD-ROM with book, 341 |
| streaming responding to browser requests, 315-317 |
| listener (P2P application) |
| communication considerations, 22 |
| connection to browser, 23-24 |
| design considerations, 29-31 |
| multiple connections, 32-33 |
| ServerSocket class (Java), 31 |
| TCPIListener class (C#), 32 |
| function of, 20 |
| listening for requests, 32-33 |
| operation, 20 |
| source code |
| check_directory class, 105-112 |
| Login class, 76-86 |
| MultiServer class, 95-105 |
| Shareddialog class, 87-94 |
| XMLParserServer class, 73-76 |

| listings |
| add-on.java, 155, 164 |
| check_directory.java, 105, 112 |
| client.java, 119, 128 |
| Declaration of CBroadcastIt class, 283, 287 |
| Declaration of CEvents class, 304, 310 |
| file_gui.java, 135, 149 |
| frmClient, 227, 242 |
| frmLogin, 201, 214 |
| frmSearch, 274, 278 |
| frmSelection, 218, 226 |
| frmShare, 245, 269 |
| Implementation for CBroadcastIt class, 288, 300 |
| login.asp, 57-61 |
| Login.java, 76-85 |
| logout.asp, 62-63, 76 |
| MultiServer.java, 95, 105 |
| search.asp, 67-71 |
| search_screen.java, 152, 154 |
| search_window.java, 130, 134 |
| ServerCommunication Class, 167, 173 |
| Shareddialog.java, 87, 94 |
| Sparsier.java, 113-114 |
| userlist.asp, 64-66 |
| userlist.xml, 66-67 |
| XMLCreator Class, 175, 182, 184 |
| XMLParser class, 184, 198 |
| XMLParserServer.java, 73, 75 |
| XMLWriter.java, 116-117 |
| live chat, streaming and, 282 |
| local searches, 42 |
| logging out, 22 |
Login ASP, 22
Login button, 76
Login class source code, 76-86
Login form source code, 200-217
login.asp, 55-61
LoginID text box, 200
LOGINXML variable, 198
logout.asp, 61-63

M
MakeSession function, CBroadcastIt class, 287
media players, 280
memory
design considerations and, 18
limitations, PDAs, 333
utilization techniques, PDAs, 333
MESSAGE structure, 198
messaging, chat client add-on, 324
receiving messages, 325-326
sending messages, 324-325
methods
declaring, CBroadcastIt class, 287
returnvector class, 75
start_connection, 129
startElement, 76
wild_card, 112
MS XML 3.0 (C# XML parser), 29
MSXML2.IXMLDOMDOCUMENT variable, 199
MSXML2.IXMLDOMELEMENT variable, 199
MSXML2.IXMLDOMNode variable, 199
MSXML2.IXMLDOMNodeList variable, 199
MSXML2.IXMLDOMNodeMap variable, 199
multimedia streaming, 280
multiple connections, design considerations, 32-33
MultiServer class source code, 95-105

N
Napster, 5-6
networks
design considerations and, 18
printing, 327-328
NetworkStream class, 40

O
objects
Dispose functions, 315
GUI, declaring, 86
Open button, 227
OpenConnection function, 330
operation flow of P2P application, 43-51
OutputStream class, 40

P
P2P history, 1-2
P2P applications
Gnutella, 6-7
Napster, 5-6
P2P models
discovery and lookup server, 4
discovery/lookup/content server, 4-5
Pure P2P, 3
simple discovery server, 3-4
p2p_lng tag, 67
P2PServer folder contents (CD-ROM with book), 344
P2PwithAddons folder (CD-ROM with book), 343
packages
CBroadcastIt class, 300
client (P2P application), 115
listener (P2P application), 75
P2P application, C# implementation, 174
packets, Gnutella, 7
parse function, 115
parsing XML files, Sparser class (Java), 112-115
PDAs (Personal Desktop Assistants)
J2ME programming considerations, 333
limitations, 333
search application, 334-340
user interface, 334
peer application class, 334
Peer Table structure, 26
peers
connecting to, 37
querying for required content, 9
tracing out, 8
performance, Java compared to C#, 72
ports
firewalls, design considerations, 19
Gnutella, 7
ranges, 19
PrepareToEncode function, CBroadcastIt class, 288
primary keys, SQL, 54
Print button
adding, 328
coding, 328-333
PRINT request type, 328
printing, networks, 327-332
processor limitations, PDAs, 333
programming
C# disadvantages, 72
constraints, PDAs, 333
J2ME considerations, PDAs, 333
Print button, 328-330
listener, 331-333
search application, PDAs, 334-340
user interface, PDAs, 334
proxy servers, IP addresses, 19
Pure P2P
content searches, 11
querying peers for content, 9
resource sharing, 10
tracing out peers, 8
Pure P2P model, 3
push requests, Gnutella, 7
put_AudioMedia function, CBroadcastIt class, 288
put_Port function, CBroadcastIt class, 288
put_Profile function, CBroadcastIt class, 288
put_UseScript function, CBroadcastIt class, 288
put_VideoMedia function, CBroadcastIt class, 288

Q
queries
required content, 9
XML, returning instead of rowsets, 28
Quit button, 200, 227

R
radio buttons, Shareddialog class, 87
radio stations, maintaining with streaming capability, 282
ReadUserName( ) function, 215
Refresh button, 227
Remember my Login ID checkbox, 76
request types
CHAT, 321
client requests and, 105
PRINT, 328
STREAMING, 283
Request XMLs, 24
requestFString( ) function, 115
requests
streaming, creating, 313
XMLWriter class, 115-118
resource sharing, 10
responding to requests, XMLWriter class, 115-118
Response XMLs, 24
responseFString( ) function, 116
returnRequest( ) function, 115
returnResponse( ) function, 116
returnvector method, 75
root searches, 13

S
SaveAndCloseDocument( ) function, 182
scalability, design considerations, 18
search application, PDAs, 334-340
Search button, 227, 244
search interface form source code, 274-279
SEARCH request type, 105
search.asp, 67-71
search_request( ) function, 155
search_screen class source code, 151-155
search_window class source code, 130-134
searches
design considerations, 35, 40
global searches, 42
local searches, 42
global requests, 13
MultiServer class (Java), 105-112
root searches, 13
search_screen class (Java), 151-155
search_window class (Java), 130-134
server-side, 11
Share.inin file, 86
shared resources, 10-12
searchfiles class, 337
security, design considerations, 19
SELECT statement, FOR XML clause, 28
select statement, SQL, 54
SendDataToListerner( ) function, 313
SendScript function, CBroadcastIt class, 287
SendURL function, CBroadcastIt class, 287
Servents, Gnutella, 6
server
application example, 13
listener relationship, application example, 14-15
server (P2P application)
design considerations, 25-26
function of, 20
ServerCommunication class source code, 167-175
servers
connected user lists, retrieving, 36-37
connecting with HTTP, 37
connecting with URL calls, 37
discovery and lookup, 4
discovery/lookup/content, 4-5
proxy, IP addresses, 19
simple discovery server, 3-4
serversearch class, 336
SERVERSEARCH structure, 198
SERVERSEARCHXML variable, 199
server-side searches, 11
ServerSocket class (Java), 31
variables
  declaring, CBroadcastIt class, 287
  global, XMLParser class, 198
Vector class, 72
video streaming, 280
ViewFiles button, 227
Visual C++, Windows Media Encoder SDK and, 282

W
Web sites
  C# P2P application executables, 167
  IANA port assignment list, 19
  software requirements for CD-ROM with book, 345-347
  Windows Media Player, 280
wild_card method, 112
windows creation, chat client add-on, 321
Windows Media Encoder, 280
Windows Media Encoder SDK
  benefits, 282
  required tools, 282
Windows Media Player
  download site, 280
  form for displaying, 314
WorkingWithXML folder (on CD-ROM with book), 343
WriteErrorLog( ) function, 216
WriteErrorResponse( ) function, 183
WriteRequest( ) function, 183
WriteSearchResponse( ) function, 183
WriteUserName( ) function, 215

X-Y-Z
Xerces (Java XML parser), 29
XML (eXtensible Markup Language)
  browser to listener communications, 23-24
  CreateDocument( ) function, 182
  importance in P2P application, 20
  messages, sending, 324-325
  messaging format, chat client add-on, 321-324
  printing, Globalize Printing add-on, 328
  queries, 28
  request and response design specifications, 24-25
  userlist.xml file, 66-67
XML module
  design considerations, 29
  parsers, 29
    MS XML 3.0, 29
    Xerces, 29
Hungry Minds, Inc.
End-User License Agreement

READ THIS. You should carefully read these terms and conditions before opening the software packet(s) included with this book (“Book”). This is a license agreement (“Agreement”) between you and Hungry Minds, Inc. (“HMI”). By opening the accompanying software packet(s), you acknowledge that you have read and accept the following terms and conditions. If you do not agree and do not want to be bound by such terms and conditions, promptly return the Book and the unopened software packet(s) to the place you obtained them for a full refund.

1. **License Grant.** HMI grants to you (either an individual or entity) a nonexclusive license to use one copy of the enclosed software program(s) (collectively, the “Software”) solely for your own personal and non-commercial purposes on a single computer (whether a standard computer or a workstation component of a multi-user network). The Software is in use on a computer when it is loaded into temporary memory (RAM) or installed into permanent memory (hard disk, CD-ROM, or other storage device). HMI reserves all rights not expressly granted herein.

2. **Ownership.** HMI is the owner of all right, title, and interest, including copyright, in and to the compilation of the Software recorded on the disk(s) or CD-ROM (“Software Media”). Copyright to the individual programs recorded on the Software Media is owned by the author or other authorized copyright owner of each program. Ownership of the Software and all proprietary rights relating thereto remain with HMI and its licensers.

3. **Restrictions on Use and Transfer.**

   (a) You may only (i) make one copy of the Software for backup or archival purposes, or (ii) transfer the Software to a single hard disk, provided that you keep the original for backup or archival purposes. You may not (i) rent or lease the Software, (ii) copy or reproduce the Software through a LAN or other network system or through any computer subscriber system or bulletin-board system, or (iii) modify, adapt, or create derivative works based on the Software.

   (b) You may not reverse engineer, decompile, or disassemble the Software. You may transfer the Software and user documentation on a permanent basis, provided that the transferee agrees to accept the terms and conditions of this Agreement and you retain no copies. If the Software is an update or has been updated, any transfer must include the most recent update and all prior versions.

4. **Restrictions on Use of Individual Programs.** You must follow the individual requirements and restrictions detailed for each individual program in the What’s on the CD-ROM appendix of this Book. These limitations are also contained in the individual license agreements recorded on the Software Media. These limitations may include a requirement that after using the program for a specified period of time, the user must pay a registration fee or discontinue use. By opening the Software packet(s), you will be agreeing to abide by the licenses and restrictions for these individual programs that are detailed in the What’s on the CD-ROM appendix and on the Software Media. None of the material on this Software Media or listed in this Book may ever be redistributed, in original or modified form, for commercial purposes.

5. **Limited Warranty.**

   (a) HMI warrants that the Software and Software Media are free from defects in materials and workmanship under normal use for a period of sixty (60) days from the date of purchase of this Book. If HMI receives notification within the warranty period of defects in materials or workmanship, HMI will replace the defective Software Media.
(b) HMI AND THE AUTHOR OF THE BOOK DISCLAIM ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WITH RESPECT TO THE SOFTWARE, THE PROGRAMS, THE SOURCE CODE CONTAINED THEREIN, AND/OR THE TECHNIQUES DESCRIBED IN THIS BOOK. HMI DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE SOFTWARE WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE SOFTWARE WILL BE ERROR FREE.

(c) This limited warranty gives you specific legal rights, and you may have other rights that vary from jurisdiction to jurisdiction.

6. Remedies.

(a) HMI’s entire liability and your exclusive remedy for defects in materials and workmanship shall be limited to replacement of the Software Media, which may be returned to HMI with a copy of your receipt at the following address: Software Media Fulfillment Department, Attn.: Peer-to-Peer Application Development: Cracking the Code, Hungry Minds, Inc., 10475 Crosspoint Blvd., Indianapolis, IN 46256, or call 1-800-762-2974. Please allow four to six weeks for delivery. This Limited Warranty is void if failure of the Software Media has resulted from accident, abuse, or misapplication. Any replacement Software Media will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer.

(b) In no event shall HMI or the author be liable for any damages whatsoever (including without limitation damages for loss of business profits, business interruption, loss of business information, or any other pecuniary loss) arising from the use of or inability to use the Book or the Software, even if HMI has been advised of the possibility of such damages.

(c) Because some jurisdictions do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitation or exclusion may not apply to you.

7. U.S. Government Restricted Rights. Use, duplication, or disclosure of the Software for or on behalf of the United States of America, its agencies and/or instrumentalities (the "U.S. Government") is subject to restrictions as stated in paragraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause of DFARS 252.227-7013, or subparagraphs (c) (1) and (2) of the Commercial Computer Software - Restricted Rights clause at FAR 52.227-19, and in similar clauses in the NASA FAR supplement, as applicable.

8. General. This Agreement constitutes the entire understanding of the parties and revokes and supersedes all prior agreements, oral or written, between them and may not be modified or amended except in a writing signed by both parties hereto that specifically refers to this Agreement. This Agreement shall take precedence over any other documents that may be in conflict herewith. If any one or more provisions contained in this Agreement are held by any court or tribunal to be invalid, illegal, or otherwise unenforceable, each and every other provision shall remain in full force and effect.
Sun Microsystems, Inc.
Binary Code License Agreement

READ THE TERMS OF THIS AGREEMENT AND ANY PROVIDED SUPPLEMENTAL LICENSE TERMS (COLLECTIVELY "AGREEMENT") CAREFULLY BEFORE OPENING THE SOFTWARE MEDIA PACKAGE. BY OPENING THE SOFTWARE MEDIA PACKAGE, YOU AGREE TO THE TERMS OF THIS AGREEMENT. IF YOU ARE ACCESSING THE SOFTWARE ELECTRONICALLY, INDICATE YOUR ACCEPTANCE OF THESE TERMS BY SELECTING THE "ACCEPT" BUTTON AT THE END OF THIS AGREEMENT. IF YOU DO NOT AGREE TO ALL THESE TERMS, PROMPTLY RETURN THE UNUSED SOFTWARE TO YOUR PLACE OF PURCHASE FOR A REFUND OR, IF THE SOFTWARE IS ACCESSED ELECTRONICALLY, SELECT THE "DECLINE" BUTTON AT THE END OF THIS AGREEMENT.

1. LICENSE TO USE. Sun grants you a non-exclusive and non-transferable license for the internal use only of the accompanying software and documentation and any error corrections provided by Sun (collectively "Software"), by the number of users and the class of computer hardware for which the corresponding fee has been paid.

2. RESTRICTIONS. Software is confidential and copyrighted. Title to Software and all associated intellectual property rights is retained by Sun and/or its licensors. Except as specifically authorized in any Supplemental License Terms, you may not make copies of Software, other than a single copy of Software for archival purposes. Unless enforcement is prohibited by applicable law, you may not modify, decompile, or reverse engineer Software. You acknowledge that Software is not designed, licensed or intended for use in the design, construction, operation or maintenance of any nuclear facility. Sun disclaims any express or implied warranty of fitness for such uses. No right, title or interest in or to any trademark, service mark, logo or trade name of Sun or its licensors is granted under this Agreement.

3. LIMITED WARRANTY. Sun warrants to you that for a period of ninety (90) days from the date of purchase, as evidenced by a copy of the receipt, the media on which Software is furnished (if any) will be free of defects in materials and workmanship under normal use. Except for the foregoing, Software is provided "AS IS". Your exclusive remedy and Sun's entire liability under this limited warranty will be at Sun's option to replace Software media or refund the fee paid for Software.

4. DISCLAIMER OF WARRANTY. UNLESS SPECIFIED IN THIS AGREEMENT, ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT THESE DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

5. LIMITATION OF LIABILITY. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. In no event will Sun's liability to you, whether in contract, tort (including negligence), or otherwise, exceed the amount paid by you for Software under this Agreement. The foregoing limitations will apply even if the above stated warranty fails of its essential purpose.

6. Termination. This Agreement is effective until terminated. You may terminate this Agreement at any time by destroying all copies of Software. This Agreement will terminate immediately without notice from Sun if you fail to comply with any provision of this Agreement. Upon Termination, you must destroy all copies of Software.

7. Export Regulations. All Software and technical data delivered under this Agreement are subject to US export control laws and may be subject to export or import regulations in other countries. You agree to comply strictly with all such laws and regulations and acknowledge that you have the responsibility to obtain such licenses to export, re-export, or import as may be required after delivery to you.
8. U.S. Government Restricted Rights. If Software is being acquired by or on behalf of the U.S. Government or by a U.S. Government prime contractor or subcontractor (at any tier), then the Government's rights in Software and accompanying documentation will be only as set forth in this Agreement; this is in accordance with 48 CFR 227.7201 through 227.7202-4 (for Department of Defense (DOD) acquisitions) and with 48 CFR 2.101 and 12.212 (for non-DOD acquisitions).

9. Governing Law. Any action related to this Agreement will be governed by California law and controlling U.S. federal law. No choice of law rules of any jurisdiction will apply.

10. Severability. If any provision of this Agreement is held to be unenforceable, this Agreement will remain in effect with the provision omitted, unless omission would frustrate the intent of the parties, in which case this Agreement will immediately terminate.

11. Integration. This Agreement is the entire agreement between you and Sun relating to its subject matter. It supersedes all prior or contemporaneous oral or written communications, proposals, representations and warranties and prevails over any conflicting or additional terms of any quote, order, acknowledgment, or other communication between the parties relating to its subject matter during the term of this Agreement. No modification of this Agreement will be binding, unless in writing and signed by an authorized representative of each party.

Java(TM) 2 Software Development Kit (J2SDK), Standard Edition, Version 1.3 SUPPLEMENTAL LICENSE TERMS

These supplemental license terms ("Supplemental Terms") add to or modify the terms of the Binary Code License Agreement (collectively, the "Agreement"). Capitalized terms not defined in these Supplemental Terms shall have the same meanings ascribed to them in the Agreement. These Supplemental Terms shall supersede any inconsistent or conflicting terms in the Agreement, or in any license contained within the Software.

1. Software Internal Use and Development License Grant. Subject to the terms and conditions of this Agreement, including, but not limited to Section 4 (Java(TM) Technology Restrictions) of these Supplemental Terms, Sun grants you a non-exclusive, non-transferable, limited license to reproduce internally and use internally the binary form of the Software complete and unmodified for the sole purpose of designing, developing and testing your Java applets and applications intended to run on the Java platform ("Programs").

2. License to Distribute Software. Subject to the terms and conditions of this Agreement, including, but not limited to Section 4 (Java (TM) Technology Restrictions) of these Supplemental Terms, Sun grants you a non-exclusive, non-transferable, limited license to reproduce and distribute the Software in binary code form only, provided that (i) you distribute the Software complete and unmodified and only bundled as part of, and for the sole purpose of running, your Programs, (ii) the Programs add significant and primary functionality to the Software, (iii) you do not distribute additional software intended to replace any component(s) of the Software, (iv) you do not remove or alter any proprietary legends or notices contained in the Software, (v) you only distribute the Software subject to a license agreement that protects Sun's interests consistent with the terms contained in this Agreement, and (vi) you agree to defend and indemnify Sun and its licensors from and against any damages, costs, liabilities, settlement amounts and/or expenses (including attorneys' fees) incurred in connection with any claim, lawsuit or action by any third party that arises or results from the use or distribution of any and all Programs and/or Software.

3. License to Distribute Redistributables. Subject to the terms and conditions of this Agreement, including but not limited to Section 4 (Java Technology Restrictions) of these Supplemental Terms, Sun grants you a non-exclusive, non-transferable, limited license to reproduce and distribute the binary form of those files specifically identified as redistributable in the Software "README" file ("Redistributables") provided that: (i) you distribute the Redistributables complete and unmodified (unless otherwise specified in the applicable README file), and only bundled as part of Programs, (ii) you do not distribute additional software intended to supersede any component(s) of the Redistributables, (iii) you do not remove or alter any proprietary legends or notices contained in or on the Redistributables, (iv) you only distribute the Redistributables pursuant to a license agreement that protects Sun's interests consistent with the terms contained in the Agreement, and (v) you agree to defend and indemnify Sun and its licensors from and against any damages, costs, liabilities, settlement amounts and/or expenses (including attorneys' fees) incurred in connection
with any claim, lawsuit or action by any third party that arises or results from the use or
distribution of any and all Programs and/or Software.

4. Java Technology Restrictions. You may not modify the Java Platform Interface ("JPI"), identified as
classes contained within the "java" package or any subpackages of the "java" package, by creating
additional classes within the JPI or otherwise causing the addition to or modification of the classes
in the JPI. In the event that you create an additional class and associated API(s) which (i) extends
the functionality of the Java platform, and (ii) is exposed to third party software developers for the
purpose of developing additional software which invokes such additional API, you must promptly
publish broadly an accurate specification for such API for free use by all developers. You may not
create, or authorize your licensees to create, additional classes, interfaces, or subpackages that are
in any way identified as "java", "javax", "sun" or similar convention as specified by Sun in any
naming convention designation.

5. Trademarks and Logos. You acknowledge and agree as between you and Sun that Sun owns the
SUN, SOLARIS, JAVA, JINI, FORTE, STAROFFICE, STARPORTAL and iPLANET trademarks
and all SUN, SOLARIS, JAVA, JINI, FORTE, STAROFFICE, STARPORTAL and iPLANET-
related trademarks, service marks, logos and other brand designations ("Sun Marks"), and you
agree to comply with the Sun Trademark and Logo Usage Requirements currently located at
http://www.sun.com/policies/trademarks. Any use you make of the Sun Marks inures to Sun's
benefit.

6. Source Code. Software may contain source code that is provided solely for reference purposes
pursuant to the terms of this Agreement. Source code may not be redistributed unless expressly
provided for in this Agreement.

7. Termination for Infringement. Either party may terminate this Agreement immediately should any
Software become, or in either party's opinion be likely to become, the subject of a claim of
infringement of any intellectual property right.

For inquiries please contact: Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, California
94303
License Agreement: Forte for Java, release 2.0
Community Edition for All Platforms

To obtain Forte for Java, release 2.0, Community Edition for All Platforms, you must agree to the software license below.

Sun Microsystems Inc., Binary Code License Agreement

READ THE TERMS OF THIS AGREEMENT AND ANY PROVIDED SUPPLEMENTAL LICENSE TERMS (COLLECTIVELY "AGREEMENT") CAREFULLY BEFORE OPENING THE SOFTWARE MEDIA PACKAGE. BY OPENING THE SOFTWARE MEDIA PACKAGE, YOU AGREE TO THE TERMS OF THIS AGREEMENT. IF YOU ARE ACCESSING THE SOFTWARE ELECTRONICALLY, INDICATE YOUR ACCEPTANCE OF THESE TERMS BY SELECTING THE "ACCEPT" BUTTON AT THE END OF THIS AGREEMENT. IF YOU DO NOT AGREE TO ALL THESE TERMS, PROMPTLY RETURN THE UNUSED SOFTWARE TO YOUR PLACE OF PURCHASE FOR A REFUND OR, IF THE SOFTWARE IS ACCESSED ELECTRONICALLY, SELECT THE "DECLINE" BUTTON AT THE END OF THIS AGREEMENT.

1. LICENSE TO USE. Sun grants you a non-exclusive and non-transferable license for the internal use only of the accompanying software and documentation and any error corrections provided by Sun (collectively "Software"), by the number of users and the class of computer hardware for which the corresponding fee has been paid.

2. RESTRICTIONS. Software is confidential and copyrighted. Title to Software and all associated intellectual property rights is retained by Sun and/or its licensors. Except as specifically authorized in any Supplemental License Terms, you may not make copies of Software, other than a single copy of Software for archival purposes. Unless enforcement is prohibited by applicable law, you may not modify, decompile, or reverse engineer Software. You acknowledge that Software is not designed, licensed or intended for use in the design, construction, operation or maintenance of any nuclear facility. Sun disclaims any express or implied warranty of fitness for such uses. No right, title or interest in or to any trademark, service mark, logo or trade name of Sun or its licensors is granted under this Agreement.

3. LIMITED WARRANTY. Sun warrants to you that for a period of ninety (90) days from the date of purchase, as evidenced by a copy of the receipt, the media on which Software is furnished (if any) will be free of defects in materials and workmanship under normal use. Except for the foregoing, Software is provided "AS IS". Your exclusive remedy and Sun's entire liability under this limited warranty will be at Sun's option to replace Software media or refund the fee paid for Software.

4. DISCLAIMER OF WARRANTY. UNLESS SPECIFIED IN THIS AGREEMENT, ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT THESE DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

5. LIMITATION OF LIABILITY. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. In no event will Sun's liability to you, whether in contract, tort (including negligence), or otherwise, exceed the amount paid by you for Software under this Agreement. The foregoing limitations will apply even if the above stated warranty fails of its essential purpose.

6. Termination. This Agreement is effective until terminated. You may terminate this Agreement at any time by destroying all copies of Software. This Agreement will terminate immediately without notice from Sun if you fail to comply with any provision of this Agreement. Upon Termination, you must destroy all copies of Software.

7. Export Regulations. All Software and technical data delivered under this Agreement are subject to US export control laws and may be subject to export or import regulations in other countries. You
agree to comply strictly with all such laws and regulations and acknowledge that you have the responsibility to obtain such licenses to export, re-export, or import as may be required after delivery to you.

8. U.S. Government Restricted Rights. If Software is being acquired by or on behalf of the U.S. Government or by a U.S. Government prime contractor or subcontractor (at any tier), then the Government's rights in Software and accompanying documentation will be only as set forth in this Agreement; this is in accordance with 48 CFR 227.7201 through 227.7202-4 (for Department of Defense (DOD) acquisitions) and with 48 CFR 2.101 and 12.212 (for non-DOD acquisitions).

9. Governing Law. Any action related to this Agreement will be governed by California law and controlling U.S. federal law. No choice of law rules of any jurisdiction will apply.

10. Severability. If any provision of this Agreement is held to be unenforceable, this Agreement will remain in effect with the provision omitted, unless omission would frustrate the intent of the parties, in which case this Agreement will immediately terminate.

11. Integration. This Agreement is the entire agreement between you and Sun relating to its subject matter. It supersedes all prior or contemporaneous oral or written communications, proposals, representations and warranties and prevails over any conflicting or additional terms of any quote, order, acknowledgment, or other communication between the parties relating to its subject matter during the term of this Agreement. No modification of this Agreement will be binding, unless in writing and signed by an authorized representative of each party.

JAVA™ DEVELOPMENT TOOLS FORTE™ FOR JAVA™, RELEASE 2.0, COMMUNITY EDITION SUPPLEMENTAL LICENSE TERMS

These supplemental license terms ("Supplemental Terms") add to or modify the terms of the Binary Code License Agreement (collectively, the "Agreement"). Capitalized terms not defined in these Supplemental Terms shall have the same meanings ascribed to them in the Agreement. These Supplemental Terms shall supersede any inconsistent or conflicting terms in the Agreement, or in any license contained within the Software.

1. Software Internal Use and Development License Grant. Subject to the terms and conditions of this Agreement, including, but not limited to Section 3 (Java(TM) Technology Restrictions) of these Supplemental Terms, Sun grants you a non-exclusive, non-transferable, limited license to reproduce internally and use internally the binary form of the Software complete and unmodified for the sole purpose of designing, developing and testing your [Java applets and] applications intended to run on the Java platform ("Programs").

2. License to Distribute Redistributables. In addition to the license granted in Section 1 (Redistributables Internal Use and Development License Grant) of these Supplemental Terms, subject to the terms and conditions of this Agreement, including, but not limited to Section 3 (Java Technology Restrictions) of these Supplemental Terms, Sun grants you a non-exclusive, non-transferable, limited license to reproduce and distribute those files specifically identified as redistributable in the Software “README” file (“Redistributables”) provided that: (i) you distribute the Redistributables complete and unmodified (unless otherwise specified in the applicable README file), and only bundled as part of your Programs, (ii) you do not distribute additional software intended to supercede any component(s) of the Redistributables, (iii) you do not remove or alter any proprietary legends or notices contained in or on the Redistributables, (iv) for a particular version of the Java platform, any executable output generated by a compiler that is contained in the Software must (a) only be compiled from source code that conforms to the corresponding version of the OEM Java Language Specification; (b) be in the class file format defined by the corresponding version of the OEM Java Virtual Machine Specification; and (c) execute properly on a reference runtime, as specified by Sun, associated with such version of the Java platform, (v) you only distribute the Redistributables pursuant to a license agreement that protects Sun's interests consistent with the terms contained in the Agreement, and (vi) you agree to defend and indemnify Sun and its licensors from and against any damages, costs, liabilities, settlement amounts and/or expenses (including attorneys' fees) incurred in connection with any
claim, lawsuit or action by any third party that arises or results from the use or distribution of any
and all Programs and/or Software.

3. Java Technology Restrictions. You may not modify the Java Platform Interface ("JPI", identified as
classes contained within the "java" package or any subpackages of the "java" package), by creating
additional classes within the JPI or otherwise causing the addition to or modification of the classes
in the JPI. In the event that you create an additional class and associated API(s) which (i) extends
the functionality of the Java platform, and (ii) is exposed to third party software developers for the
purpose of developing additional software which invokes such additional API, you must promptly
publish broadly an accurate specification for such API for free use by all developers. You may not
create, or authorize your licensees to create, additional classes, interfaces, or subpackages that are
in any way identified as "java", "javax", "sun" or similar convention as specified by Sun in any
naming convention designation.

4. Java Runtime Availability. Refer to the appropriate version of the Java Runtime Environment
binary code license (currently located at http://www.java.sun.com/jdk/index.html) for the
availability of runtime code which may be distributed with Java applets and applications.

5. Trademarks and Logos. You acknowledge and agree as between you and Sun that Sun owns the
SUN, SOLARIS, JAVA, JINI, FORTE, STAROFFICE, STARPORTAL and iPLANET trademarks
and all SUN, SOLARIS, JAVA, JINI, FORTE, STAROFFICE, STARPORTAL and iPLANET-
related trademarks, service marks, logos and other brand designations ("Sun Marks"), and you
agree to comply with the Sun Trademark and Logo Usage Requirements currently located at
http://www.sun.com/policies/trademarks. Any use you make of the Sun Marks inures to Sun's
benefit.

6. Source Code. Software may contain source code that is provided solely for reference purposes
pursuant to the terms of this Agreement. Source code may not be redistributed unless expressly
provided for in this Agreement.

7. Termination for Infringement. Either party may terminate this Agreement immediately should any
Software become, or in either party's opinion be likely to become, the subject of a claim of
infringement of any intellectual property right.

For inquiries please contact: Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, California
94303