Master of Science Thesis

Mobile File Sharing Application using Bluetooth

2005-11-18

Authors:
Henrik Andersson
Håkan Bertilsson

Abstract

The evolution of mobile phones follows that of computers. Many features that are available to computers today also exist on mobile phones. File sharing between computers is very popular today so why not create a tool that supports the possibility to do this between mobile phones too?

Bluetooth File Share is a mobile application, programmed in J2ME that supports file sharing between several mobile phones. The application allows the user to choose which files to share from his/her mobile phone, and specify search criteria to filter files from other mobile phones. Found files can be downloaded and stored on the local mobile phone.

The available features and the fact that Bluetooth communication is free gives the mobile application Bluetooth File Share a great opportunity to become a future success.

Foreword

We would like to give special thanks to Jesper Carlgren of Cybercom who has provided us with the required equipment and supported us with his expertise. We would also like to thank our mentor Ali Hamidian who has been a great support throughout the project.

Table of contents

1 Introduction 1

1.1 Background 1

1.2 Goals 1

1.3 Delimitations 1

1.4 The company Cybercom 2

2 Bluetooth 3

2.1 Overview 3

2.2 History 3

2.3 The protocol stack 4

2.4 Bluetooth networks 5

2.5 Bluetooth profiles 6

2.6 Bluetooth security 6

2.6.1 Authentication 6

2.6.2 Authorization 7

2.6.3 Encryption 7

2.7 The basic concepts of a Java Bluetooth application 7

2.7.1 Stack Initialization 8

2.7.2 Device Discovery 8

2.7.3 Device Management 8

2.7.4 Service Discovery 8

2.7.5 Communication 9

3 J2ME programming 11

3.1 Overview 11

3.2 Mobile Information Device Profile 11

3.3 What is a MIDlet? 12

3.4 Record Management System 12

4 Programs and tools 15

4.1 Hardware 15

4.2 NetBeans 4.1 15

4.3 Wireless ToolKit 2.2 15

4.3.1 Bluetooth API (JSR-82) 16

4.3.2 FileConnection (JSR-75) 17

5 Bluetooth File Share 19

5.1 General overview 19

5.2 Example programs 19

5.3 Features and GUI 20

5.4 How the program works 22

5.4.1 FileConnection 23

5.4.2 RMS 23
5.4.3 Security 24
5.5 Evaluation 24
6 Problems 25
7 Discussion 27
8 Conclusions 29
9 References 31
9.1 Literature 31
9.2 Articles 31
9.3 Web-pages 31
10 Appendices 1
10.1 Acronyms 1
10.2 Program code 2
10.2.1 DemoMIDlet.java 2
10.2.2 GUIImageServer.java 4
10.2.3 BTImageServer.java 11
10.2.4 saveImage.java 18
10.2.5 GUIImageClient.java 19
10.2.6 BTImageClient .java 25

Table of figures

Figure 1. The Bluetooth protocol stack 4
Figure 2. A scatternet with two masters (m) and four slaves (s) 5
Figure 3. Examples of packages in CLDC and MIDP 11
Figure 4. Main Menu 20
Figure 5. Search menu 20
Figure 6. Found Files menu 21
Figure 7. View Image 21
Figure 8. Share files menu 22

1 Introduction

This section gives an introduction to the project. The report begins with two chapters of theory, then the method and result are provided and finally discussion and conclusions are presented. The target audience of this report is people with some knowledge and interest in Bluetooth and/or Java programming.

1.1 Background

More and more products in our modern life are becoming more and more multifunctional. This applies especially to mobile phones that today have Bluetooth as a standard feature. Many features that for years have been used on computers are now available also in mobile phones for example games, e-mail etcetera. File sharing has been very popular on the Internet via computers so why not using your mobile telephone for the same purpose?

This project will result in an application allowing file sharing between several mobile phones using Bluetooth. The main advantage of this application, to regular file exchange, is that the user will be able to browse files from more than one mobile phone at the same time. This without bothering to manually change between different phones since this will be done by the application. The user will be able to set search criterions to find files of his/her selection, as for example music, pictures and text files etcetera. By using Bluetooth the data transmission can be done at no cost compared to using for example General Packet Radio Service (GPRS) or Universal Mobile Telecommunications Service (UMTS), where the operators often charge per downloaded kb.

1.2 Goals

The primary goal of this thesis work is to implement a program that supports file sharing between mobile phones via Bluetooth. The application should allow the user to list and download files stored in other mobile phones for example music, pictures or program files. The application is to be designed in a user friendly manner.

To be able to perform this, thorough understanding about the Bluetooth technology and Bluetooth-focused programming is required. The application will be implemented using Java 2 Micro Edition (J2ME) since it supports the target hardware. The Java code is to be implemented following Java's "code-conventions" to result in a well written code. The testing-procedures will be executed using existing emulators and three Sony Ericsson K750i mobile phones.

1.3 Delimitations

The security issues in Bluetooth are often highlighted, but since the focus of this thesis lies on the implementation of the program, these concerns will be left to others. The implementation should support file sharing in a piconet. If there is enough time the support of file sharing in a scatternet should also be examined. The primary target for the application is mobile phones created by Sony Ericsson.

1.4 The company Cybercom

This project was supported by Cybercom in Malmö[1]. Cybercom is a high-tech consultancy that uses leading-edge technologies and offers business-critical solutions within primarily telecom. The company was launched in 1995; it was listed on the Stockholm stock exchange in 1999. Cybercom has operations in 10 countries and offices in Denmark, Norway, Sweden and the UK. Cybercom has today about 400 employees.

[1] The official home page of Cybercom group

2 Bluetooth

This theoretical part explains the basic concepts of Bluetooth. Some background information about the technology and the essentials needed for this project will be given.

2.1 Overview

Bluetooth is a technology that uses short-range radio links, intended to replace cables connecting electronic devices[2]. The key features are robustness, low complexity, low power and low cost. The maximum theoretical transfer rate in Bluetooth is approximately 700 kbit/s.

Bluetooth radio modules operate in the unlicensed and globally available Industrial, Scientific and Medical (ISM) band at 2.4 GHz[3]. The Bluetooth radio is designed to operate in noisy frequency environments and uses a fast acknowledgement and frequency hopping scheme to make the link robust. Compared with other systems in the same frequency band, the Bluetooth radio hops faster and uses shorter packets. The hopping rate is 1600 hops per second[3]. The time between two hops is called a slot and each slot uses a different frequency.

2.2 History

The name Bluetooth derives from the Danish King Harald Bluetooth, who unified Denmark and Norway in the 10th century[4]. The Bluetooth technology was born in 1994. Ericsson Mobile Communications initiated a study to investigate the feasibility of a low-power, low-cost radio interface between mobile phones and their accessories. The aim was to eliminate cables between mobile phones and PC cards, headset and desktop devices, etcetera.

In 1998 a group was formed to develop the new Bluetooth wireless technology[4].
This group became known as the Bluetooth Special Interest Group (SIG). In the start at 1998, SIG consisted of five companies: Ericsson, Nokia, IBM, Toshiba and Intel.
Since the formation of the Bluetooth SIG, thousands of companies have signed the Bluetooth adopter agreement and joined the Bluetooth SIG. Among them are world leaders in the telecommunications, computing, and network industries.

The adopters are all supportive of the wireless technology and committed to developing Bluetooth products. By signing a zero-cost agreement, member companies qualify for a royalty-free license to build products based on the Bluetooth wireless technology, as well to access the Bluetooth specification and intensive training seminars[4]. The goal is to create a universal standard for a short-range, cable replacement, radio technology, to ensure devices from different manufacturers to be compatible.

[2] Bluetooth tutorial specification
[3] Mobile Communications
[4] The Official Bluetooth Website

2.3 The protocol stack

Figure 1 shows that the Bluetooth radio layer is the lowest layer in the stack of the Bluetooth specification. The radio layer controls specifications of the air interface such as frequencies, modulation and transmit power[3]. The transmit power is divided into three classes depending on the required distance of the communication, reaching from 10 cm to 100 m. The Bluetooth radio modulation scheme is Gaussian Frequency Shift Keying (GFSK) where a binary one is represented by a positive frequency deviation and a binary zero by a negative frequency deviation.

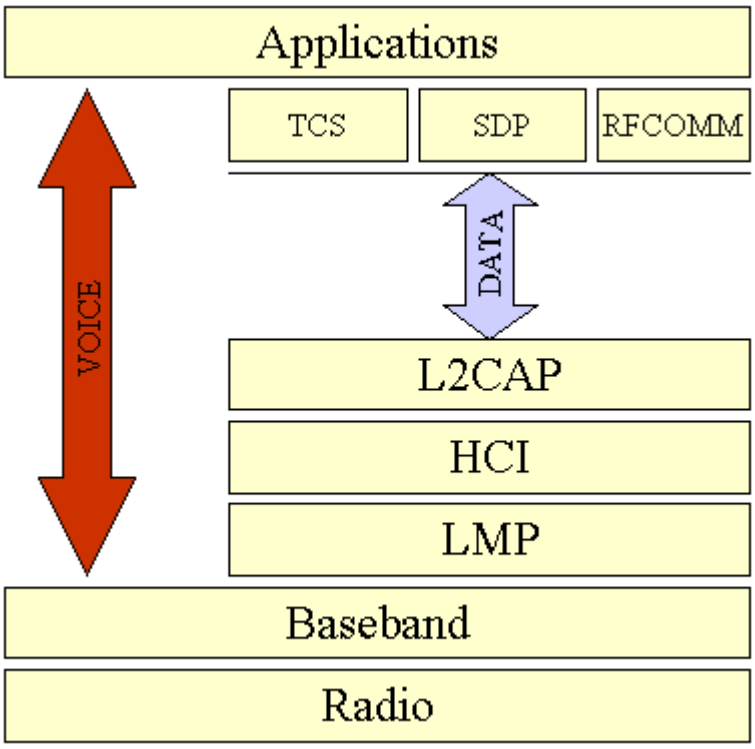

Figure 1. The Bluetooth protocol stack[5]

The Baseband is the physical layer of Bluetooth[5] . It manages basic connection establishments, packet formats and Quality of Service (QoS) parameters. The Baseband handles two types of links: Synchronous Connection-Oriented (SCO) and Asynchronous Connection-Less (ACL) link. The SCO link is a symmetric point-to-point link between a master and a single slave in the piconet whereas the ACL link is a point-to-multipoint link.

The Link Manager Protocol (LMP) controls the link setup and configuration but also handles security functions[5]. The HCI provides a command interface to the baseband controller and link manager, and access to hardware status and control registers. The Logical Link Control and Adaptation Layer Protocol (L2CAP) manage connectionless and connection-oriented services.

[5] Wireless Application Programming with J2ME and Bluetooth

RFCOMM (Radio Frequency Communications) is a simple transport protocol, which provides emulation of RS232 serial ports over the L2CAP protocol[5]. The RFCOMM protocol supports up to 60 simultaneous connections between two Bluetooth devices.

The Service Discovery Protocol (SDP) makes it possible for a device to discover what services are available on another device.

2.4 Bluetooth networks

A piconet is a collection of devices connected via Bluetooth technology in an ad hoc fashion. An ad-hoc (or "spontaneous") network is a local area network or other small network, especially one with wireless or temporary plug-in connections, in which some of the network devices are part of the network only for the duration of a communications session or, in the case of mobile or portable devices, while in some close proximity to the rest of the network[6]. A piconet starts with two connected devices, such as a portable PC and a cellular phone, and may have up to eight simultaneous connected devices. All Bluetooth devices are peer units and have identical implementations. When establishing a piconet, one unit will act as a master and the other devices will act as slaves for the duration of the piconet connection. The device that initiates the connection will become the master and the device that accepts the connection will become slave. In some exceptional situations those roles can be changed[7]. In this thesis the words client and server will be used more frequently than master and slave because it is more relevant to point out the server and client behavior in the program to understand the concept.

The Bluetooth system supports both point-to-point and point-to-multi-point connections[7]. Several piconets can be linked together to a scatternet, where each piconet is identified by a different frequency hopping sequence.

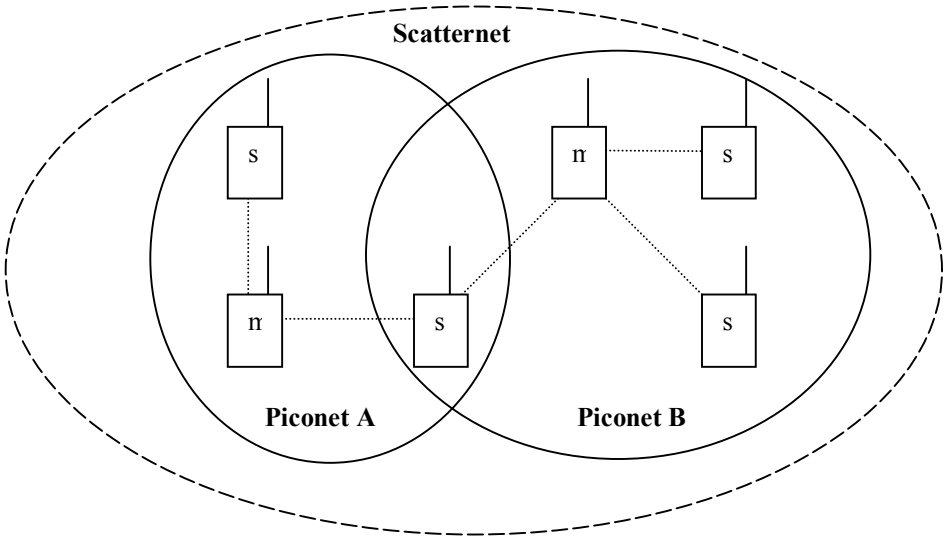

Figure 2. A scatternet with two masters (m) and four slaves (s)

[6] searchMobileComputing.com Definitions
[7] Bluetooth for Java, chapter 2

In Figure 2, there are two interconnected piconets forming a scatternet. They are able to operate within the surroundings of each other because they are using different hopping sequences, reducing mutual interference to an acceptable level. In this way it is possible to have several small groups of Bluetooth devices communicating with each other in the same area.

2.5 Bluetooth profiles

To be able to communicate via Bluetooth, devices must not only support the Bluetooth protocol but they must also support a common Bluetooth profile. A Bluetooth profile is a functionality set for Bluetooth devices. For example, if you have a mobile phone and a laptop, will the laptop be able to browse the Internet over Bluetooth via the mobile phone? Yes, if both devices support the Dialup Networking Profile. The profile concept is used to decrease the risk of interoperability problems between different manufacturers' products[8]. Below is a list of some profiles with an explanation of what they do.

- GAP (Generic Access Profile) – Device discovery and link management.
- GOEP (Generic Object Exchange Profile) – Support file transfer and object push.
- SPP (Serial Port Profile) – Setting up a virtual link between two peer devices.
- HS (Headset Profile) – Defines requirements to support the use of headsets.

2.6 Bluetooth security

Data security is a very important aspect for network communication. Data that is transmitted, both physically and wirelessly, may be caught by an eavesdropper. For application developers, the Bluetooth specification handles security in three ways; authentication, authorization and encryption[9].

2.6.1 Authentication

Authentication in Bluetooth is done using a Personal Identification Number (PIN). The PIN code is never transmitted from the client to the server. Instead the client creates a 128-bit *shared link key*, which is derived from the PIN[9]. If the PIN codes do not match, the authentication fails.

Bluetooth servers can request authentication by setting the *authenticate* parameter to *true* in the connection URL string. In the application Bluetooth File Share this is done using the Serial Port Profile (SPP) as follows[9]:

```
String url=
"btspp://localhost:F0E0D0C0B0A0009080706050403020l0;authenticate=true"
```

[8] Bluetooth Tutorial - Profiles
[9] Bluetooth for Java, chapter 9

Bluetooth clients can also require the server to authenticate it in similar way. This must be done if the server does not require authentication but the client wants to send encrypted data.

2.6.2 Authorization

Servers can choose to demand authorization from clients by adding the *authorize* parameter. Since authorization require authentication the connection URL now looks like this[9]:

```
url="btspp://localhost:F0E0D0C0B0A00090807060504030201 0;
authenticate=true;authorize=true"
```

A client cannot request authorization since it is not a requirement for encryption.

2.6.3 Encryption

Authentication and authorization makes an unwanted user unable to access the services of a Bluetooth device[9]. Encryption however aims at securing the data during a transmission. To encrypt data, an encryption algorithm and key is needed. There are two ways of encrypting information; symmetrical and asymmetrical encryption. Bluetooth uses symmetrical encryption where the same key is used to encrypt and decrypt the data. In asymmetrical encryption, two different keys are used.

In Java, Bluetooth servers can request encrypted transmission by setting the *encrypt* parameter to *true*. Since encryption requires authentication the connection URL can look like this[9]:

```
url="btspp://localhost:F0E0D0C0B0A00090807060504030201 0;
authenticate=true;encrypt=true"
```

The client can require the communication with the server to be encrypted in a similar way.

2.7 The basic concepts of a Java Bluetooth application

There are five basic steps in any kind of Bluetooth communication, not just in Java[10].

1. Stack Initialization
2. Device Discovery
3. Device Management
4. Service Discovery
5. Communication

[10] Getting started with Java and Bluetooth

2.7.1 Stack Initialization

The first thing to do when setting up a communication link is to initialize the Bluetooth stack[10]. The stack controls the Bluetooth device and is usually some piece of software or firmware. The main purpose of the initialization phase is to make the device ready for wireless communication.

2.7.2 Device Discovery

To be able to communicate with other Java Bluetooth devices the device either have to make itself visible to other devices or search for other discoverable devices in the neighbourhood[10]. The device discovery procedure is done on the client side. A Java Bluetooth device uses the Device Discovery classes to find out if there are other devices nearby. To be able to find other devices there are two classes that are used, DiscoveryAgent and DiscoveryListener. The object in use must implement the DiscoveryListener interface, this interface works like any other listener; it will notify when an event occurs, in this case, when another Bluetooth device is nearby. One good thing about the DiscoveryAgent class is that the method that searches for devices, startInquiry(), is non-blocking so other things can be done while it is searching for the devices. When the inquiry discovers a device, the Java Virtual Machine (JVM) will call the deviceDiscovered() method of the class that implemented the DiscoveryListener interface and pass a representation of the device discovered as a RemoteDevice object.

2.7.3 Device Management

Device management is performed using the classes LocalDevice and RemoteDevice[10]. These classes can give information about both the own local device and other Bluetooth devices in the vicinity. For example every Bluetooth device has its own unique address and it can be seen using the method getBluetoothAddress(). For the client to be able to find any other Bluetooth server devices it is assumed that at least one of those nearby devices are set to be discoverable. This is done by calling the setDiscoverable() method in the LocalDevice object.

2.7.4 Service Discovery

When one or more devices are found it is time to discover what services they have got to offer[10]. Just like Device Discovery, the Service Discovery is done by the DiscoveryAgent class. The method to use is searchServices() and when services are found, the JVM will call the servicesDiscovered method in the object that implemented the DiscoveryListener interface and pass a ServiceRecord object that correspond to the service. With this ServiceRecord it is possible to retrieve the remote device's URL.

Before a Bluetooth client device can do a Service Discovery on a Bluetooth server device, the Bluetooth server has to register its services in internally in the Service Discovery Database (SDDB) [10]. This is called Service Registration. To do a Service Registration the server calls Connector.open() and cast the resulting connection to a StreamConnectionNotifier. The Connector.open() method creates a ServiceRecord. This

ServiceRecord can be modified by the server. When the server sets itself in communication mode, that is calls the StreamConnectionNotifier.acceptAndOpen() method, the system automatically creates a ServiceRecord in the SDDB.

2.7.5 Communication

One way to send and receive data in Java Bluetooth is to use the RFCOMM protocol[10]. The Serial Port Profile (SPP) uses this protocol to communicate. There are two ways to start communicating in Java Bluetooth; either as a server or as a client.

After the server has done its service registration and set itself into discoverable mode it is ready to communicate. To start the communication the server creates a StreamConnectionNotifier object. This object is used to instantiate a StreamConnection object that is used to receive incoming client connections.

After a client has done device and service discovery the client can start the communication. This is done by making a stream connection to the server with the retrieved URL from the service record. When a server accepts a connection from a client it will become the slave and the client will become the master.

3 J2ME programming

The programming language used in this project is Java 2 Micro Edition (J2ME) since it is adapted for devices with less processor capacity than a computer, for example a mobile phone.

3.1 Overview

There are two configurations in J2ME that corresponds to two different kinds of devices[11]. The most advanced of them is the Connected Device Configuration (CDC) that is used by devices that have a network connection, but have less processing power than a computer. Examples of devices that fit in this category are smart phones and advanced PDAs. The other configuration has less processing power and is called the Connected Limited Device Configuration (CLDC). The CLDC, in contrast to the CDC, can only set up a non-dedicated and non-robust network connection. Most mobile phones, two-way pagers and some simple PDAs belong to this category.

3.2 Mobile Information Device Profile

The most used J2ME profile is the Mobile Information Device Profile (MIDP)[11]. Almost all mobile phones with Java support are a MIDP device. The profiles in J2ME extend the functionality of a configuration such as the CLDC.

Figure 3 presents some packages that belong to each profile.

CLDC	MIDP
Java.lang (basic core language classes)	Javax.microedition.midlet (core MIDlet classes)
Java.util (utility classes)	Javax.microedition.lcdui (user interface classes)
Java.microedition.io (network classes)	Javax.microedition.rms (data persistence classes)

Figure 3. Examples of packages in CLDC and MIDP

Some packages in the CLDC, such as java.lang and java.util, are also available in Java 2 Standard Edition (J2SE), but these are just a subset of the original packages that is optimized for micro devices[11]. In the MIDP specification, there are some requirements for a MIDP 1.0 device. It will have to support for example a minimum screen resolution of 9654 pixels and a minimum of 128 kB non-volatile memory for the MIDP implementation. In the MIDP 2.0 specification the requirements are raised to for example at least 2565 kB non-volatile memory but new functionality is also implemented. The MIDP 2.0 increases the network security with HTTPS but also simplifies the creation of games with an own package for this purpose.

[11] Bluetooth for Java, chapter 5

3.3 What is a MIDlet?

A MIDlet is a Java application that is executed on a mobile device using the MIDP[11]. A MIDlet has three states; active, paused and destroyed. One or more MIDlets packed together in a Java Archive (JAR) file form a so called MIDlet suite. The java structure of a typical MIDlet looks like this:

```
import javax.microedition.midlet.MIDlet;
public class MyApplication extends MIDlet {
      public MyApplication() {
      }
      public void startApp() {
      }
      public void pauseApp() {
      }
      public void destroyApp(boolean unconditional) {
      }
}
```

3.4 Record Management System

To store persistent data on a mobile phone Java application, the Record Management System (RMS) is often used. The RMS is a record-oriented database that is stored in the memory of the mobile device. The package *javax.microedition.rms* allows classes to read, write and sort data in the RMS. A record store is a collection of records and a record is basically a byte array with some arbitrary data[12]. Every record has a unique identifier number that is valid until the record store is deleted. The main class of the RMS is the *RecordStore*. The record store contains methods for creating, updating, deleting and querying a record store.

In MIDP programming RMS is a key feature though most mobile devices do not support local file system access for storing or retrieving data[13]. With RMS it is possible for a MIDlet to make data persistent over several invocations. MIDlets in the same MIDlet suite can use RMS to share data amongst each other. A record store is represented by a *RecordStore* object and can be retrieved by the following code segment[13]:

```
import javax.microedition.rms.*;
RecordStore rs = null;
try {
    rs = RecordStore.openRecordStore( "mydata", false );
}
catch( RecordStoreNotFoundException e ){
    // doesn't exist
}
catch( RecordStoreException e ){
    // some other error
}
```

[12] Record Managent System Basics
[13] Understanding the Record Management System

The first parameter in the *openRecordStore("mydata", false)* method is the name of the record store, the second parameter is if the record store should be created if it does not exist. If this parameter is set to *false* a *RecordStoreNotFoundException* will be thrown if the record store does not exist. If the second parameter is set to *true*, a new record store would be created if the record store does not already exist.

There can only be a single instance of a *RecordStore* object with the same name in the same MIDlet suite. If two MIDlets open a record store with the same name they both get a reference to the same record store. To close a record store the following code segment can be used[12]:

```
try {
    rs.closeRecordStore();
}
catch( RecordStoreNotOpenException e ){
    // already closed
}
catch( RecordStoreException e ){
    // some other error
}
```

If a record store has been opened by several MIDlets it will not be closed until the *closeRecordStore()* method has been called as many times as the *openRecordStore()* method has been called[12].

To add a byte array, *data*, to a record store the following code segment can be used: [12]

```
try {
    int id = rs.addRecord( data, 0, data.length );
}
catch( RecordStoreFullException e ){
    // no room left for more data
}
catch( RecordStoreNotOpenException e ){
    // store has been closed
}
catch( RecordStoreException e ){
    // general error
}
```

The first parameter in the *addRecord(data, 0, data.length)* method is a byte array, the second parameter is in what position to start in the array and the third parameter is where to stop. The value returned, *id*, is the unique identification number the record is assigned. There is also a method to delete records: [12]

```
try {
    rs.deleteRecord( id );
}
catch( InvalidRecordId e ){
    // if the recordId is invalid
}
catch( RecordStoreNotOpenException e ){
```

```
    // store has been closed
}
catch( RecordStoreException e ){
    // general error
}
```

The parameter is the identification number of the record. After the record has been deleted the record id is not reused by the record store. There are also methods for retrieving data from a record; one way of doing this is shown below: [12]

```
try {
    byte[] data = rs.getRecord( id );
}
catch( InvalidRecordIDException e ){
    // record doesn't exist
}
catch( RecordStoreNotOpenException e ){
    // store has been closed
}
catch( RecordStoreException e ){
    // general error
}
```

There is no way to modify part of the data in a record; the one thing that can be done is replacing the whole record. An example of how this can be done is shown below: [12]

```
try {
    rs.setRecord( id, data, 0, data.length );
}
catch( RecordStoreFullException e ){
    // no room left for more data
}
catch( InvalidRecordIDException e ){
    // record doesn't exist
}
catch( RecordStoreNotOpenException e ){
    // store has been closed
}
catch( RecordStoreException e ){
    // general error;
}
```

The first parameter is the identification number of the record, the second parameter, *data*, is a byte array, the third parameter states in what position to start in the array and the fourth parameter is where to stop[12].

4 Programs and tools

This section presents what programs and tools that were used during the development of the mobile application Bluetooth File Share. The software that were used can all be downloaded for free on their respective home pages. Some methods of the application are explained using Java code.

4.1 Hardware

The field tests were performed using three Sony Ericsson K750i mobile phones. The reason for this is that this model support both JSR-82 and JSR-75 that will be described below. This support is only available on newer mobile phones like for example Sony Ericsson K750i. Since mobile phones may behave differently depending on the vendor this project has concentrated on mobile phones created by Sony Ericsson.

4.2 NetBeans 4.1

NetBeans is the Integrated Development Environment (IDE) that was used to develop the program Bluetooth File Share. The IDE was chosen due to its J2ME support and compatibility with various toolkits. NetBeans is also the base in Sun Java Studio Mobility that both project members have experienced before. NetBeans also support Bluetooth emulation which in this case was a must have, to be able to develop the application.

4.3 Wireless ToolKit 2.2

Sun has created a tool for MIDlet development called the Wireless ToolKit (WTK). WTK contains an emulator and can package, compile, preverify and run CLDC and MIDP applications. WTK also contains lots of Java Specification Request (JSR) APIs that are useful for wireless programming for example[14]:

- Java Technology for the Wireless Industry 1.0 (JSR 185)
- Wireless Messaging API 2.0 (JSR 205)
- PDA Optional Packages for the J2ME Platform (JSR 75)
- Java APIs for Bluetooth (JSR 82)
- Mobile 3D Graphics API for J2ME (JSR 184)

Especially two APIs were very useful during the project. These APIs were JSR-82 Java API for Bluetooth and JSR-75 FileConnection that will be described more in detail below.

[14] Sun Java Wireless Toolkit Overview

4.3.1 Bluetooth API (JSR-82)

In order to control the Bluetooth technology of a mobile phone the JSR-82 was used. JSR-82 is an optional package defined by the Java Community Process that provides an API for Bluetooth connectivity[15]. This API contains methods to perform, for example device- and service discovery.

The code below describes how to open a connection on a Java Bluetooth device that will act as a server[10]:

```
StreamConnectionNotifier notifier = null;
StreamConnection con = null;
LocalDevice localdevice = null;
ServiceRecord servicerecord = null;
InputStream input;
OutputStream output;

// let's create a URL that contains a UUID that
// has a very low chance of conflicting with anything
String url =
  "btspp://localhost:00112233445566778899AABBCCDDEEFF;name=serialconn";
// let's open the connection with the url and
// cast it into a StreamConnectionNotifier
notifier = (StreamConnectionNotifier)Connector.open(url);

// block the current thread until a client responds
con = notifier.acceptAndOpen();

// the client has responded, so open some streams
input = con.openInputStream();
output = con.openOutputStream();

// now that the streams are open, send and
// receive some data
```

The server above is using the Bluetooth Serial Port Profile (BTSPP) as seen in the String URL that begins with *btspp://localhost*. The next part of the URL is the Universally Unique Identifier (UUID), which is *00112233445566778899AABBCCDDEEFF*[10]. This uuid is just an arbitrary identifier made up for this application. The chance of meeting someone else with the same id is close to zero. The last part of the URL is the name of the service, ;name=serialconn.

When the client wants to communicate with the BTSPP the following lines of code can be used[10]:

```
String connectionURL = serviceRecord.getConnectionURL(0, false);
StreamConnection con =(StreamConnection)Connector.open(connectionURL);
```

The first line of code retrieves the URL String to the server with the preferred service and the second line starts a connection to the server.

[15] Developing Applications with the Java APIs for Bluetooth (JSR-82)

Here is an example; in order to perform a device discovery, the two classes *DiscoveryAgent* and *DiscoveryListener* are needed[10]. These two lines of code retrieves a *LocalDevice* object and instantiates a *DiscoveryAgent* assuming *DiscoveryListener* has been implemented by the class.

```
LocalDevice localDevice = LocalDevice.getLocalDevice();
discoveryAgent = localDevice.getDiscoveryAgent();
```

To begin searching devices the method *startInquiry(DiscoveryAgent.GIAC, this)* is called like this:

```
try {
    discoveryAgent.startInquiry(DiscoveryAgent.GIAC, this);
} catch (BluetoothStateException e) {
    System.err.println("Can't start inquiry: " + e);
}
```

When a Bluetooth device is found, the JVM will call the *deviceDiscovered()* method of the class that implemented the *DiscoveryListener* interface. This method will pass a *RemoteDevice* object that represents the device discovered by the inquiry[10].

4.3.2 FileConnection (JSR-75)

The FileConnection APIs is an optional package, that is, an API set that may be licensed and added on top of J2ME configuration and profile implementations by vendors independent of the JSR process. The APIs is a part of the JSR-75 but there is no dependency between the FileConnection APIs and other APIs in the JSR-75[16].

The primary goal of the FileConnection APIs is to give access to file systems or other mounted memory cards. File connections can only be supported if the device has hardware support for file systems. The types of memory cards that contain file systems and could be supported include[16]:

- SmartMedia card
- CompactFlash® card (Registered trademark of SanDisk Corporation)
- Secure Digital card
- Memory Stick® (Registered trademark of Sony Corporation)
- MultiMediaCard

The main class of the FileConnection API is the *FileConnection* class. This is an interface and is used to gain access to files or directories on a device's memory or a memory card. A *FileConnection* object can only reference to one file or directory at the same time but can be reused and set to refer to other files or folders. In most cases, the best way to refer to another file or directory is to create a new *FileConnection* object. To open a new file connection on the local devices' memory to a file called *c:/pictures/Winter.png*, the following code segment can be used[16]:

16 fileconnection_spec_1.00

```
try {
     FileConnection fconn =
(FileConnection)Connector.open("file://localhost/c:/pictures/Winter.png
");
     // If no exception is thrown, then the URI is valid, but the file
may or may not exist.
     if (!fconn.exists())
         fconn.create();  // create the file if it doesn't exist
     fconn.close();
 }
 catch (IOException ioe) {
 }
```

In the code example the *fconn.exist()* method is used to check if the file exist. If the file does not exist it will be created. There is also a *delete()* method that can be used in a similar way as the *create()* method[16].

A *FileConnection* object has got one underlying *InputStream* and one *OutputStream*. Trying to open more than one *Input-* or *OutputStream* will result in an *IOException* being thrown. An *IOException* is also thrown if trying to open an *Output-* or *InputStream* when the file connection is closed[16].

Access to files and directories are often restricted on devices, this is to protect the device's and user's files from unauthorized access. In a situation where access is not allowed, a *java.lang.SecurityException* is thrown from the *Connector.open()* method. The *FileConnection* APIs can not access RMS databases[16].

Another class in the *FileConnection* API is the *FileSystemListener* class. This class is used for receiving status notification when adding or removing a file system root. This can be achieved by inserting or removing a card from a device or by mounting or demounting file systems to a device[16].

To add or remove a *FileSystemListener* object the *FileSystemRegistry* class is used. This class is a central registry for file system listeners. An important method of the *FileSystemRegistry* class is the *listRoots()* method. This method returns all mounted roots on the device including memory cards[16].

5 Bluetooth File Share

The application that is the result of this project is called Bluetooth File Share. This section explains how the program works and the features it provides. Some parts of the program will be explained with code examples in Java.

5.1 General overview

Bluetooth File Share is a program that allows file sharing between mobile phones. The program consists of one server and one client part. The server part gives the user an opportunity to choose what files on his/her device that other users are allowed to download. In Bluetooth File Share, this is called to publish a file. The client is allowed to select what kind of files to search for, for example music, pictures, text files etcetera. If the selected search criteria are met, the found files are displayed. A file is placed into a category depending on the three last characters in its filename for example the category Pictures contains jpg, bmp, gif, and png. The client can select to download the file, or if the file is a picture, the file can be viewed before saving.

For a client to be able to find files on a server, both sides must support and have started the application. Both sides can be active at the same time. When searching for files, the client detects all discoverable devices in range and search files from them one by one. This results in that the client can find files from more than eight phones in one search without setting up a scatternet. If two devices share a file with the same file name, the program assume it is the same file and decides to only present one of them. In order to be able to store files, a mobile phone with a memory card at location e: is required. This implies for most Sony Ericsson mobile phones.

5.2 Example programs

When the project started a lot of research in Bluetooth programming was performed. To get an idea of the opportunities with the technique, open source programs written by other developers were examined. Especially two programs turned out to be extra helpful; the BluetoothDemo and PDAPdemo. BluetoothDemo shows how to send and receive images, located at a specified path, using a client – server structure. The user can select which pictures to share with other users. This program does not use RMS and therefore does not store the settings whether the images are shared or not from one execution to another.

The PDAPdemo is an example program included in WTK2.2 that shows how to use the FileConnection API. The program can navigate in the file system of the mobile phone and view and create certain files. This helped us to get into and use the file system of the mobile phone instead of just searching for files at a specified path.

5.3 Features and GUI

When the user starts Bluetooth File Share, a menu with four alternatives *Search*, *Share*, *Help* and *About* is displayed as shown in Figure 4. As soon as the program has started the mobile phone is put into server mode and shared files are available for others to download. The *Help* and *About* alternatives presents brief how the program works and information about the program respectively.

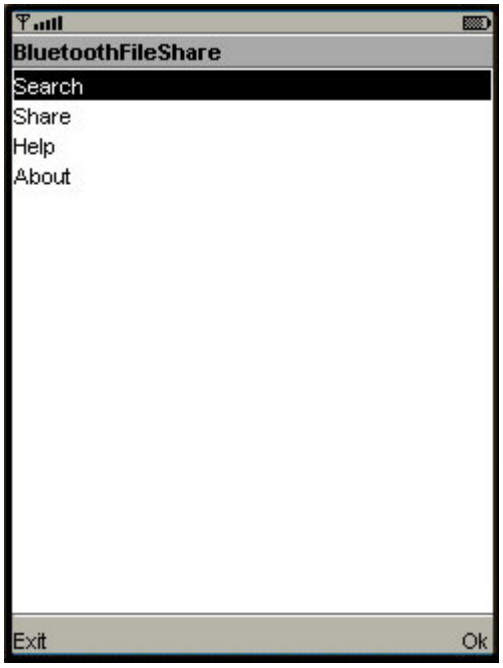

Figure 4. Main Menu

Figure 5. Search menu

If *Search* is selected, the mobile phone is also put in client mode and prepares to search for files. Several search alternatives are shown as can be seen in Figure 5, such as music, pictures, text files etcetera. The user selects the preferred file types by selecting the checkboxes next to the predefined categories and chooses *Find*.

If no files are found, an error message will be shown. This is either because no devices were found or that some files were found but their categories were not selected by the user. This generates two different system messages. Otherwise, if files are found, the found files are displayed as can be seen in Figure 6 with the options to *Download File* or *View Image*. If some of the found files are pictures, they can be viewed before saving by selecting *View Image*. If *View Image* is selected for another file type, nothing happens.

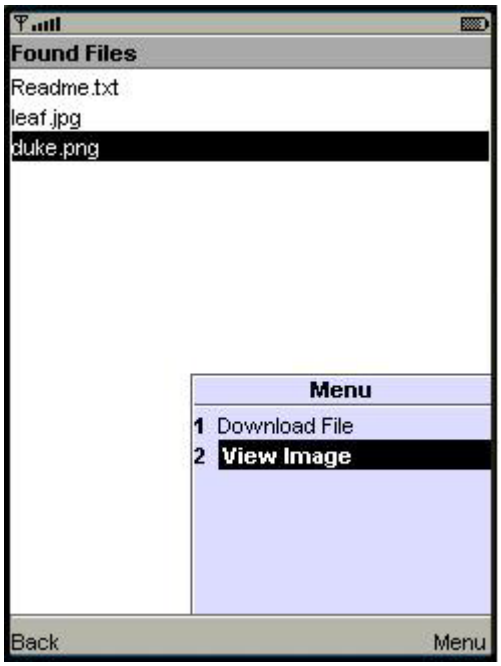

Figure 6. Found Files menu

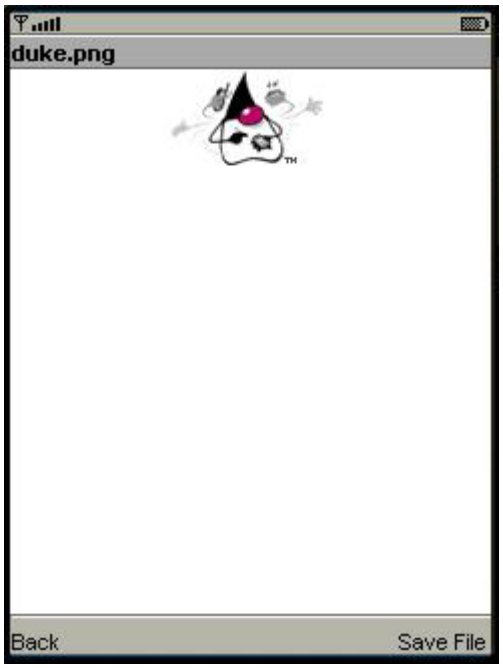

Figure 7. View Image

Figure 7 shows the picture and gives the opportunity to save the image on the phone or to go back. The file is saved at a specified location on the memory card given that the root of the memory card is *e:*. This procedure is the same as if *Download File* is selected without viewing the image before.

If *Share* is selected from the main menu, a file browser of the mobile phone is displayed as can be seen in Figure 8. Here the user can navigate through the internal memory of the mobile phone or through the memory card to choose what files to share. To make a file available to others, a file must be shared by selecting the option *Publish file*. The icon to the left of the filename is turned green when a file has been published and purple if a file is not shared. The path to the current directory is visible at the top of the screen, in this case “root1/”.

Figure 8. Share files menu

A user can view an image or a text file on the server side, before deciding to publish it or not, by selecting *View*. If the user tries to view another file type, nothing will happen. The option *Main Menu* returns the user to the main menu.

5.4 How the program works

The application is made up by two parts; one server side, to be able to share files, and one client side, to be able to search and download files. The program basically works as follows:

- When the program starts up, Bluetooth is initialized and the device is set to be discoverable. That is, a file sharing service is put in the Service Discovery Database (SDDB) and this makes published files searchable for other users. The server side then awaits incoming connections from clients. This procedure is always done and it corresponds to the server side of the application.
- A device, the client, discovers nearby devices and then for each discovered device it searches for services that is, published files.
- The client requests a certain file and the server sends the file to the client.

Since no point-to-multi-point connections are done in the program, no scatternets are established.

5.4.1 FileConnection

The fileConnection API gives access to the local file system of the mobile phone. The support of fileConnection is today (2005) only available in some new mobile phones like for example Sony Ericsson's K750i that was used in this project.

When a file is to be saved for the first time, the program must know where to store the file. Bluetooth File Share creates a folder named *download* located at *e:/MSSEMC/Media files/other/* where all files are stored. In this folder everyone has permission to read and write. This is written in Java as follows:

```
FileConnection fconn =
 (FileConnection)Connector.open("file://localhost/e:/MSSEMC/Media
files/other/download/");
fconn.mkdir();
```

When the directory has been created, the program creates an empty file with the same name as the selected file and copies the received data into that file. This is done as shown below where *fconn2* contains the URL to the empty file, and *fileData* is a vector of bytes containing the data of the file to be saved.

```
OutputStream out = fconn2.openOutputStream();
out.write(fileData);
out.flush();
```

5.4.2 RMS

In the program a *RecordStore* object is used to save settings from one execution to another. In our case a single record is used to save a String array which contains filenames that are published. This is because the user should not have to choose those files over and over again. If the program were to be further developed one can imagine that more settings would be used and therefore more records would be saved.

The *setupRecordStore()* method is called every time the program starts to retrieve which files that are shared. The following code segment shows the method:

```
public void setupRecordStore() {
        try {
            recordStore = RecordStore.openRecordStore("recordStore",
            true);
            System.out.println("recordStore open");
            RecordEnumeration enume = recordStore.enumerateRecords(
                    null, null, false );
            int id =0;
            String st = null;
            while( enume.hasNextElement() ){
                id = enume.nextRecordId();
                recordStore.getRecord(id, data, 0);
                st = din.readUTF();
                din.reset();
                Boolean b = new Boolean();
```

```
                b.setBoolean(true);
                published2.put(st, b);
            }
            enume.destroy();
        } catch (Exception e) {
            showAlert("RecordStore not open. "+ e.getMessage());
            System.out.println(e);
        }
}
```

The *RecordStore.openRecordStore("recordStore", true)* method opens the *RecordStore* recordStore and if it does not exist it is created. If the program has been executed on the device before the record store will exist and if files has been shared in previous executions of the program the record store will contain them. The *recordStore.enumerateRecords(null, null, false)* method returns an enumeration of all records in the record store, that is shared files. The *while (enume.hasNextElement()){* method extracts the filenames from the records and puts them into the *published2 HashTable* which is used during execution to manage shared files.

5.4.3 Security

This project does not focus on security issues in Bluetooth but to create a working file share application. Since devices sometimes have trouble connecting using the security methods described in Section 2.6 Bluetooth security, Bluetooth File Share does not use any of them.

5.5 Evaluation

One of our goals was to make a user friendly application and an evaluation of the application was planned to investigate whether it was easy to use or not. But since there were problems signing the MIDlet, see Section 6 Problems, this evaluation was never done. One alternative was to perform the test at a computer to get rid of the signing problems but since a computer environment would have been to unnatural to the user no evaluation was done.

6 Problems

The problems that were encountered during the project are presented in this section.

At the start of this project a list of features that the program should support was made and a couple of prototypes of the design and layout of the program were created. According to one layout the program would show the file name, file size and file type but also the name of the host who shared the file. The plan was to be able to choose how to sort the found files according to these parameters which could be good if many files are found. This plan was abandoned early in the project due to the fact that only the file name alone sometimes filled an entire row on the screen.

The most frustrating problems during this project have been when the simulator does not behave exactly as the mobile phones. Functionality that works on the simulator does not necessarily means it works on the phones. This led to many transfers and installations of the program on the mobile phone each time a test of certain functionality was made. This takes even longer time when the phone sometimes dumps memory or other errors occur during the file transfer from the computer to the phone.

One problem that have not been solved is the lack of a certificate required to sign the application. Now the user, on the mobile device, has to confirm that the program is permitted to read and write to the file system every time a new directory is opened. This problem is very annoying and this problem can be solved with a signed MIDlet. Unfortunately certificates are not for free and the certificates needed to sign the MIDlet to work properly is hard to get. Cybercom may invest in a certificate so this can be implemented in a later version.

Another feature that was planned to be implemented was the possibility to select whether to share an entire folder or not. This was not implemented since this would cause lots of key-pressing as explained above. Also the option of selecting where to save the downloaded files was overlooked for the same reason.

If two devices try to search for each other's files at exactly the same time they will not find each other's files. A similar problem occurs when two devices want to download a file from one server at the same time, this problem could probably be solved using more threads. The possibility to run the program on mobile phones from other vendors than Sony Ericsson has not been investigated due to lack of equipment and time.

7 Discussion

Below the project member's own opinions and thoughts about the project are revealed.

The goal of this project was to create an application for mobile phones that enabled the user to share and download files from several other users at the same time. We think that this project has been quite successful since our primary goal has been met. During the programming, Java's "code-conventions" has been followed to a certain extent but this could be improved in a later version.

This project has taught us lots about both Java programming and Bluetooth. We have also learned that simulators are not always trustworthy but they can be seen as a good compliment to real testing. A lot of information and help is available on the Internet for people who want to learn about Bluetooth-programming with Java. We did expect to encounter more problems that would take longer time to solve than we actually did.

Bluetooth range from 10 cm to 100 m depending of the transmitted power as explained in Section 2.3 The protocol stack. Most mobile phones today using Bluetooth only reach a radius of 10 m which results in that fewer files are found than if the search radius would be 100 m. We believe that the future will bring more applications for Bluetooth. Mobile devices will have more battery capacity, high capacity memory cards, increased transmission rate and they will reach longer. This will greatly enhance the use of our program.

With more time available, we would have implemented more features into Bluetooth File Share such as; the possibility to share a directory and its sub directories, choose where to store the downloaded files, point-to-multipoint connections, etcetera. Since Cybercom has shown interest in continuing the development of this application, many of the features discussed above may be implemented in a later version.

We have been speaking with several people working at Cybercom if they believe there is a future for Bluetooth File Share. The program has been received with curiosity and got many positive comments for the innovative idea and possibilities. One pessimistic comment we received was that due to the fact that mobile operators (such as Vodafone and Telia), has got great influence of the mobile manufacturers (such as Sony Ericsson and Samsung), the program could be hard to distribute. For operators to be able to charge people they want them to download music, ring tones etcetera via UMTS or GPRS and not for free using Bluetooth. This is an important comment that forces us to distribute the application in other ways, if that decision is made.

8 Conclusions

Bluetooth File Share is an application that supports simultaneous communication between several mobile phones using Bluetooth. The goal of this thesis work was to create an application that supports file exchange between mobile phones. To be able to do this for free, Bluetooth was the most effective technology to use. In the application the user is given the possibility to share only the files he/she desires and also the possibility to search for different file types, from other mobile devices such as music, pictures or text files.

This project has given valuable experience about both Java programming and Bluetooth. A thing to remember is that simulators are not always trustworthy but can be seen as a good compliment to real testing. A lot of information and help is available on the Internet for people who want to learn more about Bluetooth-programming with Java.

We believe that the future will bring more applications using Bluetooth and more battery capacity for mobile phones. This and high capacity memory cards together with increased transmission rate will greatly enhance the use of our program. If the problem with the MIDlet signing and minor bugs will be solved the future of Bluetooth File Share looks very bright.

9 References

Below are the references used to write this thesis.

9.1 Literature

[3] Jochen Schiller, "Mobile Communications" chapter 7.5, 2003

[7] Bruce Hopkins and Ranjith Anthony, "Bluetooth for Java" chapter 2, 2003

[9] Bruce Hopkins and Ranjith Anthony, "Bluetooth for Java" chapter 9, 2003

[11] Bruce Hopkins and Ranjith Anthony, "Bluetooth for Java" chapter 5, 2003

9.2 Articles

[5] Qusay H. Mahmoud, 2003, "Wireless Application Programming with J2ME and Bluetooth", acc. 2005-07-06.
http://developers.sun.com/techtopics/mobility/midp/articles/bluetooth1/index.html

[12] Eric Giguere, 2001-02-20, "Record Management System Basics", acc. 2005-07-05
http://developers.sun.com/techtopics/mobility/midp/ttips/rmsbasics/index.html

[13] Eric Giguere, 2004-02, "Databases and MIDP, Part 1: Understanding the Record Management System", acc. 2005-10-21
http://developers.sun.com/techtopics/mobility/midp/articles/databaserms/index.html

[15] Sony Ericsson, 2004 "Developing Applications with the Java APIs for Bluetooth (JSR-82)", acc. 2005-10-21
http://www.microjava.com/articles/Bluetooth-jsr-82-training.pdf

[16] Fileconnection specification 1.0
http://www.jcp.org/aboutJava/communityprocess/final/jsr075/

9.3 Web-pages

[1] The official home page of Cybercom Group, acc. 2005-07-05
http://www.cybercomgroup.com/templates/CCPage.aspx?id=3064

[2] Palo Wireless, "Bluetooth tutorial specification", acc. 2005-10-05
http://www.palowireless.com/infotooth/tutorial.asp

[4] The Official Bluetooth Website, acc. 2005-11-01
http://www.bluetooth.com/about/

[6] searchMobileComputing.com Definitions, acc. 2005-11-17
http://searchmobilecomputing.techtarget.com/sDefinition/0,290660,sid40_gci213462,00.html

[8] Palo Wireless, “Bluetooth Tutorial – Profiles”, acc. 2005-10-21
http://www.palowireless.com/infotooth/tutorial/profiles.asp

[10] Bruce Hopkins, “Getting Started with Java and Bluetooth”, acc. 2005-10-03
http://today.java.net/pub/a/today/2004/07/27/bluetooth.html

[14] Sun Java Wireless Toolkit Overview, acc. 2005-10-18
http://java.sun.com/products/sjwtoolkit/overview.html

10 Appendices

There are two appendices attached to this report, Acronyms and Program code.

10.1 Acronyms

Acronym	Short for	Explanation
ACL	Asynchronous Connection-Less	A point-to-multipoint link between the master and all the slaves participating on the piconet
API	Application Programming Interface	A set of public methods
CDC	Connected Device Configuration	J2ME configuration for smart phones and advanced PDAs.
CLDC	Connected Limited Device Configuration	J2ME configuration for most mobile phones and PDAs.
GAP	Generic Access Profile	Bluetooth profile that support device discovery and link management
GFSK	Gaussian Frequency Shift Keying	The modulation scheme used by Bluetooth radio.
GOEP	Generic Object Exchange Profile	Bluetooth profile that support file transfer and object push.
GPRS	General Packet Radio Service	Packet based communication service
HCI	Host Controller Interface	The driver interface for the physical bus that connects LMP and L2CAP
HS	Headset profile	Bluetooth profile that defines requirements for the support of headsets.
IDE	Integrated Development Environment	An editor that lets you edit, compile and debug all from within the same program.
JAR	Java ARchive	Contains the class files of a Java MIDlet
J2ME	Java 2 Micro Edition	A programming language for mobile telephones.
J2SE	Java 2 Standard Edition	A programming language for computers.
JSR	Java Specification Request	Where requests for changes to the Java language are presented
JVM	Java Virtual Machine	The machine code for an ideal Java CPU
L2CAP	Logical Link Control and Adaption Protocol	Receives application data and adapts it to the Bluetooth format. Handles quality of service.
LMP	Link Manager Protocol	Establishes connections and manages piconets. Also responsible for security services.
MIDlet	Mobile Information Device appLET	Tiny Java programs
PDA	Personal Digital Assistant	Handheld computer
PIN	Personal Identification Number	Security code used during authentication.
RFCOMM	Radio Frequency Communications	A Bluetooth transport protocol using RS232.
SCO	Synchronous Connection-Oriented	Symmetric point-to-point link between a master and a single slave in the piconet
SDDB	Service Discovery Database	Database where the server contains services.
SDP	Service Discovery Protocol	Makes it possible for a device to discover what services are available on another device.
SIG	Special Interest Group	Group of companies that develops Bluetooth.
SPP	Serial Port Profile	Bluetooth profile that sets up a virtual link between two devices.
UMTS	Universal Mobile Telecommunications Service	3G mobile system
WTK	Wireless ToolKit	Toolkit containing APIs for wireless communication.
UUID	Universally Unique Identifier	Each service and service attribute is uniquely identified by a UUID.

10.2 Program code

The program Bluetooth File Share consists of six classes; DemoMIDlet.java, GUIImageServer.java, BTImageServer.java, GUIImageClient.java, BTImageClient.java and saveImage.java. The commented code is shown below.

10.2.1 DemoMIDlet.java

```
/*
 * DemoMIDlet.java
 */
package example.bluetooth.demo;
import javax.microedition.midlet.MIDlet;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.List;

import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;

/**
 * The main class displays the Main Menu.
 */
public final class DemoMIDlet extends MIDlet implements CommandListener {

    /** Keeps the help message of this demo. */
    private final String helpText = "Select \"Share\" to choose which files to share " +
            "and select \"Search\" to search for files. Before you share files you " +
            "can view them by choosing \"View\". This functionality only works " +
            "on pictures and text files. You can also view found pictures before you " +
        "save them. Downloaded files are stored at e:/MSSEMC/Media files/other/download/";

        /** Keeps the about message of this demo. */
    private final String aboutText = "Bluetooth File Share is a M.Sc thesis at the " +
            "Department of Communication Systems at Lund Institute of Technology " +
            "created by Henrik Andersson and Håkan Bertilsson 2005.";

    /** The messages are shown in this demo this amount of time. */
    static final int ALERT_TIMEOUT = 2000;

    /** Soft button for exiting the demo. */
    private final Command EXIT_CMD = new Command("Exit", Command.EXIT, 2);

    /** Soft button for launching a client or sever. */
    private final Command OK_CMD = new Command("Ok", Command.SCREEN, 1);

    /** A list of menu items */
    private static final String[] elements = { "Search", "Share", "Help", "About" };

    /** A menu list instance */
    private final List menu = new List("BluetoothFileShare", List.IMPLICIT,
            elements, null);

    /** A GUI part of server that publishes images. */
    private GUIImageServer imageServer;

    /** A GUI part of client that receives image from client */
    private GUIImageClient imageClient;

    private final Alert helpScreen = new Alert("Help");

    private final Alert aboutScreen = new Alert("About");

    private final Command backCommand = new Command("Back", Command.BACK, 2);

    /** Shows the help message. */
```

```
private final Command helpCommand = new Command("Help", Command.HELP, 1);

private final Command aboutCommand = new Command("About", Command.HELP, 1);

/**
 * Constructs main screen of the MIDlet.
 */
public DemoMIDlet() {
    menu.addCommand(EXIT_CMD);
    menu.addCommand(OK_CMD);
    menu.setCommandListener(this);

    // prepare help screen
    helpScreen.addCommand(backCommand);
    helpScreen.setTimeout(Alert.FOREVER);
    helpScreen.setString(helpText);
    helpScreen.setCommandListener(this);

    aboutScreen.addCommand(backCommand);
    aboutScreen.setTimeout(Alert.FOREVER);
    aboutScreen.setString(aboutText);
    aboutScreen.setCommandListener(this);
}

/**
 * Creates the view and action buttons.
 */
public void startApp() {
    show();

    //Start imageServer
    imageServer = new GUIImageServer(this);
}

/**
 * Destroys the application.
 */
protected void destroyApp(boolean unconditional) {
    if (imageServer != null) {
        imageServer.destroy();
    }

    if (imageClient != null) {
        imageClient.destroy();
    }
}

/**
 * Does nothing.
 */
protected void pauseApp() {}

/**
 * Responds to commands issued on "client or server" form.
 */
public void commandAction(Command c, Displayable d) {
    if (c == EXIT_CMD) {
        destroyApp(true);
        notifyDestroyed();
        return;
    }

    if (c == backCommand && d == helpScreen || d == aboutScreen) {
        Display.getDisplay(this).setCurrent(menu);
        return;
    }

    if (c == helpCommand) {
        Display.getDisplay(this).setCurrent(helpScreen);
        return;
    }
```

```
        if (c == aboutCommand) {
            Display.getDisplay(this).setCurrent(aboutScreen);
            return;
        }

        switch (menu.getSelectedIndex()) {
        case 0: // Search, call imageClient
            imageClient = new GUIImageClient(this);
            break;
        case 1: // Share files
            //Start method that shows which files to share
            imageServer.shareFiles();

            break;
        case 2: // Help
            //Show text
            Display.getDisplay(this).setCurrent(helpScreen);
            break;
        case 3: // About
            //Show text
            Display.getDisplay(this).setCurrent(aboutScreen);
            break;
        default:
            System.err.println("Unexpected choice...");
            break;
        }
    }

    /** Shows main menu of MIDlet on the screen. */
    void show() {
        Display.getDisplay(this).setCurrent(menu);
    }

    /**
     * Returns the displayable object of this screen -
     * it is required for Alert construction for the error
     * cases.
     */
    Displayable getDisplayable() {
        return menu;
    }
} // end of class 'DemoMIDlet' definition
```

10.2.2 GUIImageServer.java

```
/*
 * GUIImageServer.java
 */
package example.bluetooth.demo;
import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Graphics;
import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.List;
import javax.microedition.lcdui.Ticker;
import java.io.IOException;
import java.util.Vector;

import java.util.*;
import java.io.*;
import javax.microedition.io.*;
import javax.microedition.io.file.*;
import java.util.Hashtable;

import javax.microedition.io.Connector;
import javax.microedition.lcdui.ImageItem;
import javax.microedition.lcdui.*;
```

```
import javax.microedition.rms.RecordStore;
import javax.microedition.rms.*;

/**
 * Allows to customize the file list to be published,
 * creates the corresponding service record to describe this list
 * and send the files to clients by request.
 */
final class GUIImageServer implements CommandListener {

    /** This command goes to main screen. */
    private final Command backCommand = new Command("Back", Command.BACK, 2);

    /** Adds the selected file to the published list. */
    private final Command addCommand = new Command("Publish file",
            Command.SCREEN, 1);

    /** Removes the selected file from the published list. */
    private final Command removeCommand = new Command("Unpublish file",
            Command.SCREEN, 1);

    private Command viewCommand = new Command("View", Command.ITEM, 1);

    private Command mainMenuCommand = new Command("Main Menu", Command.EXIT, 3);

    /** The list control to configure files. */
    private List imagesList = new List("Bluetooth FileShare", List.IMPLICIT);

    /** Keeps the parent MIDlet reference to process specific actions. */
    private DemoMIDlet parent;

    /** The list of images file names. */
    private Vector imagesNames;

    /** These images are used to indicate the picture is published. */
    private Image onImage, offImage;

    /** Keeps an information about what images are published. */
    private boolean[] published;

    /** This object handles the real transmission. */
    private BTImageServer bt_server;

    private GUIImageClient imageClient;

    /* special string denotes upper directory */
    private final static String UP_DIRECTORY = "..";

    /* special string that denotes upper directory accessible by this browser.
     * this virtual directory contains all roots.
     */
    private final static String MEGA_ROOT = "/";

    /* separator string as defined by FC specification */
    private final static String SEP_STR = "/";

    /* separator character as defined by FC specification */
    private final static char   SEP = '/';

    private String currDirName;

    private Hashtable published2;

    /*Used for RMS */
    private RecordStore recordStore;

    private byte[] data = new byte[200]; //200 är ett bra tal...
    private ByteArrayInputStream bin = new ByteArrayInputStream( data );
    private DataInputStream din =new DataInputStream( bin );

    TextBox viewer;
    String fileNamePath="";
```

```
 private final Form imageScreen = new Form("Filename");

 /** Constucts images server GUI. */
 GUIImageServer(DemoMIDlet parent) {
     this.parent = parent;
     currDirName = MEGA_ROOT;
     published = new boolean[100];
     published2 = new Hashtable();
     setupRecordStore();
     bt_server = new BTImageServer(this);
     setupIdicatorImage();
     System.out.println(imagesList.size());

     // This is done to avoid initialization problems
     FileSystemRegistry.listRoots();
 }

 /*
  * Returns the list with fileinfo, that is if the file is published or not
  */
 public Hashtable getPublished() {
     return published2;
 }
 /**
  * Setup the publlished list using a recordStore and a hashtable
  */

 public void setupRecordStore() {
     //Opens the RecordStore to retrieve the filenames of published files,
     //if there is no previously saved recordStore a new one is created.
     try {
         recordStore = RecordStore.openRecordStore("recordStore", true);
         System.out.println("recordStore open");
         RecordEnumeration enume = recordStore.enumerateRecords(
                 null, null, false );
         int id =0;
         String st = null;
         while( enume.hasNextElement() ){
             id = enume.nextRecordId();
             //recordStore.deleteRecord(id);
             System.out.println("före getR");
             recordStore.getRecord(id, data, 0);
             System.out.println("efter getR");
             st = din.readUTF();
             System.out.println(st);
             din.reset();
             Boolean b = new Boolean();
             b.setBoolean(true);
             published2.put(st, b);
         }
         enume.destroy();
     } catch (Exception e) {
         showAlert("RecordStore not open. "+ e.getMessage());
         System.out.println(e);
     }
 }

 /**
  * Updates the recordStore, that is changes the state of the image, published
  * or not published.
  */
public boolean updateRecordStore(String fileName, boolean published) {
    try {
        int numRecords = recordStore.getNumRecords();
        boolean inRecord = false;
        String s = null;
        System.out.println("innan felet");
        RecordEnumeration enume = recordStore.enumerateRecords(null, null, false );
        int i =0;
        while( enume.hasNextElement() && !inRecord){
             i = enume.nextRecordId();
             System.out.println("före getR");
```

```
                recordStore.getRecord(i, data, 0);
                System.out.println("efter getR");
                s = din.readUTF();
                din.reset();
                inRecord=fileName.equals(s);
            }
           enume.destroy();
           if(inRecord && published==false){
               recordStore.deleteRecord(i);
           } else if(!inRecord && published) {
               ByteArrayOutputStream bout = new ByteArrayOutputStream();
               DataOutputStream dout = new DataOutputStream( bout );
               dout.writeUTF( fileName );
               dout.flush();
               byte[] b = bout.toByteArray();
               recordStore.addRecord(b, 0, b.length);
               bout.reset();
           }
           //showAlert("recordStore updated successfully");
           return true;
       }catch (Exception e){
           showAlert("RecordStore not updated. " +e.getMessage());
           System.out.println(e);
           return false;
       }
   }


    /**
     * Process the command event.
     */
    public void commandAction(Command c, Displayable d) {
        if (c == viewCommand) {
            List curr = (List)d;
            final String currFile = curr.getString(curr.getSelectedIndex());
            new Thread(new Runnable() {
                public void run() {
                    if (currFile.endsWith(SEP_STR) || currFile.equals(UP_DIRECTORY)) {
                        traverseDirectory(currFile);
                    } else {
                        // Show file contents
                        showFile(currFile);
                    }
                }
            }).start();
            return;
        }

        if (c == mainMenuCommand && d == imagesList) {
            destroy();
            parent.show();
            return;
        }

        if (c == backCommand && d == imageScreen || d == viewer) {
            Display.getDisplay(parent).setCurrent(imagesList);
            return;
        }

        if (c == backCommand && d == imagesList) {
            if (MEGA_ROOT.equals(currDirName)) {
                destroy();
                parent.show();
                return;
            } else {
                traverseDirectory(UP_DIRECTORY);
                return;
            }
        }

        /*
         * Changing the state of base of published images
```

```
         */
        int index = imagesList.getSelectedIndex();

        boolean published2State;
        published2State=((Boolean)
        published2.get((currDirName+imagesList.getString(index)))).getBoolean();
        String fileName2=imagesList.getString(index);

        // nothing to do
        if ((c == addCommand) == published2State) {
            return;
        }

        Boolean b = new Boolean();
        b.setBoolean(c == addCommand);
        published2.put(currDirName+fileName2, b);
        boolean bo = updateRecordStore(
                currDirName+fileName2, b.getBoolean());

        Image stateImg = c == addCommand ? onImage : offImage;
        imagesList.set(index, imagesList.getString(index), stateImg);

        // update bluetooth service information
        if (!bt_server.changeImageInfo(currDirName+fileName2, b.getBoolean())) {

            // either a bad record or SDDB is buzy
            Alert al = new Alert("Error", "Can't update base", null,
                    AlertType.ERROR);
            al.setTimeout(DemoMIDlet.ALERT_TIMEOUT);
            Display.getDisplay(parent).setCurrent(al, imagesList);

            b.setBoolean(!b.getBoolean());
            published2.put(currDirName+fileName2, b);
            bo = updateRecordStore(
                    currDirName+fileName2, b.getBoolean());
            stateImg = b.getBoolean() ? onImage : offImage;

            imagesList.set(index, imagesList.getString(index), stateImg);
        }
    }

    public void shareFiles(){
        boolean isBTReady=true;
        showCurrDir();
    }

    //Show the file if it is a text or a picture else do nothing
    void showFile(String fileName) {
        String fileType = fileName.substring(fileName.length()-4, fileName.length());

        if(fileType.equals(".png") || fileType.equals(".jpg")
               || fileType.equals(".gif") || fileType.equals(".bmp")
               || fileType.equals(".PNG") || fileType.equals(".JPG")
               || fileType.equals(".GIF") || fileType.equals(".BMP")) {

            byte[] imgData = bt_server.getImageData(currDirName+fileName);
            Image img = null;

            try {
                img = Image.createImage(imgData, 0, imgData.length);
            } catch (Exception e) {
                showAlert("Wrong file data. " + e.getMessage());

            }

        imageScreen.deleteAll();
        imageScreen.append(new ImageItem("", img,
                ImageItem.LAYOUT_CENTER | ImageItem.LAYOUT_VCENTER,
                "Downloaded file: " + fileName));
        imageScreen.setTitle(fileName);
        imageScreen.addCommand(backCommand);
        imageScreen.setCommandListener(this);
```

```
        Display.getDisplay(parent).setCurrent(imageScreen);

        } else if (fileType.equals(".txt") || fileType.equals(".TXT")) {
            try {
                FileConnection fc = (FileConnection)
                Connector.open("file://localhost/" + currDirName + fileName);
                if (!fc.exists()) {
                    throw new IOException("File does not exists");
                }

                InputStream fis = fc.openInputStream();
                byte[] b = new byte[1024];

                int length = fis.read(b, 0, 1024);

                fis.close();
                fc.close();

                viewer = new TextBox(fileName, null, 1024,
                        TextField.ANY | TextField.UNEDITABLE);

                viewer.addCommand(backCommand);
                viewer.setCommandListener(this);

                if (length > 0) {
                    viewer.setString(new String(b, 0, length));
                }

                Display.getDisplay(parent).setCurrent(viewer);
            } catch (Exception e) {
                Alert alert = new Alert("Error!",
                        "Can not access file " + fileName
                        + " in directory " + currDirName
                        + "\nException: " + e.getMessage(),
                        null,
                        AlertType.ERROR);
                alert.setTimeout(Alert.FOREVER);
                Display.getDisplay(parent).setCurrent(alert);
            }
        }

    }

    /** Destroys this component. */
    void destroy() {
    }

    /**
     * Creates the image to indicate the base state.
     */
    private void setupIdicatorImage() {

        // create "on" image
        try {
            onImage = Image.createImage("/images/st-on.png");
        } catch (IOException e) {

            // provide off-screen image then
            onImage = createIndicatorImage(12, 12, 0, 255, 0);
        }

        // create "off" image
        try {
            offImage = Image.createImage("/images/st-off.png");
        } catch (IOException e) {

            // provide off-screen image then
            offImage = createIndicatorImage(12, 12, 255, 0, 0);
        }
    }

    /**
```

```
 * Creates the off-screen image with specified size an color.
 */
private Image createIndicatorImage(int w, int h, int r, int g, int b) {
    Image res = Image.createImage(w, h);
    Graphics gc = res.getGraphics();
    gc.setColor(r, g, b);
    gc.fillRect(0, 0, w, h);
    return res;
}

/**
 * Show file list in the current directory .
 */
void showCurrDir() {
    Enumeration e;
    FileConnection currDir = null;
    String fileNamePath="";
    try {
        if (MEGA_ROOT.equals(currDirName)) {
            e = FileSystemRegistry.listRoots();
            imagesList = new List(currDirName, List.IMPLICIT);
        } else {
            currDir = (FileConnection)Connector.open("file://localhost/" +
                    currDirName);

            e = currDir.list();
            imagesList = new List(currDirName, List.IMPLICIT);
            // not root - draw UP_DIRECTORY
            imagesList.append(UP_DIRECTORY, offImage);
        }

        while (e.hasMoreElements()) {
            String fileName = (String)e.nextElement();

            if (currDirName!=MEGA_ROOT){
                fileNamePath=currDirName+fileName;
            } else {
                fileNamePath=fileName;
            }

            if(!published2.containsKey(fileNamePath)) {
                published2.put(fileNamePath, new Boolean());
                boolean bo = updateRecordStore(
                        fileNamePath, false);
            }

            if (((Boolean) published2.get(fileNamePath)).getBoolean()){
                imagesList.append(fileName, onImage);
            } else {
                imagesList.append(fileName, offImage);
            }
        }

        imagesList.setSelectCommand(viewCommand);

        imagesList.addCommand(viewCommand);
        imagesList.addCommand(backCommand);
        imagesList.addCommand(addCommand);
        imagesList.addCommand(removeCommand);
        imagesList.addCommand(mainMenuCommand);
        imagesList.setCommandListener(this);

        if (currDir != null) {
            currDir.close();
        }

        Display.getDisplay(parent).setCurrent(imagesList);
    } catch (IOException ioe) {
        ioe.printStackTrace();
    }
}
```

```
    void traverseDirectory(String fileName) {
    /* In case of directory just change the current directory
     * and show it
     */
        if (currDirName.equals(MEGA_ROOT)) {
            if (fileName.equals(UP_DIRECTORY)) {
                // can not go up from MEGA_ROOT
                return;
            }
            currDirName = fileName;
        } else if (fileName.equals(UP_DIRECTORY)) {
            // Go up one directory
            int i = currDirName.lastIndexOf(SEP, currDirName.length()-2);
            if (i != -1) {
                currDirName = currDirName.substring(0, i+1);
            } else {
                currDirName = MEGA_ROOT;
            }
        } else {
            currDirName = currDirName + fileName;
        }
        showCurrDir();
    }

    public void showAlert(String alert){
        Alert al = new Alert("System message", alert, null,
                AlertType.ERROR);
        al.setTimeout(7000);
        Display.getDisplay(parent).setCurrent(al, parent.getDisplayable());
    }
} // end of class 'GUIImageServer' definition

class Boolean {
    private boolean shared;

    public Boolean(){
        shared=false;
    }

    public boolean getBoolean(){
        return shared;
    }

    public void setBoolean(boolean set) {
        shared=set;
    }
}
```

10.2.3 BTImageServer.java

```
/*
 * BTImageServer.java
 */
package example.bluetooth.demo;

// jsr082 API
import javax.bluetooth.DataElement;
import javax.bluetooth.DiscoveryAgent;
import javax.bluetooth.LocalDevice;
import javax.bluetooth.ServiceRecord;
import javax.bluetooth.ServiceRegistrationException;
import javax.bluetooth.UUID;

// midp/cldc API
import javax.microedition.io.Connector;
import javax.microedition.io.StreamConnection;
import javax.microedition.io.StreamConnectionNotifier;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.DataInputStream;
import java.io.OutputStream;
```

```
import java.io.DataOutputStream;
import java.util.Vector;
import java.util.Hashtable;
import java.util.*;


/**
 * Established the BT service, accepts connections
 * and send the requested files.
 */
final class BTImageServer implements Runnable {

    /** Describes this server */
    private static final UUID PICTURES_SERVER_UUID =
            new UUID("F0E0D0C0B0A000908070605040302010", false);

    /** The attribute id of the record item with file names. */
    private static final int IMAGES_NAMES_ATTRIBUTE_ID = 0x4321;

    /** Keeps the local device reference. */
    private LocalDevice localDevice;

    /** Accepts new connections. */
    private StreamConnectionNotifier notifier;

    /** Keeps the information about this server. */
    private ServiceRecord record;

    /** Keeps the parent reference to process specific actions. */
    private GUIImageServer parent;

    /** Becomes 'true' when this component is finilized. */
    private boolean isClosed;

    /** Creates notifier and accepts clients to be processed. */
    private Thread accepterThread;

    /** Process the particular client from queue. */
    private ClientProcessor processor;

    /** Optimization: keeps the table of data elements to be published. */
    private final Hashtable dataElements = new Hashtable();

    /**
     * Constructs the bluetooth server, but it is initialized
     * in the different thread to "avoid dead lock".
     */
    BTImageServer(GUIImageServer parent) {
        this.parent = parent;

        // we have to initialize a system in different thread...
        accepterThread = new Thread(this);
        accepterThread.start();
    }

    /**
     * Accepts a new client and send him/her a requested file.
     */
    public void run() {
        boolean isBTReady = false;

        try {

            // create/get a local device
            localDevice = LocalDevice.getLocalDevice();
            System.out.println(localDevice.getBluetoothAddress());

            // set we are discoverable
            if (!localDevice.setDiscoverable(DiscoveryAgent.GIAC)) {
                // Some implementations always return false, even if
                // setDiscoverable successful
            }
```

```
            // prepare a URL to create a notifier
            StringBuffer url = new StringBuffer("btspp://");

            // indicate this is a server
            url.append("localhost").append(':');

            // add the UUID to identify this service
            url.append(PICTURES_SERVER_UUID.toString());

            // add the name for our service
            url.append(";name=Picture Server");

            // request all of the client not to be authorized
            // some devices fail on authorize=true
            url.append(";authorize=false");

            // create notifier now
            notifier = (StreamConnectionNotifier) Connector.open(
                    url.toString());

            // and remember the service record for the later updates
            record = localDevice.getRecord(notifier);

            // create a special attribute with images names
            DataElement base = new DataElement(DataElement.DATSEQ);
            record.setAttributeValue(IMAGES_NAMES_ATTRIBUTE_ID, base);

            // remember we've reached this point.
            isBTReady = true;
        } catch (Exception e) {
            System.err.println("Can't initialize bluetooth: " + e);
        }

        // nothing to do if no bluetooth available
        if (!isBTReady) {
            return;
        }

        Enumeration en = parent.getPublished().keys();
        String key = null;
        while (en.hasMoreElements()) {
            key = (String) en.nextElement();
            changeImageInfo(key,true);
        }
        // ok, start processor now
        processor = new ClientProcessor();

        // ok, start accepting connections then
        while (!isClosed) {
            StreamConnection conn = null;

            try {
                conn = notifier.acceptAndOpen();
            } catch (IOException e) {

                // wrong client or interrupted - continue anyway
                continue;
            }
            processor.addConnection(conn);
        }
    }

    /**
     * Updates the service record with the information
     * about the published images availability.
     * This method is invoked after the caller has cheched
     * already that the real action should be done.
     */
    boolean changeImageInfo(String name, boolean isPublished) {
        // ok, get the record from service
        DataElement base = record.getAttributeValue(IMAGES_NAMES_ATTRIBUTE_ID);
```

```
    // check the corresponding DataElement object is created already
    DataElement de = (DataElement) dataElements.get(name);

    // if no, then create a new DataElement that describes this image
    if (de == null) {
        de = new DataElement(DataElement.STRING, name);
        dataElements.put(name, de);
    }

    Enumeration e=dataElements.elements();
    while(e.hasMoreElements()) {
        System.out.println("element: " +e.nextElement().toString());
    }

    // we know this data element has DATSEQ type
    if (isPublished) {
        base.addElement(de);
    } else {
        if (!base.removeElement(de)) {
            System.err.println("Error: item was not removed for: " + name);
            return false;
        }
    }
    record.setAttributeValue(IMAGES_NAMES_ATTRIBUTE_ID, base);

    try {
        localDevice.updateRecord(record);
    } catch (ServiceRegistrationException sere) {
        System.err.println("Can't update record now for: " + name);
        return false;
    }
    return true;
}

/**
 * Destroy a work with bluetooth - exits the accepting
 * thread and close notifier.
 */
void destroy() {
    isClosed = true;

    // finilize notifier work
    if (notifier != null) {
        try {
            notifier.close();
        } catch (IOException e) {} // ignore
    }

    // wait for acceptor thread is done
    try {
        accepterThread.join();
    } catch (InterruptedException e) {} // ignore

    // finilize processor
    if (processor != null) {
        processor.destroy(true);
    }
    processor = null;
}

/**
 * Reads the filename from the specified connection
 * and sends this file through this connection, then
 * close it after all.
 */
private void processConnection(StreamConnection conn) {
    // read the filename first
    String imgName = readImageName(conn);

    // check this file is published and get the filename
```

```
        Hashtable published = new Hashtable();
        published = parent.getPublished();
        byte[] imgData = null;

        Boolean b = new Boolean();

        //Check that the file is still published
        if (!((Boolean) published.get(imgName)).getBoolean()) {
            imgData = "F".getBytes();

        } else {
            // load image data into buffer to be send
            imgData = getImageData(imgName);
        }

        // send image data now
        sendImageData(imgData, conn);

        // close connection
        try {
            conn.close();
        } catch (IOException e) {} // ignore
    }

    /** Send image data. */
    private void sendImageData(byte[] imgData, StreamConnection conn) {
        if (imgData == null) {
            return;
        }
        DataOutputStream out = null;
        int nbrOfSentBytes;
        int lapCount=0;
        try {
            out = conn.openDataOutputStream();
            System.out.println("out= "+out);
            //Här läggs längden på datan till
            out.write(imgData.length >> 24);
            out.write(imgData.length >> 16);
            out.write(imgData.length >> 8);
            out.write(imgData.length & 0xff);

             /* Here the transmission data is divided in smaller parts. This have
              * to be done due to phone problems.*/
            int byteNbr = 0;
            int breakpoint = 10;
            int nbrOfDataParts = imgData.length/breakpoint;
            int remainder = imgData.length%breakpoint;
            byte[] imgDataPart = null;
            if(remainder!=0){
                nbrOfDataParts++;
            }
            //Create the first part array
            if(nbrOfDataParts>0){
                if(nbrOfDataParts>1){
                    imgDataPart= new byte[breakpoint];
                } else{
                    imgDataPart= new byte[remainder];
                }
            }
            while(byteNbr+lapCount*breakpoint!=imgData.length){
                imgDataPart[byteNbr]=imgData[byteNbr+lapCount*breakpoint];
                byteNbr++;
                if(byteNbr==breakpoint){
                    lapCount++;
                    byteNbr=0;
                    out.write(imgDataPart, 0, imgDataPart.length);
                    out.flush();
                    if(lapCount+1==nbrOfDataParts && remainder!=0){
                        imgDataPart = new byte[remainder];
                    } else {
                        imgDataPart = new byte[breakpoint];
                    }
```

```
            }
        }
        out.write(imgDataPart, 0, imgDataPart.length);
        out.flush();
    } catch (IOException e) {
        parent.showAlert("Can't send file data: " + e.getMessage());
    }

    // close output stream anyway
    if (out != null) {
        try {
            out.close();
        } catch (IOException e) {} // ignore
    }
}

/** Reads image name from specified connection. */
private String readImageName(StreamConnection conn) {
    String imgName = null;
    DataInputStream in = null;
    System.out.println("readImageName");

    try {
        in = conn.openDataInputStream();
        int length = in.read(); // 'name' length is 1 byte

        if (length <= 0) {
            throw new IOException("Can't read name length");
        }
        byte[] nameData = new byte[length];
        length = 0;

        while (length != nameData.length) {
            int n = in.read(nameData, length, nameData.length - length);

            if (n == -1) {
                throw new IOException("Can't read name data");
            }
            length += n;
        }

        imgName = new String(nameData);

    } catch (IOException e) {
        System.err.println(e);
    } catch (NullPointerException ne) {
        System.err.println("Nullfel: "+ne);
    }

    // close input stream anyway
    if (in != null) {
        try {
            in.close();
        } catch (IOException e) {} // ignore
        catch (NullPointerException ne) {
            System.out.println("null!!!!");
        }
    }
    return imgName;
}

/** Reads images data from MIDlet archive to array. */
public byte[] getImageData(String imgName) {
    if (imgName == null) {
        return null;
    }

    imgName="file://localhost/"+imgName;

    InputStream in=null;
    try {
        in = Connector.openInputStream(imgName);
```

```
    } catch (IOException ioe) {
         parent.showAlert(ioe.getMessage());
    }

    // read image data and create a byte array
    byte[] buff = new byte[1024];
    ByteArrayOutputStream baos = new ByteArrayOutputStream(1024);

    try {
        while (true) {
            System.out.println("före in.read(buff)= " +buff.toString());
            int length = in.read(buff);

            if (length == -1) {
                break;
            }
            baos.write(buff, 0, length);
        }
    } catch (IOException e) {
        System.err.println("Can't get image data: imgName=" + imgName + " :"
                + e);
        return null;
    } catch (NullPointerException ne) {
        System.err.println("Nullpontfel!!! " +ne);
    }

    return baos.toByteArray();
}

/**
 * Orginizes the queue of clients to be processed,
 * processes the clients one by one until destroyed.
 */
private class ClientProcessor implements Runnable {
    private Thread processorThread;
    private Vector queue = new Vector();
    private boolean isOk = true;

    ClientProcessor() {
        processorThread = new Thread(this);
        processorThread.start();
    }

    public void run() {
        while (!isClosed) {

            // wait for new task to be processed
            synchronized (this) {
                if (queue.size() == 0) {
                    try {
                        wait();
                    } catch (InterruptedException e) {
                        System.err.println("Unexpected exception: " + e);
                        destroy(false);
                        return;
                    }
                }
            }

            // send the image to specified connection
            StreamConnection conn;

            synchronized (this) {

                // may be awaked by "destroy" method.
                if (isClosed) {
                    return;
                }
                conn = (StreamConnection) queue.firstElement();
                queue.removeElementAt(0);
                processConnection(conn);
            }
```

```
            }
        }

        /** Adds the connection to queue and notifys the thread. */
        void addConnection(StreamConnection conn) {
            synchronized (this) {
                queue.addElement(conn);
                notify();
            }
        }

        /** Closes the connections and . */
        void destroy(boolean needJoin) {
            StreamConnection conn;

            synchronized (this) {
                notify();

                while (queue.size() != 0) {
                    conn = (StreamConnection) queue.firstElement();
                    queue.removeElementAt(0);

                    try {
                        conn.close();
                    } catch (IOException e) {} // ignore
                }
            }

            // wait until dispatching thread is done
            try {
                processorThread.join();
            } catch (InterruptedException e) {} // ignore
        }
    }

    public void log(String l) {
        System.out.println(l);
    }
} // end of class 'BTImageServer' definition
```

10.2.4 saveImage.java

```
/*
 * saveImage.java
 */

package example.bluetooth.demo;

import javax.microedition.io.*;
import javax.microedition.io.file.*;
import java.io.*;
import java.lang.Thread;

/**
 *
 * @author Henrik     Andersson
 */


public class saveImage extends Thread {
    GUIImageClient parent;


    /** Creates a new instance of saveImage */
    public saveImage(GUIImageClient parent) {
        this.parent=parent;

    }

    public void run() {
        //Save the picture BTImageClient (via GUIImageClient)
        parent.saveImgPassage();
```

```
    }

}
```

10.2.5 GUIImageClient.java

```
/*
 * GUIImageClient.java
 */
package example.bluetooth.demo;

// midp GUI classes
import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.Gauge;
import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.ImageItem;
import javax.microedition.lcdui.List;
import javax.microedition.lcdui.StringItem;

import javax.microedition.lcdui.ChoiceGroup;
import javax.microedition.lcdui.Choice;
import java.util.Vector;

// midp/cldc classes
import java.io.IOException;
import java.util.Hashtable;
import java.util.Enumeration;
import javax.microedition.io.Connector;
/**
 * Provides a GUI to present the download options
 * to used, gives a chance to make a choice,
 * finally shows the downloaded image.
 */
final class GUIImageClient implements CommandListener {

    /** This command goes to demo main screen. */
    private final Command SCR_MAIN_BACK_CMD = new Command("Back", Command.BACK,
            2);

    /** Starts the proper services search. */
    private final Command SCR_MAIN_SEARCH_CMD = new Command("Find", Command.OK,
            1);

    /** Cancels the device/services discovering. */
    private final Command SCR_SEARCH_CANCEL_CMD = new Command("Cancel",
            Command.BACK, 2);

    /** This command goes to client main screen. */
    private final Command SCR_IMAGES_BACK_CMD = new Command("Back",
            Command.BACK, 2);

    private final Command updateCommand = new Command("Back", Command.BACK, 1);

    public final Command viewCommand = new Command("View Image", Command.OK, 1);

    private final Command downloadCommand = new Command("Download File", Command.OK, 1);

    /** Cancels the image download. */
    private final Command SCR_LOAD_CANCEL_CMD = new Command("Cancel",
            Command.BACK, 2);

    /** This command goes from image screen to images list one. */
    private final Command SCR_SHOW_BACK_CMD = new Command("Back", Command.BACK,
            2);

    /** Save the downloaded image. */
```

```
private final Command SAVE_CMD = new Command("Save File", Command.OK,
        1);

/** The main screen of the client part. */
private final Form mainScreen = new Form("Search for...");

/** The screen with found images names. */
private final List listScreen = new List("Found Files", List.IMPLICIT);

/** The screen with download image. */
private final Form imageScreen = new Form("Filename");

/** Keeps the parent MIDlet reference to process specific actions. */
public DemoMIDlet parent;

/** This object handles the real transmission. */
private BTImageClient bt_client;

private ChoiceGroup searchAlternatives;

public Command c_state;

public Hashtable fileList;

private boolean[] marked;

/** Constucts client GUI. */
GUIImageClient(DemoMIDlet parent) {
    this.parent = parent;
    bt_client = new BTImageClient(this);
    mainScreen.addCommand(SCR_MAIN_BACK_CMD);
    mainScreen.addCommand(SCR_MAIN_SEARCH_CMD);
    mainScreen.setCommandListener(this);

    //listScreen.addCommand(SCR_IMAGES_BACK_CMD);
    listScreen.addCommand(downloadCommand);
    listScreen.addCommand(updateCommand);
    listScreen.addCommand(viewCommand);
    listScreen.setCommandListener(this);

    imageScreen.addCommand(SCR_SHOW_BACK_CMD);
    imageScreen.addCommand(SAVE_CMD);
    imageScreen.setCommandListener(this);

    fileList=new Hashtable();
}

/**
 * Process the command events.
 */
public void commandAction(Command c, Displayable d) {

    // back to demo main screen
    if (c == SCR_MAIN_BACK_CMD) {
        destroy();
        parent.show();
        return;
    }

    // starts file (device/services) search
    if (c == SCR_MAIN_SEARCH_CMD || c==updateCommand) {
        c_state=c;
        System.out.println("c_state= "+c_state);
        Form f = new Form("");

        if(c==updateCommand) {
            log("destroy");
            bt_client.destroy();
            bt_client = new BTImageClient(this);
            f.deleteAll();
        }
```

```
    f.addCommand(SCR_SEARCH_CANCEL_CMD);
    f.setCommandListener(this);
    f.append(new Gauge("Searching files...", false, Gauge.INDEFINITE,
            Gauge.CONTINUOUS_RUNNING));

    Display.getDisplay(parent).setCurrent(f);
    bt_client.requestSearch();
    return;
}

// cancels device/services search
if (c == SCR_SEARCH_CANCEL_CMD) {
    bt_client.cancelSearch();
    Display.getDisplay(parent).setCurrent(mainScreen);
    return;
}

// back to client main screen
if (c == SCR_IMAGES_BACK_CMD) {
    bt_client.requestLoad(null);
    Display.getDisplay(parent).setCurrent(mainScreen);
    return;
}

// starts image download
if (c == viewCommand || c == downloadCommand) {
    c_state=c;
    List l = (List) d;
    String markedRow=l.getString(l.getSelectedIndex());
    log("markedRow== "+markedRow);

    if (c_state==viewCommand) {
        String ext=bt_client.getExtension(markedRow);
       if (ext.equals(".png") || ext.equals(".gif") || ext.equals(".jpg") ||
               ext.equals(".bmp")
        || ext.equals(".PNG") || ext.equals(".GIF") || ext.equals(".JPG") ||
               ext.equals(".BMP")) {
            log("picture=true "+ext);
        } else {
            log("picture=false "+ext);
            showAlert("The selected file is not a picture. To download the file
              choose \"Download File\"." );
            return;
        }

    }

    Form f = new Form("");
    f.addCommand(SCR_LOAD_CANCEL_CMD);
    f.setCommandListener(this);
    f.append(new Gauge("Loading file...", false, Gauge.INDEFINITE,
            Gauge.CONTINUOUS_RUNNING));

    Display.getDisplay(parent).setCurrent(f);

    //Get filename

    String loadFile=fileList.get(markedRow)+markedRow;
    bt_client.requestLoad(loadFile);
    return;
}

// cancel load
if (c == SCR_LOAD_CANCEL_CMD) {
    bt_client.cancelLoad();
    Display.getDisplay(parent).setCurrent(listScreen);
    return;
}

// back to client main screen
if (c == SCR_SHOW_BACK_CMD) {
    Display.getDisplay(parent).setCurrent(listScreen);
```

```
            return;
        }

        // Store file on the phone
        if (c == SAVE_CMD) {
            saveImage saveImg = new saveImage(this);
            saveImg.start();
            Display.getDisplay(parent).setCurrent(imageScreen);
            return;
        }
    }

    public Command getC_state() {
        return c_state;
    }
    public Command getViewCommand() {
        return viewCommand;
    }

    /**
     * We have to provide this method due to "do not do network
     * operation in command listener method" restriction, which
     * is caused by crooked midp design.
     *
     * This method is called by BTImageClient after it is done
     * with bluetooth initialization and next screen is ready
     * to appear.
     */
    void completeInitialization(boolean isBTReady) {

        // bluetooth was initialized successfully.
        if (isBTReady) {
            mainScreen.deleteAll();
            mainScreen.append(createSearchAlternatives());
            if(getC_state()==updateCommand){
                searchAlternatives.setSelectedFlags(marked);
            }

            Display.getDisplay(parent).setCurrent(mainScreen);
            return;
        }

        // something wrong
        Alert al = new Alert("Error", "Can't inititialize bluetooth", null,
                AlertType.ERROR);
        al.setTimeout(DemoMIDlet.ALERT_TIMEOUT);
        Display.getDisplay(parent).setCurrent(al, parent.getDisplayable());
    }

    /** Creates a menu with search criterions */
    private ChoiceGroup createSearchAlternatives() {
        searchAlternatives = new ChoiceGroup("", Choice.MULTIPLE);
        searchAlternatives.append("All", null);
        searchAlternatives.append("Music", null);
        searchAlternatives.append("Pictures", null);
        searchAlternatives.append("Programs/Games", null);
        searchAlternatives.append("Movies", null);
        searchAlternatives.append("Text files", null);
        return searchAlternatives;
    }

    public Vector getSearchExtensions() {
        Vector searchExtensions = new Vector();
        marked = new boolean[6];

        if (searchAlternatives.isSelected(0)) {
            searchExtensions.addElement("All");
            marked[0]=true;
        } else {
            if (searchAlternatives.isSelected(1)) { //Music
                searchExtensions.addElement(".mp3");
                searchExtensions.addElement(".wav");
```

```
            searchExtensions.addElement(".mid");
            searchExtensions.addElement(".MP3");
            searchExtensions.addElement(".WAV");
            searchExtensions.addElement(".MID");
            marked[1]=true;
        }
        if (searchAlternatives.isSelected(2)) { //Pictures
            searchExtensions.addElement(".jpg");
            searchExtensions.addElement(".gif");
            searchExtensions.addElement(".png");
            searchExtensions.addElement(".bmp");
             searchExtensions.addElement(".JPG");
            searchExtensions.addElement(".GIF");
            searchExtensions.addElement(".PNG");
            searchExtensions.addElement(".BMP");
            marked[2]=true;
        }
        if (searchAlternatives.isSelected(3)) { //Programs/Games
            searchExtensions.addElement(".jad");
            searchExtensions.addElement(".jar");
             searchExtensions.addElement(".JAD");
            searchExtensions.addElement(".JAR");
            marked[3]=true;
        }
        if (searchAlternatives.isSelected(4)) { //Movies
            searchExtensions.addElement(".3gp");
            searchExtensions.addElement(".rm");
            searchExtensions.addElement(".wm");
            searchExtensions.addElement(".avi");
            searchExtensions.addElement(".mpg");
            searchExtensions.addElement(".3GP");
            searchExtensions.addElement(".RM");
            searchExtensions.addElement(".WM");
            searchExtensions.addElement(".AVI");
            searchExtensions.addElement(".MPG");
            marked[4]=true;
        }
        if (searchAlternatives.isSelected(5)) {
            searchExtensions.addElement(".txt");
            searchExtensions.addElement(".TXT");
            marked[5]=true;
        }

    }
    return searchExtensions;
}


/** Destroys this component. */
void destroy() {

    // finilize the image client work
    bt_client.destroy();
}

/**
 * Informs the error during the images search.
 */
void informSearchError(String resMsg) {
    Alert al = new Alert("Error", resMsg, null, AlertType.ERROR);
    al.setTimeout(DemoMIDlet.ALERT_TIMEOUT);
    Display.getDisplay(parent).setCurrent(al, mainScreen);
}

/**
 * Informs the error during the selected image load.
 */
void informLoadError(String resMsg) {
    Alert al = new Alert("Error", resMsg, null, AlertType.ERROR);
    al.setTimeout(DemoMIDlet.ALERT_TIMEOUT);
    Display.getDisplay(parent).setCurrent(al, listScreen);
}
```

```
/**
 * Shows the downloaded image.
 */
void showImage(Image img, String imgName) {
    imageScreen.deleteAll();

    imageScreen.append(new ImageItem("", img,
            ImageItem.LAYOUT_CENTER | ImageItem.LAYOUT_VCENTER,
            "Downloaded file: " + imgName));
    imageScreen.setTitle(bt_client.getFileName(imgName));
    Display.getDisplay(parent).setCurrent(imageScreen);
}

/**
 * Shows the available images names.
 *
 * @returns false if no images names were found actually
 */
boolean showImagesNames(Hashtable base) {
    Enumeration keys = base.keys();


    // no images actually
    if (!keys.hasMoreElements()) {

        informSearchError("No files with the selected extensions were found. \nPlease
           try again");
        return false;
    }

    // prepare the list to be shown
    while (listScreen.size() != 0) {
        listScreen.delete(0);
    }

    while (keys.hasMoreElements()) {
        String key=(String) keys.nextElement();
        String filePath=bt_client.getFilePath(key);
        String fileName=bt_client.getFileName(key);

        fileList.put(fileName, filePath);
        log("filePath= "+filePath);
        log("fileName= "+fileName);
        log("fileList="+fileList.get(fileName));

        listScreen.append(fileName, null);
    }
    Display.getDisplay(parent).setCurrent(listScreen);
    return true;
}

public void saveImgPassage() {
    bt_client.saveImg();
}

public void showAlert(String alert){
    Alert al = new Alert("System message", alert, null,
            AlertType.ERROR);
    al.setTimeout(6000);
    Display.getDisplay(parent).setCurrent(al, listScreen);
}

public void showAlertStartMenu(String alert){
    Alert al = new Alert("System message", alert, null,
            AlertType.ERROR);
    al.setTimeout(6000);
    Display.getDisplay(parent).setCurrent(al, parent.getDisplayable());
}

public void log(String l) {
    System.out.println(l);
```

```
    }
} // end of class 'GUIImageClient' definition
```

10.2.6 BTImageClient .java

```
*
 * BTImageClient.java
 */
package example.bluetooth.demo;

// jsr082 API
import javax.bluetooth.BluetoothStateException;
import javax.bluetooth.DataElement;
import javax.bluetooth.DeviceClass;
import javax.bluetooth.DiscoveryAgent;
import javax.bluetooth.DiscoveryListener;
import javax.bluetooth.LocalDevice;
import javax.bluetooth.RemoteDevice;
import javax.bluetooth.ServiceRecord;
import javax.bluetooth.UUID;

// midp/cldc API
import javax.microedition.io.Connector;
import javax.microedition.io.StreamConnection;
import javax.microedition.lcdui.Image;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.Enumeration;
import java.util.Hashtable;
import java.util.Vector;

import javax.microedition.io.*;
import javax.microedition.io.file.*;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.*;

/**
 * Initialize BT device, search for BT services,
 * presents them to user and picks his/her choice,
 * finally download the choosen image and present
 * it to user.
 */
final class BTImageClient implements Runnable, DiscoveryListener {

    /** Describes this server */
    private static final UUID PICTURES_SERVER_UUID =
            new UUID("F0E0D0C0B0A000908070605040302010", false);

    /** The attribute id of the record item with images names. */
    private static final int IMAGES_NAMES_ATTRIBUTE_ID = 0x4321;

    /** Shows the engine is ready to work. */
    private static final int READY = 0;

    /** Shows the engine is searching bluetooth devices. */
    private static final int DEVICE_SEARCH = 1;

    /** Shows the engine is searching bluetooth services. */
    private static final int SERVICE_SEARCH = 2;

    /** Keeps the current state of engine. */
    private int state = READY;

    /** Keeps the discovery agent reference. */
    private DiscoveryAgent discoveryAgent;

    /** Keeps the parent reference to process specific actions. */
    private GUIImageClient parent;

    /** Becomes 'true' when this component is finilized. */
    private boolean isClosed;
```

```
/** Proccess the search/download requests. */
private Thread processorThread;

/** Collects the remote devices found during a search. */
private Vector /* RemoteDevice */ devices = new Vector();

/** Collects the services found during a search. */
private Vector /* ServiceRecord */ records = new Vector();

/** Keeps the device discovery return code. */
private int discType;

/** Keeps the services search IDs (just to be able to cancel them). */
private int[] searchIDs;

/** Keeps the image name to be load. */
private String imageNameToLoad;

/** Keeps the table of {name, Service} to process the user choice. */
private Hashtable base = new Hashtable();

/** Informs the thread the download should be canceled. */
private boolean isDownloadCanceled;

/** Optimization: keeps service search patern. */
private UUID[] uuidSet;

/** Optimization: keeps attributes list to be retrieved. */
private int[] attrSet;

byte[] imgData;

FileConnection fconn;

/** Contains the selected extensions */
Vector extensions;

/** Is it a picture or not? */
boolean picture;

boolean ok;

private int serviceCount;

private boolean isSearchStarted;

private static final String downloadPath = "e:/MSSEMC/Media files/other/Download/";

/**
 * Constructs the bluetooth server, but it is initialized
 * in the different thread to "avoid dead lock".
 */
BTImageClient(GUIImageClient parent) {
    ok=true;
    this.parent = parent;
    imgData = null;

    // we have to initialize a system in different thread...
    processorThread = new Thread(this);
    processorThread.start();
}

/**
 * Process the search/download requests.
 */
public void run() {

    // initialize bluetooth first
    boolean isBTReady = false;

    try {
```

```
            // create/get a local device and discovery agent
            LocalDevice localDevice = LocalDevice.getLocalDevice();
            discoveryAgent = localDevice.getDiscoveryAgent();

            // remember we've reached this point.
            isBTReady = true;
        } catch (Exception e) {
            System.err.println("Can't initialize bluetooth: " + e);
        }
        parent.completeInitialization(isBTReady);

        // nothing to do if no bluetooth available
        if (!isBTReady) {
            return;
        }

        // initialize some optimization variables
        uuidSet = new UUID[2];

        // ok, we are interesting in btspp services only
        uuidSet[0] = new UUID(0x1101);

        // and only known ones, that allows pictures
        uuidSet[1] = PICTURES_SERVER_UUID;

        // we need an only service attribute actually
        attrSet = new int[1];

        // it's "images names" one
        attrSet[0] = IMAGES_NAMES_ATTRIBUTE_ID;

        // start processing the images search/download
        processImagesSearchDownload();
    }

    /**
     * Invoked by system when a new remote device is found -
     * remember the found device.
     */
    public void deviceDiscovered(RemoteDevice btDevice, DeviceClass cod) {

        // same device may found several times during single search
        if (devices.indexOf(btDevice) == -1) {
            devices.addElement(btDevice);
        }
    }

    /**
     * Invoked by system when device discovery is done.
     * <p>
     * Use a trick here - just remember the discType
     * and process its evaluation in another thread.
     */
    public void inquiryCompleted(int discType) {
        this.discType = discType;

        synchronized (this) {
            notify();
        }
    }

    public void servicesDiscovered(int transID, ServiceRecord[] servRecord) {
        for (int i = 0; i < servRecord.length; i++) {
            records.addElement(servRecord[i]);
        }
    }

    public void serviceSearchCompleted(int transID, int respCode) {
        // first, find the service search transaction index
        int index = -1;
        for (int i = 0; i < searchIDs.length; i++) {
```

```
            if (searchIDs[i] == transID) {
                index = i;
                break;
            }
        }

        // error - unexpected transaction index
        if (index == -1) {
            System.err.println("Unexpected transaction index: " + transID);
            // FIXME: process the error case
        } else {
            searchIDs[index] = -1;
        }

        synchronized (this) {
            System.out.println("serviceSearchComleted sync o notify");
            notify();
        }
    }

    /** Sets the request to search the devices/services. */
    void requestSearch() {
        synchronized (this) {
            notify();
        }
    }

    /** Cancel's the devices/services search. */
    void cancelSearch() {
        synchronized (this) {
            if (state == DEVICE_SEARCH) {
                discoveryAgent.cancelInquiry(this);
            } else if (state == SERVICE_SEARCH) {
                for (int i = 0; i < searchIDs.length; i++) {
                    discoveryAgent.cancelServiceSearch(searchIDs[i]);
                }
            }
        }
    }

    /** Sets the request to load the specified image. */
    void requestLoad(String name) {
        synchronized (this) {
            imageNameToLoad = name;
            notify();
        }
    }

    /** Cancel's the image download. */
    void cancelLoad() {
        /*
         * The image download process is done by
         * this class's thread (not by a system one),
         * so no need to wake up the current thread -
         * it's running already.
         */
        isDownloadCanceled = true;
    }

    /**
     * Destroy a work with bluetooth - exits the accepting
     * thread and close notifier.
     */
    void destroy() {
        synchronized (this) {
            isClosed = true;
            isDownloadCanceled = true;
            notify();
        }

        // wait for acceptor thread is done
        try {
```

```
            processorThread.join();
        } catch (InterruptedException e) {} // ignore
    }

    /**
     * Processes images search/download until component is closed
     * or system error has happen.
     */
    private synchronized void processImagesSearchDownload() {
        while (!isClosed) {
            // wait for new search request from user
            state = READY;

            try {
                wait();
            } catch (InterruptedException e) {
                System.err.println("Unexpected interuption: " + e);
                return;
            }

            // check the component is destroyed
            if (isClosed) {
                return;
            }

            // search for devices
            if (!searchDevices()) {
                return;
            } else if (devices.size() == 0) {
                continue;
            }

            // search for services now
            state = SERVICE_SEARCH;
            records.removeAllElements();
            searchIDs = new int[devices.size()];
            isSearchStarted = false;
            serviceCount = 0;
            while(serviceCount < devices.size()){
                if (!searchServices()) {
                    return;
                }
                serviceCount++;
            }

            // ok, something was found - present the result to user now
            if (!presentUserSearchResults()) {
                // services are found, but no names there
                continue;
            }

            // the several download requests may be processed
            while (true) {
                System.out.println("while-satsen");
                // this download is not canceled, right?
                isDownloadCanceled = false;

                // ok, wait for download or need to wait for next search
                try {
                    wait();
                } catch (InterruptedException e) {
                    System.err.println("Unexpected interuption: " + e);
                    return;
                }

                // check the component is destroyed
                if (isClosed) {
                    System.out.println("isClosed");
                    return;
                }

                // this means "go to begining"
```

```
            if (imageNameToLoad == null) {
                System.out.println("imageNameToLoad==null");
                break;
            }

            log("imageNameToLoad= "+imageNameToLoad);
            String ext=getExtension(imageNameToLoad);
            if (ext.equals(".png") || ext.equals(".gif") || ext.equals(".jpg") ||
                  ext.equals(".bmp")
            || ext.equals(".PNG") || ext.equals(".GIF") || ext.equals(".JPG") ||
                  ext.equals(".BMP")) {
                log("picture=true");
                picture=true;
            } else {
                log("picture=false "+ext);
                picture=false;
            }

            Image img=null;
            // load selected image data
            boolean imageDataisCorrect = loadImage();

            Command cs=parent.getC_state();
            Command csv=parent.getViewCommand();
            log("cs= " +cs+", csv= "+csv);
            //It is a picture and viewCommand was selected
            if (picture && cs==csv && imageDataisCorrect) {

                if (ok) {
                    try {

                        log("imgData is a picture");
                        img = Image.createImage(imgData, 0, imgData.length);

                    } catch (Exception e) {
                        parent.showAlert("Wrong data received, please try again");
                        continue;
                    }

                } else {
                    log("not ok");
                    break;
                }

            } else if(imageDataisCorrect) {
                log("Save file");
                saveImg();
                return;
            }

            // FIXME: this never happen
            if (isClosed) {
                return;
            }

            if (isDownloadCanceled) {
                continue; // may be next image to be download
            }

            if (img == null) {
                parent.informLoadError("Can't load file: "
                        + imageNameToLoad);
                continue; // may be next image to be download
            }

            // ok, show image to user
            parent.showImage(img, imageNameToLoad);

            // may be next image to be download
            continue;
        }
    }
```

```
}

/**
 * Search for bluetooth devices.
 */
private boolean searchDevices() {

    // ok, start a new search then
    state = DEVICE_SEARCH;
    devices.removeAllElements();

    try {
        discoveryAgent.startInquiry(DiscoveryAgent.GIAC, this);
    } catch (BluetoothStateException e) {
        System.err.println("Can't start inquiry now: " + e);
        parent.informSearchError("Can't start device search");
        return true;
    }

    try {
        wait(); // until devices are found
    } catch (InterruptedException e) {
        System.err.println("Unexpected interuption: " + e);
        return false;
    }

    // this "wake up" may be caused by 'destroy' call
    if (isClosed) {
        return false;
    }

    // no?, ok, let's check the return code then
    switch (discType) {
        case INQUIRY_ERROR:
            parent.informSearchError("Device discovering error...");

            // fall through

        case INQUIRY_TERMINATED:

            // make sure no garbage in found devices list
            devices.removeAllElements();

            // nothing to report - go to next request
            break;

        case INQUIRY_COMPLETED:
            if (devices.size() == 0) {
                parent.informSearchError("No devices in range");
            }

            // go to service search now
            break;
        default:

            System.err.println("system error:"
                    + " unexpected device discovery code: " + discType);
            destroy();
            return false;
    }
    return true;
}

/**
 * Search for proper service.
 */
private boolean searchServices() {
    RemoteDevice rd = (RemoteDevice) devices.elementAt(serviceCount);
    try {
        searchIDs[serviceCount] = discoveryAgent.searchServices(attrSet, uuidSet,
                rd, this);
    } catch (BluetoothStateException e) {
```

```
            System.err.println("Can't search services for: "
                    + rd.getBluetoothAddress() + " due to " + e);
            searchIDs[serviceCount] = -1;
            //continue;
        }
        isSearchStarted = true;
        // at least one of the services search should be found
        if (!isSearchStarted) {
            parent.informSearchError("Can't search services.");
            return true;
        }

        try {
            wait(); // until services are found
        } catch (InterruptedException e) {
            System.err.println("Unexpected interuption: " + e);
            return false;
        }

        // this "wake up" may be caused by 'destroy' call
        if (isClosed) {
            return false;
        }

        // actually, no services were found
        if (records.size() == 0) {
            parent.informSearchError("No proper services were found");
        }
        return true;
    }

    /**
     * Gets the collection of the images titles (names)
     * from the services, prepares a hashtable to match
     * the image name to a services list, presents the images names
     * to user finally.
     */
    private boolean presentUserSearchResults() {
        base.clear();

        //Kollar vilka filändelser som är markerade
        extensions = new Vector();
        extensions=parent.getSearchExtensions();

        for (int i = 0; i < records.size(); i++) {
            ServiceRecord sr = (ServiceRecord) records.elementAt(i);

            // get the attribute with images names
            DataElement de = sr.getAttributeValue(IMAGES_NAMES_ATTRIBUTE_ID);

            if (de == null) {
                System.err.println("Unexpected service - missed attribute");
                continue;
            }

            // get the images names from this attribute
            Enumeration enume = (Enumeration) de.getValue();

            while (enume.hasMoreElements()) {
                de = (DataElement) enume.nextElement();
                String name = (String) de.getValue();

                //If the extension is chosen, show the file
                if (extensions.contains(getExtension(name))
                       || extensions.contains("All") ) {

                    // name may be stored already
                    Object obj = base.get(name);

                    // that's either the ServiceRecord or Vector
                    if (obj != null) {
                        Vector v;
```

```
                    if (obj instanceof ServiceRecord) {
                        v = new Vector();
                        v.addElement(obj);
                    } else {
                        v = (Vector) obj;
                    }
                    v.addElement(sr);
                    obj = v;
                } else {
                    obj = sr;
                }
                base.put(name, obj);
            }

        }
    }
    return parent.showImagesNames(base);
}


/**
 * Loads selected image data.
 */
private boolean loadImage() {
    if (imageNameToLoad == null) {
        System.err.println("Error: imageNameToLoad=null");
        return false;
    }
    // ok, get the list of service records
    ServiceRecord[] sr = null;
    Object obj = base.get(imageNameToLoad);

    if (obj == null) {
        System.err.println("Error: no record for: " + imageNameToLoad);
        return false;
    } else if (obj instanceof ServiceRecord) {
        sr = new ServiceRecord[] { (ServiceRecord) obj };
    } else {
        Vector v = (Vector) obj;
        sr = new ServiceRecord[v.size()];

        for (int i = 0; i < v.size(); i++) {
            sr[i] = (ServiceRecord) v.elementAt(i);
        }
    }

    // now try to load the image from each services one by one
    for (int i = 0; i < sr.length; i++) {
        StreamConnection conn = null;
        String url = null;

        // the process may be canceled
        if (isDownloadCanceled) {
            return false;
        }
        // first - connect
        try {
            url = sr[i].getConnectionURL(
                    ServiceRecord.NOAUTHENTICATE_NOENCRYPT, false);
            conn = (StreamConnection) Connector.open(url);
        } catch (IOException e) {
            System.err.println("Note: can't connect to: " + url);
            // ignore
            continue;
        } catch (NullPointerException ne) {
            System.err.println("Nullpointer!!!!!!");
        }

        // then open a stream and write a name
        try {
            OutputStream out = conn.openOutputStream();
```

```
    out.write(imageNameToLoad.length()); // length is 1 byte
    out.write(imageNameToLoad.getBytes());
    out.flush();
    out.close();
} catch (IOException e) {
    System.err.println("Can't write to server for: " + url);

    // close stream connection
    try {
        System.out.println("try close connection");
        conn.close();
    } catch (IOException ee) {} // ignore
    continue;
} catch (NullPointerException ne){
    System.out.println("NUUUUUL!!!!!!!!");
}

// then open a stream and read an image
InputStream in=null;

try {
    in = conn.openInputStream();

    // read a length first
    System.out.println("try in.read()= " +in);
    int length = in.read() << 24;
    length |= in.read() << 16;
    length |= in.read() << 8;
    length |= in.read();

    if (length <= 0) {
        throw new IOException("Can't read a length");
    }

    //If the chosen file does not exists
    if (length==1) {
        parent.showAlert("The file is not shared anymore. \nPlease try
           again");
        //requestSearch();
        ok=false;

    } else {

        // read the image now
        imgData = new byte[length];
        length = 0;

        while (length != imgData.length && !isDownloadCanceled) {
            int n = in.read(imgData, length, imgData.length - length);

            if (n == -1) {
                throw new IOException("Can't read a file data");
            }
            length += n;
        }
        if(isDownloadCanceled){
            System.out.println("DOWNLOADCANCELED!!!!!!!!!!!!"+length);
            return false;
        }

    }
    in.close();
} catch (IOException e) {
    parent.showAlert("Cant read from server for: "+url);

    continue;
} catch (NullPointerException ne){
    System.out.println("Nullfel: "+ne);
} finally {

    // close stream connection anyway
    try {
```

```
                    conn.close();
                } catch (Exception e) {
                    parent.showAlert(e.getMessage());
                } // ignore
            }

        }
        return true;
    }

    public void saveImg() {
        try {
            System.out.println("I saveImg");

            FileConnection fconn2 = (FileConnection)
               Connector.open("file://localhost/"+downloadPath);

            // If no exception is thrown, then the URI is valid, but the file may or may
               not exist.
            if (!fconn2.exists()) {
                fconn2.mkdir();  // create the directory if it doesn't exist
            }
            fconn2.setReadable(true);
            fconn2.setWritable(true);
            fconn2.close();

            String fileName= getFileName(imageNameToLoad);
            fconn2 = (FileConnection)
               Connector.open("file://localhost/"+downloadPath+fileName);

            if (!fconn2.exists()) {
                fconn2.create();
                fconn2.setReadable(true);
                fconn2.setWritable(true);

                OutputStream out = fconn2.openOutputStream();
                out.write(imgData);
                out.flush();
                parent.showAlertStartMenu(fileName+" has been successfully saved at; "
                      +downloadPath+fileName);
            } else {
                parent.showAlertStartMenu(fileName+" does already exist at; "
                      +downloadPath+fileName);
            }
            fconn2.close();
        } catch (Exception e) {
            parent.showAlertStartMenu("The file could not be saved. " +e.getMessage());
        }
    }

    String getFilePath(String name) {
        String filePath = name.substring(0, name.lastIndexOf('/')+1);
        return filePath;
    }
    String getFileName(String name) {
        String fileName = name.substring(name.lastIndexOf('/')+1, name.length());
        return fileName;
    }
        public String getExtension(String s) {
        String name;
        name=s.substring(s.length()-4, s.length()); //Get extension
        return name;
    }

    public void log(String l) {
        System.out.println(l);
    }

} // end of class 'BTImageClient' definition
```