



Jaypee University of Information Technology
Solan (H.P.)

LEARNING RESOURCE CENTER

Acc. Num. SP04010 Call Num:

General Guidelines:

- ◆ Library books should be used with great care.
- ◆ Tearing, folding, cutting of library books or making any marks on them is not permitted and shall lead to disciplinary action.
- ◆ Any defect noticed at the time of borrowing books must be brought to the library staff immediately. Otherwise the borrower may be required to replace the book by a new copy.
- ◆ The loss of LRC book(s) must be immediately brought to the notice of the Librarian in writing.

Learning Resource Centre-JUIT



SP04010

P2P System

**Submitted in partial fulfillment of the Degree of
Bachelor of Technology**

By

Vasu Gupta(041211)

Praneet Tandon(041279)

Kavitt Sharma(041421)



DEPARTMENT OF COMPUTER SCIENCE

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY

WAKNAGHAT

MAY-2008

Abstract

Transferring of data in connected machines is not easy. Either one has to connect remotely to another machine or use external media to transfer data. Usage of external media requires a person to physically go to a machine, copy data first to the external device and then onto his/her own machine. In the case of a Remote connection, the data transfer rate is slow and till the time data is being copied, one machine cannot be used as it becomes locked as soon as a remote connection is made. Our tool helps in transferring files from one machine to another without any obstruction of work of any of the connected users. All files being shared by one client are accessible to all the clients who are logged in. Hence this tool helps in transferring files shared on one machine to another on a network with greater efficiency.

CERTIFICATE

This is to certify that the work entitled, "P2P System" submitted by Vasu Gupta, Praneet Tandon and Kavitt Sharma in partial fulfillment for the award of degree of Bachelor of Technology in Computer Science of Jaypee University of Information Technology has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.



Supervisor: _____

Mr. Vipin Arora

Lecturer,

Department of Computer Science and Information Technology,

Jaypee University of Information Technology,

Waknaghat, Solan-172315, Himachal Pradesh,

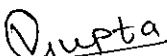
INDIA.


ACKNOWLEDGEMENT

We wish to express our earnest gratitude to Mr. Vipin Arora, for providing us invaluable guidance and suggestions, which inspired us to submit this project on time.

We would also like to thank all the staff members of Computer Science and Engineering Department of Jaypee University of Information Technology, Waknaghat, for providing us all the facilities required for the completion of this project.

Last but not least we wish to thank all our classmates and friends for their timely suggestions and cooperation during the period of our project.


Vasu Gupta (041211)


Praneet Tandon (041279)



Kavitt Sharma (041421)

Table of Contents

List of figures	V
List of abbreviations	VII
1. Project Basics	1
i. Project Aim	1
ii. Project Scope	1
iii. Introduction to the Platform	2
iv. Chapterization and Theoretical Basics	6
2. Project Design	7
i. Methodology and Approach	7
ii. Implementation	8
iii. Software Development Cycle	22
iv. ER Diagram	23
v. Data Flow Diagram	24
vi. Flow Chart	27
vii. Event Diagram	28
viii. Testing	29
3. Graphical User Interface Details	33
4. Source Code	39
5. Conclusion and Future Scope	89
6. References	90
7. Installation Guide	91

List of figures

Fig 2.1: User interface of the Project

Fig 2.2.1: Basics of Discovery Service

Fig 2.2.2: Search queries

Fig 2.2.3: File Upload Information Display

Fig 2.2.4: Download Information Display

Fig 2.2.5: Downloading Display

Fig 2.3: Lifecycle of the project

Fig 2.4: Entity-Relationship Diagram

Fig 2.5.1: Zero-Level Data Flow Diagram

Fig 2.5.2: Level-One Data Flow Diagram

Fig 2.5.3: Level-Two Data Flow Diagram

Fig 2.6: Flowchart

Fig 2.7: Event Diagram

Fig 2.8.1: Graph Based Method

Fig 2.8.2: Boundary Value Analysis

Fig 3.1.1: User Interface of Prototype

Fig 3.1.2: User Interface of Final Tool

Fig 3.1.3: Download Information Display

Fig 3.1.4: File Upload Information Display

Fig 3.1.5: Client Settings Display

List of Abbreviations

1. P2P – Peer to peer
2. .Net – Visual Basic. Net 2005 (VB. Net)
3. IIS - Internet Indexing Service
4. NAT - Network Address Translation
5. GUID – Global unique Identifier
6. LAN – Local Area Network
7. SQL – Structured Query Language

All other abbreviations that were used sparingly have been explained wherever they were used.

1. PROJECT BASICS:

1.1 PROJECT AIM:

The project aims on the development of a user friendly tool which can transfer files across a network. It will be used to transfer shared files from one machine to another.

1.2 PROJECT SCOPE:

The tool is a Peer to Peer system. It will enable users to transfer files shared on machines across a network. It will allow them to continue working without any hindrance as the files they request are being copied in the background. It is a better option than a Remote Desktop Connection which stops the usage of the machine that is being accessed.

1.3 Introduction to the Platform

VB.Net

Visual Basic .NET provides the easiest, most productive language and tool for rapidly building Windows and Web applications. Visual Basic .NET comes with enhanced visual designers, increased application performance, and a powerful integrated development environment (IDE). It also supports creation of applications for wireless, Internet-enabled hand-held devices. The following are the features of Visual Basic .NET.

1. Powerful Windows-based Applications

Visual Basic .NET comes with features such as a powerful new forms designer, an in-place menu editor, and automatic control anchoring and docking. Visual Basic .NET delivers new productivity features for building more robust applications easily and quickly. With an improved integrated development environment (IDE) and a significantly reduced startup time, Visual Basic .NET offers fast, automatic formatting of code as you type, an enhanced object browser and XML designer, and much more.

2. Building Web-based Applications

With Visual Basic .NET we can create Web applications using the shared Web Forms Designer and the familiar "drag and drop" feature. You can double-click and write code to respond to events. Visual Basic .NET 2003 comes with an enhanced HTML Editor for working with complex Web pages.

3. Simplified Deployment

With Visual Basic .NET we can build applications more rapidly and deploy and maintain them with efficiency. Visual Basic .NET 2003 and .NET Framework 1.1 makes "DLL Hell" a thing of the past. Side-by-side versioning enables multiple versions of the same component to live safely on the same machine so that applications can use a specific version of a component. XCOPY-deployment and Web auto-download of Windows-

based applications combine the simplicity of Web page deployment and maintenance with the power of rich, responsive Windows-based applications.

4. Powerful, Flexible, Simplified Data Access

You can tackle any data access scenario easily with ADO.NET and ADO data access. The flexibility of ADO.NET enables data binding to any database, as well as classes, collections, and arrays, and provides true XML representation of data. Seamless access to ADO enables simple data access for connected data binding scenarios. Using ADO.NET, Visual Basic .NET can gain high-speed access to MS SQL Server, Oracle, DB2, Microsoft Access, and more.

5. Improved Coding

You can code faster and more effectively. A multitude of enhancements to the code editor, smart listing of code for greater readability and a background compiler for real-time notification of syntax errors transforms into a rapid application development (RAD) coding machine.

6. Direct Access to the Platform

Visual Basic developers can have full access to the capabilities available in .NET Framework 1.1. Developers can easily program system services including the event log, performance counters and file system. The new Windows Service project template enables to build real Microsoft Windows NT Services. Programming against Windows Services and creating new Windows Services is not available in Visual Basic .NET Standard, it requires Visual Studio 2003 Professional, or higher.

7. Full Object-Oriented Constructs

One can create reusable, enterprise-class code using full object-oriented constructs. Language features include full implementation inheritance, encapsulation, and

polymorphism. Structured exception handling provides a global error handler and eliminates spaghetti code.

8. XML Web Services

XML Web services enable you to call components running on any platform using open Internet protocols. Working with XML Web services is easier where enhancements simplify the discovery and consumption of XML Web services that are located within any firewall. XML Web services can be built as easily as you would build any class in Visual Basic 6.0. The XML Web service project template builds all underlying Web service_infrastructure.

9. Mobile Applications

Visual Basic .NET 2003 and the .NET Framework 1.1 offer integrated support for developing mobile Web applications for more than 200 Internet-enabled mobile devices. These new features give developers a single, mobile Web interface and programming model to support a broad range of Web devices.

10. COM Interoperability

One can maintain existing code without the need to recode. COM interoperability enables you to leverage your existing code assets and offers seamless bi-directional communication between Visual Basic 6.0 and Visual Basic .NET applications.

11. Reuse Existing Investments

You can reuse all your existing ActiveX Controls. Windows Forms in Visual Basic .NET 2003 provide a robust container for existing ActiveX controls. In addition, full support for existing ADO code and data binding enable a smooth transition to Visual Basic .NET 2003.

12. Upgrade Wizard

We can upgrade our code to receive all of the benefits of Visual Basic .NET 2003. The Visual Basic .NET Upgrade Wizard, available in Visual Basic .NET 2003 Standard Edition, and higher, upgrades up to 95 percent of existing Visual Basic code and forms to Visual Basic .NET with new support for Web classes and User Controls.

1.4 CHAPTERISATION AND THEORETICAL BASIS:

- 1 Project Overview, approach, methodology.
- 2 Project design including finalization of specification.
- 3 First review of project design.
- 4 Implementation of prototype, input, output and user Interface.
- 5 Second review of project design.
- 6 Coding design-flowchart, file design, data representation, testing procedure.
- 7 Coding.
- 8 Final Testing and Releases.
- 9 Final documentation and completion.

PROJECT DESIGN:

2.1 METHODOLOGY & APPROACH:

This project is divided into two modules

1. GUI development.
2. Implementation.

GUI development module gives the complementary view of the desired system behavior which is user compatible and easy to use. File Transfer will help in realization of the goal.

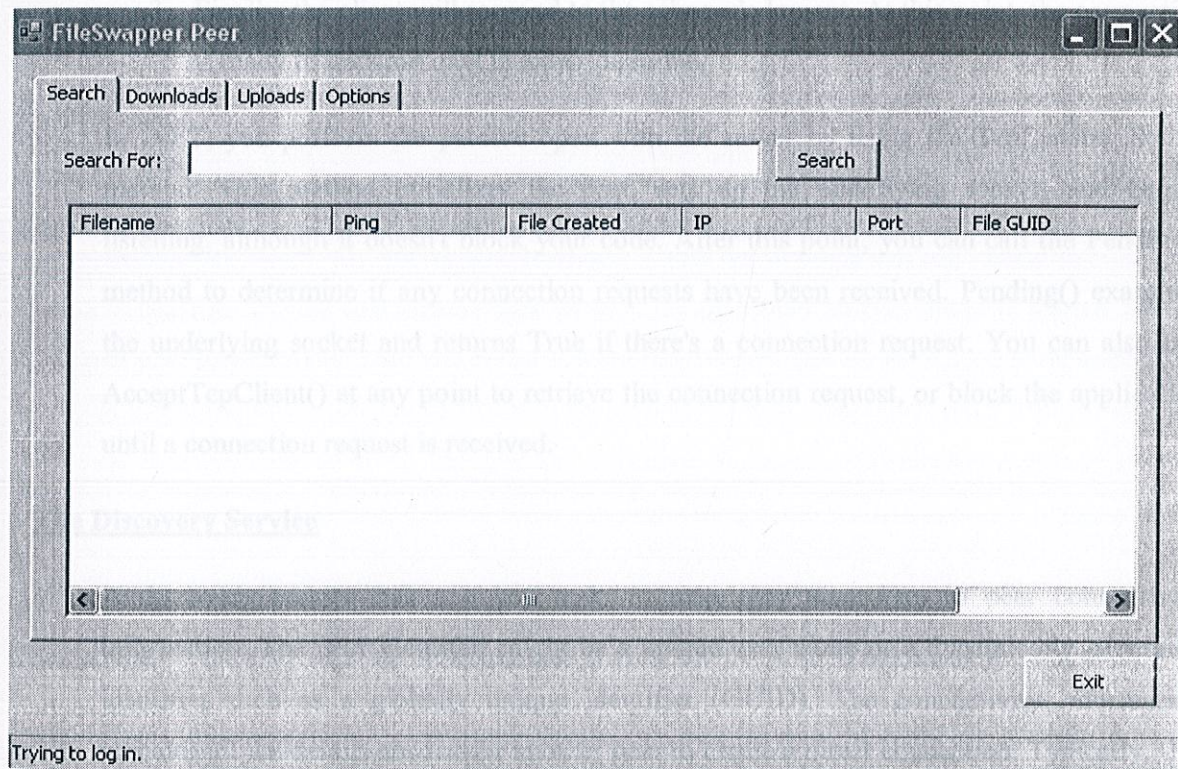


Fig 2.1: User Interface of the Project

2.2 Implementation:

Communicating with TCP

TCP connections require a three-stage handshaking mechanism:

1. First, the server must enter listening mode by performing a passive open. At this point, the server will be idle, waiting for an incoming request.
2. A client can then use the IP address and port number to perform an active open. The server will respond with an acknowledgment message in a predetermined format that incorporates the client sequence number.
3. Finally, the client will respond to the acknowledgment. At this point, the connection is ready to transmit data in either direction.

In .NET, you perform the passive open with the server by using the `TcpListener.Start()` method. This method initializes the port, sets up the underlying socket, and begins listening, although it doesn't block your code. After this point, you can call the `Pending()` method to determine if any connection requests have been received. `Pending()` examines the underlying socket and returns `True` if there's a connection request. You can also call `AcceptTcpClient()` at any point to retrieve the connection request, or block the application until a connection request is received.

The Discovery Service

A discovery service has one key task: to map peer identifiers to peer connectivity information. The peer identifier might be a unique user name or a dynamically generated identifier such as a globally unique identifier (GUID). The connectivity information includes all the details needed for another peer to create a direct connection. Typically, this includes an IP address and port number, although this information could be wrapped up in a higher-level construct. For example, the coordination server that we used in the Remoting chat application stores a proxy (technically, that encapsulates the IP address and port number as well as other details such as the remote class type and version).

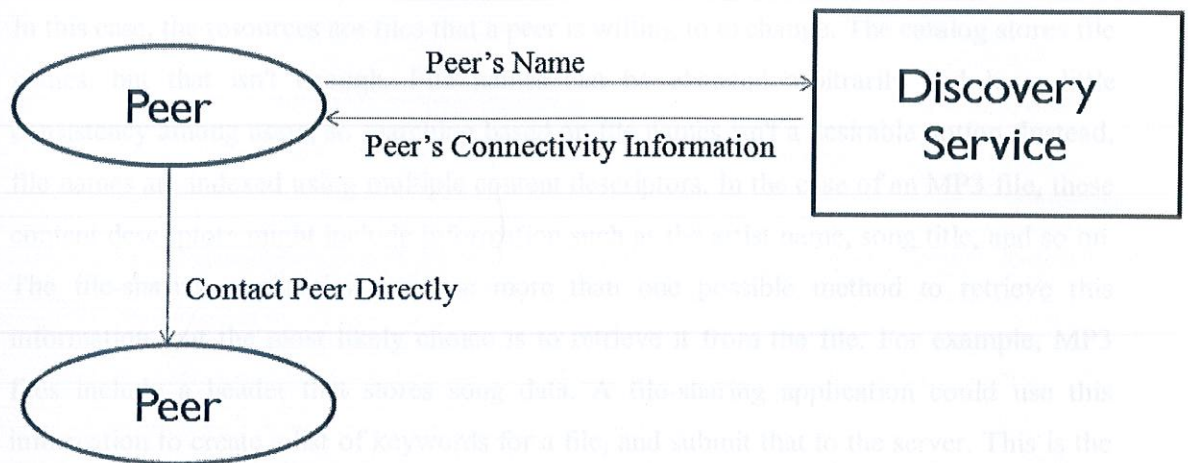


Fig 2.2.1: Basics of Discovery Service

In addition, a discovery service might provide information about the resources a peer provides. For example, in the file-sharing application, a peer creates a query based on a file name or keyword. The server then responds with a list of peers that can satisfy that request. In order to provide this higher-level service, the discovery service needs to store a catalog of peer information. This makes the system more dependent on its central component, and it limits the ways that you can search, because the central component must expect the types of searches and have all the required catalogs. However, if your searches are easy to categorize, this approach greatly improves performance and reduces network bandwidth.

The Registration Database

The registration database stores a list of all the peers that are currently available and the information needed to connect to them. It also uses a basic cataloging system, whereby each peer uploads a catalog of available resources shortly after logging in. When a peer needs a specific resource, it calls a web-service method. The web service attempts to find peers that can provide the resource and returns a list of search matches with the required peer-connectivity information.

In this case, the resources are files that a peer is willing to exchange. The catalog stores file names, but that isn't enough. File names can be changed arbitrarily and have little consistency among users, so searching based on file names isn't a desirable option. Instead, file names are indexed using multiple content descriptors. In the case of an MP3 file, these content descriptors might include information such as the artist name, song title, and so on. The file-sharing application can use more than one possible method to retrieve this information, but the most likely choice is to retrieve it from the file. For example, MP3 files include a header that stores song data. A file-sharing application could use this information to create a list of keywords for a file, and submit that to the server. This is the approach taken in our sample registration database.

Creating the Database

The registration database consists of three tables. These tables include the following:

1. The Peers table lists currently connected peers, each of which is assigned a unique GUID. The peer-connectivity information includes the numeric IP address (stored as a string in dotted notation) and port number. The Peers table also includes a LastUpdate time, which allows an expiration policy to be used to remove old peer registration records.
2. The Files table lists shared files, the peer that's sharing them, and the date stamp on the file. Each file has a unique GUID, thereby ensuring that they can be tracked individually.
3. The Keywords table lists a single-word descriptor for a file. You'll notice that the Keywords table is linked to both the Files table and the Peers table. This makes it easier to delete the keywords related to a peer if the peer registration expires, without having to retrieve a list of shared files.

Working of the Discovery Service

Now that the actual data-access logic has been written, the actual discovery web service will need very little code. For the most part, its methods simply wrap the P2PDatabase component. All exceptions are caught, logged, and suppressed, so that sensitive information will not be returned to the client, who is in no position to correct low-level database errors anyway.

A typical interaction with the discovery service goes as follows:

1. The client generates a new GUID to identify itself, records its current IP address and port, and calls Register() with this information.
2. The client inspects the files that it's sharing, creates the keywords lists, and calls PublishFiles() to submit the catalog.
3. After this point, the client calls RefreshRegistration() periodically, to prevent its login information from expiring.
4. Optionally, the client calls SearchForFile() with any queries.
5. The client ends the session by calling Unregister().

An Overview of FileSwapper

The FileSwapper application is built around a single. This form uses multiple tables and allows users to initiate searches, configure settings, and monitor uploads and downloads.

FileSwapper divides its functionality into a small army of classes, including the following:

- SwapperClient, which is the main form class. It delegates as much work as possible to other classes and uses a timer to periodically update its login information with the discovery service.
- Global, which includes the data that's required application-wide (for example, registry settings).

- App, which includes shared methods for some of the core application tasks such as Login(), Logout(), and PublishFiles(), and also provides access to the various application threads.
- KeywordUtil and MP3Util, which provide a few shared helper methods for analyzing MP3 files and parsing the keywords that describe them.
- RegistrySettings, which provides access to the application's configuration settings, along with methods for saving and loading them.
- ListViewItemWrapper, which performs thread-safe updating of a ListViewItem.
- Search, which contacts the discovery service with a search request on a separate thread (allowing long-running searches to be easily interrupted).
- FileServer and FileUpload, which manage the incoming connections and transfer shared files to interested peers.
- FileDownloadQueue and FileDownloadClient, which manage in-progress downloads from other peers.
- Messages, which defines constants used for peer-to-peer communication.

The file-transfer process is fairly easy. Once a peer locates another peer that has an interesting file, it opens a direct TCP/IP connection and sends a download request. However, the application is still fairly complex because it needs to handle several tasks that require multithreading at once. Because every peer acts as both a client and a server, every application needs to simultaneously monitor for new incoming connections that are requesting files. In addition, the application must potentially initiate new outgoing connections to download other files. Not only does the client need to perform uploading and downloading at the same time, but it also needs to be able to perform multiple uploads or downloads at once (within reason). In order to accommodate this design, a separate thread needs to work continuously to schedule new uploads or downloads as required.

Global Data and Tasks

The FileSwapper application uses two classes that consist of shared members: Global and App. These classes act like global modules and are available from any location in the FileSwapper code. That means that you don't need to create an instance of these classes—instead, their properties and methods are always available and you can access them through the class name.

The Global class stores data that's required by multiple objects in the application. As with any application, it's always best to keep the amount of global data to a minimum. A large number of global variables usually indicates a poor structure that really isn't object-oriented. All the data in the Global class is held using public shared variables, making it widely available. For example, any code in the application can use the Global.Identity property to access information about the current computer's IP address and port number settings.

The App class also relies on shared variables to store a common set of information. It actually stores references to three separate objects, each of which will be executed on an independent thread. Using the App class, your startup code can easily initialize the threads on startup and abort them when the application is about to end. The App class also includes a private shared variable that references the web-service proxy. This ensures that no other part of the application can access the discovery service directly—instead, the application must call one of the App class methods.

Utility Functions

There are three utility classes in the FileSwapper: RegistrySettings, MP3Util, and KeywordUtil. All of them use shared methods to provide helper functions.

The first class, RegistrySettings, wraps access to the Windows registry. It allows the application to store and retrieve machine-specific information. You could replace this class with code that reads and writes settings in an application configuration file, but the drawback would be that multiple users couldn't load the same client application file from a network (as they would end up sharing the same configuration file).

The RegistrySettings class provides five settings as public variables and two methods. The Load() method retrieves the values from the specified key and configures the public variables. The Save() method stores the current values in the appropriate locations.

The final utility class is KeywordUtil. It includes a single shared method—ParseKeywords()—that takes a string which contains a list of keywords, and splits it into words wherever a space, comma, or period is found. This step is performed using the built-in String.Split() method. Thus, if you index an MP3 file that has the artist "Linkin Park," the keyword list will include two entries: "Linkin" and "Park". This allows a peer to search with both or only one of these terms.

Thread-Safe ListViewItem Updates

This is usually the case when updating one of the three main ListView controls in the FileSwapper: the upload status display, the download status display, and the search-result listing.

For the first two cases, there's a direct mapping between threads and ListViewItem items. For example, every concurrent upload requires exactly one ListViewItem to display ongoing status information. To simplify the task of creating and updating the ListViewItem, FileSwapper includes a wrapper class called ListViewItemWrapper.

The Main Form

When the main form first loads, it reads the registry, updates the configuration window with the retrieved settings, starts the other threads, and then logs in. The login is actually a multiple step procedure. First, the peer information is submitted with the App.Login() method. Next, the file catalog is created and submitted with the App.PublishFiles() method. Finally, the timer is enabled to automatically update the login information as required.

While the peer is sending data to the discovery web service, the mouse pointer is changed to an hourglass, and the text in the status bar panel is updated to reflect what's taking place.

Searches

The Search class is the first of three custom-threaded objects used by FileSwapper. As part of any search, FileSwapper attempts to contact each peer with a network ping (the equivalent of asking "are you there?"). FileSwapper measures the time it takes for a response and any errors that occur, and then displays this information in the search results. This allows the user to decide where to send a download request, depending on which peer is fastest.

The drawback of this approach is that pinging each peer could take a long time, especially if some peers are unreachable. This in itself isn't a problem, provided the user has some way to cancel a long-running search and start a new one. To implement this approach, the Search class uses custom threading code.

Threading the Search class may seem easy, but it runs into the classic userinterface problem. In order to display the results in the ListView, the user-interface code must be marshaled to the main application thread using the `Control.Invoke()` method. This isn't difficult, but it is an added complication.

The Search class needs to track several pieces of information:

- The thread it's using to execute the search.
- Its current state (searching, not searching).
- The search keywords.
- The ListView where it should write search results.
- The SearchResults it retrieves.
- The ping times it calculates.

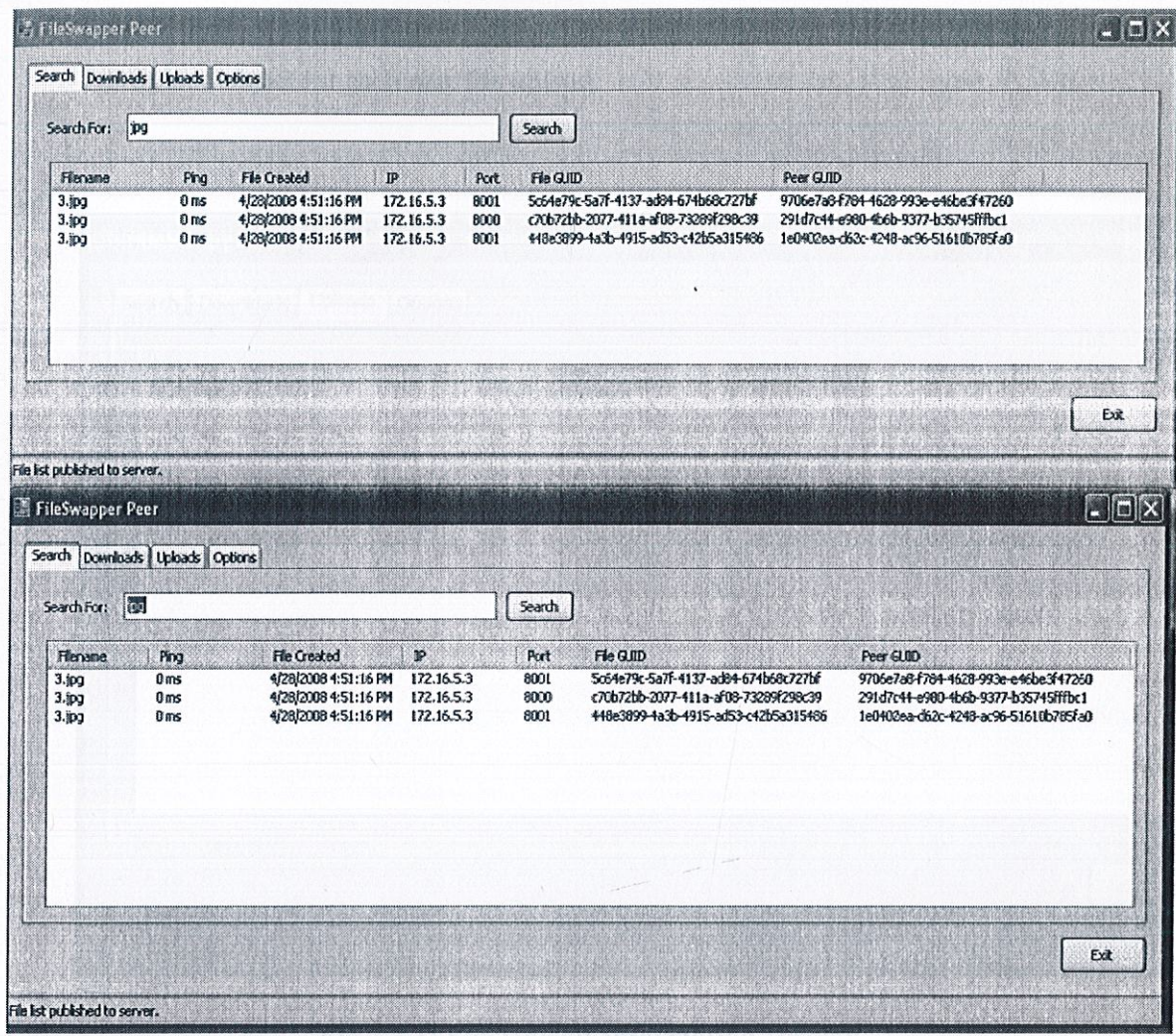


Fig 2.2.2: Search queries

Uploads

The file uploading and downloading logic represents the heart of the FileSwapper application. The user needs the ability not only to perform both of these operations at the same time, but also to serve multiple upload requests or download multiple files in parallel. To accommodate this requirement, we must use a two-stage design, in which one class is responsible for creating new upload or download objects as needed. In the case of an

upload, this is the FileServer class. The FileServer waits for requests and creates a FileUpload object for each new file upload.

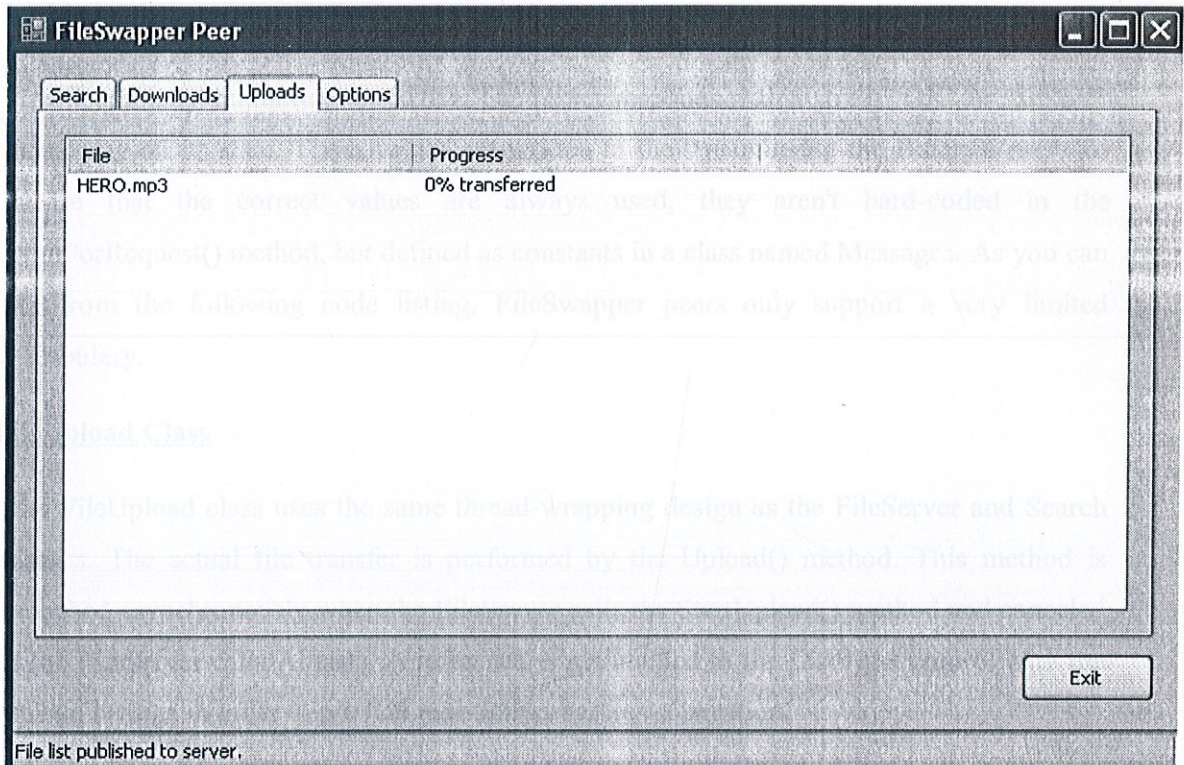


Fig 2.2.3: File Upload Information Display

The FileServer Class

The FileServer class listens for connection requests on the defined port using a TcpListener. It follows the same pattern as the asynchronous Search class:

- The thread used to monitor the port is stored in a private member variable.
- The thread is created with a call to StartWaitForRequest(), and aborted with a call to Abort(). The actual monitoring code exists in the WaitForRequest() method.
- The ListView that tracks uploads is stored in a private member variable.

This framework is shown in the following code listing. One of the differences you'll notice is that an additional member variable is used to track individual upload threads. The `Abort()` method doesn't just stop the thread that's waiting for connection requests—it also aborts all the threads that are currently transferring files.

`FileSwapper` peers communicate using simple string messages. A peer requests a file for downloading by submitting its GUID. The server responds with a string "OK" or "BUSY" depending on its state. These values are written to the stream using the `BinaryWriter`. To ensure that the correct values are always used, they aren't hard-coded in the `WaitForRequest()` method, but defined as constants in a class named `Messages`. As you can see from the following code listing, `FileSwapper` peers only support a very limited vocabulary.

The FileUpload Class

The `FileUpload` class uses the same thread-wrapping design as the `FileServer` and `Search` classes. The actual file transfer is performed by the `Upload()` method. This method is launched asynchronously when the `FileServer` calls the `StartUpload()` method and canceled if the `FileServer` calls `Abort()`. A reference is maintained to the `ListView` control with the upload listings in order to provide real-time progress information.

Downloads

The file-downloading process is similar to the file-uploading process. A `FileDownloadQueue` class creates `FileDownloadClient` instances to serve new user requests, provided the maximum number of simultaneous downloads hasn't been reached. Download progress information is written directly to the download `ListView` display, using the thread-safe `ListViewItemWrapper`.

A download operation begins when a user double-clicks an item in the `ListView` search results, thereby triggering the `ItemActivate` event. The form code handles the event, checks that the requested file hasn't already been submitted to the `FileDownloadQueue`, and then adds it. This code demonstrates another advantage of using GUIDs to uniquely identify all

files on the peer-to-peer network: it allows each peer to maintain a history of downloaded files.

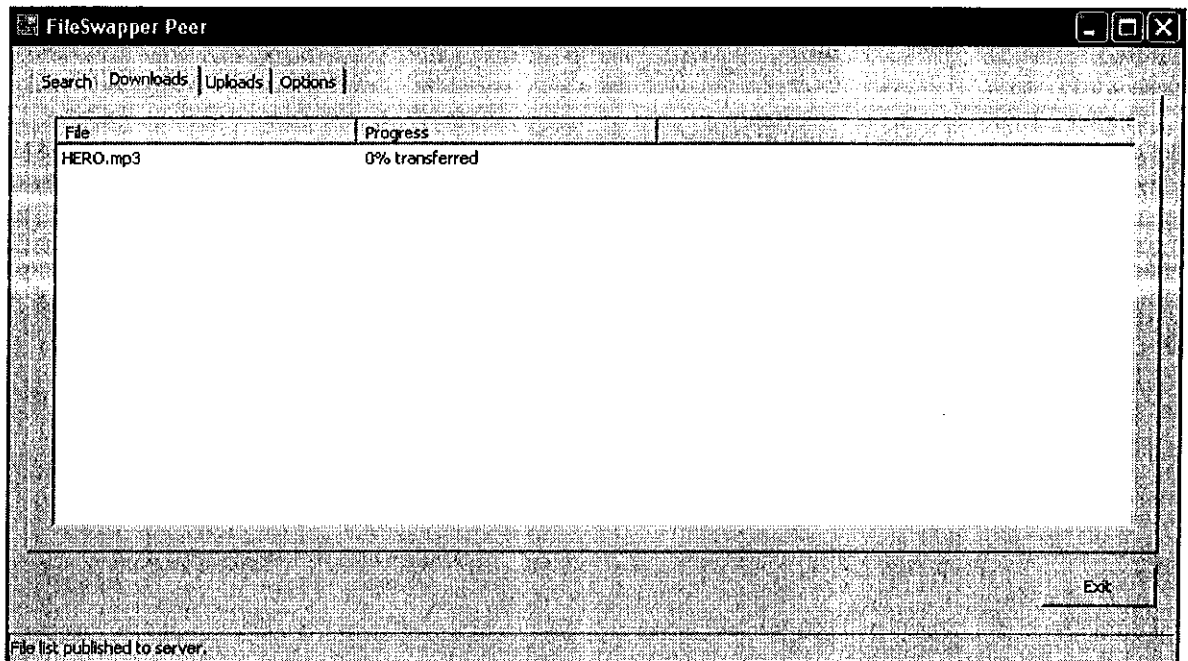


Fig 2.2.4: Download Information Display

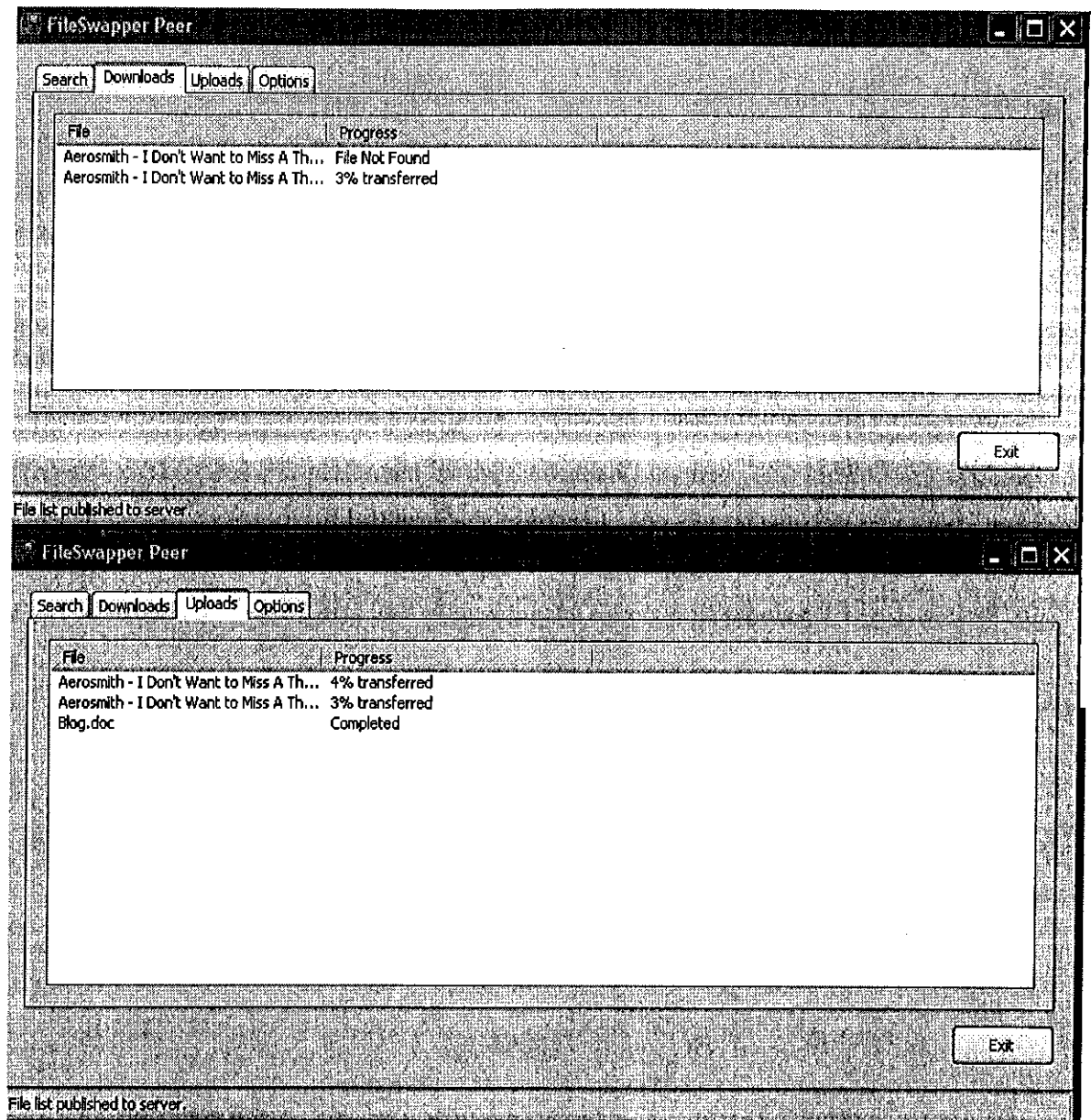


Fig 2.2.5: Downloading Display

The FileDownloadQueue Class

The FileDownloadQueue tracks and schedules ongoing downloads. When the user requests a file, it's added to the QueuedFiles collection. If the maximum download thread count hasn't yet been reached, the file is removed from this collection and a new FileDownloadClient object is created to serve the request. All active FileDownloadClient objects are tracked in the DownloadThreads collection.

The FileDownloadClient Class

The FileDownloadClient uses the same thread-wrapping design as the FileUpload class. The actual file transfer is performed by the Download() method. This method is launched asynchronously when the FileDownloadQueue calls the StartDownload() method, and canceled if the FileDownloadQueue calls Abort(). The current SharedFile and ListViewItem information is tracked using a private DisplayFile property.



Fig 2.1 Lifecycle of the project

2.3 SOFTWARE DEVELOPMENT LIFECYCLE:

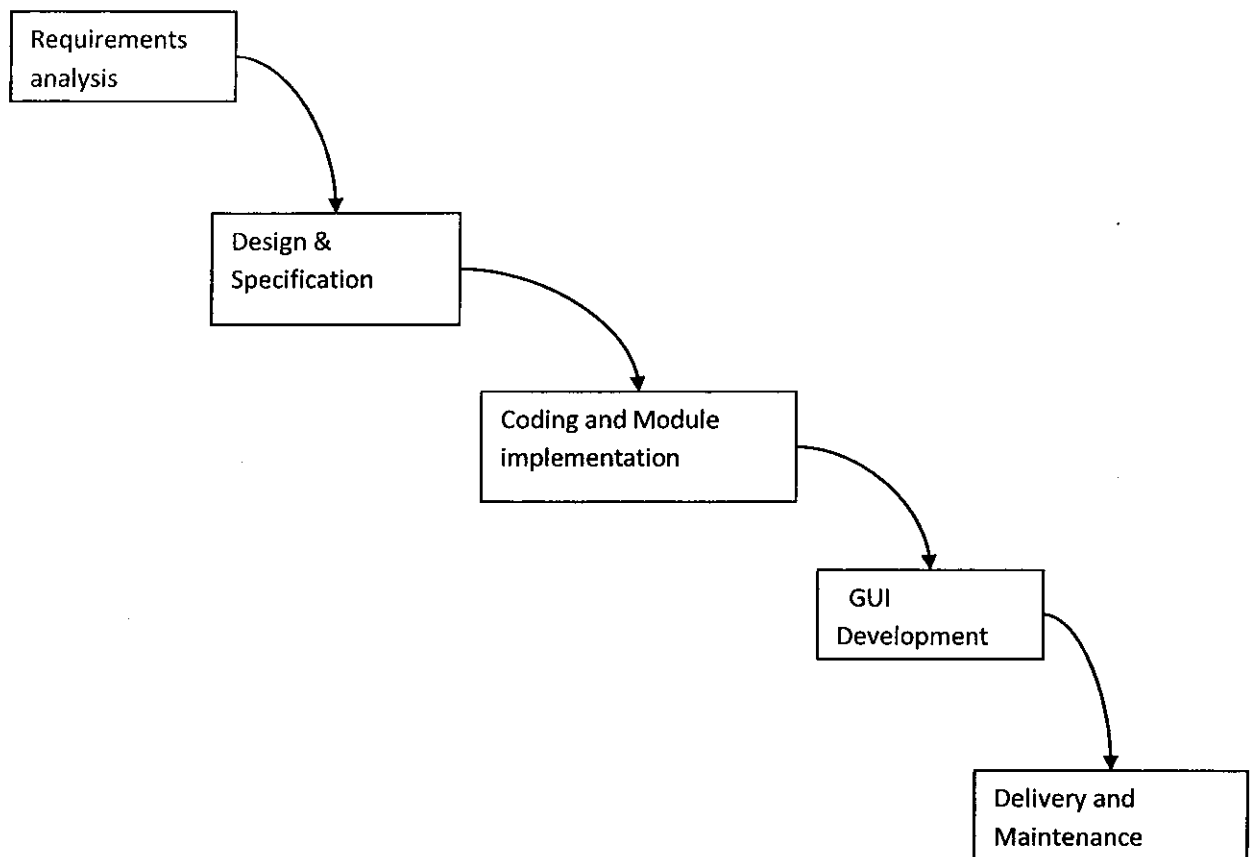


Fig 2.3 Lifecycle of the project

2.4 Entity-Relationship Diagram:

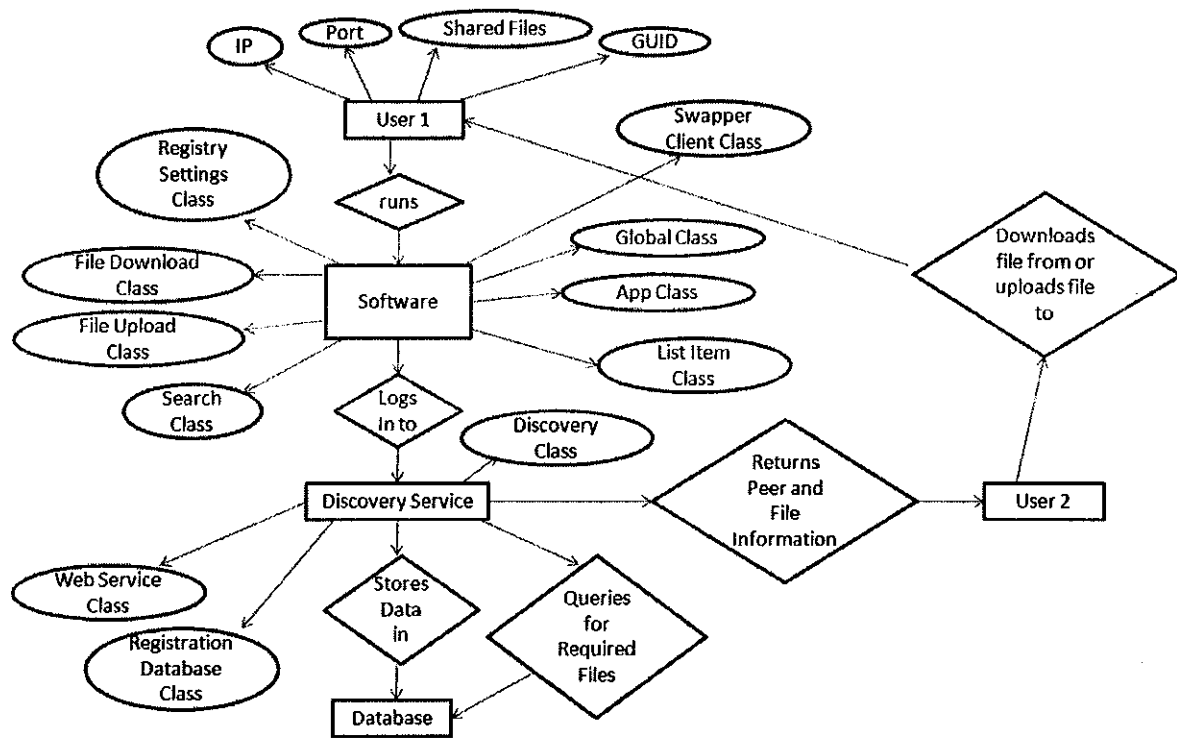


Fig 2.4 Entity-Relationship Diagram

2.5 Data Flow Diagram:

2.5.1 Zero-Level Data Flow Diagram:

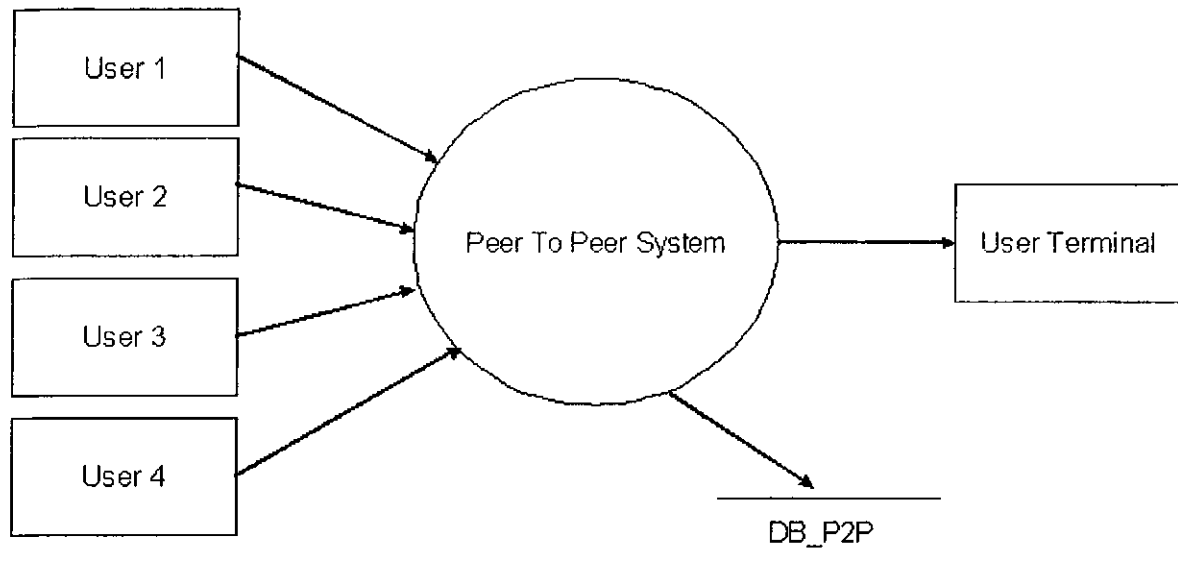


Fig 2.5.1 Zero-Level Data Flow Diagram

2.5.2 Level-One Data Flow Diagram:

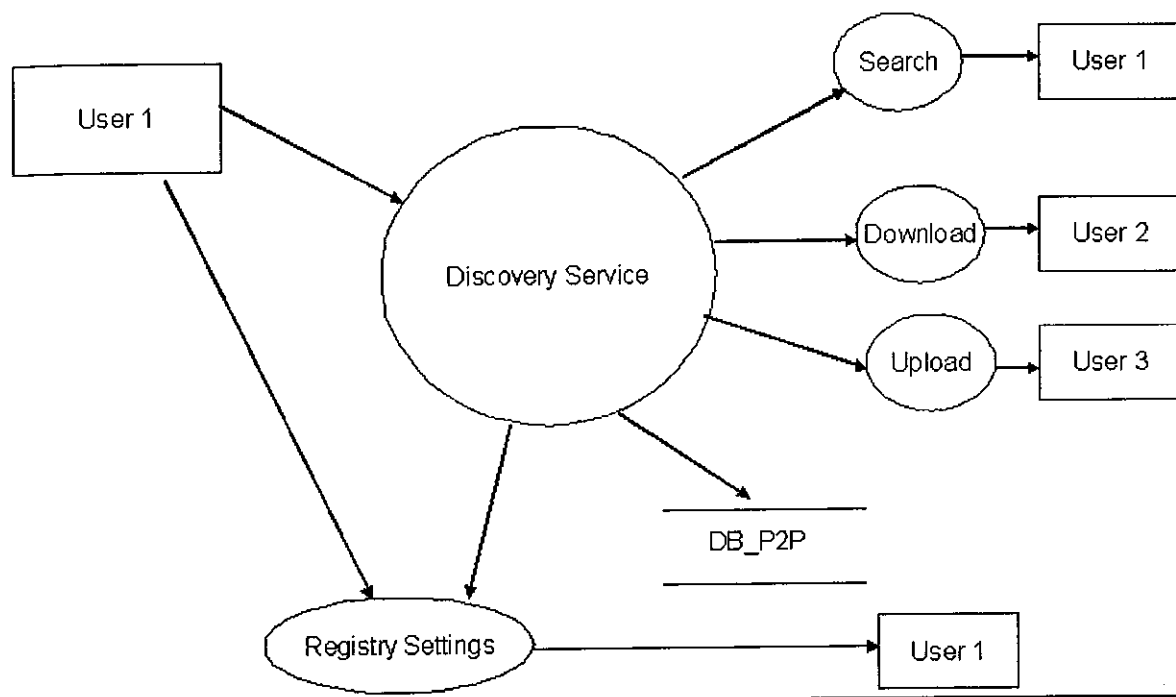


Fig 2.5.2 Level-One Data Flow Diagram

2.5.3 Level-Two Data Flow Diagram:

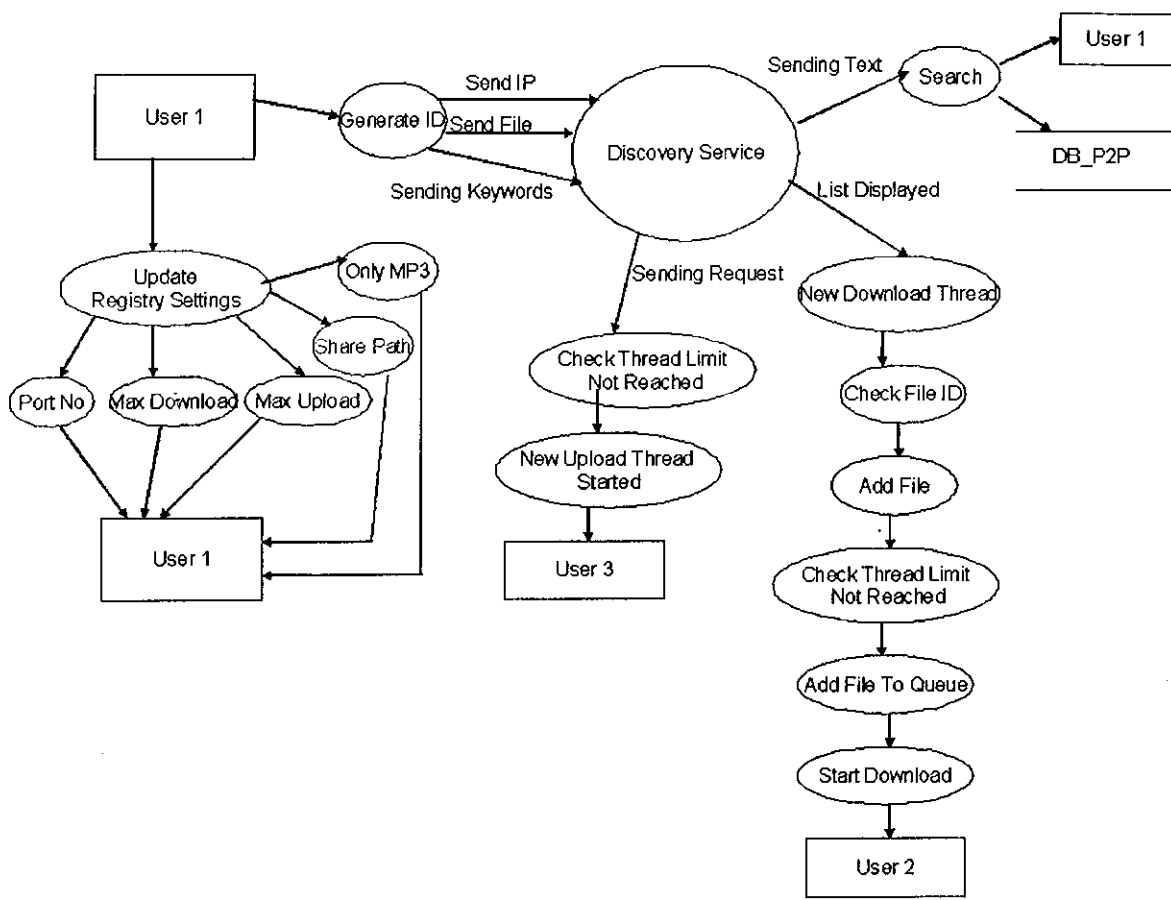


Fig 2.5.3 Level-Two Data Flow Diagram

2.6 FLOWCHART:

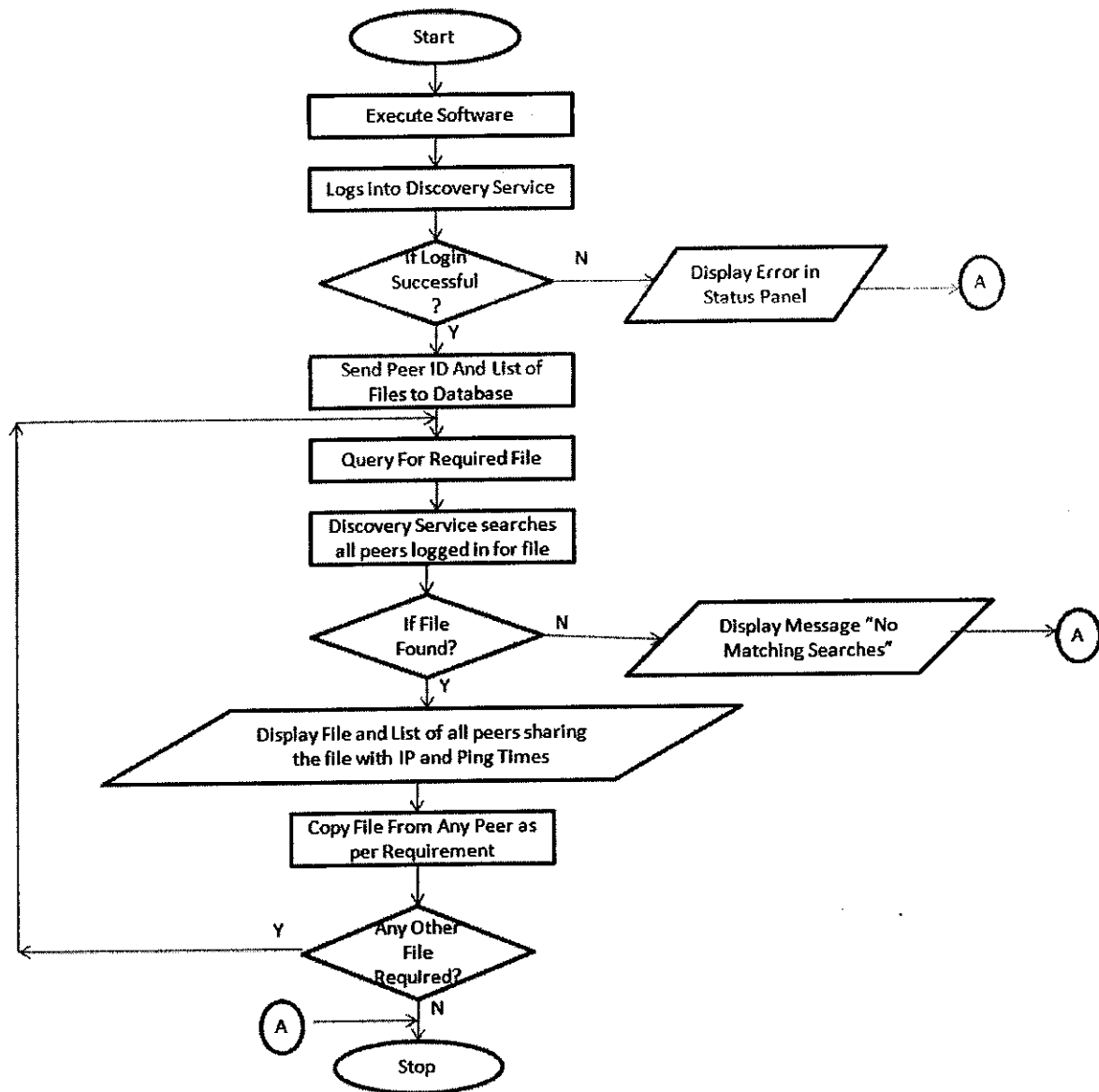


Fig 2.6 Flowchart

2.7 EVENT DIAGRAM:

Event diagrams are constructed through event lists which are list of stimuli coming from the environment to which the system must respond. According to the responses events are described as Flow Oriented Temporal and Control events.

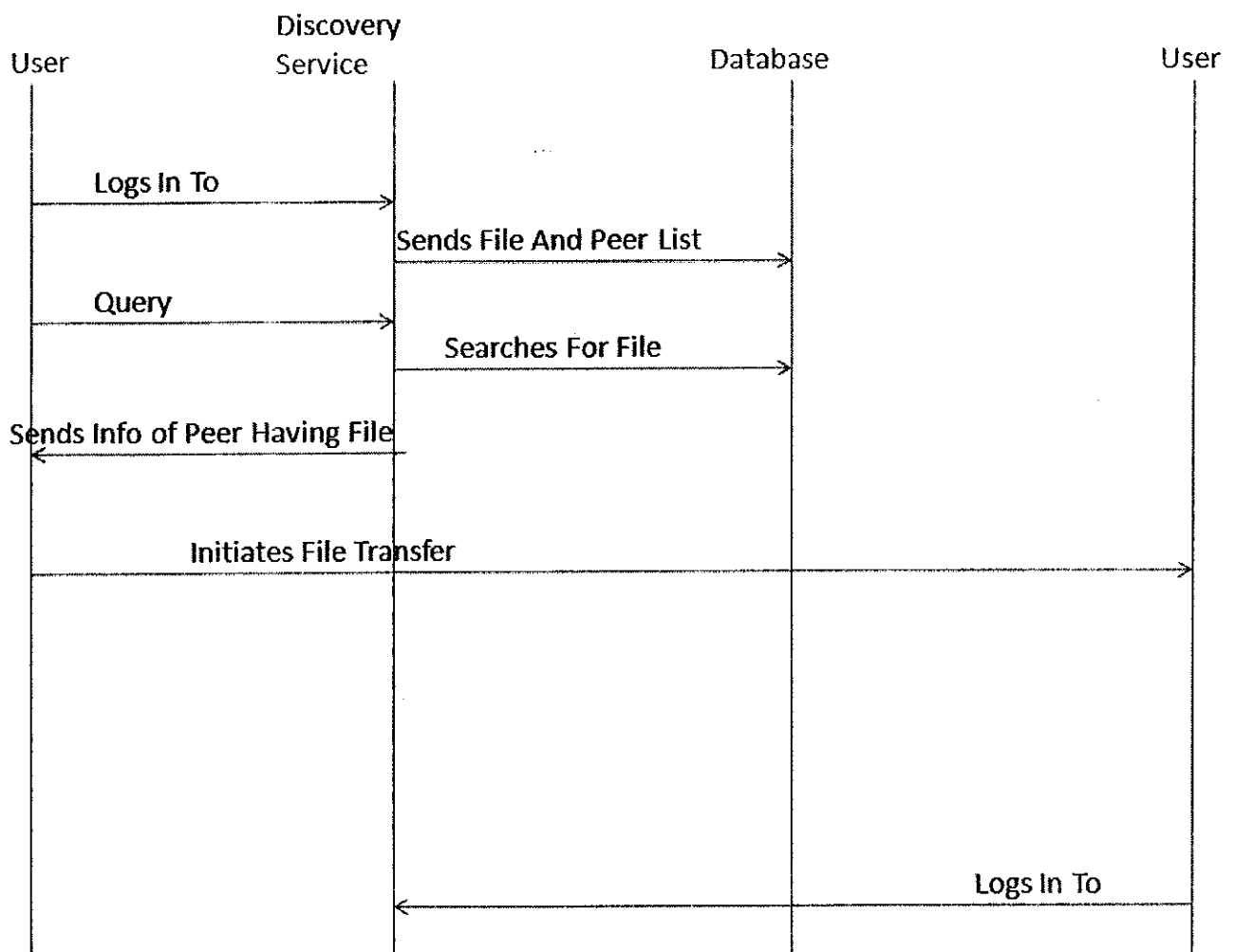


Fig 2.7 Event Diagram

2.8 TESTING:

Unit Testing:

The first consideration for any peer-to-peer application is how peers will discover one another on the network and retrieve the information they need to communicate.

We identified four units namely Web-Service Client, Registration Database Discovery Service & File Swapper.

Web-Service Client Class was tested whether it can cross different firewalls or not as it is requested like a web page over a HTTP channel which allows it to do so. Web Services are created to serve a single client request and are destroyed immediately when the request ends & needs to be created again for a new request so it was tested for multiple requests at a time.

Registration Database Class was tested for the IP string, port, Filenames, tags, keywords, user ID stored is valid or not. It was also tested for the updates and deletes performed by the client is reflected in the database class at the same time.

Discovery Service simply wraps the P2P Database component. All exceptions are caught, logged, and suppressed. Discovery Service was first tested on the localhost whether it is successfully running and identifying the user and the files shared by the user.

After testing on a single user it was tested on the Local Area Network by adding the discovery file of a user to other users connected to the service.

Integrated Testing:

The units were combined together to build the application which was checked whether threads cause any exception handling or show any atypical behavior.

It was successfully tested for a maximum of 10 users at a time & is scalable also.

BLACK BOX Testing:

Graph Based Test Method

In this method, the first step is to understand the objects that are modeled in the software and the relationships that connect these objects. Once this has been accomplished, the next step is to define a series of tests that verify “all objects have the expected relationship to one another”. Stated in another way, software testing begins by creating a graph of important objects and their relationships and then devising a series of tests that will cover the graph so that each object and relationship is exercised and errors are uncovered.

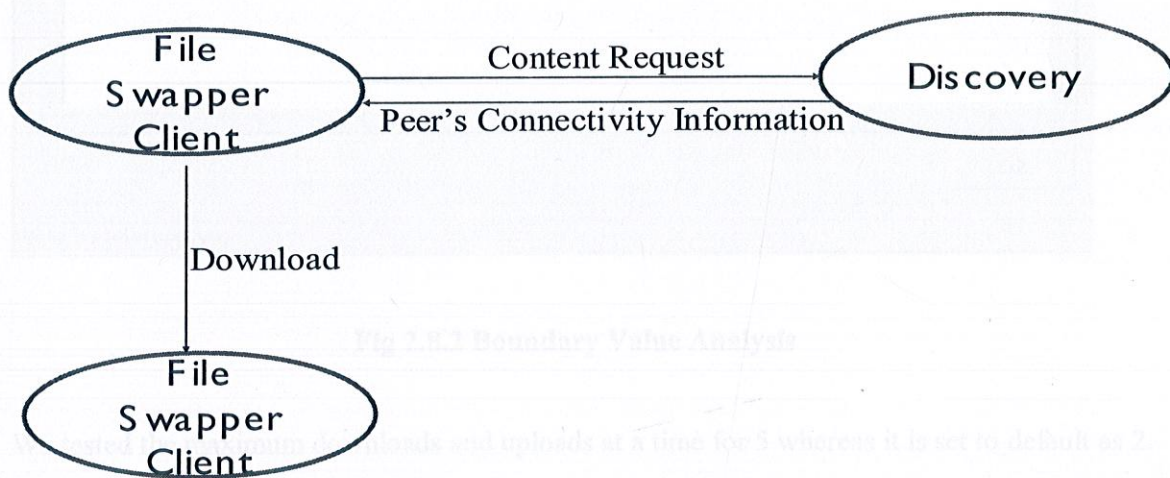


Fig 2.8.1: Graph Based Method

Boundary Value Analysis

Boundary value analysis is a test case design technique that complements equivalence partitioning. Rather than selecting any element of an equivalence class, BVA leads to the selection of test cases at the edges of the class. Rather than focusing solely on input conditions, BVA derives test cases from the output domain as well.

It was tested for downloading and uploading multiple files at a time also when the download limit is reached the file is queued for downloading later once in progress files are completed.

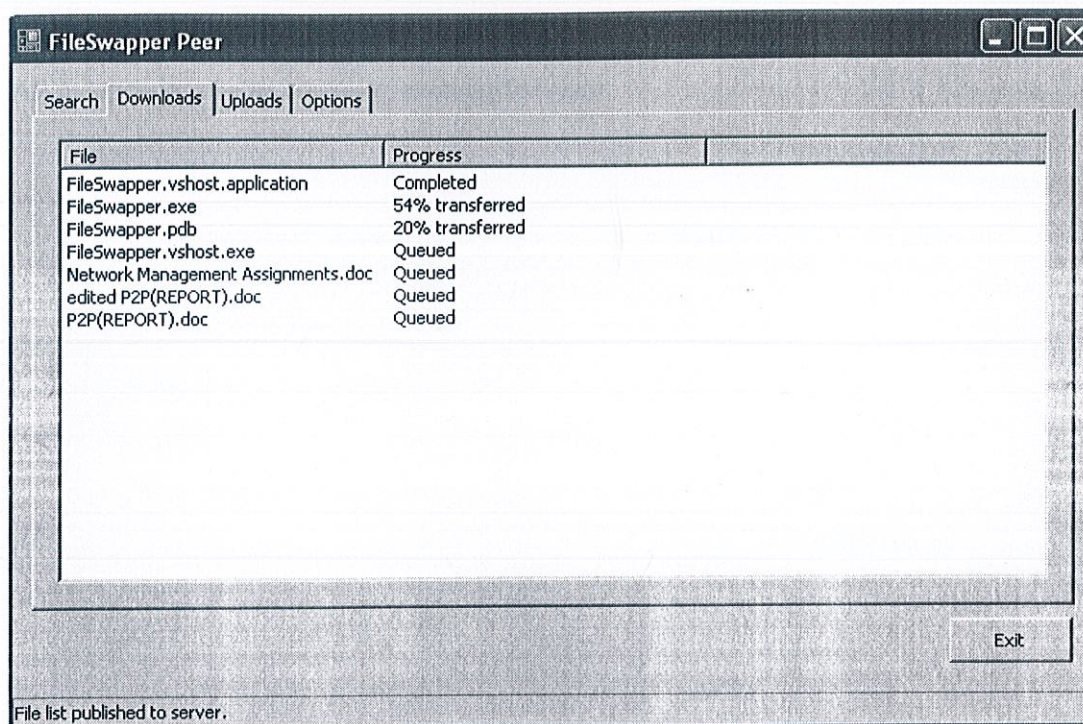


Fig 2.8.2 Boundary Value Analysis

We tested the maximum downloads and uploads at a time for 5 whereas it is set to default as 2. Also Ports should range from 1024 to 65000.

It was specifically tested for sharing Mp3 files only & then implemented for sharing different types of file.

White Box Testing

It is a test case design that uses the control structure described as part of component-level design to derive test cases.

Using white-box testing following points were kept in consideration:

- All Data Structures were used.
- All logical decisions were tested for True & False cases.

- All loops were tested at boundary values.
- All independent paths are being exercised at once.

3. GUI Details

We have implemented the prototype with a very simple interface. We have implemented the client and server on the same windows form so as to provide simplicity of use.

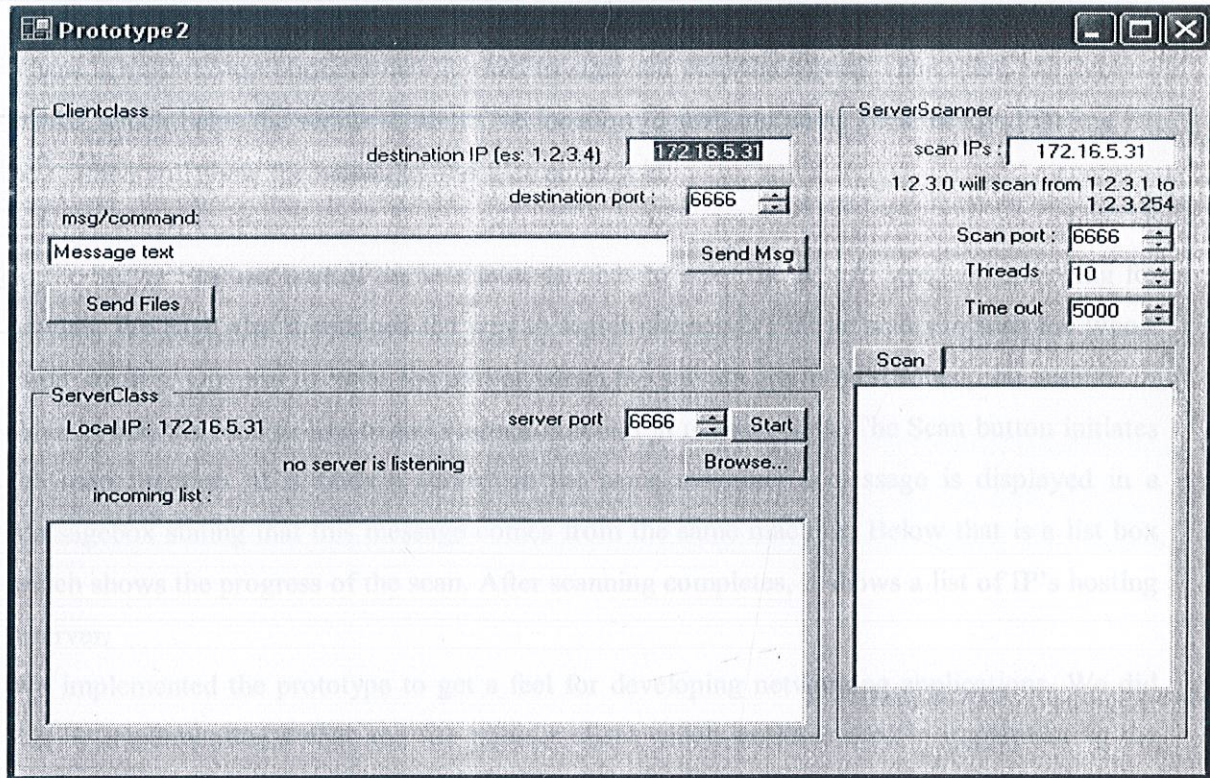


Fig 3.1.1:GUI of Prototype

In the Client part of the interface, one has to insert the destination IP, that is, the IP of the server and the Port on which the server is running. There is a list box in which the user can type the message he/she wants to send to the server. Then there is a send message button which executes the send function to send this message to the server. Also there is a button for the transfer of files. This button opens the select file to send dialog box. One can browse through their hard disk drive to select the files required to be sent. As soon as the open button is pressed, the function to send the selected file is executed.

In the Server part of the interface, the IP of the server is displayed. One can set the port on which the server should run. We have set the default port as 6666. The port selection part enables users to create a server which will not be shown by the server scanner unless the port is known. This allows server host to receive messages and files from only those users who know the specific port of his/her server. This helps in avoiding useless files and messages coming in. There is a Start/Stop button which is used to start and stop the server. Then there is a browse button which helps the server to select the location to save incoming files. Below that is a list box which will show the messages and files coming in.

In the Server Scanner part of the interface, one has to input the IP's to scan when looking for servers. We have also mentioned the way to search the servers if one wants to scan the whole range of IP's. One has to input the port at which the servers are to be scanned, the number of threads, and the time to live of the scanner function in nanoseconds. The Scan button initiates the scan function. If it finds a server on the same machine, a message is displayed in a messagebox stating that this message comes from the same machine. Below that is a list box which shows the progress of the scan. After scanning completes, it shows a list of IP's hosting a server.

We implemented the prototype to get a feel for developing networking applications. We did this by implementing a chat and file transfer client which we have used in implementing the final tool. We have used the same methodology for transferring files and used the prototype as a base for our work towards achieving our end goal.

We have implemented the final tool with a very simple interface.

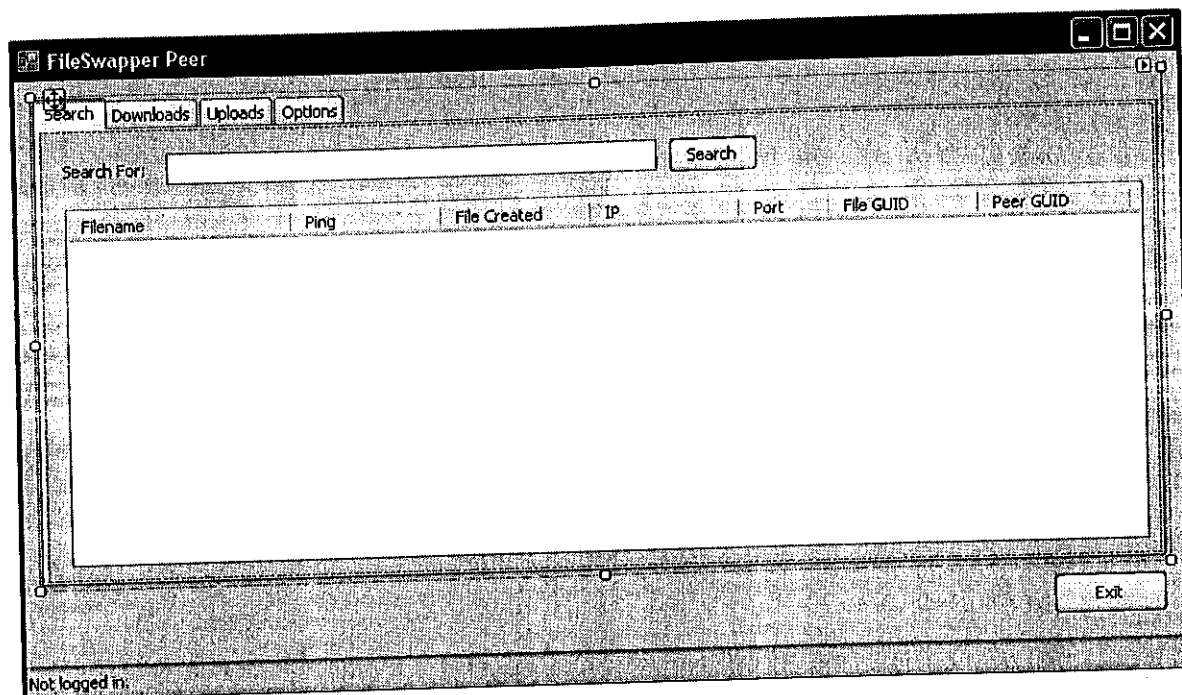


Fig 3.1.2 User Interface

Upon loading the interface, a form is shown with a search textbox and below it there is a listbox which is used for listing the files with the various attributes. The attributes being displayed are Filename, Ping of that filename, Time of Creation of File, IP on which file is shared, Port on which file is shared, Unique File GUID and Unique Peer GUID. There is a status bar at the base which when code is executed shows 'Trying to Log in'. If login is successful it displays 'File List Published to Server' else shows 'Not Logged In'. There is also an Exit Button for closing the Client. The Status Bar and the Exit button are constant in all the forms.

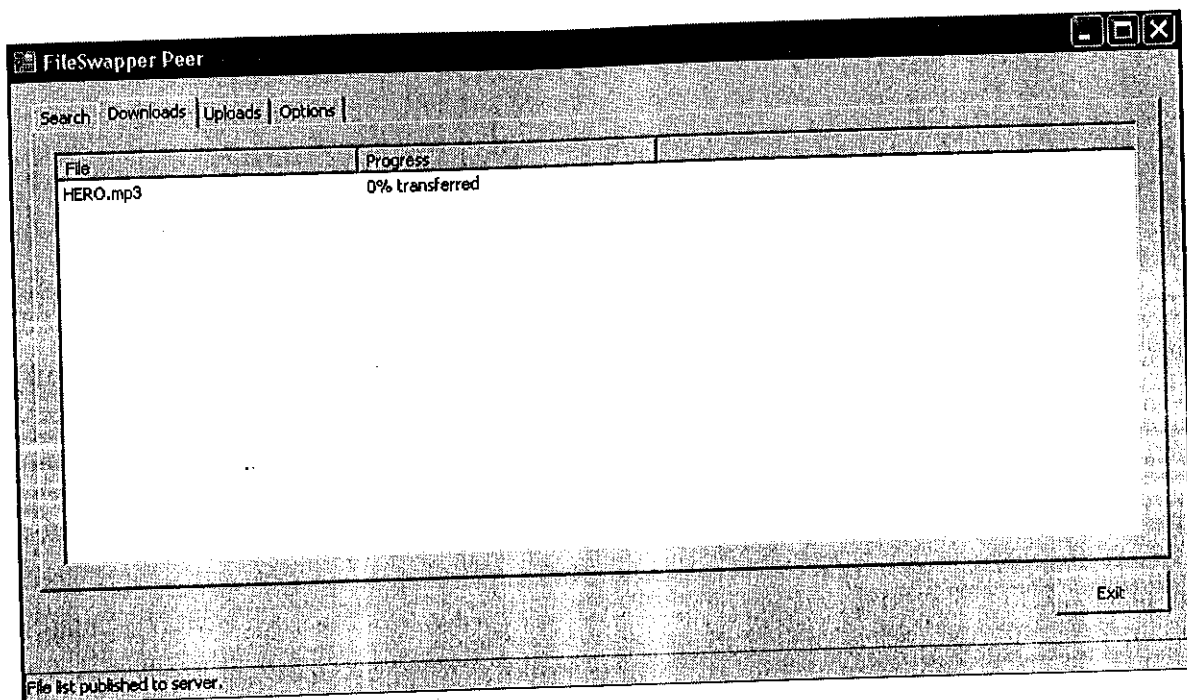


Fig 3.1.3: Download Information Display

This is the form used for seeing the progress of the files being downloaded. It shows the name of the file and the progress of the file transferred.

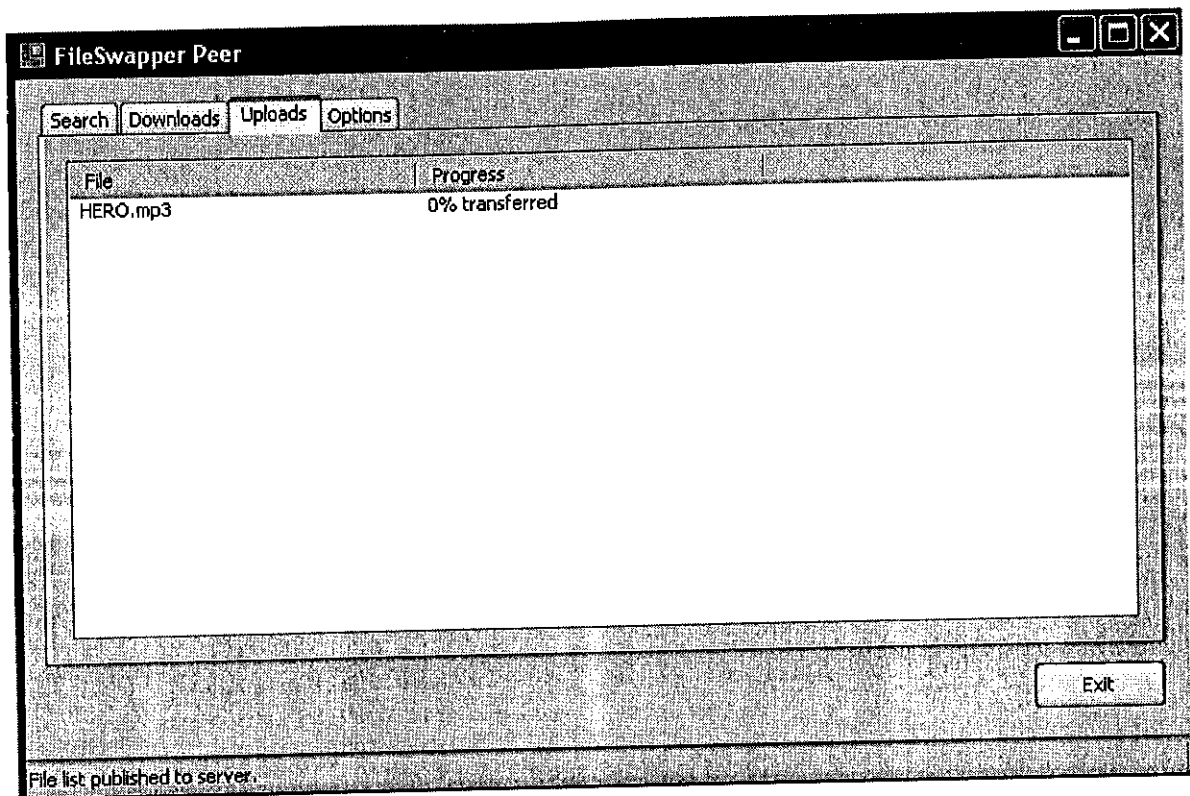


Fig 3.1.4: File Upload Information Display

This form is used to show the uploads, that is, the file that is being transferred from this machine and the progress of the transfer.

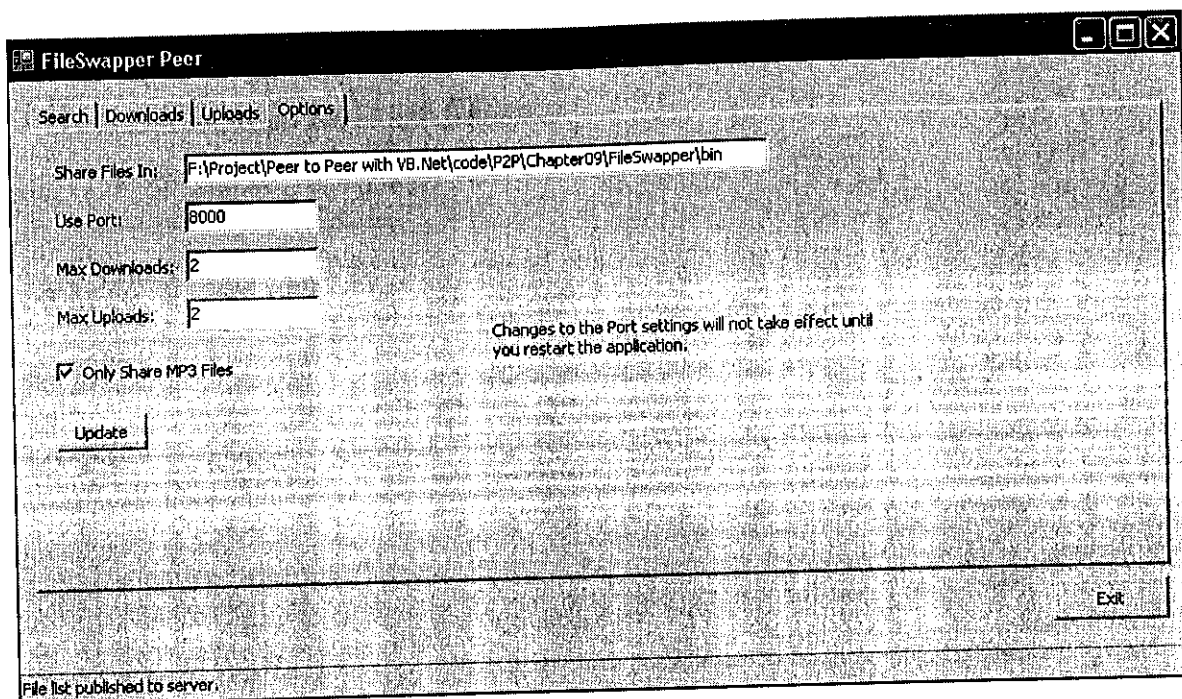


Fig 3.1.5: Client Settings Display

This is the form which has the options for the client which are editable. Here the user can set or change the path in which files are shared and downloaded, change the port no so as to ensure a private network, and set the maximum number of uploads and downloads. Also there is an option of only sharing MP3 files as we had earlier based the coding on Napster and Kaazaa, and then broadened the scope to include all types of files. The update button saves the configuration changes if any are made.

4. Source Code

Code executed at the loading of the form:

```
Imports FileSwapper.localhost

Public Class SwapperClient
    Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

    Public Sub New()
        MyBase.New()

        'This call is required by the Windows Form Designer.
        InitializeComponent()

        'Add any initialization after the InitializeComponent() call
    End Sub

    'Form overrides dispose to clean up the component list.
    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    Friend WithEvents TabPage1 As System.Windows.Forms.TabPage
```


Friend WithEvents tabPage2 As System.Windows.Forms.TabPage
Friend WithEvents tabPage3 As System.Windows.Forms.TabPage
Friend WithEvents cmdExit As System.Windows.Forms.Button
Friend WithEvents statusBar1 As System.Windows.Forms.StatusBar
Friend WithEvents pnlState As System.Windows.Forms.StatusBarPanel
Friend WithEvents txtSharePath As System.Windows.Forms.TextBox
Friend WithEvents txtPort As System.Windows.Forms.TextBox
Friend WithEvents label1 As System.Windows.Forms.Label
Friend WithEvents label2 As System.Windows.Forms.Label
Friend WithEvents cmdSearch As System.Windows.Forms.Button
Friend WithEvents label3 As System.Windows.Forms.Label
Friend WithEvents txtKeywords As System.Windows.Forms.TextBox
Friend WithEvents lstSearchResults As System.Windows.Forms.ListView
Friend WithEvents tabPage4 As System.Windows.Forms.TabPage
Friend WithEvents lstUploads As System.Windows.Forms.ListView
Friend WithEvents chkMP3Only As System.Windows.Forms.CheckBox
Friend WithEvents cmdUpdate As System.Windows.Forms.Button
Friend WithEvents filename As System.Windows.Forms.ColumnHeader
Friend WithEvents IP As System.Windows.Forms.ColumnHeader
Friend WithEvents Ping As System.Windows.Forms.ColumnHeader
Friend WithEvents CreatedDate As System.Windows.Forms.ColumnHeader
Friend WithEvents Port As System.Windows.Forms.ColumnHeader
Friend WithEvents tmrRefreshRegistration As System.Windows.Forms.Timer
Friend WithEvents fileGUID As System.Windows.Forms.ColumnHeader
Friend WithEvents peerGUID As System.Windows.Forms.ColumnHeader
Friend WithEvents label4 As System.Windows.Forms.Label
Friend WithEvents txtDownloads As System.Windows.Forms.TextBox
Friend WithEvents txtUploads As System.Windows.Forms.TextBox
Friend WithEvents label5 As System.Windows.Forms.Label
Friend WithEvents file As System.Windows.Forms.ColumnHeader
Friend WithEvents progress As System.Windows.Forms.ColumnHeader
Friend WithEvents lstDownloads As System.Windows.Forms.ListView
Friend WithEvents columnHeader1 As System.Windows.Forms.ColumnHeader
Friend WithEvents columnHeader2 As System.Windows.Forms.ColumnHeader
Friend WithEvents tbPages As System.Windows.Forms.TabControl
Friend WithEvents label6 As System.Windows.Forms.Label

```

<System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
    Me.components = New System.ComponentModel.Container()
    Me.tbPages = New System.Windows.Forms.TabControl()
    Me.TabPage1 = New System.Windows.Forms.TabPage()
    Me.txtKeywords = New System.Windows.Forms.TextBox()
    Me.Label3 = New System.Windows.Forms.Label()
    Me.cmdSearch = New System.Windows.Forms.Button()
    Me.lstSearchResults = New System.Windows.Forms.ListView()
    Me.Filename = New System.Windows.Forms.ColumnHeader()
    Me.Ping = New System.Windows.Forms.ColumnHeader()
    Me.CreatedDate = New System.Windows.Forms.ColumnHeader()
    Me.IP = New System.Windows.Forms.ColumnHeader()
    Me.Port = New System.Windows.Forms.ColumnHeader()
    Me.FileGUID = New System.Windows.Forms.ColumnHeader()
    Me.PeerGUID = New System.Windows.Forms.ColumnHeader()
    Me.TabPage2 = New System.Windows.Forms.TabPage()
    Me.lstDownloads = New System.Windows.Forms.ListView()
    Me.ColumnHeader1 = New System.Windows.Forms.ColumnHeader()
    Me.ColumnHeader2 = New System.Windows.Forms.ColumnHeader()
    Me.TabPage4 = New System.Windows.Forms.TabPage()
    Me.lstUploads = New System.Windows.Forms.ListView()
    Me.File = New System.Windows.Forms.ColumnHeader()
    Me.Progress = New System.Windows.Forms.ColumnHeader()
    Me.TabPage3 = New System.Windows.Forms.TabPage()
    Me.Label6 = New System.Windows.Forms.Label()
    Me.Label5 = New System.Windows.Forms.Label()
    Me.txtUploads = New System.Windows.Forms.TextBox()
    Me.txtDownloads = New System.Windows.Forms.TextBox()
    Me.Label4 = New System.Windows.Forms.Label()
    Me.chkMP3Only = New System.Windows.Forms.CheckBox()
    Me.cmdUpdate = New System.Windows.Forms.Button()
    Me.Label2 = New System.Windows.Forms.Label()
    Me.Label1 = New System.Windows.Forms.Label()
    Me.txtPort = New System.Windows.Forms.TextBox()
    Me.txtSharePath = New System.Windows.Forms.TextBox()
    Me.cmdExit = New System.Windows.Forms.Button()

```

```

Me.StatusBar1 = New System.Windows.Forms.StatusBar()
Me.pnlState = New System.Windows.Forms.StatusBarPanel()
Me.tmrRefreshRegistration = New
System.Windows.Forms.Timer(Me.components)
Me.tbPages.SuspendLayout()
Me.TabPage1.SuspendLayout()
Me.TabPage2.SuspendLayout()
Me.TabPage4.SuspendLayout()
Me.TabPage3.SuspendLayout()
CType(Me.pnlState,
System.ComponentModel.ISupportInitialize).BeginInit()
Me.SuspendLayout()
'
'tbPages
'
Me.tbPages.Anchor = ((System.Windows.Forms.AnchorStyles.Top Or
System.Windows.Forms.AnchorStyles.Bottom) _
Or System.Windows.Forms.AnchorStyles.Left) _
Or System.Windows.Forms.AnchorStyles.Right)
Me.tbPages.Controls.AddRange(New System.Windows.Forms.Control()
(Me.TabPage1, Me.TabPage2, Me.TabPage4, Me.TabPage3))
Me.tbPages.Location = New System.Drawing.Point(12, 12)
Me.tbPages.Name = "tbPages"
Me.tbPages.SelectedIndex = 0
Me.tbPages.Size = New System.Drawing.Size(644, 328)
Me.tbPages.TabIndex = 0
'
'TabPage1
'
Me.TabPage1.Controls.AddRange(New System.Windows.Forms.Control()
(Me.txtKeywords, Me.Label3, Me.cmdSearch, Me.lstSearchResults))
Me.TabPage1.Location = New System.Drawing.Point(4, 22)
Me.TabPage1.Name = "TabPage1"
Me.TabPage1.Size = New System.Drawing.Size(636, 302)
Me.TabPage1.TabIndex = 0
Me.TabPage1.Text = "Search"
'

```



```

'txtKeywords
'
Me.txtKeywords.Location = New System.Drawing.Point(84, 16)
Me.txtKeywords.Name = "txtKeywords"
Me.txtKeywords.Size = New System.Drawing.Size(328, 21)
Me.txtKeywords.TabIndex = 7
Me.txtKeywords.Text = ""
'
'Label3
'
Me.Label3.Location = New System.Drawing.Point(12, 20)
Me.Label3.Name = "Label3"
Me.Label3.Size = New System.Drawing.Size(64, 16)
Me.Label3.TabIndex = 6
Me.Label3.Text = "Search For:"
'
'cmdSearch
'
Me.cmdSearch.FlatStyle = System.Windows.Forms.FlatStyle.System
Me.cmdSearch.Location = New System.Drawing.Point(420, 15)
Me.cmdSearch.Name = "cmdSearch"
Me.cmdSearch.Size = New System.Drawing.Size(60, 24)
Me.cmdSearch.TabIndex = 5
Me.cmdSearch.Text = "Search"
'
'lstSearchResults
'
Me.lstSearchResults.Activation =
System.Windows.Forms.ItemActivation.TwoClick
Me.lstSearchResults.Anchor = (((System.Windows.Forms.AnchorStyles.Top
Or System.Windows.Forms.AnchorStyles.Bottom) _
Or System.Windows.Forms.AnchorStyles.Left) _
Or System.Windows.Forms.AnchorStyles.Right)
Me.lstSearchResults.Columns.AddRange(New
System.Windows.Forms.ColumnHeader() {Me.Filename, Me.Ping, Me.CreatedDate,
Me.IP, Me.Port, Me.FileGUID, Me.PeerGUID})
Me.lstSearchResults.FullRowSelect = True

```

```

Me.lstSearchResults.HeaderStyle =
System.Windows.Forms.ColumnHeaderStyle.Nonclickable
Me.lstSearchResults.Location = New System.Drawing.Point(16, 52)
Me.lstSearchResults.MultiSelect = False
Me.lstSearchResults.Name = "lstSearchResults"
Me.lstSearchResults.Size = New System.Drawing.Size(612, 240)
Me.lstSearchResults.TabIndex = 0
Me.lstSearchResults.View = System.Windows.Forms.View.Details
'
' TabPage7.Controls.Add(Page7)
'
' TabPage7.Location = New System.Drawing.Point(16, 224)
Me.FileName.Text = "Filename"
Me.FileName.Width = 150
'
' TabPage2.TabIndex = 1
' Ping
'
Me.Ping.Text = "Ping"
Me.Ping.Width = 100
'
' lstDownloads.Anchor = ((System.Windows.Forms.AnchorStyles.Top |
System
' CreatedDate
'
Me.CreatedDate.Text = "File Created"
Me.CreatedDate.Width = 100
'
' IP
'
Me.IP.Text = "IP"
Me.IP.Width = 100
'
' lstDownloads.TabIndex = 2
' Port
'
Me.Port.Text = "Port"
'
' FileGUID
'
Me.FileGUID.Text = "File GUID"
Me.FileGUID.Width = 100

```

```

'
'PeerGUID
'
Me.PeerGUID.Text = "Peer GUID"
Me.PeerGUID.Width = 100
'
'TabPage2
'
Me.TabPage2.Controls.AddRange(New System.Windows.Forms.Control()
{Me.lstDownloads})
Me.TabPage2.Location = New System.Drawing.Point(4, 22)
Me.TabPage2.Name = "TabPage2"
Me.TabPage2.Size = New System.Drawing.Size(636, 302)
Me.TabPage2.TabIndex = 1
Me.TabPage2.Text = "Downloads"
'
'lstDownloads
'
Me.lstDownloads.Anchor = (((System.Windows.Forms.AnchorStyles.Top Or
System.Windows.Forms.AnchorStyles.Bottom) _
Or System.Windows.Forms.AnchorStyles.Left) _
Or System.Windows.Forms.AnchorStyles.Right)
Me.lstDownloads.Columns.AddRange(New
System.Windows.Forms.ColumnHeader() {Me.ColumnHeader1, Me.ColumnHeader2})
Me.lstDownloads.FullRowSelect = True
Me.lstDownloads.Location = New System.Drawing.Point(12, 13)
Me.lstDownloads.Name = "lstDownloads"
Me.lstDownloads.Size = New System.Drawing.Size(612, 276)
Me.lstDownloads.TabIndex = 3
Me.lstDownloads.View = System.Windows.Forms.View.Details
'
'ColumnHeader1
'
Me.ColumnHeader1.Text = "File"
Me.ColumnHeader1.Width = 200
'
'ColumnHeader2

```

```

'
Me.ColumnHeader2.Text = "Progress"
Me.ColumnHeader2.Width = 200
'
'TabPage4
'
Me.TabPage4.Controls.AddRange(New System.Windows.Forms.Control()
{Me.lstUploads})
Me.TabPage4.Location = New System.Drawing.Point(4, 22)
Me.TabPage4.Name = "TabPage4"
Me.TabPage4.Size = New System.Drawing.Size(636, 302)
Me.TabPage4.TabIndex = 3
Me.TabPage4.Text = "Uploads"
'
'lstUploads
'
Me.lstUploads.Anchor = ((System.Windows.Forms.AnchorStyles.Top Or
System.Windows.Forms.AnchorStyles.Bottom) _
Or System.Windows.Forms.AnchorStyles.Left) _
Or System.Windows.Forms.AnchorStyles.Right)
Me.lstUploads.Columns.AddRange(New
System.Windows.Forms.ColumnHeader() {Me.File, Me.Progress})
Me.lstUploads.FullRowSelect = True
Me.lstUploads.Location = New System.Drawing.Point(12, 13)
Me.lstUploads.Name = "lstUploads"
Me.lstUploads.Size = New System.Drawing.Size(612, 276)
Me.lstUploads.TabIndex = 2
Me.lstUploads.View = System.Windows.Forms.View.Details
'
'File
'
Me.File.Text = "File"
Me.File.Width = 200
'
'Progress
'
Me.Progress.Text = "Progress"

```



```

Me.Progress.Width = 200
'
'TabPage3
'

Me.TabPage3.Controls.AddRange(New System.Windows.Forms.Control()
{Me.Label6, Me.Label5, Me.txtUploads, Me.txtDownloads, Me.Label4,
Me.chkMP3Only, Me.cmdUpdate, Me.Label2, Me.Label1, Me.txtPort,
Me.txtSharePath})

Me.TabPage3.Location = New System.Drawing.Point(4, 22)
Me.TabPage3.Name = "TabPage3"
Me.TabPage3.Size = New System.Drawing.Size(636, 302)
Me.TabPage3.TabIndex = 2
Me.TabPage3.Text = "Options"
'
'Label6
'
Me.Label6.Location = New System.Drawing.Point(300, 132)
Me.Label6.Name = "Label6"
Me.Label6.Size = New System.Drawing.Size(268, 108)
Me.Label6.TabIndex = 10
Me.Label6.Text = "Changes to the Port settings will not take effect
until you restart the applicati" & _
"on."
'
'Label5
'
Me.Label5.Location = New System.Drawing.Point(12, 116)
Me.Label5.Name = "Label5"
Me.Label5.Size = New System.Drawing.Size(84, 16)
Me.Label5.TabIndex = 9
Me.Label5.Text = "Max Uploads:"
'
'txtUploads
'

Me.txtUploads.Location = New System.Drawing.Point(100, 112)
Me.txtUploads.Name = "txtUploads"
Me.txtUploads.Size = New System.Drawing.Size(88, 21)

```

```

Me.txtUploads.TabIndex = 8
Me.txtUploads.Text = ""
'
'txtDownloads
'
Me.txtDownloads.Location = New System.Drawing.Point(100, 80)
Me.txtDownloads.Name = "txtDownloads"
Me.txtDownloads.Size = New System.Drawing.Size(88, 21)
Me.txtDownloads.TabIndex = 7
Me.txtDownloads.Text = ""
'
'Label4
'
Me.Label4.Location = New System.Drawing.Point(12, 84)
Me.Label4.Name = "Label4"
Me.Label4.Size = New System.Drawing.Size(88, 16)
Me.Label4.TabIndex = 6
Me.Label4.Text = "Max Downloads:"
'
'chkMP3Only
'
Me.chkMP3Only.FlatStyle = System.Windows.Forms.FlatStyle.System
Me.chkMP3Only.Location = New System.Drawing.Point(12, 148)
Me.chkMP3Only.Name = "chkMP3Only"
Me.chkMP3Only.Size = New System.Drawing.Size(164, 24)
Me.chkMP3Only.TabIndex = 5
Me.chkMP3Only.Text = "Only Share MP3 Files"
'
'txtSharePath
'
Me.cmdUpdate.Location = New System.Drawing.Point(12, 188)
Me.cmdUpdate.Name = "cmdUpdate"
Me.cmdUpdate.Size = New System.Drawing.Size(60, 24)
Me.cmdUpdate.TabIndex = 4
Me.cmdUpdate.Text = "Update"
'

```



```

'Label2
'
Me.Label2.Location = New System.Drawing.Point(12, 52)
Me.Label2.Name = "Label2"
Me.Label2.Size = New System.Drawing.Size(84, 16)
Me.Label2.TabIndex = 3
Me.Label2.Text = "Use Port:"
'
'Label1
'
Me.Label1.Location = New System.Drawing.Point(12, 20)
Me.Label1.Name = "Label1"
Me.Label1.Size = New System.Drawing.Size(84, 16)
Me.Label1.TabIndex = 2
Me.Label1.Text = "Share Files In:"
'
'txtPort
'
Me.txtPort.Location = New System.Drawing.Point(100, 48)
Me.txtPort.Name = "txtPort"
Me.txtPort.Size = New System.Drawing.Size(88, 21)
Me.txtPort.TabIndex = 1
Me.txtPort.Text = ""
'
'txtSharePath
'
Me.txtSharePath.Location = New System.Drawing.Point(100, 16)
Me.txtSharePath.Name = "txtSharePath"
Me.txtSharePath.Size = New System.Drawing.Size(388, 21)
Me.txtSharePath.TabIndex = 0
Me.txtSharePath.Text = ""
'
'cmdExit
'
Me.cmdExit.Anchor = (System.Windows.Forms.AnchorStyles.Bottom Or
System.Windows.Forms.AnchorStyles.Right)
Me.cmdExit.FlatStyle = System.Windows.Forms.FlatStyle.System

```



```

Me.cmdExit.Location = New System.Drawing.Point(580, 348)
Me.cmdExit.Name = "cmdExit"
Me.cmdExit.Size = New System.Drawing.Size(75, 28)
Me.cmdExit.TabIndex = 1
Me.cmdExit.Text = "Exit"
'
'StatusBar1
'

Me.StatusBar1.Location = New System.Drawing.Point(0, 392)
Me.StatusBar1.Name = "StatusBar1"
Me.StatusBar1.Panels.AddRange(New
System.Windows.Forms.StatusBarPanel() {Me.pnlState})
Me.StatusBar1.ShowPanels = True
Me.StatusBar1.Size = New System.Drawing.Size(668, 22)
Me.StatusBar1.SizingGrip = False
Me.StatusBar1.TabIndex = 2
Me.StatusBar1.Text = "StatusBar1"
'
'pnlState
'

Me.pnlState.AutoSize =
System.Windows.Forms.StatusBarPanelAutoSize.Spring
Me.pnlState.Text = "Not logged in."
Me.pnlState.Width = 668
'
'tmrRefreshRegistration
'

Me.tmrRefreshRegistration.Interval = 300000
'
'SwapperClient
'

Me.AutoScaleBaseSize = New System.Drawing.Size(5, 14)
Me.ClientSize = New System.Drawing.Size(668, 414)
Me.Controls.AddRange(New System.Windows.Forms.Control()
{Me.StatusBar1, Me.cmdExit, Me.tbPages})

```

```
Me.Font = New System.Drawing.Font("Tahoma", 8.25!,  
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, CType(0,  
Byte))
```

```
Me.Name = "SwapperClient"
```

```
Me.Text = "FileSwapper Peer"
```

```
Me.tbPages.ResumeLayout(False)
```

```
Me.TabPage1.ResumeLayout(False)
```

```
Me.TabPage2.ResumeLayout(False)
```

```
Me.TabPage4.ResumeLayout(False)
```

```
Me.TabPage3.ResumeLayout(False)
```

```
CType(Me.pnlState,
```

```
System.ComponentModel.ISupportInitialize).EndInit()
```

```
Me.ResumeLayout(False)
```

```
End Sub
```

```
End Sub
```

```
#End Region
```

```
Private Sub cmdExit_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles cmdExit.Click
```

```
Me.Close()
```

```
End Sub
```

```
Private Sub SwapperClient_Load(ByVal sender As Object, ByVal e As  
System.EventArgs) Handles MyBase.Load
```

```
Me.Show()
```

```
Me.Refresh()
```

```
' Read registry.
```

```
[Global].Settings.Load()
```

```
txtSharePath.Text = [Global].Settings.SharePath
```

```
txtPort.Text = [Global].Settings.Port
```

```
chkMP3Only.Checked = [Global].Settings.ShareMP3Only
```

```
txtUploads.Text = [Global].Settings.MaxUploadThreads
```

```
txtDownloads.Text = [Global].Settings.MaxDownloadThreads
```

```

' Create the search, download, and upload objects.
' They will create their own threads.
App.SearchThread = New Search(lstSearchResults)
App.DownwnloadThread = New FileDownloadQueue(lstDownloads)
App.UploadThread = New FileServer(lstUploads)
App.UploadThread.StartWaitForRequest()

[Global].Identity.Port = [Global].Settings.Port
DoLogin()
AddHandler AppDomain.CurrentDomain.UnhandledException, AddressOf
UnhandledException

End Sub

Private Sub DoLogin()

    Me.Cursor = Cursors.WaitCursor

    ' Log in.
    pnlState.Text = "Trying to log in."
    App.Login()
    If Not [Global].LoggedIn Then
        pnlState.Text = "Not logged in."
        Me.Cursor = Cursors.Default
        Return
    End If

    ' Submit list of files.
    pnlState.Text = "Sending file information..."
    If App.PublishFiles() Then
        pnlState.Text = "File list published to server."
    Else
        pnlState.Text = "Could not publish file list."
    End If

    ' Refresh login information every five minutes.

```



```
tmrRefreshRegistration.Start()
```

```
Me.Cursor = Cursors.Default
```

```
End Sub
```

```
Private Sub SwapperClient_Closed(ByVal sender As Object, ByVal e As  
System.EventArgs) Handles MyBase.Closed
```

```
App.Logout()
```

```
App.DownwnloadThread.Abort()
```

```
App.SearchThread.Abort()
```

```
App.UploadThread.Abort()
```

```
End Sub
```

```
Private Sub cmdSearch_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles cmdSearch.Click
```

```
If App.SearchThread.Searching Then
```

```
App.SearchThread.Abort()
```

```
End If
```

```
App.SearchThread.StartSearch(txtKeywords.Text)
```

```
End Sub
```

```
Private Sub cmdUpdate_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles cmdUpdate.Click
```

```
[Global].Settings.Port = Val(txtPort.Text)
```

```
[Global].Settings.SharePath = txtSharePath.Text
```

```
[Global].Settings.ShareMP3Only = chkMP3Only.Checked
```

```
[Global].Settings.MaxDownloadThreads = Val(txtDownloads.Text)
```

```
[Global].Settings.MaxUploadThreads = Val(txtUploads.Text)
```

```
[Global].Settings.Save()
```

```
' Log back in.
App.Logout()
[Global].Identity.Port = [Global].Settings.Port
DoLogin()
```

End Sub

```
Private Sub tmRefreshRegistration_Tick(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles tmrRefreshRegistration.Tick
```

```
' Fires every five minutes to update registration.
App.RefreshLogin()
' Currently no steps are taken to refresh subscribed file list.
```

End Sub

```
Private Sub lstSearchResults_ItemActivate(ByVal sender As Object, ByVal e
As System.EventArgs) Handles lstSearchResults.ItemActivate
    Dim File As SharedFile
    File = CType(CType(sender, ListView).SelectedItem(0).Tag,
SharedFile)
```

```
If App.DownwnloadThread.CheckForFile(File) Then
    MessageBox.Show("You are already downloading this file.",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Information)
```

```
'ElseIf File.Peer.Guid.ToString() =
Global.Identity.Guid.ToString() Then
    'MessageBox.Show("This is a local file.", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Information)
```

Else

```
App.DownwnloadThread.AddFile(File)
If Not App.DownwnloadThread.Working Then
    App.DownwnloadThread.StartAllocateWork()
End If
```

```
tbPages.SelectedTab = tbPages.TabPages(1)
```

```
End If
```

```
End Sub
```

```
Public Sub UnhandledException(ByVal sender As Object, _  
    ByVal e As UnhandledExceptionEventArgs)
```

```
    ' Log the error.
```

```
    Trace.Write(e.ExceptionObject.ToString())
```

```
    ' Log out of the discovery service.
```

```
    App.Logout()
```

```
End Sub
```

```
End Class
```


Code for Assembly Information (Deployment Information):

Imports System.Reflection

Imports System.Runtime.InteropServices

' General Information about an assembly is controlled through the following
' set of attributes. Change these attribute values to modify the information
' associated with an assembly.

' Review the values of the assembly attributes

<Assembly: AssemblyTitle("")>

<Assembly: AssemblyDescription("")>

<Assembly: AssemblyCompany("")>

<Assembly: AssemblyProduct("")>

<Assembly: AssemblyCopyright("")>

<Assembly: AssemblyTrademark("")>

<Assembly: CLSCompliant(True)>

'The following GUID is for the ID of the typelib if this project is exposed
to COM

<Assembly: Guid("3CBD4A2B-8A5A-4DC3-93DA-711EBA992F76")>

' Version information for an assembly consists of the following four values:

'

' Major Version

' Minor Version

' Build Number

' Revision

'

' You can specify all the values or you can default the Build and Revision
Numbers

' by using the '*' as shown below:

<Assembly: AssemblyVersion("1.0.*")>

Code for the Downloading Function:

```
Imports FileSwapper.localhost
Imports System.Threading
Imports System.Net
Imports System.Net.Sockets
Imports System.IO

Public Class FileDownloadQueue

    Private AllocateWorkThread As System.Threading.Thread
    Private _Working As Boolean
    Private ListView As ListView

    Private QueuedFiles As New ArrayList()
    Private DownloadThreads As New ArrayList()

    Public Sub New(ByVal linkedControl As ListView)
        ListView = linkedControl
    End Sub

    Public Sub AddFile(ByVal file As SharedFile)
        ' Add shared file.
        SyncLock QueuedFiles
            QueuedFiles.Add(New DisplayFile(file, ListView))
        End SyncLock
    End Sub

    Public Function CheckForFile(ByVal file As SharedFile) As Boolean

        Dim Item As DisplayFile
        For Each Item In QueuedFiles
            If Item.File.Guid.ToString() = file.Guid.ToString() Then Return
        Next

        True
    End Function
End Class
```

```

    Dim DownloadThread As FileDownloadClient
    For Each DownloadThread In DownloadThreads
        If DownloadThread.File.Guid.ToString() = file.Guid.ToString()
Then Return True
        Next

    Return False
End Function

Private Sub AllocateWork()
    Do
        ' Remove completed.
        Dim i As Integer
        For i = DownloadThreads.Count - 1 To 0 Step -1
            Dim DownloadThread As FileDownloadClient
            DownloadThread = CType(DownloadThreads(i),
FileDownloadClient)
            If Not DownloadThread.Working Then
                SyncLock DownloadThreads
                    DownloadThreads.Remove(DownloadThread)
                End SyncLock
            End If
        Next

        ' Allocate new while threads are available.
        Do While QueuedFiles.Count > 0 And DownloadThreads.Count <
[Global].Settings.MaxDownloadThreads
            Dim DownloadThread As New FileDownloadClient(QueuedFiles(0))
            SyncLock DownloadThreads
                DownloadThreads.Add(DownloadThread)
            End SyncLock
            SyncLock QueuedFiles
                QueuedFiles.RemoveAt(0)
            End SyncLock
            DownloadThread.StartDownload()
        Loop

```



```

        Thread.Sleep(TimeSpan.FromSeconds(10))
    Loop
End Sub

Public Sub StartAllocateWork()
    If _Working Then
        Throw New ApplicationException("Already in progress.")
    Else
        _Working = True

        AllocateWorkThread = New Threading.Thread(AddressOf AllocateWork)
        AllocateWorkThread.Start()
    End If
End Sub

Public Sub Abort()
    If _Working Then
        AllocateWorkThread.Abort()
        'AllocateWorkThread.Join()

        ' Abort all download threads.
        Dim DownloadThread As FileDownloadClient
        For Each DownloadThread In DownloadThreads
            DownloadThread.Abort()
        Next

        _Working = False
    End If
End Sub

Public ReadOnly Property Working() As Boolean
    Get
        Return _Working
    End Get
End Property

```

End Class

Public Class FileDownloadClient

Private DisplayFile As DisplayFile

Private DownloadThread As System.Threading.Thread

Private _Working As Boolean

Public Sub New(ByVal file As DisplayFile)

Me.DisplayFile = file

End Sub

Private Client As TcpClient

Private Sub Download()

DisplayFile.ListViewItem.ChangeStatus("Connecting...")

' Connect.

Dim Completed As Boolean = False

Do

' (Add error handling.)

Client = New TcpClient()

Client.Connect(Dns.GetHostEntry(DisplayFile.File.Peer.IP).AddressList(0),
Val(DisplayFile.File.Peer.Port))

Dim r As New BinaryReader(Client.GetStream())

Dim Response As String = r.ReadString()

If Response = Messages.Ok Then

DisplayFile.ListViewItem.ChangeStatus("Connected")

Dim w As New BinaryWriter(Client.GetStream())

' Request file.

w.Write(DisplayFile.File.Guid.ToString())

```

' Write file.
Dim TotalBytes As Integer = r.ReadInt32()
If TotalBytes = Messages.FileNotFound Then
    DisplayFile.ListViewItem.ChangeStatus("File Not Found")

Else
    ' Write temporary file.
    Dim FullPath As String =
Path.Combine([Global].Settings.SharePath, File.Guid.ToString() & ".tmp")
    Dim Download As New FileInfo(FullPath)

    Dim TotalBytesRead, BytesRead As Integer

    Dim fs As FileStream = Download.Create()
    Dim Buffer(1024) As Byte
    Dim Percent As Single
    Dim LastWrite As DateTime = DateTime.Now
    Do
        BytesRead = r.Read(Buffer, 0, Buffer.Length)
        fs.Write(Buffer, 0, BytesRead)
        TotalBytesRead += BytesRead

        If DateTime.Now.Subtract(LastWrite).TotalSeconds > 2
Then
            LastWrite = DateTime.Now
            Percent = Math.Round((TotalBytesRead /
TotalBytes) * 100, 0)
            DisplayFile.ListViewItem.ChangeStatus(Percent.ToString() & "% transferred")
        End If
    Loop While BytesRead > 0
    fs.Close()

    ' Ensure a unique name is chosen.

```



```

        Dim FileNames() As String =
Directory.GetFiles([Global].Settings.SharePath)
        Dim FinalPath As String =
Path.Combine([Global].Settings.SharePath, File.FileName)
        Dim i As Integer
        Do While Array.IndexOf(FileNames, FinalPath) <> -1
            i += 1
            FinalPath = Path.Combine([Global].Settings.SharePath,
Path.GetFileNameWithoutExtension(File.FileName) & i.ToString() &
Path.GetExtension(File.FileName))
        Loop

        ' Rename file.
        System.IO.File.Move(FullPath, FinalPath)

        ' We could also contact the server here to update the
file
        ' subsription information.

        DisplayFile.ListViewItem.ChangeStatus("Completed")
    End If

    Client.Close()
    Completed = True

ElseIf Response = Messages.Busy Then
    DisplayFile.ListViewItem.ChangeStatus("Busy - Will Retry")
    Client.Close()
Else
    DisplayFile.ListViewItem.ChangeStatus("Error - Will Retry")
    Client.Close()
End If
If Not Completed Then Thread.Sleep(TimeSpan.FromSeconds(10))

Loop Until Completed

```

```
_Working = False
```

```
End Sub
```

```
Public Sub StartDownload()
```

```
    If _Working Then
```

```
        Throw New ApplicationException("Already in progress.")
```

```
    Else
```

```
        _Working = True
```

```
        DownloadThread = New Threading.Thread(AddressOf Download)
```

```
        DownloadThread.Start()
```

```
    End If
```

```
End Sub
```

```
Public Sub Abort()
```

```
    If _Working Then
```

```
        'Client.Close()
```

```
        DownloadThread.Abort()
```

```
        'DownloadThread.Join()
```

```
        _Working = False
```

```
    End If
```

```
End Sub
```

```
Public ReadOnly Property Working() As Boolean
```

```
    Get
```

```
        Return _Working
```

```
    End Get
```

```
End Property
```

```
Public ReadOnly Property File() As SharedFile
```

```
    Get
```

```
        Return DisplayFile.File
```

```
    End Get
```

```
End Property
```

```
End Class
```

```
Public Class DisplayFile
```

```
    Private _ListViewItem As ListViewItemWrapper
```

```
    Private _File As SharedFile
```

```
    Public ReadOnly Property File() As SharedFile
```

```
        Get
```

```
            Return _File
```

```
        End Get
```

```
    End Property
```

```
    Public ReadOnly Property ListViewItem() As ListViewItemWrapper
```

```
        Get
```

```
            Return _ListViewItem
```

```
        End Get
```

```
    End Property
```

```
    Public Sub New(ByVal file As SharedFile, ByVal linkedControl As ListView)
```

```
        _ListViewItem = New ListViewItemWrapper(linkedControl, file.FileName,  
"Queued")
```

```
        _File = file
```

```
    End Sub
```

```
End Class
```

```
Public Class ListViewItemWrapper
```

```
    Private ListView As ListView
```

```
    Private ListViewItem As ListViewItem
```

```
    Private RowName As String
```

```
    Private RowStatus As String
```

```
    Public Sub New(ByVal listView As ListView, ByVal rowName As String, ByVal  
rowStatus As String)
```

```
        Me.ListView = listView
```



```

    Me.RowName = rowName
    Me.RowStatus = rowStatus

    ' Marshal the operation to the user interface thread.
    listView.Invoke(New MethodInvoker(AddressOf AddListViewItem))
End Sub

' This code executes on the user interface thread.
Private Sub AddListViewItem()
    ' Create new ListView item.
    ListViewItem = New ListViewItem(RowName)
    ListViewItem.SubItems.Add(RowStatus)
    listView.Items.Add(ListViewItem)
End Sub

Public Sub ChangeStatus(ByVal rowStatus As String)
    Me.RowStatus = rowStatus

    ' Marshal the operation to the user interface thread.
    listView.Invoke(New MethodInvoker(AddressOf RefreshListViewItem))
End Sub

' This code executes on the user interface thread.
Private Sub RefreshListViewItem()
    ListViewItem.SubItems(1).Text = RowStatus
End Sub

End Class

```

Code for the Global Class:

```
Imports System.Net
Imports System.IO
Imports System.Text
Imports FileSwapper.localhost
```

```
Public Class [Global]
```

```
    ' Contains information about the current peer.
```

```
    Public Shared LoggedIn As Boolean = False
```

```
    Public Shared Identity As New Peer()
```

```
    ' Lists files that are available for other peers.
```

```
    Public Shared SharedFiles() As SharedFile
```

```
    ' Provides access to configuration settings that are stored in the
    registry.
```

```
    Public Shared Settings As New RegistrySettings()
```

```
End Class
```

```
Public Class App
```

```
    ' Holds a reference to the web service proxy.
```

```
    Private Shared Discovery As New DiscoveryService()
```

```
    Public Shared SearchThread As Search
```

```
    Public Shared DownwnloadThread As FileDownloadQueue
```

```
    Public Shared UploadThread As FileServer
```

```
    Public Shared Sub Login()
```

```
        [Global].Identity.Guid = Guid.NewGuid
```

```

        [Global].Identity.IP =
Dns.GetHostEntry(Dns.GetHostName).AddressList(0).ToString()
        [Global].LoggedIn = Discovery.Register([Global].Identity)

End Sub

Public Shared Sub Logout()
    If [Global].LoggedIn Then Discovery.Unregister([Global].Identity)
End Sub

Public Shared Sub RefreshLogin()
    If [Global].LoggedIn Then
Discovery.RefreshRegistration([Global].Identity)
    End Sub

Public Shared Function PublishFiles() As Boolean

    Try
        ' Perform a failsafe check in case old registry settings
        ' that point to a directory that no longer exists.
        If Not Directory.Exists([Global].Settings.SharePath) Then
            [Global].Settings.SharePath = Application.StartupPath
        End If

        Dim Dir As New DirectoryInfo([Global].Settings.SharePath)

        Dim Files() As FileInfo = Dir.GetFiles()
        Dim FileList As New ArrayList()
        Dim File As FileInfo
        Dim IsMP3 As Boolean

        For Each File In Files

            IsMP3 = Path.GetExtension(File.Name).ToLower() = ".mp3"

            If Path.GetExtension(File.Name).ToLower() = ".tmp" Then
                ' Ignore all temporary files.

```



```

ElseIf (Not IsMP3) And [Global].Settings.ShareMP3Only Then
    ' Ignore non-MP3 file depending on setting.
Else
    Dim SharedFile As New SharedFile()
    SharedFile.Guid = Guid.NewGuid()
    SharedFile.FileName = File.Name
    SharedFile.FileCreated = File.CreationTime

    If IsMP3 Then
        SharedFile.Keywords =
MP3Util.GetMP3Keywords(File.FullName)
    Else
        ' Determine some other way to set keywords,
        ' perhaps by filename or depending on the file
        ' type.
        ' The default (no keywords), will prevent the
        ' file from appearing in a search.
    End If

    FileList.Add(SharedFile)
End If
Next

[Global].SharedFiles =
CType(FileList.ToArray(GetType(SharedFile)), SharedFile())

Return Discovery.PublishFiles([Global].SharedFiles,
[Global].Identity)

Catch Err As Exception
    MessageBox.Show(Err.ToString())
End Try

End Function

Public Shared Function SearchForFile(ByVal keywords() As String) As
SharedFile()

```

```
Return Discovery.SearchForFile(keywords)
End Function
```

```
End Class
```

```
Public Class KeywordUtil
```

```
Private Shared NoiseWords() As String = {"the", "for", "and", "or"}
```

```
Public Shared Function ParseKeywords(ByVal keywordString As String) As
String()
```

```
    ' Split the list of words into an array.
```

```
    Dim Keywords() As String
```

```
    Dim Delimiters() As Char = {" ", ",", ".", ""}
```

```
    Keywords = keywordString.Split(Delimiters)
```

```
    ' Add each valid word into an ArrayList.
```

```
    Dim FilteredWords As New ArrayList()
```

```
    Dim Word As String
```

```
    For Each Word In Keywords
```

```
        If Word.Trim() <> "" And Word.Length > 1 Then
```

```
            If Array.IndexOf(NoiseWords, Word.ToLower()) = -1 Then
```

```
                FilteredWords.Add(Word)
```

```
            End If
```

```
        End If
```

```
    Next
```

```
    ' Convert the ArrayList into a normal string array.
```

```
    Return FilteredWords.ToArray(GetType(String))
```

```
End Function
```

```
End Class
```

```
Public Class MP3Util
```

```
    Public Shared Function GetMP3Keywords(ByVal filename As String) As
String()
```

```

Dim fs As New FileStream(filename, FileMode.Open)

' Read the MP3 tag.
fs.Seek(0 - 128, SeekOrigin.End)
Dim Tag(2) As Byte
fs.Read(Tag, 0, 3)

If Encoding.ASCII.GetString(Tag).Trim() = "TAG" Then

    Dim KeywordString As New StringBuilder()
    ' Title.
    KeywordString.Append(GetTagData(fs, 30))
    ' Artist.
    KeywordString.Append(" ")
    KeywordString.Append(GetTagData(fs, 30))
    ' Album.
    KeywordString.Append(" ")
    KeywordString.Append(GetTagData(fs, 30))
    ' Year.
    KeywordString.Append(" ")
    KeywordString.Append(GetTagData(fs, 4))
    ' Comment.
    KeywordString.Append(" ")
    KeywordString.Append(GetTagData(fs, 30))

    fs.Close()
    Dim Keywords() As String =
KeywordUtil.ParseKeywords(KeywordString.ToString())
    Return Keywords
Else
    fs.Close()
    Dim EmptyArray() As String = {}
    Return EmptyArray
End If

End Function

```

```
Public Shared Function GetTagData(ByVal stream As Stream, ByVal length As Integer) As String
```

```
    Dim Bytes(length - 1) As Byte  
    stream.Read(Bytes, 0, length)
```

```
    Dim TagData As String = Encoding.ASCII.GetString(Bytes)
```

```
    ' Trim nulls.
```

```
    Dim TrimChars() As Char = {" ", vbNullChar}
```

```
    TagData = TagData.Trim(TrimChars)
```

```
    Return TagData
```

```
End Function
```

```
End Class
```


Code for the Registry Class:

```
Imports Microsoft.Win32
```

```
Public Class RegistrySettings
```

```
    Public SharePath As String
```

```
    Public ShareMP3Only As Boolean
```

```
    Public MaxUploadThreads As Integer
```

```
    Public MaxDownloadThreads As Integer
```

```
    Public Port As Integer
```

```
    Public Sub Load()
```

```
        Dim Key As RegistryKey
```

```
        Key = Microsoft.Win32.Registry.LocalMachine.CreateSubKey( _  
            "Software\FilesSwapper\Settings")
```

```
        SharePath = Key.GetValue("SharePath", Application.StartupPath)
```

```
        Port = CType(Key.GetValue("LocalPort", "8000"), Integer)
```

```
        ShareMP3Only = CType(Key.GetValue("OnlyShareMP3", "True"), Boolean)
```

```
        MaxUploadThreads = CType(Key.GetValue("MaxUploadThreads", "2"),
```

```
Integer)
```

```
        MaxDownloadThreads = CType(Key.GetValue("MaxDownloadThreads", "2"),
```

```
Integer)
```

```
    End Sub
```

```
    Public Sub Save()
```

```
        Dim Key As RegistryKey
```

```
        Key = Microsoft.Win32.Registry.LocalMachine.CreateSubKey( _  
            "Software\FilesSwapper\Settings")
```

```
        Key.SetValue("SharePath", SharePath)
```

```
        Key.SetValue("LocalPort", Port.ToString())
```

```
        Key.SetValue("OnlyShareMP3", ShareMP3Only.ToString())
```

```
Key.SetValue("MaxUploadThreads", MaxUploadThreads.ToString())  
Key.SetValue("MaxDownloadThreads", MaxDownloadThreads.ToString())
```

```
End Sub
```

```
End Class
```

Code for the Search Class:

```
Imports FileSwapper.localhost
Imports System.Threading.Thread
```

```
Public Class Search
```

```
    Private SearchThread As System.Threading.Thread
```

```
    Private ListView As ListView
```

```
    Private SearchResults() As SharedFile
```

```
    Private PingTimes As New Hashtable()
```

```
    Private _Searching As Boolean = False
```

```
    Private Keywords() As String
```

```
    Public Sub New(ByVal linkedControl As ListView)
```

```
        ListView = linkedControl
```

```
    End Sub
```

```
    Private Sub Search()
```

```
        SearchResults = App.SearchForFile(Me.Keywords)
```

```
        _Searching = False
```

```
        PingRecipients()
```

```
        Try
```

```
            ListView.Invoke(New MethodInvoker(AddressOf UpdateInterface))
```

```
        Catch
```

```
            ' An error could occur here if the search is cancelled and the
```

```
            ' class is destroyed before the invoke finishes.
```

```
        End Try
```

```
    End Sub
```

```
    Private Sub PingRecipients()
```

```

    PingTimes.Clear()
    Dim File As SharedFile
    For Each File In SearchResults
        Dim PingTime As Integer =
PingUtility.Pinger.GetPingTime(File.Peer.IP)
        If PingTime = -1 Then
            PingTimes.Add(File.Guid, "Error")
        Else
            PingTimes.Add(File.Guid, PingTime.ToString() & " ms")
        End If
    Next

End Sub

Private Sub UpdateInterface()
    ListView.Items.Clear()
    If SearchResults.Length = 0 Then
        MessageBox.Show("No matches found.", "Error",
        MessageBoxButtons.OK, MessageBoxIcon.Information)
    Else
        Dim File As SharedFile
        For Each File In SearchResults
            Dim Item As ListViewItem = ListView.Items.Add(File.FileName)
            Item.SubItems.Add(PingTimes(File.Guid).ToString())
            Item.SubItems.Add(File.FileCreated)
            Item.SubItems.Add(File.Peer.IP)
            Item.SubItems.Add(File.Peer.Port)
            Item.SubItems.Add(File.Guid.ToString())
            Item.SubItems.Add(File.Peer.Guid.ToString())

            ' Store the SharedFile object for easy access later.
            Item.Tag = File
        Next
    End If
End Sub

Public Sub StartSearch(ByVal keywordString As String)

```



```

    If _Searching Then
        Throw New ApplicationException("Cancel current search first.")
    Else
        _Searching = True
        SearchResults = Nothing
        Keywords = KeywordUtil.ParseKeywords(keywordString)
        SearchThread = New Threading.Thread(AddressOf Search)
        SearchThread.Start()
    End If
End Sub

Public Sub Abort()
    If _Searching Then

        SearchThread.Abort()
        SearchThread.Join()

        _Searching = False
    End If
End Sub

Public Function GetSearchResults() As SharedFile()
    If _Searching = False Then
        Return SearchResults
    Else
        Return Nothing
    End If
End Function

Public ReadOnly Property Searching() As Boolean
    Get
        Return _Searching
    End Get
End Property
End Class

```

Code for the Uploading Function:

```
Imports FileSwapper.localhost
Imports System.Threading
Imports System.Net
Imports System.Net.Sockets
Imports System.IO

Public Class FileServer

    Private WaitForRequestThread As System.Threading.Thread
    Private _Working As Boolean
    Private ListView As ListView

    Private UploadThreads As New ArrayList()

    Public Sub New(ByVal linkedControl As ListView)
        ListView = linkedControl
    End Sub

    Private Listener As TcpListener

    Public Sub WaitForRequest()
        Dim localaddr As System.Net.IPAddress
        localaddr = Dns.GetHostEntry(Dns.GetHostName()).AddressList(0)
        Listener = New TcpListener(localaddr, [Global].Settings.Port)
        Listener.Start()
        Do
            ' (Error handling code).

            ' Block until connection received.
            Dim Client As TcpClient = Listener.AcceptTcpClient()

            ' Check for completed requests.
            ' This will free up space for new requests.
            Dim UploadThread As FileUpload
            Dim i As Integer
```

```

For i = (UploadThreads.Count - 1) To 0 Step -1
    UploadThread = CType(UploadThreads(i), FileUpload)
    If UploadThread.Working = False Then
        UploadThreads.Remove(UploadThread)
    End If
Next

Dim s As NetworkStream = Client.GetStream()
Dim w As New BinaryWriter(s)
If UploadThreads.Count > [Global].Settings.MaxUploadThreads Then
    w.Write(Messages.Busy)
    s.Close()
Else
    w.Write(Messages.Ok)
    Dim Upload As New FileUpload(s, ListView)
    UploadThreads.Add(Upload)
    Upload.StartUpload()
End If
Loop
End Sub

Public Sub StartWaitForRequest()
    If _Working Then
        Throw New ApplicationException("Already in progress.")
    Else
        _Working = True

        WaitForRequestThread = New Threading.Thread(AddressOf
WaitForRequest)
        WaitForRequestThread.Start()
    End If
End Sub

Public Sub Abort()
    If _Working Then
        Listener.Stop()
    End If
End Sub

```

```

        WaitForRequestThread.Abort()
        'WaitForRequestThread.Join()

        ' Abort all upload threads.
        Dim UploadThread As FileUpload
        For Each UploadThread In UploadThreads
            UploadThread.Abort()
        Next

        _Working = False
    End If
End Sub

Public ReadOnly Property Working() As Boolean
    Get
        Return _Working
    End Get
End Property

End Class

Public Class FileUpload

    Private Stream As NetworkStream
    Private UploadThread As System.Threading.Thread
    Private _Working As Boolean

    Private ListView As ListView

    Public Sub New(ByVal stream As NetworkStream, ByVal listView As ListView)
        Me.Stream = stream
        Me.ListView = listView
    End Sub

    Private Sub Upload()

        ' Connect.

```



```

Dim w As New BinaryWriter(Stream)
Dim r As New BinaryReader(Stream)

' Read file request.
Dim FileRequest As String = r.ReadString()

Dim File As SharedFile
Dim Filename As String
Filename = ""
For Each File In [Global].SharedFiles
    If File.Guid.ToString() = FileRequest Then
        Filename = File.FileName
        Exit For
    End If
Next

' Check file is shared.
If Filename = "" Then
    w.Write(Messages.FileNotFound)
Else

    ' Create ListView.
    Dim ListViewItem As New ListViewItemWrapper(ListView, Filename,
"Initializing")

    ' (Add error handling.)
    ' Open file.
    Dim Upload As New
FileInfo(Path.Combine([Global].Settings.SharePath, Filename))

    ' Read file.
    Dim TotalBytes As Integer = Upload.Length
    w.Write(TotalBytes)

    Dim TotalBytesRead, BytesRead As Integer

    Dim fs As FileStream = Upload.OpenRead()

```

```

Dim Buffer(1024) As Byte
Dim Percent As Single
Dim LastWrite As DateTime = DateTime.MinValue
Do
    BytesRead = fs.Read(Buffer, 0, Buffer.Length)
    w.Write(Buffer, 0, BytesRead)
    TotalBytesRead += BytesRead
    Percent = Math.Round((TotalBytesRead / TotalBytes) * 100, 0)
    If DateTime.Now.Subtract(LastWrite).TotalSeconds > 2 Then
        LastWrite = DateTime.Now
        ListViewItem.ChangeStatus(Percent.ToString() & "%
transferred")
    End If
    Thread.Sleep(TimeSpan.FromSeconds(1))
Loop While BytesRead > 0

fs.Close()

ListViewItem.ChangeStatus("Completed")

End If

Stream.Close()

_Working = False
End Sub

Public Sub StartUpload()
    If _Working Then
        Throw New ApplicationException("Already in progress.")
    Else
        _Working = True
        UploadThread = New Threading.Thread(AddressOf Upload)
        UploadThread.Start()
    End If
End Sub

```

```
Public Sub Abort()  
    If _Working Then  
        UploadThread.Abort()  
        'UploadThread.Join()  
        _Working = False  
    End If  
End Sub  
  
Public ReadOnly Property Working() As Boolean  
    Get  
        Return _Working  
    End Get  
End Property  
  
End Class
```

Code for the Discovery Service:

Code for the structures used:

Namespace Discovery

Public Class Peer

Public Guid As Guid

Public IP As String

Public Port As Integer

End Class

Public Class SharedFile

Public Guid As Guid

Public FileName As String

Public FileCreated As Date

Public Peer As New Peer()

Public Keywords() As String

End Class

End Namespace

Code for the Global Class:

```
Imports System.Web
Imports System.Web.SessionState

Namespace Discovery

Public Class [Global]
    Inherits System.Web.HttpApplication

#Region " Component Designer Generated Code "

    Public Sub New()
        MyBase.New()

        'This call is required by the Component Designer.
        InitializeComponent()

        'Add any initialization after the InitializeComponent() call

    End Sub

    'Required by the Component Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Component Designer
    'It can be modified using the Component Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
    components = New System.ComponentModel.Container()
End Sub

#End Region
```

```

Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
    ' Fires when the application is started
End Sub

Sub Session_Start(ByVal sender As Object, ByVal e As EventArgs)
    ' Fires when the session is started
End Sub

Sub Application_BeginRequest(ByVal sender As Object, ByVal e As
EventArgs)
    ' Fires at the beginning of each request
End Sub

Sub Application_AuthenticateRequest(ByVal sender As Object, ByVal e As
EventArgs)
    ' Fires upon attempting to authenticate the use
End Sub

Sub Application_Error(ByVal sender As Object, ByVal e As EventArgs)
    ' Fires when an error occurs
End Sub

Sub Session_End(ByVal sender As Object, ByVal e As EventArgs)
    ' Fires when the session ends
End Sub

Sub Application_End(ByVal sender As Object, ByVal e As EventArgs)
    ' Fires when the application ends
End Sub

End Class

End Namespace

```

Code for the Discovery Service Page:

```
Imports System.Web.Services
Namespace Discovery

Public Class DiscoveryService
    Inherits System.Web.Services.WebService

    #Region " Web Services Designer Generated Code "

    Public Sub New()
        MyBase.New()

        'This call is required by the Web Services Designer.
        InitializeComponent()

        'Add your own initialization code after the InitializeComponent()
call

    End Sub

    'Required by the Web Services Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Web Services
Designer
    'It can be modified using the Web Services Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
        components = New System.ComponentModel.Container()
    End Sub

    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
        'CODEGEN: This procedure is required by the Web Services Designer
        'Do not modify it using the code editor.
        If disposing Then
            If Not (components Is Nothing) Then
```

```

        components.Dispose()
    End If
End If
MyBase.Dispose(disposing)
End Sub

```

```

#End Region

```

```

Private DB As New P2PDatabase()

```

```

<WebMethod()> _

```

```

Public Function Register(ByVal peer As Peer) As Boolean

```

```

    Try
        DB.AddPeer(peer)
        Return True
    Catch err As Exception
        Trace.Write(err.ToString)
        Return False
    End Try

```

```

End Function

```

```

<WebMethod()> _

```

```

Public Function RefreshRegistration(ByVal peer As Peer) As Boolean

```

```

    Try
        DB.RefreshPeer(peer)
        Return True
    Catch err As Exception
        Trace.Write(err.ToString)
        Return False
    End Try

```

```

End Function

```

```

<WebMethod()> _

```



```

Public Sub Unregister(ByVal peer As Peer)

    Try
        DB.DeletePeerAndFiles(peer)
    Catch err As Exception
        Trace.Write(err.ToString)
    End Try
End Sub

<WebMethod()> _
Public Function PublishFiles(ByVal files() As SharedFile, ByVal peer
As Peer) As Boolean

    Try
        DB.AddFileInfo(files, peer)
        Return True
    Catch err As Exception
        Trace.Write(err.ToString)
        Return False
    End Try

End Function

<WebMethod()> _
Public Function SearchForFile(ByVal keywords() As String) As
SharedFile()

    Try
        Return DB.GetFileInfo(keywords)
    Catch err As Exception
        Trace.Write(err.ToString)
        Dim EmptyArray() As SharedFile = {}
        Return EmptyArray
    End Try
End Function

End Class
End Namespace

```

5. Conclusion and Future Scope:

This software is basically a File Transfer system to be used on any Local Area Network. It is useful for usage in organizations where files have to be sent to others frequently or the work is being shared by a team and they have to proof-read the work of others before presenting it to a client. It can also be used by students on a LAN to transfer various files which are useful to their studies or for entertainment.

In future, the feature of Load-Sharing can be added, that is, the file can be copied in parts from various users who are sharing it so as to share the load of the machines. Moreover, security features and encryption could be provided in the software. It can be scaled to be used over the internet, so that people requiring files from remote locations can easily obtain them.

6. References:

- Tanenbaum, Andrew S. *Computer Networks Fourth Edition*. Prentice Hall
- O'Reilly, *Peer to Peer: Harnessing the Power of Disruptive Technologies*
- Holzner, Steven *Visual Basic 6 Black Book*. The Coriolis Group
- Reid, Fiach *Network Programming in .Net*. Digital Press.

7. Installation Guide:

i. Contents of CD

- The CD contains two main folders, Part 1 and Part 2. In the folder 'part 1' there is a folder Setup that contains sub folders debug and release. It also contains .Net Framework 1.1. In the folder 'part 2', there is a folder Setup which has sub folders dotnetfx, Fileswapper, instmsi and the setup.exe file.

ii. Software Requirements

- Both the setups require the Computer running them to have Windows Xp or Windows 98 installed on them.
- The Part 1 code requires .Net Framework 1.1 which has to be installed before the software is setup on the machine. It also requires a .dll file which is inbuilt into the setup.
- The Part 2 code requires .Net Framework 2.0 which is inbuilt into the setup of the software. It will be installed automatically if not found on the machine. It also requires 1 machine to have Internet Indexing Services(IIS) and SQL server to be installed. This machine will act as Server for the files being shared. The IP of the machine running the server has to one provided by us as it is inbuilt into the setup and the clients will not run unless it is the same. Here we have put the IP as 172.16.10.254.

iii. Hardware Requirements

- The machines running the codes should have minimum 128MB RAM and all should have a Pentium III or higher processor.

iv. Test Layout with Expected output.

- There is no specified test layout.
- In the Part 1 code the client will run and all those having started a server on the same port will be able to chat and transfer files with each other. A client can also scan for all servers running on a specified port.
- In the Part 2 code the client will run and in the Options form there is a path to a folder on which the files are to be shared. All the other clients will similarly share their files and in the Search option in the main form, one can search the

required file and if it is found, it will be displayed in the List Box and it can be downloaded by double-clicking it.

