Appendix A: Application Source Code

BluetoothDiscovererMidlet.java

This Java file is dedicated to the overall functional flow of the application.

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
import javax.bluetooth.*;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import javax.microedition.rms.*;

import java.util.*;

public class BluetoothDiscovererMidlet extends MIDlet implements CommandListener {

Vector mDevices;
int mTotalDevice;

Vector vContextSave = new Vector();
String []contextInitial = new String[]{ "Computing Studio",
                                        "Lecture Theatre",
                                        "Office",
                                        "Meeting Room",
                                        "Library",
                                        "Home",
                                        "Other"};

Form deleteConfirmForm = new Form("Delete");
Command okCommandDelete = new Command("Ok", Command.OK, 0);
Command cancelCommandDelete = new Command("Cancel", Command.CANCEL, 0);

Form mainForm = new Form("Bluetooth Device Discoverer");;
Form choiceForm = new Form("Delete Database");
Form detailsForm = new Form("Details");
List saveList = new List("Save", List.IMPLICIT);

Form searchForm = new Form("Search");
TextField deviceSearch = new TextField("Device", "", 50, TextField.ANY);

ChoiceGroup searchChoiceGroup = new ChoiceGroup("Context", ChoiceGroup.POPUP);
List deviceList = new List("Devices", List.IMPLICIT);
List browseList = new List("Devices", List.IMPLICIT);
List mainList = new List("BT Main", List.IMPLICIT);

DeviceDiscoverer deviceDiscoverer;
ServiceDiscoverer serviceDiscoverer;

Command exitCommand = new Command("Exit", Command.EXIT, 0);;
Command okCommand = new Command("Ok", Command.OK, 0);
Command cancelCommand = new Command("Cancel", Command.CANCEL, 0);
Command serviceSearchCommand = new Command("Search", Command.OK, 0);;
Command saveCommnd = new Command("Save", Command.ITEM, 1);;
Command searchCommnd = new Command("Search", Command.ITEM, 1);
Command backFromDeviceList = new Command("Back", Command.BACK, 1);;
```

```
Command backFromSave = new Command("Back", Command.BACK, 1);;
Command backFromBrowse = new Command("Back", Command.BACK, 1);
Command backFromDetails = new Command("Back", Command.BACK, 1);
Command backFromSearch = new Command("Back", Command.BACK, 1);
Command deleteFrmDetail = new Command("Delete", Command.OK, 1);;

Form addContext = new Form("New Context");
TextField newContext = new TextField("Context", "", 50, TextField.ANY);
Command backFromAddContext = new Command("Back", Command.BACK, 1);
Command okAddContext = new Command("Ok", Command.OK, 0);

// a simple constructor
public BluetoothDiscovererMidlet() {
String contexts = geString("Context");

if(contexts == null)
{
contexts = contextInitial[0];
for(int i=1;i<contextInitial.length;i++)
{
contexts += "|" + contextInitial[i];
}
saveString(contexts, "Context");
}

String []sArray = split(contexts, "|");

searchChoiceGroup.append("", null);

for(int i=0;i<sArray.length;i++)
{
vContextSave.addElement(sArray[i]);
searchChoiceGroup.append(sArray[i], null);
saveList.append(sArray[i], null);
}

deleteConfirmForm.append("Do you really want to delete?");
deleteConfirmForm.addCommand(okCommandDelete);
deleteConfirmForm.addCommand(cancelCommandDelete);
deleteConfirmForm.setCommandListener(this);

addContext.append(newContext);
addContext.addCommand(okAddContext);
addContext.addCommand(backFromAddContext);
addContext.setCommandListener(this);

choiceForm.append("To remove all records, choose \"OK\"");
choiceForm.addCommand(okCommand);
choiceForm.addCommand(cancelCommand);
choiceForm.setCommandListener(this);

mainForm.addCommand(exitCommand);
mainForm.setCommandListener(this);

deviceList.setSelectCommand(null);
deviceList.addCommand(saveCommnd);
deviceList.addCommand(backFromDeviceList);
deviceList.setCommandListener(this);

saveList.addCommand(backFromSave);
saveList.setCommandListener(this);

browseList.addCommand(backFromBrowse);
```

```
browseList.setCommandListener(this);

detailsForm.addCommand(backFromDetails);
detailsForm.addCommand(deleteFrmDetail);
detailsForm.setCommandListener(this);

mainList.append("DISCOVER BT DEVICES", null);
mainList.append("BROWSE BT DATABASE", null);
mainList.append("SEARCH BT DATABASE", null);
mainList.append("RESET BT DATABASE", null);
mainList.append("ADD CONTEXT", null);

mainList.addCommand(exitCommand);
mainList.setCommandListener(this);

searchForm.append(deviceSearch);
searchForm.append(searchChoiceGroup);
searchForm.addCommand(searchCommnd);
searchForm.addCommand(backFromSearch);
searchForm.setCommandListener(this);
}

public void startApp()
{
Display.getDisplay(this).setCurrent(mainList);
}

public Display getDisplay()
{
return Display.getDisplay(this);
}

public void pauseApp() {}

public void destroyApp(boolean unconditional) {}

/**
  * This method takes the input from the phone, and exits the app,
  * finds new Bluetooth mDevices, or searches for services on
  * mDevices that have been already found.
  */

// this function shows an alert. After user presses OK, disp parameter is shown
public void showAlert(String sHeader, String sMessage, final Displayable disp,
       AlertType alertType)
{
Alert alert = new Alert(sHeader, sMessage, null, alertType);
alert.setTimeout(Alert.FOREVER);

alert.setCommandListener(new CommandListener()
{
public void commandAction(Command command, Displayable displayable)
{
if (command == Alert.DISMISS_COMMAND) {
// exit the app
if (disp == null)
{
// stop the app when alert is dismissed
destroyApp(false);
notifyDestroyed();
}
else
{
```

```
getDisplay().setCurrent(disp);
}
}
}
});

getDisplay().setCurrent(alert);

}
// reads database and creates device vector for browsing
void readDatabase()
{
// reads number of record
int count = getInt("RecordCount");
mDevices = new Vector();

for(int i=0;i<count;i++)
{
// reads a record
String []sArray = split(geString("" + i), "|");
int deviceCount = Integer.parseInt(sArray[1]);
int index = 2;

for(; deviceCount != 0; deviceCount--)
{
BlueToothDevice currrentDevice = new BlueToothDevice(sArray[index++]);
currrentDevice.context = sArray[0];

int serviceCount = Integer.parseInt(sArray[index++]);
for(; serviceCount != 0; serviceCount--)
{
currrentDevice.serviceNames.addElement(sArray[index++]);
}

// stores the device
currrentDevice.recordId = i;
mDevices.addElement(currrentDevice);
}
}
}

boolean matchSearch(Vector tempDevices, String []deviceName)
{
String context =
searchChoiceGroup.getString(searchChoiceGroup.getSelectedIndex());
BlueToothDevice currentDevice = (BlueToothDevice)tempDevices.elementAt(0);

// matches against context
if(currentDevice.context.indexOf(context) > -1)
{
for(int i=0; i< tempDevices.size(); i++)
{
currentDevice = (BlueToothDevice)tempDevices.elementAt(i);

// device name contains any word of the search device.
String name = currentDevice.deviceName.toLowerCase();
for(int j=0;j<deviceName.length; j++)
{
if(name.indexOf(deviceName[j]) > -1)
{
return true;
}
}
```

```
}
}

return false;
}

void readSearchDatabase()
{
String []deviceName = split(deviceSearch.getString(), " ");
for(int i=0;i<deviceName.length; i++)
{
deviceName[i] = deviceName [i].toLowerCase();
}

// reads number of record
int count = getInt("RecordCount");
mDevices = new Vector();

for(int i=0;i<count;i++)
{
//reads a record
String []sArray = split(geString("" + i), "|");
Vector tempDevices = new Vector();

int deviceCount = Integer.parseInt(sArray[1]);
int index = 2;

for(; deviceCount != 0; deviceCount--)
{
BlueToothDevice currrentDevice = new BlueToothDevice(sArray[index++]);
currrentDevice.context = sArray[0];

int serviceCount = Integer.parseInt(sArray[index++]);
for(; serviceCount != 0; serviceCount--)
{
currrentDevice.serviceNames.addElement(sArray[index++]);
}

// stores the device
currrentDevice.recordId = i;
tempDevices.addElement(currrentDevice);
}

if(matchSearch(tempDevices, deviceName))
{
for(int j=0; j<tempDevices.size() ;j++)
{
mDevices.addElement(tempDevices.elementAt(j));
}
}
}
}
// it is called from ServiceDiscoverer on every serviceSearchCompleted()
public void serviceSerchComplete()
{
mTotalDevice--;

try
{
Thread.sleep(3000);
}
catch(Exception e)
{
```

```
}
// service search is finished
if(mTotalDevice <= 0)
{
try
{
Thread.sleep(3000);
}
catch(Exception e)
{
}

try
{
deviceList.deleteAll();
}
catch(Exception e)
{}
for(int i=0;i<mDevices.size();i++)
{
deviceList.append(((BlueToothDevice)mDevices.elementAt(i)).deviceName, null);
}

if(mDevices.size() > 0)
{
showAlert("DEVICES FOUND", ""+ deviceList.size() + " Devices found", deviceList,
      AlertType.INFO);
}
else
{
showAlert("", "No device found", mainList, AlertType.INFO);
}
}
else
// devices left for service search
{
new
ServiceDiscoverer(this,(RemoteDevice)deviceDiscoverer.remoteDevices.elementAt
      (mTotalDevice-1)).start();
}
}

public List getSaveList()
{
// holds the maximum match for each context
int []matchCount = new int[vContextSave.size()];
// this to check against multiple device with same friendly name
boolean []flag = new boolean [mDevices.size()];

for(int i=0;i<vContextSave.size();i++)
{
// initialised to zero
matchCount[i] = 0;
}
// reads number of records
int count = getInt("RecordCount");

for(int i=0;i<count;i++)
{
// reads a record
String []sArray = split(geString("" + i), "|");
int deviceCount = Integer.parseInt(sArray[1]);
int index = 2;
```

```
int currentMatch = 0;
int divider = getMax(deviceCount, mDevices.size());

for(int j=0;j<mDevices.size();j++)
{
flag[j] = false;
}

// transverses through a record
for(; deviceCount != 0; deviceCount--)
{
String deviceName = sArray[index++];

// skips the services
int serviceCount = Integer.parseInt(sArray[index++]);
index += serviceCount;

for(int j=0;j<mDevices.size();j++)
{
BlueToothDevice currentDevice = (BlueToothDevice)mDevices.elementAt(j);

if(currentDevice.deviceName.equalsIgnoreCase(deviceName) && flag[j] == false)
{
// this will force the device with exact same friendly name to be matched twice
currentMatch++;
flag[j] = true;
break;
}
}
}
// converting to a percentage
currentMatch = (currentMatch*100)/divider;

for(int j=0;j<vContextSave.size();j++)
{
String context = (String)vContextSave.elementAt(j);
// matching the saved context
if(context.equalsIgnoreCase(sArray[0]))
{
// storing the maximum percentage match
if(matchCount[j] < currentMatch)
{
matchCount[j] = currentMatch;
}
break;
}
}
}

// building the saved list
for(int i=0;i<vContextSave.size();i++)
{
String context = (String)vContextSave.elementAt(i);
saveList.set(i, context + " ("+ matchCount[i] +"%)", null);
}
return saveList;
}

int getMax(int x, int y)
{
if(x>y)
return x;
return y;
```

```
}

public void commandAction(Command c, Displayable disp) {

if(c == exitCommand)
{
notifyDestroyed();
}
else if(disp == mainForm)
{
// starts the service search of 1st device
if(c == serviceSearchCommand)
{
try
{
mainForm.deleteAll();
}
catch(Exception e)
{}
mainForm.removeCommand(serviceSearchCommand);
mDevices = new Vector();
updateStatus("Searching for Bluetooth services in the area...");
mTotalDevice = deviceDiscoverer.remoteDevices.size();
serviceDiscoverer = new ServiceDiscoverer(this,
       (RemoteDevice)deviceDiscoverer.remoteDevices.elementAt(mTotalDevice-1));
serviceDiscoverer.start();
}
}
else if(disp == deleteConfirmForm)
{
if(c == cancelCommandDelete)
{
getDisplay().setCurrent(detailsForm);
}
else
{
// get the Id of the device to be deleted
int deviceId = browseList.getSelectedIndex();
// get the device from vector
BlueToothDevice delDevice = (BlueToothDevice) mDevices.elementAt(deviceId);


// delete from vector
mDevices.removeElementAt(deviceId);
// delete from display list
browseList.delete(deviceId);

//reads a record
String []sArray = split(geString("" + delDevice.recordId), "|");
Vector tempDevices = new Vector();

int deviceCount = Integer.parseInt(sArray[1]);
int index = 2;

for(; deviceCount != 0; deviceCount--)
{
BlueToothDevice currrentDevice = new BlueToothDevice(sArray[index++]);
currrentDevice.context = sArray[0];

int serviceCount = Integer.parseInt(sArray[index++]);
for(; serviceCount != 0; serviceCount--)
{
currrentDevice.serviceNames.addElement(sArray[index++]);
```

```
}

// this device is not to be deleted
if(!currrentDevice.deviceName.equalsIgnoreCase(delDevice.deviceName))

// stores the device
tempDevices.addElement(currrentDevice);
}

// save the record except the deleted device
String saveString = sArray[0]+ "|" + tempDevices.size();

for(int j=0;j<tempDevices.size();j++)
{
BlueToothDevice currentDevice = (BlueToothDevice)tempDevices.elementAt(j);
saveString += "|" + currentDevice.deviceName + "|" +
       currentDevice.serviceNames.size();

for(int i=0; i<currentDevice.serviceNames.size() ;i++)
{
saveString += "|" + (String)currentDevice.serviceNames.elementAt(i);
}
}

saveString(saveString, ""+ delDevice.recordId);

showAlert("INFO", "Device is deleted", browseList, AlertType.INFO);
}

}
else if(disp == mainList)
{
if(c == List.SELECT_COMMAND)
{
switch(mainList.getSelectedIndex())
{
// user chooses "DISCOVER BT DEVICES"
case 0:
try
{
mainForm.deleteAll();
}
catch(Exception e)
{
}
deviceDiscoverer = new DeviceDiscoverer(this);
getDisplay().setCurrent(mainForm);
break;

// user chooses "BROWSE BT DATABASE"
case 1:
readDatabase();
try
{
browseList.deleteAll();
}
catch(Exception e)
{}
for(int i=0;i<mDevices.size(); i++)
{
BlueToothDevice currentDevice = (BlueToothDevice)mDevices.elementAt(i);
browseList.append(currentDevice.deviceName + " - " + currentDevice.context, null);
}
```

```
getDisplay().setCurrent(browseList);
break;

// user chooses "SEARCH BT DATABASE"
case 2:
searchChoiceGroup.setSelectedIndex(0, true);
deviceSearch.setString("");
getDisplay().setCurrent(searchForm);
break;

// user chooses "RESET BT DATABASE"
case 3:
getDisplay().setCurrent(choiceForm);
break;

// user chooses "ADD CONTEXT"
case 4:
getDisplay().setCurrent(addContext);
break;
}
}
}
else if(disp == addContext)
{
if(c == backFromAddContext)
{
getDisplay().setCurrent(mainList);
}
else if(c == okAddContext)
{
String context = newContext.getString();
for(int i=0;i<vContextSave.size();i++)
{
if(((String)vContextSave.elementAt(i)).equalsIgnoreCase(context))
{
showAlert("Error", "This context already exists", addContext, AlertType.ERROR);
return;
}
}



vContextSave.addElement(context);
searchChoiceGroup.append(context, null);
saveList.append(context, null);

String contexts = (String)vContextSave.elementAt(0);
for(int i=1;i<vContextSave.size();i++)
{
contexts += "|" + (String)vContextSave.elementAt(i);
}
saveString(contexts, "Context");
showAlert("INFO", "The context is saved", mainList, AlertType.INFO);
}
}
else if(disp == choiceForm)
{
if(c == cancelCommand)
{
getDisplay().setCurrent(mainList);
}
else if(c == okCommand)
{
```

```
// reads number of record
int count = getInt("RecordCount");

try
{
for(int i=0;i<count;i++)
{
// deleting each record
RecordStore.deleteRecordStore(""+i);
}

RecordStore.deleteRecordStore("RecordCount");
showAlert("", "Database deleted", mainList, AlertType.INFO);
}
catch(Exception e)
{
showAlert("", "Failed to delete Database", mainList, AlertType.ERROR);
}

}
}
else if(disp == searchForm)
{
if(c == backFromSearch)
{
getDisplay().setCurrent(mainList);
}
// user initiates a device search
else if(c == searchCommnd)
{
readSearchDatabase();
if(mDevices.size() == 0)
{
showAlert("", "No device found", searchForm, AlertType.INFO);
}
else
{
try
{
browseList.deleteAll();
}
catch(Exception e)
{}
for(int i=0;i<mDevices.size(); i++)
{
BlueToothDevice currentDevice = (BlueToothDevice)mDevices.elementAt(i);
browseList.append(currentDevice.deviceName + " - " + currentDevice.context,
null);
}

getDisplay().setCurrent(browseList);
}
}
}
else if(disp == browseList)
{
if(c == backFromBrowse)
{
getDisplay().setCurrent(mainList);
}
// user chooses a device to display details
else if(c == List.SELECT_COMMAND)
{
```

```
try
{
detailsForm.deleteAll();
}
catch(Exception e)
{}

BlueToothDevice currentDevice =
       (BlueToothDevice)mDevices.elementAt(browseList.getSelectedIndex());
detailsForm.append(new StringItem("Device", currentDevice.deviceName + "\n\n"));
detailsForm.append(new StringItem("Context", currentDevice.context + "\n\n"));

for(int i=0;i<currentDevice.serviceNames.size();i++)
{
detailsForm.append(new StringItem("Service #" +(i+1),
       (String)currentDevice.serviceNames.elementAt(i)+ "\n"));
}
getDisplay().setCurrent(detailsForm);
}
}
else if(disp == deviceList)
{
// user chooses a device for saving in the RMS
if(c == saveCommnd)
{
getDisplay().setCurrent(getSaveList());
}
else if(c == backFromDeviceList)
{
getDisplay().setCurrent(mainList);
}
}
else if(disp == saveList)
{

/**
  * The device is saved into RMS in the following format:
  * context|deviceCount|deviceNameA|serviceCountA|service1|service2|...
  * deviceNameB|serviceCountB|service1|service2|
  */

if(c == List.SELECT_COMMAND)
{
int recordId = getInt("RecordCount");

String saveString = vContextSave.elementAt(saveList.getSelectedIndex())
                    + "|" + mDevices.size();
for(int j=0;j<mDevices.size();j++)
{
BlueToothDevice currentDevice = (BlueToothDevice)mDevices.elementAt(j);
saveString += "|" + currentDevice.deviceName + "|" +
                    currentDevice.serviceNames.size();
for(int i=0; i<currentDevice.serviceNames.size() ;i++)
{
saveString += "|" + (String)currentDevice.serviceNames.elementAt(i);
}
}

saveString(saveString, ""+recordId);
recordId++;
saveInt(recordId, "RecordCount");
showAlert("Notification", "Devices are saved into the Database", deviceList,
                    AlertType.INFO);
```

```
}
else if(c == backFromSave)
{
getDisplay().setCurrent(deviceList);
}
}
else if(disp == detailsForm)
{
if(c == backFromDetails)
{
getDisplay().setCurrent(browseList);
}
else if(c == deleteFrmDetail)
{
getDisplay().setCurrent(deleteConfirmForm);
}
}
}

// saves "id" into database named "btDB"
public void saveString(String id, String btDB)
{
RecordStore rs = null;
try
{
rs = RecordStore.openRecordStore(btDB, true);;
ByteArrayOutputStream baos = new ByteArrayOutputStream();
DataOutputStream dos = new DataOutputStream(baos);
dos.write(id.getBytes());
// get all records
RecordEnumeration re = rs.enumerateRecords(null, null, false);
if (re.hasNextElement()) {
int iRecordID = re.nextRecordId();
rs.setRecord(iRecordID, baos.toByteArray(), 0, baos.size());
}
else {
rs.addRecord(baos.toByteArray(), 0, baos.size());
}
}

catch (Exception e) {
}
finally
{
try
{
if(rs != null)
rs.closeRecordStore();
}
catch(Exception e)
{}
}
}

public String geString(String btDB)
{
String result = null;
RecordStore rs = null;

try {
rs = RecordStore.openRecordStore(btDB, true);
// all records
RecordEnumeration re = rs.enumerateRecords(null, null, false);
```

```
if (re.hasNextElement()) {
byte[] data = re.nextRecord();
result = new String(data);
}
}

catch (Exception e) {
}
finally
{
try{
if(rs != null)
rs.closeRecordStore();
}
catch(Exception e)
{}
}
return result;
}

// gets integer from database named "btDB"
public int getInt(String btDB)
{
// not started
int result = 0;
RecordStore rs = null;
try {
rs = RecordStore.openRecordStore(btDB, true);
// all records
RecordEnumeration re = rs.enumerateRecords(null, null, false);
if (re.hasNextElement()) {
byte[] data = re.nextRecord();
DataInputStream dis = new DataInputStream(new ByteArrayInputStream(data));
result = dis.readInt();
}
}

catch (Exception e) {
}
finally
{
try{
if(rs != null)
rs.closeRecordStore();
}
catch(Exception e)
{}
}
return result;
}

// saves "value" into database named "btDB"
public void saveInt(int value, String btDB)
{
RecordStore rs = null;
try {
rs = RecordStore.openRecordStore(btDB, true);;
ByteArrayOutputStream baos = new ByteArrayOutputStream();
DataOutputStream dos = new DataOutputStream(baos);
dos.writeInt(value);

// all records
RecordEnumeration re = rs.enumerateRecords(null, null, false);
```

```
if (re.hasNextElement()) {
int iRecordID = re.nextRecordId();
rs.setRecord(iRecordID, baos.toByteArray(), 0, baos.size());
}
else {
rs.addRecord(baos.toByteArray(), 0, baos.size());
}
}

catch (Exception e) {
}
finally
{
try{
if(rs != null)
rs.closeRecordStore();
}
catch(Exception e)
{}
}
}

// this method updates the Midlet's form
public void updateStatus(String status){
mainForm.append(new StringItem(null,  status + "\n\n" ));
}

/**
  * This method splits a string of "abc|def|ghi" into a
  * string array of "abc", "def" and "ghi"
  */
public String[] split(String sStr, String match) {
Vector vTemp = new Vector();
int ind;
int indStart = 0;
try{

while ((ind = sStr.indexOf(match,indStart)) != -1)
{
vTemp.addElement(sStr.substring(indStart, ind));
indStart = ind+1;
}
vTemp.addElement(sStr.substring(indStart));
}
catch(Exception e)
{
}

String[] sTemp = new String[vTemp.size()];
vTemp.copyInto(sTemp);

return sTemp;
}
}
```

DeviceDiscoverer.java

This Java file is written to perform the Bluetooth device discovery.

```
import javax.bluetooth.*;
import java.util.*;

public class DeviceDiscoverer implements DiscoveryListener {

BluetoothDiscovererMidlet midlet;
Vector remoteDevices = new Vector();

DiscoveryAgent discoveryAgent;

/**
 * A simple constructor that starts the Bluetooth device discovery
 * process according the JSR-82 API. In order for remote Bluetooth
 * devices to be found, then each Bluetooth device must be in
 * "discoverable" mode.
 */

public DeviceDiscoverer(BluetoothDiscovererMidlet midlet) {
this.midlet = midlet;
try {
LocalDevice localDevice = LocalDevice.getLocalDevice();
discoveryAgent = localDevice.getDiscoveryAgent();

midlet.updateStatus("Searching for Bluetooth devices in the area...");
discoveryAgent.startInquiry(DiscoveryAgent.GIAC, this);
}

catch(Exception e) {
e.printStackTrace();
}
}

/**
 * This method is called each time when a remote
 * Bluetooth device has been discovered.
 */
public void deviceDiscovered(RemoteDevice remoteDevice, DeviceClass cod) {
try{
remoteDevices.addElement(remoteDevice);
midlet.updateStatus("Device Discovered: " + remoteDevice.getFriendlyName(true));
}

catch(Exception e){
e.printStackTrace();
}
}
```

```
/**
 * This method is called when the
 * Bluetooth device discovery process has ended.
 */
public void inquiryCompleted(int discType) {
String inqStatus = null;

if (discType == DiscoveryListener.INQUIRY_COMPLETED) {
inqStatus = "Inquiry completed. " + remoteDevices.size() + " Devices found";
}
else if (discType == DiscoveryListener.INQUIRY_TERMINATED) {
inqStatus = "Inquiry terminated";
}
else if (discType == DiscoveryListener.INQUIRY_ERROR) {
inqStatus = "Inquiry error";
}

midlet.updateStatus(inqStatus);
midlet.mainForm.deleteAll();
midlet.mainForm.addCommand(midlet.serviceSearchCommand);
midlet.updateStatus("Click on the Search button to search for the all services
                    available");
}

/**
  * This method is not used, but any implementation of DiscoveryListerer.java
  * must include these methods
  * @param transID int
  * @param servRecord ServiceRecord[]
  */
public void servicesDiscovered(int transID, ServiceRecord[] servRecord){}

/**
  * This method is not used, but any implementation of
  * DiscoveryListerer.java must include these methods
  * @param transID int
  * @param servRecord ServiceRecord[]
  */
public void serviceSearchCompleted(int transID, int respCode) {}
}
```

ServiceDiscoverer.java

This Java file is written to perform the Bluetooth services search of devices discovered.

```
import javax.bluetooth.*;
import java.io.*;
import java.util.Vector;

public class ServiceDiscoverer extends Thread implements DiscoveryListener {

UUID[] uuidSet = {new UUID(0x0003) };
int[] attrSet = {0x0100, 0x0003, 0x0004};

BluetoothDiscovererMidlet midlet = null;
ServiceRecord serviceRecord = null;
RemoteDevice remoteDevice = null;
String connectionURL = null;

BlueToothDevice blueToothDevice;
/**
* A simple constructor
* @param midlet BluetoothDiscovererMidlet
* @param remoteDevice RemoteDevice
*/

public ServiceDiscoverer(BluetoothDiscovererMidlet midlet, RemoteDevice
                         remoteDevice){
this.midlet = midlet;
this.remoteDevice = remoteDevice;
}

/**
 * ServiceDiscoverer.java extends Thread,
 * so the run method allows the
 * class to run in a separate thread.
 */

public void run(){

try {
LocalDevice localDevice = LocalDevice.getLocalDevice();
DiscoveryAgent discoveryAgent = localDevice.getDiscoveryAgent();

String deviceName;
try
{
deviceName = remoteDevice.getFriendlyName(false);
blueToothDevice = new BlueToothDevice(deviceName);
}
catch(IOException e)
{
deviceName = remoteDevice.getBluetoothAddress();
blueToothDevice = new BlueToothDevice(deviceName);
}
midlet.updateStatus("Device Discovered:"  + deviceName);
discoveryAgent.searchServices(attrSet, uuidSet, remoteDevice, this);
}
catch(Exception e) {
midlet.serviceSerchComplete();
}
}
```

```
/**
 * This method is called when a new service is discovered
 * on a remote Bluetooth device
 */
public void servicesDiscovered(int transID, ServiceRecord[] servRecord) {

for(int i = 0; i < servRecord.length; i++) {

DataElement serviceNameElement = servRecord[i].getAttributeValue(0x0100);
String temp_serviceName = (String)serviceNameElement.getValue();
String serviceName = temp_serviceName.trim();
midlet.updateStatus("-" + serviceName );

blueToothDevice.serviceNames.addElement(serviceName);
}
}

/**
 * This method is called when the service search process
 * is completed for a remote Bluetooth device.
 */
public void serviceSearchCompleted(int transID, int respCode) {

String searchStatus = null;

if (respCode == DiscoveryListener.SERVICE_SEARCH_DEVICE_NOT_REACHABLE) {
midlet.updateStatus("Device not reachable");
}
else if (respCode == DiscoveryListener.SERVICE_SEARCH_NO_RECORDS) {
midlet.updateStatus("Service not available");
}
else if (respCode == DiscoveryListener.SERVICE_SEARCH_COMPLETED) {

midlet.mDevices.addElement(blueToothDevice);
//do nothing
}
else if (respCode == DiscoveryListener.SERVICE_SEARCH_TERMINATED) {
midlet.updateStatus("Service search terminated");
}
else if (respCode == DiscoveryListener.SERVICE_SEARCH_ERROR) {
midlet.updateStatus("Service search error");
}

midlet.serviceSerchComplete();
}
/**
 * This method is not used, but any implementation of DiscoveryListerer.java
 * must include these methods
 * @param transID int
 * @param servRecord ServiceRecord[]
 */
public void inquiryCompleted(int discType){}

/**
  * This method is not used, but any implementation of DiscoveryListerer.java
  * must include these methods
  * @param transID int
  * @param servRecord ServiceRecord[]
  */
public void deviceDiscovered(RemoteDevice btDevice, DeviceClass cod){}
}
```

BlueToothDevice.java

This Java file is written to hold all the information of a device together in the application.

```
import java.util.*;

public class BlueToothDevice {

Vector serviceNames;
String deviceName;
String context;
int recordId;

// creates a new instance of BlueToothDevice
public BlueToothDevice(String name) {
deviceName = name;
serviceNames = new Vector();
}
}
```