# A10. Multi-Platform Bluetooth Remote Control Source Code

## A10.1 J2ME

### A10.1.1 Package: gui

#### DeviceScreen.java

**package** gui;

**import** appLogic.\* ;

**import** javax.microedition.lcdui.\* ;

//Displays information about a Bluetooth device.

**public** **class** DeviceScreen **extends** Form **implements** CommandListener {

**private** Main main ;

**private** String callingclass ;

**private** String address ;

**private** String name ;

**private** **int** majorClass ;

**private** **int** minorClass ;

**private** **int** serviceClass ;

**static** **final** Command *backCommand* = **new** Command("Back", Command.*SCREEN*, 1);

**public** DeviceScreen() {

**super**("Device info") ;

addCommand(*backCommand*) ;

setCommandListener(**this**) ;

}

**protected** **void** setMain(Main m) {

main = m ;

}

**public** **void** setCallingClass(String cc) {

callingclass = cc ;

}

//Shows the information about the device

**protected** **void** setDevice(BTDevice btd) {

address = btd.getAddress() ;

name = btd.getName() ;

majorClass = btd.getMajorClass() ;

minorClass = btd.getMinorClass() ;

serviceClass = btd.getServiceClass() ;

deleteAll() ;

String mc = "" ;

**if**(majorClass == Constants.*MAJOR\_CLASS\_COMPUTER*) {

mc = "Computer" ;

}

**else** **if**(majorClass == Constants.*MAJOR\_CLASS\_PHONE*) {

mc = "Phone" ;

}

**else** **if**(majorClass == Constants.*MAJOR\_CLASS\_IMAGING*) {

mc = "Imaging device" ;

}

**if**(btd.getName().equals("")) {

append("Device name:\n" + "- Not found\nBT address:\n" + "- " + address +

"\nMajor class:\n" + "- " + mc + "(" + Integer.*toString*(majorClass) + ")" +

"\nMinor class:\n" + "- " + Integer.*toString*(minorClass) + "\nService class:\n" + "- " + Integer.*toString*(serviceClass)) ;

}

**else** {

append("Device name:\n" + "- " + name + "\nBT address:\n" + "- " + address +

"\nMajor class:\n" + "- " + mc + "(" + Integer.*toString*(majorClass) + ")" +

"\nMinor class:\n" + "- " + Integer.*toString*(minorClass) + "\nService class:\n" + "- " + Integer.*toString*(serviceClass)) ;

}

}

//Registers when a button has been pushed

**public** **void** commandAction(Command c, Displayable d) {

**if**(c == *backCommand*) {

**if**(callingclass.equals(Constants.*CLASS\_PROFILEADMIN*))

main.showProfileAdmin() ;

**else** **if**(callingclass.equals(Constants.*CLASS\_REMOTECONTROLMENU*))

main.showRemoteControlMenu() ;

}

}

}

#### GuiSingleton.java

**package** gui;

**import** guiController.\* ;

//Ensures that there is only one instance of each user interface object

//(the only exception is the Message class).

**public** **class** GuiSingleton {

**private** **static** GuiSingleton *guisingleton* = **null** ;

//gui classes

**private** Intro intro = **null** ;

**private** MainGui maingui = **null** ;

**private** Loading loading = **null** ;

**private** RemoteControlMenu rcmenu = **null** ;

**private** RemoteControlGui rcgui = **null** ;

**private** RemoteControlOptions rcoptions = **null** ;

**private** RemoteControlKeyEdit rckeyedit = **null** ;

**private** RemoteControlPointerSpeed rcpointer = **null** ;

**private** ProfileAdmin padmin = **null** ;

**private** DeviceScreen ds = **null** ;

//Controller classes

**private** ProfileAdminCont pacont = **null** ;

**private** RemoteControlCont rccont = **null** ;

//Private constructor to make sure it is only created once

**private** GuiSingleton() {

}

**public** **static** **synchronized** GuiSingleton getGuiSingleton() {

**if**(*guisingleton* == **null**) {

*guisingleton* = **new** GuiSingleton() ;

}

**return** *guisingleton* ;

}

**public** **synchronized** Intro getIntro() {

**if**(intro == **null**) {

intro = **new** Intro() ;

}

**return** intro ;

}

**public** **synchronized** MainGui getMainGui() {

**if**(maingui == **null**) {

maingui = **new** MainGui() ;

}

**return** maingui ;

}

**public** **synchronized** Loading getLoading() {

**if**(loading == **null**) {

loading = **new** Loading() ;

}

**return** loading ;

}

**public** **synchronized** ProfileAdmin getProfileAdmin() {

**if**(padmin == **null**) {

padmin = **new** ProfileAdmin() ;

}

**return** padmin ;

}

**public** **synchronized** DeviceScreen getDeviceScreen() {

**if**(ds == **null**) {

ds = **new** DeviceScreen() ;

}

**return** ds ;

}

**public** **synchronized** RemoteControlMenu getRemoteControlMenu() {

**if**(rcmenu == **null**) {

rcmenu = **new** RemoteControlMenu() ;

}

**return** rcmenu ;

}

**public** **synchronized** RemoteControlGui getRemoteControlGui() {

**if**(rcgui == **null**) {

rcgui = **new** RemoteControlGui() ;

}

**return** rcgui ;

}

**public** **synchronized** RemoteControlOptions getRemoteControlOptions() {

**if**(rcoptions == **null**) {

rcoptions = **new** RemoteControlOptions() ;

}

**return** rcoptions ;

}

**public** **synchronized** RemoteControlKeyEdit getRemoteControlKeyEdit() {

**if**(rckeyedit == **null**) {

rckeyedit = **new** RemoteControlKeyEdit() ;

}

**return** rckeyedit ;

}

**public** **synchronized** RemoteControlPointerSpeed getRemoteControlPointerSpeed() {

**if**(rcpointer == **null**) {

rcpointer = **new** RemoteControlPointerSpeed() ;

}

**return** rcpointer ;

}

**public** **synchronized** ProfileAdminCont getProfileAdminCont() {

**if**(pacont == **null**) {

pacont = **new** ProfileAdminCont() ;

}

**return** pacont ;

}

**public** **synchronized** RemoteControlCont getRemoteControlCont() {

**if**(rccont == **null**) {

rccont = **new** RemoteControlCont() ;

}

**return** rccont ;

}

}

#### Intro.java

**package** gui;

**import** appLogic.\* ;

**import** java.io.\* ;

**import** javax.microedition.lcdui.\* ;

//Displays a loading screen while the application is started.

**public** **class** Intro **extends** Canvas **implements** CommandListener {

**private** Main main ;

**private** Image image ;

**private** **int** wcenter ;

**private** **int** hcenter ;

**private** **boolean** ready ;

**private** Command startCommand = **new** Command("Start", Command.*OK*, 1) ;

**private** Command exitCommand = **new** Command("Exit", Command.*BACK*, 2) ;

**public** Intro() {

**try** {

image = Image.*createImage*(Constants.*PICTURE\_INTRO*) ;

wcenter = (getWidth() / 2) - (image.getWidth() / 2) ;

hcenter = (getHeight() / 2) - (image.getHeight() / 2) ;

}

**catch**(IOException io) {

}

ready = **false** ;

addCommand(startCommand) ;

addCommand(exitCommand) ;

setCommandListener(**this**) ;

}

**public** **void** setMain(Main m) {

main = m ;

}

**public** **void** setReady() {

ready = **true** ;

repaint() ;

}

**protected** **void** paint(Graphics g) {

g.setColor(Constants.*BACKGROUND\_COLOUR*) ;

g.fillRect(0, 0, getWidth(), getHeight()) ;

g.drawImage(image, wcenter, hcenter, Graphics.*TOP* | Graphics.*LEFT*) ;

g.setColor(Constants.*TEXT\_COLOUR*) ;

String start = "" ;

**if**(ready)

start = "Press start" ;

**else**

start = "Please wait" ;

g.drawString(start, getWidth() / 2, (getHeight() / 2) + (image.getHeight() / 2) + 14, Graphics.*HCENTER* | Graphics.*BASELINE*) ;

}

**public** **void** commandAction(Command c, Displayable d) {

**if**(c == exitCommand) {

main.exitApp() ;

}

**else** **if**(c == startCommand) {

**if**(ready)

main.showMainGui() ;

}

}

}

#### Loading.java

**package** gui;

**import** appLogic.\* ;

**import** javax.microedition.lcdui.\* ;

//Displays a loading screen for each task that takes some time to finish

//(like searching for Bluetooth devices).

//It is used to show that the program is still running while a time consuming

//task is being executed in the background.

**public** **class** Loading **extends** Canvas {

**private** Main main ;

**private** CounterThread ct ;

**private** Thread thread ;

**private** Font font ;

**private** **int** fheight ;

**private** **int** maxprogress ;

**private** **int** progress ;

**private** String timeouttype ;

**private** String name ;

**private** String callingclass ;

**private** **boolean** cont ;

**public** Loading() {

ct = **new** CounterThread() ;

callingclass = "" ;

font = Font.*getDefaultFont*() ;

fheight = font.getHeight() ;

}

**protected** **void** setMain(Main m) {

main = m ;

}

**public** **void** setName(String n, String type, String cc) {

name = n ;

timeouttype = type ;

callingclass = cc ;

}

**public** **void** startCount() {

**if**(timeouttype.equals(Constants.*LOADING\_TIMEOUT\_UNDEFINED*)) {

maxprogress = 5 ;

progress = 0 ;

cont = **true** ;

thread = **new** Thread(ct) ;

thread.start() ;

}

}

**public** **void** stoppCount(String cc) {

callingclass = cc ;

thread.interrupt() ;

}

**protected** **void** paint(Graphics g) {

g.setColor(Constants.*BACKGROUND\_COLOUR*) ;

g.fillRect(0, 0, getWidth(), getHeight()) ;

g.setColor(Constants.*TEXT\_COLOUR*) ;

g.drawString(name, (getWidth() / 2), (getHeight() / 2), Graphics.*HCENTER* | Graphics.*BASELINE*) ;

**if**(progress == 1) {

g.fillRect(((getWidth() / 2) - 20), (getHeight() / 2) + fheight, 10, fheight) ;

}

**else** **if**(progress == 2) {

g.fillRect(((getWidth() / 2) - 20), (getHeight() / 2) + fheight, 10, fheight) ;

g.fillRect((getWidth() / 2) - 5, (getHeight() / 2) + fheight, 10, fheight) ;

}

**else** **if**(progress == 3) {

g.fillRect(((getWidth() / 2) - 20), (getHeight() / 2) + fheight, 10, fheight) ;

g.fillRect((getWidth() / 2) - 5, (getHeight() / 2) + fheight, 10, fheight) ;

g.fillRect((getWidth() / 2) + 10, (getHeight() / 2) + fheight, 10, fheight) ;

}

}

**private** **class** CounterThread **extends** Thread {

**public** **void** run() {

**while**(cont) {

**try** {

**for**(**int** i = 0; i <= maxprogress; i++) {

repaint() ;

Thread.*sleep*(500) ;

progress++ ;

**if**(progress == 4) {

progress = 0 ;

}

}

}

**catch**(InterruptedException ie) {

cont = **false** ;

}

}

**if**(callingclass.equals(Constants.*CLASS\_PROFILEADMIN*))

main.showProfileAdmin() ;

**else** **if**(callingclass.equals(Constants.*CLASS\_REMOTECONTROLMENULOADING*))

main.showRemoteControlMenu() ;

**else** **if**(callingclass.equals(Constants.*CLASS\_REMOTECONTROLMENU*))

main.showRemoteControlGui() ;

**else** **if**(callingclass.equals(Constants.*CLASS\_REMOTECONTROLGUI*))

main.showRemoteControlGui2() ;

**else** **if**(callingclass.equals(Constants.*CLASS\_REMOTECONTROLOPTIONS*))

main.showRemoteControlOptions() ;

}

}

}

#### Main.java

**package** gui;

**import** appLogic.\* ;

**import** guiController.\* ;

**import** javax.microedition.midlet.\* ;

**import** javax.microedition.lcdui.\*;

**import** javax.bluetooth.\* ;

//Starts the application and has responsibility for showing each user interface.

**public** **class** Main **extends** MIDlet {

**private** Intro intro ;

**private** Loading loading ;

**private** DeviceScreen ds ;

**private** MainGui maingui ;

**private** ProfileAdmin padmin ;

**private** RemoteControlMenu rcmenu ;

**private** RemoteControlGui rcgui ;

**private** ProfileAdminCont pacont ;

**private** RemoteControlCont rccont ;

**private** RemoteControlOptions rcoptions ;

**private** RemoteControlKeyEdit rckeyedit ;

**private** RemoteControlPointerSpeed pointer ;

**private** Display display ;

**public** Main() {

}

//Loads the classes needed to start the application.

**private** **void** loadClasses() {

loading = (GuiSingleton.*getGuiSingleton*()).getLoading() ;

loading.setMain(**this**) ;

ds = (GuiSingleton.*getGuiSingleton*()).getDeviceScreen() ;

ds.setMain(**this**) ;

maingui = (GuiSingleton.*getGuiSingleton*()).getMainGui() ;

maingui.setMain(**this**) ;

rcoptions = (GuiSingleton.*getGuiSingleton*()).getRemoteControlOptions() ;

rcoptions.setMain(**this**) ;

padmin = (GuiSingleton.*getGuiSingleton*()).getProfileAdmin() ;

padmin.setMain(**this**) ;

padmin.setRemoteControlOptions(rcoptions) ;

pacont = (GuiSingleton.*getGuiSingleton*()).getProfileAdminCont() ;

pacont.setProfileAdmin(padmin) ;

rcmenu = (GuiSingleton.*getGuiSingleton*()).getRemoteControlMenu() ;

rcmenu.setMain(**this**) ;

rcgui = (GuiSingleton.*getGuiSingleton*()).getRemoteControlGui() ;

rcgui.setMain(**this**) ;

rcgui.setRemoteControlOptions(rcoptions) ;

rckeyedit = (GuiSingleton.*getGuiSingleton*()).getRemoteControlKeyEdit() ;

rckeyedit.setMain(**this**) ;

rckeyedit.setRemoteControlOptions(rcoptions) ;

pointer = (GuiSingleton.*getGuiSingleton*()).getRemoteControlPointerSpeed();

pointer.setRemoteControlKeyEdit(rckeyedit) ;

rccont = (GuiSingleton.*getGuiSingleton*()).getRemoteControlCont() ;

rccont.setRemoteControlGui(rcgui) ;

rccont.setRemoteControlMenu(rcmenu) ;

rccont.setRemoteControlOptions(rcoptions) ;

intro.setReady() ;

}

**protected** **void** destroyApp(**boolean** b) **throws** MIDletStateChangeException {

}

**protected** **void** pauseApp() {

}

//It first test whether the bluetooth is turned on. This does not work on all device

//since some will allow you the get the local device even if the bluetooth is turned off.

**protected** **void** startApp() **throws** MIDletStateChangeException {

**try** {

LocalDevice local = LocalDevice.*getLocalDevice*() ;

display = Display.*getDisplay*(**this**) ;

intro = (GuiSingleton.*getGuiSingleton*()).getIntro() ;

intro.setMain(**this**) ;

display.setCurrent(intro) ;

loadClasses() ;

}

**catch**(BluetoothStateException bse) {

display.setCurrent((**new** Message(**this**, Constants.*ERROR\_BLUETOOTH\_OFF*, Constants.*CLASS\_MAIN*)).returnMessage()) ;

}

}

//GUI operations.

**protected** **void** showMainGui() {

intro = **null** ;

maingui.repaint() ;

display.setCurrent(maingui) ;

}

**protected** **void** showProfileAdmin() {

padmin.openRMS() ;

padmin.repaint() ;

display.setCurrent(padmin) ;

}

**protected** **void** showLoadingRunning(Loading l) {

l.repaint() ;

display.setCurrent(l) ;

}

**protected** **void** showDeviceScreen(DeviceScreen ds) {

display.setCurrent(ds) ;

}

**protected** **void** showMessageScreen(Message m) {

display.setCurrent(m.returnMessage()) ;

}

**protected** **void** showRemoteControlLoading() {

padmin.openRMS() ;

rcmenu.repaint() ;

rcmenu.showScreen() ;

}

**protected** **void** showRemoteControlMenu() {

rcmenu.repaint() ;

display.setCurrent(rcmenu) ;

}

**protected** **void** showRemoteControlGui() {

rcgui.openRMS() ;

rcgui.setMenu() ;

rcgui.repaint() ;

display.setCurrent(rcgui) ;

}

**protected** **void** showRemoteControlGui2() {

rcgui.repaint() ;

display.setCurrent(rcgui) ;

}

**protected** **void** showRemoteControlOptions() {

rcoptions.repaint() ;

display.setCurrent(rcoptions) ;

}

**protected** **void** showRemoteControlKeyEdit(RemoteControlKeyEdit rckeyedit) {

rckeyedit.repaint() ;

display.setCurrent(rckeyedit) ;

}

**protected** **void** showRemoteControlPointerSpeed(RemoteControlPointerSpeed rcpointer) {

display.setCurrent(rcpointer) ;

}

//Exit application

**protected** **void** exitApp() {

notifyDestroyed() ;

}

}

#### MainGui.java

**package** gui;

**import** appLogic.\* ;

**import** javax.microedition.lcdui.\* ;

**import** java.util.\* ;

//The main menu

**public** **class** MainGui **extends** Canvas **implements** CommandListener {

**private** Main main ;

**private** Font font ;

**private** **int** fheight ;

**private** **int** wcenter ;

**private** **int** hcenter ;

**private** **int** position ;

**private** Vector menu ;

**private** Command openCommand = **new** Command("Open", Command.*OK*, 1) ;

**private** Command exitCommand = **new** Command("Exit", Command.*BACK*, 2) ;

**public** MainGui() {

font = Font.*getDefaultFont*() ;

fheight = font.getHeight() ;

wcenter = getWidth() / 2 ;

hcenter = getHeight() / 2 ;

position = 0 ;

menu = **new** Vector() ;

menu.addElement("Remote Control") ;

menu.addElement("Profile admin") ;

menu.addElement("Exit") ;

addCommand(openCommand) ;

addCommand(exitCommand) ;

setCommandListener(**this**) ;

repaint() ;

}

**protected** **void** setMain(Main m) {

main = m ;

}

**private** **int** getPlace(**int** number) {

**int** temp = (fheight + 1) \* (menu.size() + 1) ;

temp = temp / 2 ;

temp = hcenter - temp ;

**return** temp +((fheight + 1) \* number) ;

}

**protected** **void** paint(Graphics g) {

g.setColor(Constants.*BACKGROUND\_COLOUR*) ;

g.fillRect(0, 0, getWidth(), getHeight()) ;

**for**(**int** i = 0; i < menu.size(); i++) {

**if**(position == i) {

g.setColor(Constants.*MENUITEM\_BOX\_COLOUR*) ;

g.fillRect(0, getPlace(i) + 5, getWidth(), fheight) ;

g.setColor(Constants.*TEXT\_COLOUR\_SELECTED*) ;

}

**else** **if** (g.getColor() == Constants.*TEXT\_COLOUR\_SELECTED* || g.getColor() == Constants.*BACKGROUND\_COLOUR*){

g.setColor(Constants.*TEXT\_COLOUR*) ;

}

g.drawString((String)menu.elementAt(i), wcenter, getPlace(i + 1), Graphics.*HCENTER* | Graphics.*BASELINE*) ;

}

}

**protected** **void** keyPressed(**int** keyCode) {

**switch**(keyCode) {

//UP

**case** -1:

**case** 50:

**if**(position > 0)

position-- ;

**else**

position = (menu.size() - 1) ;

**break** ;

//DOWN

**case** -2:

**case** 56:

**if**(position < (menu.size() - 1))

position++ ;

**else**

position = 0 ;

**break** ;

//SELECT

**case** 53:

**case** -5:

commandControl() ;

**break** ;

}

repaint() ;

}

**private** **void** commandControl() {

**if**(position == 0) {

position = 0 ;

main.showRemoteControlLoading() ;

}

**else** **if**(position == 1) {

position = 0 ;

main.showProfileAdmin() ;

}

**else** **if**(position == 2) {

main.exitApp() ;

}

}

**public** **void** commandAction(Command c, Displayable d) {

**if**(c == openCommand) {

commandControl() ;

}

**else** **if**(c == exitCommand) {

main.exitApp() ;

}

}

}

#### Message.java

**package** gui ;

**import** appLogic.\* ;

**import** javax.microedition.lcdui.\* ;

//Displays an information message.

//This is for example used to let the user know that the KeyAction object is saved.

//The message object is created when needed.

**public** **class** Message **implements** CommandListener {

**private** Alert alert ;

**private** Main main ;

**private** String obj ;

**private** **static** **final** Command *okCommand* = **new** Command("Ok", Command.*EXIT*, 1) ;

**public** Message(Main m, String s, String o) {

obj = o ;

main = m ;

alert = **new** Alert(**null**) ;

alert.setString(s) ;

alert.addCommand(*okCommand*) ;

alert.setCommandListener(**this**) ;

}

**public** Alert returnMessage() {

**return** alert ;

}

**public** **void** commandAction(Command c, Displayable d) {

**if**(c == *okCommand*) {

**if**(obj.equals(Constants.*CLASS\_MAIN*)) {

main.exitApp() ;

}

**else** **if**(obj.equals(Constants.*CLASS\_PROFILEADMIN*)) {

main.showProfileAdmin() ;

}

**else** **if**(obj.equals(Constants.*CLASS\_REMOTECONTROLGUI*)) {

main.showRemoteControlGui() ;

}

**else** **if**(obj.equals(Constants.*CLASS\_REMOTECONTROLOPTIONS*)) {

main.showRemoteControlOptions() ;

}

**else** **if**(obj.equals(Constants.*CLASS\_REMOTECONTROLERROR*)) {

main.showMainGui() ;

}

**else** **if**(obj.equals(Constants.*CLASS\_REMOTECONTROLMENU*)) {

main.showRemoteControlMenu() ;

}

}

}

}

#### ProfileAdmin.java

**package** gui;

**import** appLogic.\* ;

**import** guiController.\* ;

**import** javax.microedition.lcdui.\* ;

**import** javax.microedition.lcdui.game.Sprite;

**import** java.io.\* ;

**import** java.util.\* ;

//Handles all the operations on the Bluetooth devices and also contains the menu for the Custom map keys option.

**public** **class** ProfileAdmin **extends** Canvas **implements** CommandListener {

**private** Main main ;

**private** ProfileAdminCont pacont ;

**private** Loading loading ;

**private** DeviceScreen ds ;

**private** RemoteControlOptions rcoptions ;

**private** **int** showmenu ;

**private** Vector menu ;

**private** Vector searchlist ;

**private** Vector saved ;

**private** Font fontmenu ;

**private** Font fontlist ;

**private** **int** fmenuheight ;

**private** **int** flistheight ;

**private** **int** wcenter ;

**private** **int** hcenter ;

**private** **int** position ;

**private** **int** size ;

**private** **int** count ;

**private** Image image ;

**private** Command selectCommand = **new** Command("Select", Command.*OK*, 1) ;

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

**private** Command infoCommand = **new** Command("Info", Command.*OK*, 1) ;

**private** Command saveCommand = **new** Command("Save", Command.*SCREEN*, 2) ;

**private** Command deleteCommand = **new** Command("Delete", Command.*SCREEN*, 2) ;

**public** ProfileAdmin() {

loading = (GuiSingleton.*getGuiSingleton*()).getLoading() ;

pacont = (GuiSingleton.*getGuiSingleton*()).getProfileAdminCont() ;

ds = (GuiSingleton.*getGuiSingleton*()).getDeviceScreen() ;

showmenu = 0 ;

searchlist = **new** Vector() ;

menu = **new** Vector() ;

menu.addElement("Custom map keys") ;

menu.addElement("Open device list") ;

menu.addElement("Search computers") ;

menu.addElement("Search phones") ;

menu.addElement("Search all") ;

menu.addElement("Back") ;

saved = **new** Vector() ;

fontmenu = Font.*getFont*(Font.*FACE\_SYSTEM*, Font.*STYLE\_PLAIN*, Font.*SIZE\_MEDIUM*);

fmenuheight = fontmenu.getHeight() ;

fontlist = Font.*getFont*(Font.*FACE\_SYSTEM*, Font.*STYLE\_PLAIN*, Font.*SIZE\_SMALL*) ;

flistheight = fontlist.getHeight() ;

wcenter = getWidth() / 2 ;

hcenter = getHeight() / 2 ;

showmenu = 0 ;

position = 0 ;

size = 0 ;

count = 0 ;

**try** {

image = Image.*createImage*(Constants.*PICTURE\_ARROW*) ;

}

**catch**(IOException io) {

}

addCommand(backCommand) ;

setCommandListener(**this**) ;

repaint() ;

}

**protected** **void** setMain(Main m) {

main = m ;

}

**protected** **void** setRemoteControlOptions(RemoteControlOptions rco) {

rcoptions = rco ;

}

**protected** **void** openRMS() {

pacont.openRMS() ;

}

//Returns the place on the screen for the gui-element

**private** **int** getPlace(**int** number, String screen) {

**int** temp = 0 ;

**if**(screen.equals("list")) {

**int** listheight = (flistheight + 1) \* (searchlist.size() + 1) ;

**if**(listheight > getHeight()) {

**return** (flistheight + 1) \* number ;

}

**else** {

temp = (flistheight + 1) \* (searchlist.size() + 1) ;

temp = temp / 2 ;

temp = hcenter - temp ;

**return** temp +((flistheight + 1) \* number) ;

}

}

**else** {

temp = (fmenuheight + 1) \* (menu.size() + 1) ;

temp = temp / 2 ;

temp = hcenter - temp ;

**return** temp +((fmenuheight + 1) \* number) ;

}

}

//Returns the number of lines that can be shown on the screen.

**private** **int** getNumberOfLines() {

**return** (getHeight() / (flistheight + 1)) - 1 ;

}

//When a new device is found, this is called and the list is updated

**public** **void** updateDevices(BTDevice btd, String s) {

searchlist.addElement(btd) ;

saved.addElement(s) ;

showDevices(searchlist) ;

}

//Shows the devices and resets the selected position

**private** **void** showDevices(Vector v) {

**if**(v != **null**)

searchlist = v ;

position = 0 ;

count = 0 ;

size = searchlist.size() ;

repaint() ;

}

//Is called when the search is completed, the loading gui is stopped

**public** **void** searchCompleted() {

loading.stoppCount(Constants.*CLASS\_PROFILEADMIN*) ;

repaint() ;

}

//Paints the correct method based on the variable showmenu

//showmenu 0: Menu

//showmenu 1: Search list

//showmenu 2: Saved device list

**protected** **void** paint(Graphics g) {

g.setColor(Constants.*BACKGROUND\_COLOUR*) ;

g.fillRect(0, 0, getWidth(), getHeight()) ;

**if**(showmenu == 0) {

removeCommand(infoCommand) ;

removeCommand(saveCommand) ;

removeCommand(deleteCommand) ;

addCommand(selectCommand) ;

paintMenu(g) ;

}

**else** **if**(showmenu == 1){

removeCommand(saveCommand) ;

removeCommand(selectCommand) ;

addCommand(infoCommand) ;

addCommand(deleteCommand) ;

paintSearch(g) ;

}

**else** **if**(showmenu == 2) {

removeCommand(selectCommand) ;

removeCommand(deleteCommand) ;

addCommand(saveCommand) ;

addCommand(infoCommand) ;

paintSearch(g) ;

}

}

//Creates the menu

**private** **void** paintMenu(Graphics g) {

g.setFont(fontmenu) ;

**for**(**int** i = 0; i < menu.size(); i++) {

**if**(position == i) {

g.setColor(Constants.*MENUITEM\_BOX\_COLOUR*) ;

g.fillRect(0, getPlace(i, "menu") + 5, getWidth(), fmenuheight) ;

g.setColor(Constants.*TEXT\_COLOUR\_SELECTED*) ;

}

**else** **if** (g.getColor() == Constants.*TEXT\_COLOUR\_SELECTED* || g.getColor() == Constants.*BACKGROUND\_COLOUR*){

g.setColor(Constants.*TEXT\_COLOUR*) ;

}

g.drawString((String)menu.elementAt(i), wcenter, getPlace((i + 1), "menu"), Graphics.*HCENTER* | Graphics.*BASELINE*) ;

}

}

//Creates the search list or the saved device list

**private** **void** paintSearch(Graphics g) {

**if**(searchlist.size() > 0) {

g.setFont(fontlist) ;

**if**((searchlist.size() + 1) > getNumberOfLines()) {

size = getNumberOfLines() ;

}

**else** {

size = searchlist.size() ;

}

**if**(position < (searchlist.size() - 1) && position == ((size - 1) + count) &&

(searchlist.size() + 1) > getNumberOfLines()) {

count++ ;

}

**else** **if**(count > 0 && position == count && (searchlist.size() + 1) > getNumberOfLines()) {

count-- ;

}

**int** records = 0 ;

**for**(**int** i = count; i < (size + count); i++) {

**if**(position == i) {

g.setColor(Constants.*MENUITEM\_BOX\_COLOUR*) ;

g.fillRect(0, getPlace((i - count), "list") + 5, getWidth(), flistheight) ;

g.setColor(Constants.*TEXT\_COLOUR\_SELECTED*) ;

}

**else** **if** (g.getColor() == Constants.*TEXT\_COLOUR\_SELECTED* || g.getColor() == Constants.*BACKGROUND\_COLOUR*){

g.setColor(Constants.*TEXT\_COLOUR*) ;

}

BTDevice temp = (BTDevice)searchlist.elementAt(i) ;

String name = "" ;

**if**(saved.size() > 0 && ((String)saved.elementAt(i)).equals(Constants.*BTDEVICE\_NEW*))

name = "(New) " + temp.getName() ;

**else**

name = temp.getName() ;

**if**(name.equals("") || name.equals("(New) ")) {

name = name + temp.getAddress() ;

}

**else** **if**(fontlist.stringWidth(name + ": " + temp.getAddress()) < (getWidth() - 15)){

name = name + ": " + temp.getAddress() ;

}

g.drawString(name, wcenter, getPlace(((i - count) + 1), "list"), Graphics.*HCENTER* | Graphics.*BASELINE*) ;

records = i ;

}

**if**(records < (searchlist.size() - 1) && (searchlist.size() > getNumberOfLines()) && position < (searchlist.size()- 2)) {

g.drawRegion(image, 0, 0, image.getWidth(), image.getHeight(), Sprite.*TRANS\_ROT180*,

(getWidth() - image.getWidth()) - 2, getPlace(((records - count) + 1), "list"), Graphics.*LEFT* | Graphics.*TOP*);

}

**if**(records > (getNumberOfLines() - 1)) {

g.drawImage(image, (getWidth() - image.getWidth()) - 2, getPlace(0, "list"), Graphics.*LEFT* | Graphics.*TOP*) ;

}

}

}

**protected** **void** keyPressed(**int** keyCode) {

**switch**(keyCode) {

//UP

**case** -1:

**case** 50:

**if**(showmenu == 0) {

**if**(position > 0)

position-- ;

**else**

position = (menu.size() - 1) ;

}

**else** {

**if**(position > 0)

position-- ;

**else** {

count = (searchlist.size() - size) ;

position = (searchlist.size() - 1) ;

}

}

**break** ;

//DOWN

**case** -2:

**case** 56:

**if**(showmenu == 0) {

**if**(position < (menu.size() - 1))

position++ ;

**else**

position = 0 ;

}

**else** {

**if**(position < (searchlist.size() - 1))

position++ ;

**else** {

position = 0 ;

count = 0 ;

}

}

**break** ;

//SELECT

**case** 53:

**case** -5:

commandControl() ;

**break** ;

}

repaint() ;

}

//Handles the commands, both from Command-actions and keyPressed-actions

**private** **void** commandControl() {

**if**(showmenu == 0 && position == 0) {

searchlist.removeAllElements() ;

saved.removeAllElements() ;

rcoptions.setCallingClass(Constants.*CLASS\_PROFILEADMIN*) ;

rcoptions.setKeyActions(**null**) ;

}

**else** **if**(showmenu == 0 && position == 1) {

searchlist.removeAllElements() ;

saved.removeAllElements() ;

showmenu = 1 ;

loading.setName("Opening device list", Constants.*LOADING\_TIMEOUT\_UNDEFINED*, Constants.*CLASS\_PROFILEADMIN*) ;

loading.startCount() ;

main.showLoadingRunning(loading) ;

showDevices(pacont.getStoredDevices()) ;

loading.stoppCount(Constants.*CLASS\_PROFILEADMIN*) ;

}

**else** **if**(showmenu == 0 && position == 2) {

searchlist.removeAllElements() ;

saved.removeAllElements() ;

showmenu = 2 ;

loading.setName("Searching for computers", Constants.*LOADING\_TIMEOUT\_UNDEFINED*, Constants.*CLASS\_PROFILEADMIN*) ;

loading.startCount() ;

main.showLoadingRunning(loading) ;

pacont.searchDevices(Constants.*MAJOR\_CLASS\_COMPUTER*) ;

}

**else** **if**(showmenu == 0 && position == 3) {

searchlist.removeAllElements() ;

saved.removeAllElements() ;

showmenu = 2 ;

loading.setName("Searching for phones", Constants.*LOADING\_TIMEOUT\_UNDEFINED*, Constants.*CLASS\_PROFILEADMIN*) ;

loading.startCount() ;

main.showLoadingRunning(loading) ;

pacont.searchDevices(Constants.*MAJOR\_CLASS\_PHONE*) ;

}

**else** **if**(showmenu == 0 && position == 4) {

searchlist.removeAllElements() ;

saved.removeAllElements() ;

showmenu = 2 ;

loading.setName("Searching for BT devices", Constants.*LOADING\_TIMEOUT\_UNDEFINED*, Constants.*CLASS\_PROFILEADMIN*) ;

loading.startCount() ;

main.showLoadingRunning(loading) ;

pacont.searchDevices(Constants.*MAJOR\_CLASS\_ALL*) ;

}

**else** **if**(showmenu == 0 && position == 5) {

position = 0 ;

main.showMainGui() ;

}

**else** {

ds.setCallingClass(Constants.*CLASS\_PROFILEADMIN*) ;

ds.setDevice((BTDevice)searchlist.elementAt(position)) ;

main.showDeviceScreen(ds) ;

}

}

**public** **void** commandAction(Command c, Displayable d) {

**if**(c == selectCommand) {

commandControl() ;

}

**else** **if**(showmenu == 0 && c == backCommand) {

position = 0 ;

main.showMainGui() ;

}

**else** **if**(c == backCommand) {

showmenu = 0 ;

position = 0 ;

repaint() ;

}

**else** **if**(c == saveCommand) {

**if**(position >= 0) pacont.storeDevice((BTDevice)searchlist.elementAt(position)) ;

}

**else** **if**(c == infoCommand) {

**if**(position >= 0) {

ds.setCallingClass(Constants.*CLASS\_PROFILEADMIN*) ;

ds.setDevice((BTDevice)searchlist.elementAt(position)) ;

main.showDeviceScreen(ds) ;

}

}

**else** **if**(c == deleteCommand && showmenu == 1) {

pacont.deleteDevice(position) ;

}

}

**public** **void** objectSaved(String event) {

Message m = **new** Message(main, event, Constants.*CLASS\_PROFILEADMIN*) ;

main.showMessageScreen(m) ;

**if**(event.equals(Constants.*BTDEVICE\_DELETED*)) {

showDevices(pacont.getStoredDevices()) ;

}

}

}

#### RemoteControlGui.java

**package** gui;

**import** guiController.\* ;

**import** appLogic.\* ;

**import** java.io.\* ;

**import** java.util.\* ;

**import** javax.microedition.lcdui.\* ;

**import** javax.microedition.lcdui.game.\* ;

//Displays the remote control interface with a list of all the mapped keys.

//It also displays the action of the pressed key.

**public** **class** RemoteControlGui **extends** Canvas **implements** Runnable {

**private** Main main ;

**private** RemoteControlCont rccont ;

**private** RemoteControlOptions rcoptions ;

**private** Loading loading ;

**private** KeyAction ka ;

**private** Vector keyActions ;

**private** Font font ;

**private** Font font2 ;

**private** Font font3 ;

**private** **int** fheight ;

**private** **int** fheight2 ;

**private** **int** lines ;

**private** **boolean** connected ;

**private** **int** keypressed ;

**private** Thread thread ;

**private** **int**[] keys = {

Constants.*KEY\_POINTER\_UP*,

Constants.*KEY\_POINTER\_DOWN*,

Constants.*KEY\_POINTER\_LEFT*,

Constants.*KEY\_POINTER\_RIGHT*,

Constants.*KEY\_POINTER\_SELECT*,

Constants.*KEY\_LEFTSOFTKEY*,

Constants.*KEY\_RIGHTSOFTKEY*,

*KEY\_NUM1*,

*KEY\_NUM2*,

*KEY\_NUM3*,

*KEY\_NUM4*,

*KEY\_NUM5*,

*KEY\_NUM6*,

*KEY\_NUM7*,

*KEY\_NUM8*,

*KEY\_NUM9*,

*KEY\_NUM0*,

*KEY\_STAR*,

*KEY\_POUND*

} ;

**public** RemoteControlGui() {

rccont = (GuiSingleton.*getGuiSingleton*()).getRemoteControlCont() ;

loading = (GuiSingleton.*getGuiSingleton*()).getLoading() ;

font = Font.*getFont*(Font.*FACE\_SYSTEM*, Font.*STYLE\_PLAIN*, Font.*SIZE\_SMALL*);

font2 = Font.*getFont*(Font.*FACE\_SYSTEM*, Font.*STYLE\_BOLD*, Font.*SIZE\_SMALL*);

fheight = font.getHeight() ;

keyActions = **new** Vector() ;

keypressed = 0 ;

font3 = Font.*getFont*(Font.*FACE\_SYSTEM*, Font.*STYLE\_BOLD*, Font.*SIZE\_LARGE*);

fheight2 = font3.getHeight() ;

lines = (getHeight() / fheight2) ;

thread = **new** Thread(**this**) ;

setFullScreenMode(**true**) ;

}

**protected** **void** setMain(Main m) {

main = m ;

}

**public** **void** setRemoteControlOptions(RemoteControlOptions rco) {

rcoptions = rco ;

}

**public** **void** setMenu() {

keypressed = 0 ;

}

//Is called if the application is disconnected from the server

**public** **void** disconnect(String event) {

connected = **false** ;

stoppThread() ;

keyActions.removeAllElements() ;

Message m = **new** Message(main, event, Constants.*CLASS\_REMOTECONTROLERROR*) ;

main.showMessageScreen(m) ;

}

//Returns number of lines that can be displayed on the screen

**private** **int** getNumberOfLines() {

**return** (getHeight() / fheight) ;

}

//Opens the key action database

**protected** **void** openRMS() {

loading.setName("Opening key mappings", Constants.*LOADING\_TIMEOUT\_UNDEFINED*, Constants.*CLASS\_REMOTECONTROLGUI*) ;

loading.startCount() ;

main.showLoadingRunning(loading) ;

**if**(keyActions.size() == 0) {

rccont.openRMS(Constants.*CLASS\_REMOTECONTROLGUI*) ;

}

**else** {

connected = **true** ;

loading.stoppCount(Constants.*CLASS\_REMOTECONTROLGUI*) ;

}

repaint() ;

}

//Updates a row in the list with key actions

**public** **void** updateKeyActions(KeyAction ka, **int** pos) {

keyActions.setElementAt(ka, pos) ;

}

//Updates the entire key actions list

**public** **void** setKeyActions(Vector v) {

keyActions = v ;

repaint() ;

}

//Is called when the key actions are saved and/or loaded

**public** **void** objectSaved(String event) {

loading.stoppCount(Constants.*CLASS\_REMOTECONTROLGUI*) ;

**if**(event.equals(Constants.*DEFAULT\_KEYACTIONS\_SAVED*) || event.equals(Constants.*KEYACTIONS\_LOADED*)) {

keyActions = rccont.getKeyActions() ;

connected = **true** ;

repaint() ;

}

**else** {

main.showRemoteControlMenu() ;

}

}

**public** **void** stoppThread() {

**if**(thread.isAlive())

thread.interrupt() ;

}

**public** **void** dismiss() {

keypressed = 0 ;

repaint() ;

}

**public** **void** run() {

**try** {

Thread.*sleep*(2000) ;

dismiss() ;

}

**catch**(InterruptedException ie) {

}

}

**private** **void** setKeyAction(KeyAction k) {

ka = k ;

}

//Sends the keyAction based on the button pressed.

//Number 18 (# on the phone) will always be exit

**protected** **void** keyPressed(**int** keyCode) {

**if**(keyActions.size() > 0 && connected) {

**if**(keys[18] == keyCode) {

keyActions.removeAllElements() ;

stoppThread() ;

rccont.endRemoteControl() ;

main.showMainGui() ;

}

**else** {

**for**(**int** i = 0; i < (keys.length - 1); i++) {

**int** key = keys[i] ;

**if**(key == keyCode) {

KeyAction tmp = (KeyAction)keyActions.elementAt(i) ;

**if**((tmp.getKeyActionType()).equals(Constants.*KEY\_ACTION\_TYPE\_PRESS\_RELEASE*) && (((tmp.getKey()).equals(KeyConstants.*KEY\_UP*)) || ((tmp.getKey()).equals(KeyConstants.*KEY\_DOWN*)) || ((tmp.getKey()).equals(KeyConstants.*KEY\_LEFT*)) || ((tmp.getKey()).equals(KeyConstants.*KEY\_RIGHT*)))) {

tmp.setKeyAction(KeyConstants.*KEY\_PRESS*) ;

**if**(rccont.setKeyAction(tmp)) {

tmp.removeKeyAt(1) ;

setKeyAction(tmp) ;

keypressed = 1 ;

repaint() ;

}

}

**else** {

**if**(rccont.setKeyAction(tmp)) {

setKeyAction(tmp) ;

keypressed = 1 ;

repaint() ;

}

}

**break** ;

}

}

}

}

}

**public** **void** keyReleased(**int** keyCode) {

**for**(**int** i = 0; i < (keys.length - 1); i++) {

**int** key = keys[i] ;

**if**(key == keyCode) {

KeyAction tmp = (KeyAction)keyActions.elementAt(i) ;

**if**((tmp.getKeyActionType()).equals(Constants.*KEY\_ACTION\_TYPE\_PRESS\_RELEASE*) && (((tmp.getKey()).equals(KeyConstants.*KEY\_UP*)) || ((tmp.getKey()).equals(KeyConstants.*KEY\_DOWN*)) || ((tmp.getKey()).equals(KeyConstants.*KEY\_LEFT*)) ||

((tmp.getKey()).equals(KeyConstants.*KEY\_RIGHT*)))) {

tmp.setKeyAction(KeyConstants.*KEY\_RELEASE*) ;

**if**(rccont.setKeyAction(tmp)) {

tmp.removeKeyAt(1) ;

setKeyAction(tmp) ;

keypressed = 1 ;

repaint() ;

}

}

**break** ;

}

}

}

**protected** **void** paint(Graphics g) {

**if**(keypressed == 0) {

paintList(g) ;

}

**else** **if**(keypressed == 1) {

paintKeyPressed(g) ;

}

}

**private** **void** paintKeyPressed(Graphics g) {

g.setFont(font3) ;

g.setColor(Constants.*KEYPRESSEDSCREEN\_BACKGROUND*) ;

g.fillRect(0, 0, getWidth(), getHeight()) ;

g.setColor(Constants.*KEYPRESSED\_TEXT\_COLOUR*) ;

Vector key = ka.getKeys() ;

**if**(key.size() > 0) {

**int** size = 0 ;

**if**(key.size() > lines) {

size = lines ;

}

**else** {

size = key.size() ;

}

**for**(**int** i = 0; i < key.size() && i <= lines; i++) {

g.drawString((String)key.elementAt(i), (getWidth() / 2), (((getHeight() / 2) - ((size \* (fheight2 + 1)) / 2))) + (fheight \* i), Graphics.*HCENTER* | Graphics.*BASELINE*) ;

}

}

**if**(thread.isAlive())

thread.interrupt() ;

thread = **new** Thread(**this**) ;

thread.start() ;

}

**private** **void** paintList(Graphics g) {

g.setFont(font) ;

g.setColor(Constants.*BACKGROUND\_COLOUR*) ;

g.fillRect(0, 0, getWidth(), getHeight()) ;

g.setColor(Constants.*TEXT\_COLOUR*) ;

**int** indent = (font2.stringWidth("SELECT: ") + 4) ;

**if**(keyActions.size() > 0) {

**try** {

Image im = Image.*createImage*(Constants.*PICTURE\_ARROW*) ;

**for**(**int** i = 0; i < 4; i++) {

**switch**(i) {

**case** 0:

g.drawImage(im, 2, 2, Graphics.*TOP* | Graphics.*LEFT*) ;

**break**;

**case** 1:

g.drawRegion(im, 0, 0, im.getWidth(), im.getHeight(), Sprite.*TRANS\_ROT180*,

2, ((fheight + 2) \* i), Graphics.*LEFT* | Graphics.*TOP*);

**break** ;

**case** 2:

g.drawRegion(im, 0, 0, im.getWidth(), im.getHeight(), Sprite.*TRANS\_ROT270*,

2, ((fheight + 2) \* i), Graphics.*LEFT* | Graphics.*TOP*);

**break** ;

**case** 3:

g.drawRegion(im, 0, 0, im.getWidth(), im.getHeight(), Sprite.*TRANS\_ROT90*,

2, ((fheight + 2) \* i), Graphics.*LEFT* | Graphics.*TOP*);

**break** ;

}

KeyAction ka = ((KeyAction)keyActions.elementAt(i)) ;

**if**(ka.getKeys().size() > 0) {

String add = ka.getKey() ;

**if**((ka.getKeys()).size() > 1) {

add = add + "+" ;

}

g.drawString(add, indent, ((fheight + 2) \* i), Graphics.*TOP* | Graphics.*LEFT*) ;

}

}

}

**catch**(IOException io) {

}

**int** counter = 0 ;

**int** width = 0 ;

**for**(counter = 4; counter < getNumberOfLines() && counter < keyActions.size(); counter++) {

KeyAction temp = (KeyAction)keyActions.elementAt(counter) ;

String keyName = temp.getKeyName() ;

String assignedKey = "" ;

g.setFont(font2) ;

g.drawString(keyName + ":", 2, ((fheight + 2) \* counter) + 2, Graphics.*TOP* | Graphics.*LEFT*) ;

**if**((temp.getKeys()).size() > 0) {

assignedKey = temp.getKey() ;

**if**((temp.getKeys()).size() > 1)

assignedKey = assignedKey + "+" ;

g.setFont(font) ;

g.drawString(assignedKey, indent, ((fheight + 2) \* counter) + 2, Graphics.*TOP* | Graphics.*LEFT*) ;

}

**if**(font.stringWidth(assignedKey) > width)

width = font.stringWidth(assignedKey) ;

}

**int** counter2 = 4 ;

**if**(keyActions.size() > getNumberOfLines()) {

**for**(**int** i = (counter - 1); (i - counter) < getNumberOfLines() && i < keyActions.size(); i++) {

KeyAction temp = (KeyAction)keyActions.elementAt(i) ;

String keyName = temp.getKeyName() ;

String assignedKey = "" ;

g.setFont(font2) ;

g.drawString(keyName + ":", (width + indent + 12), ((fheight + 2) \* (i - (counter - 1))), Graphics.*TOP* | Graphics.*LEFT*) ;

**if**((temp.getKeys()).size() > 0) {

assignedKey = temp.getKey() ;

**if**((temp.getKeys()).size() > 1)

assignedKey = assignedKey + "+" ;

g.setFont(font) ;

g.drawString(assignedKey , ((width + indent + 16) + font2.stringWidth(keyName)), ((fheight + 2) \* (i - (counter - 1))), Graphics.*TOP* | Graphics.*LEFT*) ;

}

counter2++ ;

}

}

}

}

}

#### RemoteControlKeyEdit.java

**package** gui;

**import** appLogic.\* ;

**import** java.io.\* ;

**import** java.util.\* ;

**import** javax.microedition.lcdui.\* ;

**import** javax.microedition.lcdui.game.Sprite;

//Displays a KeyAction object and presents the edit options the user has.

**public** **class** RemoteControlKeyEdit **extends** Canvas **implements** CommandListener {

**private** KeyAction key ;

**private** RemoteControlOptions rcoptions ;

**private** Main main ;

**private** RemoteControlPointerSpeed rcpointer ;

**private** Font header ;

**private** Font font ;

**private** Image image ;

**private** **int** fheight ;

**private** **int** hheight ;

**private** **int** wcenter ;

**private** **int** menu ;

**private** **int** position ;

**private** **int** count ;

**private** **int** top ;

**private** **int** size ;

**private** Vector menutext ;

**private** **int** menutextheight ;

**private** **int** number ;

**private** **int** max ;

**private** Command addCommand = **new** Command("Add", Command.*SCREEN*, 1) ;

**private** Command removeCommand = **new** Command("Remove", Command.*SCREEN*, 1) ;

**private** Command selectCommand = **new** Command("Select", Command.*SCREEN*, 1) ;

**private** Command saveCommand = **new** Command("Save", Command.*SCREEN*, 2) ;

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

**public** RemoteControlKeyEdit() {

rcpointer =(GuiSingleton.*getGuiSingleton*()).getRemoteControlPointerSpeed() ;

header = Font.*getFont*(Font.*FACE\_SYSTEM*, Font.*STYLE\_PLAIN*, Font.*SIZE\_LARGE*);

font = Font.*getFont*(Font.*FACE\_SYSTEM*, Font.*STYLE\_PLAIN*, Font.*SIZE\_SMALL*);

**try** {

image = Image.*createImage*(Constants.*PICTURE\_ARROW*) ;

}

**catch**(IOException io) {

}

hheight = header.getHeight() ;

fheight = font.getHeight() ;

wcenter = getWidth() / 2 ;

menu = 0 ;

position = 0 ;

count = 0 ;

top = 0 ;

size = 0 ;

number = 0 ;

max = 0 ;

menutext = **new** Vector() ;

menutext.addElement("Add key/pointer speed") ;

menutext.addElement("Remove key action") ;

menutext.addElement("Set action type") ;

menutextheight = (fheight + 1) \* (menutext.size()) ;

addCommand(backCommand) ;

setCommandListener(**this**) ;

}

**protected** **void** setMain(Main m) {

main = m ;

}

**public** **void** setKeyAction(KeyAction ka) {

key = ka ;

}

**public** **void** setRemoteControlOptions(RemoteControlOptions rco) {

rcoptions = rco ;

}

//Calculates the number of lines that can be shown on the screen

**private** **int** getNumberOfLines() {

**int** tmp = (getHeight() - (top + 4)) / (fheight + 1) ;

**if**(menu == 1) {

**if**((KeyConstants.*KEY\_VALUES*).length < tmp) {

**return** (KeyConstants.*KEY\_VALUES*).length ;

}

**else** {

**return** tmp ;

}

}

**else** **if**(menu == 2) {

**if**((key.getKeys()).size() < tmp) {

**return** (key.getKeys()).size() ;

}

**else** {

**return** tmp ;

}

}

**else**

**return** tmp ;

}

//Creates the screen based on the menu variable

//menu 0: menu

//menu 1: add key action

//menu 2: remove key action

//menu 3: set key action type

**protected** **void** paint(Graphics g) {

top = hheight + 5 ;

g.setColor(Constants.*BACKGROUND\_COLOUR*) ;

g.fillRect(0, 0, getWidth(), getHeight()) ;

g.setColor(Constants.*TEXT\_COLOUR*) ;

g.setFont(header) ;

**int** width2 = (header.stringWidth(key.getKeyName())) ;

**int** width = wcenter - ((font.stringWidth(key.getKeyActionType()) + width2) / 2) ;

g.drawString(key.getKeyName(), width, 2, Graphics.*LEFT* | Graphics.*TOP*) ;

g.drawLine(width, (hheight + 2), width + width2, (hheight + 2)) ;

g.setFont(font) ;

g.drawString("(" + key.getKeyActionType() + ")", width + width2 + 3, (hheight - fheight) + 2, Graphics.*LEFT* | Graphics.*TOP*) ;

**if**(menu == 0) {

paintMenu(g) ;

}

**else** **if**(menu == 1) {

paintKeyEditAddKey(g) ;

}

**else** **if**(menu == 2) {

paintKeyEditRemoveKey(g) ;

}

**else** **if**(menu == 3) {

paintSetActionType(g) ;

}

}

**private** **void** paintMenu(Graphics g) {

removeCommand(addCommand) ;

removeCommand(removeCommand) ;

addCommand(selectCommand) ;

addCommand(saveCommand) ;

addCommand(backCommand) ;

**int** height = 0 ;

**int** totalheight = top + ((fheight + 1) \* ((key.getKeys()).size() - 1)) ;

**int** columns = totalheight / ((getHeight() - top) - menutextheight) ;

**int** lines = (getHeight() - (menutextheight + top)) / (fheight + 1) ;

**int** indent = 0 ;

Vector keys = **new** Vector() ;

keys = key.getKeys() ;

**if**(totalheight >= menutextheight) {

**for**(**int** a = 0; a < keys.size(); a++) {

**if**(indent < (font.stringWidth((String)keys.elementAt(a)))) {

indent = (font.stringWidth((String)keys.elementAt(a))) ;

}

}

}

indent += 1 ;

**if**((key.getKeys()).size() > 0) {

**for**(**int** c = 0; c <= columns; c++) {

**for**(**int** i = (lines \* c); i < (lines \* (c + 1)) && i < (key.getKeys()).size(); i++) {

height = top + ((fheight + 1) \* (i - (lines \* c))) ;

g.drawString((String)keys.elementAt(i), ((getWidth() / ((columns + 1) + 1)) + (indent \* c)), height, Graphics.*LEFT* | Graphics.*TOP*) ;

}

}

}

**for**(**int** i2 = 0; i2 < menutext.size(); i2++) {

**if**(position == i2) {

g.setColor(Constants.*MENUITEM\_BOX\_COLOUR*) ;

g.fillRect(0, getHeight() - (menutextheight) + ((fheight + 1) \* i2), getWidth() + 5, fheight) ;

g.setColor(Constants.*TEXT\_COLOUR\_SELECTED*) ;

g.drawString((String)menutext.elementAt(i2), wcenter, getHeight() - (menutextheight) + ((fheight + 1) \* i2), Graphics.*HCENTER* | Graphics.*TOP*) ;

}

**else** {

g.setColor(Constants.*TEXT\_COLOUR*) ;

g.drawString((String)menutext.elementAt(i2), wcenter, getHeight() - (menutextheight) + ((fheight + 1) \* i2), Graphics.*HCENTER* | Graphics.*TOP*) ;

}

}

}

**private** **void** paintKeyEditAddKey(Graphics g) {

removeCommand(saveCommand) ;

removeCommand(removeCommand) ;

removeCommand(selectCommand) ;

addCommand(addCommand) ;

size = getNumberOfLines() ;

**if**(size > 0) {

**if**(position < ((KeyConstants.*KEY\_VALUES*).length - 1) && position == ((size - 1) + count) &&

((KeyConstants.*KEY\_VALUES*).length + 1) > getNumberOfLines()) {

count++ ;

}

**else** **if**(count > 0 && position == count && ((KeyConstants.*KEY\_VALUES*).length + 1) > getNumberOfLines()) {

count-- ;

}

**int** records = 0 ;

**for**(**int** i = count; i < (size + count); i++) {

**if**(position == i) {

g.setColor(Constants.*MENUITEM\_BOX\_COLOUR*) ;

g.fillRect(0, (top + 4) + ((fheight + 1) \* (i - count)), getWidth() + 5, fheight) ;

g.setColor(Constants.*TEXT\_COLOUR\_SELECTED*) ;

}

**else** **if**(g.getColor() == Constants.*TEXT\_COLOUR\_SELECTED* || g.getColor() == Constants.*BACKGROUND\_COLOUR*){

g.setColor(Constants.*TEXT\_COLOUR*) ;

}

g.drawString((KeyConstants.*KEY\_VALUES*)[i], wcenter, (top + 4) + ((fheight + 1) \* (i - count)), Graphics.*HCENTER* | Graphics.*TOP*) ;

records = i ;

}

**if**(records < ((KeyConstants.*KEY\_VALUES*).length - 1) && ((KeyConstants.*KEY\_VALUES*).length > getNumberOfLines()) && position < ((KeyConstants.*KEY\_VALUES*).length - 2)) {

g.drawRegion(image, 0, 0, image.getWidth(), image.getHeight(), Sprite.*TRANS\_ROT180*,

(getWidth() - image.getWidth()) - 2, (top + 4) + ((fheight + 1) \* (records - count)), Graphics.*LEFT* | Graphics.*TOP*);

}

**if**(records > (getNumberOfLines() - 1)) {

g.drawImage(image, (getWidth() - image.getWidth()) - 2, (top + 4), Graphics.*LEFT* | Graphics.*TOP*) ;

}

}

}

**private** **void** paintKeyEditRemoveKey(Graphics g) {

removeCommand(saveCommand) ;

removeCommand(addCommand) ;

removeCommand(selectCommand) ;

addCommand(removeCommand) ;

size = getNumberOfLines() ;

Vector keyvalues = key.getKeys() ;

**if**(position < (keyvalues.size() - 1) && position == ((size - 1) + count) &&

(keyvalues.size() + 1) > getNumberOfLines()) {

count++ ;

}

**else** **if**(count > 0 && position == count && (keyvalues.size() + 1) > getNumberOfLines()) {

count-- ;

}

**int** records = 0 ;

**for**(**int** i = count; i < (size + count); i++) {

**if**(position == i) {

g.setColor(Constants.*MENUITEM\_BOX\_COLOUR*) ;

g.fillRect(0, (top + 4) + ((fheight + 1) \* (i - count)), getWidth() + 5, fheight) ;

g.setColor(Constants.*TEXT\_COLOUR\_SELECTED*) ;

}

**else** {

g.setColor(Constants.*TEXT\_COLOUR*) ;

}

g.drawString((String)keyvalues.elementAt(i), wcenter, (top + 4) + ((fheight + 1) \* (i - count)), Graphics.*HCENTER* | Graphics.*TOP*) ;

records = i ;

}

**if**(records < (keyvalues.size() - 1) && (keyvalues.size() > getNumberOfLines()) && position < (keyvalues.size()- 2)) {

g.drawRegion(image, 0, 0, image.getWidth(), image.getHeight(), Sprite.*TRANS\_ROT180*,

(getWidth() - image.getWidth()) - 2, (top + 4) + ((fheight + 1) \* (records - count)), Graphics.*LEFT* | Graphics.*TOP*);

}

**if**(records > (getNumberOfLines() - 1)) {

g.drawImage(image, (getWidth() - image.getWidth()) - 2, (top + 4), Graphics.*LEFT* | Graphics.*TOP*) ;

}

}

**private** **void** paintSetActionType(Graphics g) {

removeCommand(addCommand) ;

removeCommand(removeCommand) ;

removeCommand(selectCommand) ;

**for**(**int** i = number; i < max; i++) {

**if**(position == i) {

g.setColor(Constants.*MENUITEM\_BOX\_COLOUR*) ;

g.fillRect(0, (top + 4) + ((fheight + 1) \* (i - number)), getWidth() + 5, fheight) ;

g.setColor(Constants.*TEXT\_COLOUR\_SELECTED*) ;

}

**else** **if**(g.getColor() == Constants.*TEXT\_COLOUR\_SELECTED* || g.getColor() == Constants.*BACKGROUND\_COLOUR*) {

g.setColor(Constants.*TEXT\_COLOUR*) ;

}

g.drawString((Constants.*KEY\_ACTION\_TYPES*)[i], wcenter, (top + 4) + ((fheight + 1) \* (i - number)), Graphics.*HCENTER* | Graphics.*TOP*) ;

}

}

**private** **void** commandControl() {

**if**(position == 0 && menu == 0) {

**if**((key.getKeys()).size() > 0 && ((key.getKey()).equals(KeyConstants.*KEY\_UP*) || (key.getKey()).equals(KeyConstants.*KEY\_DOWN*)

|| (key.getKey()).equals(KeyConstants.*KEY\_LEFT*) || (key.getKey()).equals(KeyConstants.*KEY\_RIGHT*))) {

rcpointer.setCurrentAmount(key.getValue()) ;

main.showRemoteControlPointerSpeed(rcpointer) ;

}

**else** {

position = 0 ;

menu = 1 ;

repaint() ;

}

}

**else** **if**(position == 1 && menu == 0) {

position = 0 ;

menu = 2 ;

repaint() ;

}

**else** **if**(position == 2 && menu == 0) {

**if**((key.getKeys()).size() == 1 && !(key.getKey().equals(KeyConstants.*KEY\_LEFTMOUSEBUTTON*)) && !(key.getKey().equals(KeyConstants.*KEY\_UP*)) && !(key.getKey().equals(KeyConstants.*KEY\_DOWN*)) &&

!(key.getKey().equals(KeyConstants.*KEY\_LEFT*)) && !(key.getKey().equals(KeyConstants.*KEY\_RIGHT*))) {

number = 2 ;

max = 3 ;

}

**else** **if**((key.getKeys()).size() > 1) {

number = 0 ;

max = 2 ;

}

**else** **if**((key.getKeys()).size() == 1 && ((key.getKey().equals(KeyConstants.*KEY\_LEFTMOUSEBUTTON*)) || (key.getKey().equals(KeyConstants.*KEY\_UP*)) ||

(key.getKey().equals(KeyConstants.*KEY\_DOWN*)) || (key.getKey().equals(KeyConstants.*KEY\_LEFT*))

|| (key.getKey().equals(KeyConstants.*KEY\_RIGHT*)))) {

number = 2 ;

max = 4 ;

}

position = number ;

menu = 3 ;

repaint() ;

}

}

**protected** **void** keyPressed(**int** keyCode) {

**switch**(keyCode) {

//UP

**case** -1:

**case** 50:

**if**(menu == 0) {

**if**(position > 0)

position-- ;

**else** {

position = (menutext.size() - 1) ;

}

}

**else** **if**(menu == 1) {

**if**(position > 0)

position-- ;

**else** {

position = ((KeyConstants.*KEY\_VALUES*).length - 1) ;

count = ((KeyConstants.*KEY\_VALUES*).length - size) ;

}

}

**else** **if**(menu == 2) {

**if**(position > 0)

position-- ;

**else** {

position = ((key.getKeys()).size() - 1) ;

count = ((key.getKeys()).size() - size) ;

}

}

**else** **if**(menu == 3) {

**if**(position > number)

position-- ;

**else** {

position = max - 1 ;

}

}

**break** ;

//DOWN

**case** -2:

**case** 56:

**if**(menu == 0) {

**if**(position < (menutext.size() - 1))

position++ ;

**else** {

position = 0 ;

}

}

**else** **if**(menu == 1) {

**if**(position < (KeyConstants.*KEY\_VALUES*).length - 1) {

position++ ;

}

**else** {

position = 0 ;

count = 0 ;

}

}

**else** **if**(menu == 2) {

**if**(position < ((key.getKeys()).size() - 1)) {

position++ ;

}

**else** {

position = 0 ;

count = 0 ;

}

}

**else** **if**(menu == 3) {

**if**(position < (max - 1)) {

position++ ;

}

**else** {

position = number ;

}

}

**break** ;

//SELECT

**case** 53:

**case** -5:

**if**(menu == 0) {

commandControl() ;

}

**else** **if**(menu == 1) {

addKey() ;

}

**else** **if**(menu == 2) {

removeKey() ;

}

**else** **if**(menu == 3) {

saveActionType() ;

}

**break** ;

}

repaint() ;

}

//Adds a key to the key action. The mouse direction key types can only have 1 action.

**private** **void** addKey() {

**if**((key.getKeys()).size() == 0 && (((KeyConstants.*KEY\_VALUES*)[position]).equals(KeyConstants.*KEY\_UP*) || ((KeyConstants.*KEY\_VALUES*)[position]).equals(KeyConstants.*KEY\_DOWN*) || ((KeyConstants.*KEY\_VALUES*)[position]).equals(KeyConstants.*KEY\_LEFT*) || ((KeyConstants.*KEY\_VALUES*)[position]).equals(KeyConstants.*KEY\_RIGHT*))) {

**if**((key.getValue()) == -1)

key.setValue(Constants.*KEY\_INITIAL\_MOUSE\_VALUE*) ;

key.setKeyAction((KeyConstants.*KEY\_VALUES*)[position]) ;

count = 0 ;

position = 0 ;

menu = 0 ;

repaint() ;

}

**else** **if**((key.getKeys()).size() != 0 && (((KeyConstants.*KEY\_VALUES*)[position]).equals(KeyConstants.*KEY\_UP*) || ((KeyConstants.*KEY\_VALUES*)[position]).equals(KeyConstants.*KEY\_DOWN*) || ((KeyConstants.*KEY\_VALUES*)[position]).equals(KeyConstants.*KEY\_LEFT*) || ((KeyConstants.*KEY\_VALUES*)[position]).equals(KeyConstants.*KEY\_RIGHT*))) {

count = 0 ;

position = 0 ;

menu = 0 ;

repaint() ;

}

**else** {

key.setKeyAction((KeyConstants.*KEY\_VALUES*)[position]) ;

**if**((key.getKeys()).size() > 1) {

key.setKeyActionType(Constants.*KEY\_ACTION\_TYPE\_PROGRESSION*) ;

}

count = 0 ;

position = 0 ;

menu = 0 ;

repaint() ;

}

}

//Removes a key action

**private** **void** removeKey() {

key.removeKeyAt(position) ;

**if**((key.getKeys()).size() < 2) {

key.setKeyActionType(Constants.*KEY\_ACTION\_TYPE\_SINGLE*) ;

}

**if**((key.getKeys()).size() == 0) {

key.setValue(-1) ;

}

count = 0 ;

position = 0 ;

menu = 0 ;

repaint() ;

}

//Saves the key action.

**private** **void** saveKey() {

**if**(menu == 1) {

main.showRemoteControlOptions() ;

}

**else** **if**(menu == 3) {

position = 0 ;

count = 0 ;

menu = 0 ;

repaint() ;

}

rcoptions.updateKeyActions(key) ;

}

//Saves the selected key action type

**private** **void** saveActionType() {

**if**((key.getKeys()).size() > 1 && position >= 0) {

key.setKeyActionType((Constants.*KEY\_ACTION\_TYPES*)[position]) ;

saveKey() ;

}

**else** **if**((key.getKeys()).size() == 1 && position == 2) {

key.setKeyActionType((Constants.*KEY\_ACTION\_TYPES*)[position]) ;

saveKey() ;

}

**else** **if**((key.getKeys()).size() == 1 && position == 3 && (key.getKey().equals(KeyConstants.*KEY\_LEFTMOUSEBUTTON*) ||

key.getKey().equals(KeyConstants.*KEY\_UP*) || key.getKey().equals(KeyConstants.*KEY\_DOWN*) ||

key.getKey().equals(KeyConstants.*KEY\_LEFT*) || key.getKey().equals(KeyConstants.*KEY\_RIGHT*))) { key.setKeyActionType((Constants.*KEY\_ACTION\_TYPES*)[position]) ;

saveKey() ;

}

**else** {

rcoptions.objectSaved(Constants.*ERROR\_KEYACTION\_WRONG\_INPUT*) ;

position = 0 ;

count = 0 ;

menu = 0 ;

repaint() ;

}

}

//If the action is a mouse directional type, the pointer speed can be changed.

**public** **void** setPointerSpeed(**int** amount) {

**if**(amount > -1) {

key.setValue(amount) ;

saveKey() ;

}

**else** {

main.showRemoteControlKeyEdit(**this**) ;

}

}

**public** **void** commandAction(Command c, Displayable d) {

**if**(menu == 0) {

**if**(c == saveCommand) {

saveKey() ;

}

**else** **if**(c == selectCommand) {

commandControl() ;

}

**else** **if**(c == backCommand) {

position = 0 ;

menu = 0 ;

count = 0 ;

main.showRemoteControlOptions() ;

}

}

**else** **if**(menu == 1) {

**if**(c == addCommand) {

addKey() ;

}

**else** **if**(c == backCommand) {

menu = 0 ;

position = 0 ;

count = 0 ;

repaint() ;

}

}

**else** **if**(menu == 2) {

**if**(c == removeCommand) {

removeKey() ;

}

**else** **if**(c == backCommand) {

menu = 0 ;

position = 0 ;

count = 0 ;

repaint() ;

}

}

**else** **if**(menu == 3) {

**if**(c == saveCommand) {

removeCommand(saveCommand) ;

saveActionType() ;

}

**else** **if**(c == backCommand) {

menu = 0 ;

position = 0 ;

count = 0 ;

removeCommand(saveCommand) ;

repaint() ;

}

}

}

}

#### RemoteControlMenu.java

**package** gui;

**import** appLogic.\* ;

**import** guiController.\* ;

**import** javax.microedition.lcdui.\* ;

**import** javax.microedition.lcdui.game.Sprite;

**import** java.io.\* ;

**import** java.util.\* ;

//Displays the list of Bluetooth devices that are stored.

//The user can select a device and connect to the server application.

**public** **class** RemoteControlMenu **extends** Canvas **implements** CommandListener {

**private** Main main ;

**private** Loading loading ;

**private** RemoteControlCont rccont ;

**private** DeviceScreen ds ;

**private** Image image ;

**private** Vector devicelist ;

**private** Font font ;

**private** **int** fheight ;

**private** **int** wcenter ;

**private** **int** hcenter ;

**private** **int** position ;

**private** **int** size ;

**private** **int** count ;

**private** Command connectCommand = **new** Command("Connect", Command.*OK*, 1) ;

**private** Command infoCommand = **new** Command("Info", Command.*SCREEN*, 2) ;

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

**public** RemoteControlMenu() {

rccont = (GuiSingleton.*getGuiSingleton*()).getRemoteControlCont() ;

loading = (GuiSingleton.*getGuiSingleton*()).getLoading() ;

ds = (GuiSingleton.*getGuiSingleton*()).getDeviceScreen() ;

font = Font.*getFont*(Font.*FACE\_SYSTEM*, Font.*STYLE\_PLAIN*, Font.*SIZE\_SMALL*) ;

fheight = font.getHeight() ;

wcenter = getWidth() / 2 ;

hcenter = getHeight() / 2 ;

position = 0 ;

count = 0 ;

size = 0 ;

**try** {

image = Image.*createImage*(Constants.*PICTURE\_ARROW*) ;

}

**catch**(IOException io) {

}

devicelist = **new** Vector() ;

addCommand(connectCommand) ;

addCommand(infoCommand) ;

addCommand(backCommand) ;

setCommandListener(**this**) ;

}

**protected** **void** setMain(Main m) {

main = m ;

}

//If the service is found.

//The status variable will be true if the connection if accepted.

**public** **void** connectedService(**boolean** status) {

**if**(status)

loading.stoppCount(Constants.*CLASS\_REMOTECONTROLMENU*) ;

**else** {

loading.stoppCount(Constants.*CLASS\_REMOTECONTROLMENULOADING*) ;

}

}

//If the service is not found

**public** **void** serviceError(String event) {

loading.stoppCount(Constants.*CLASS\_MESSAGE*) ;

Message m = **new** Message(main, event, Constants.*CLASS\_REMOTECONTROLMENU*) ;

main.showMessageScreen(m) ;

}

**public** **void** showScreen() {

loading.setName("Opening device list", Constants.*LOADING\_TIMEOUT\_UNDEFINED*, Constants.*CLASS\_REMOTECONTROLMENULOADING*) ;

loading.startCount() ;

main.showLoadingRunning(loading) ;

devicelist = rccont.getStoredDevices() ;

loading.stoppCount(Constants.*CLASS\_REMOTECONTROLMENULOADING*) ;

}

**private** **int** getPlace(**int** number) {

**int** temp = 0 ;

**int** listheight = (fheight + 1) \* (devicelist.size() + 1) ;

**if**(listheight > getHeight()) {

**return** (fheight + 1) \* number ;

}

**else** {

temp = (fheight + 1) \* (devicelist.size() + 1) ;

temp = temp / 2 ;

temp = hcenter - temp ;

**return** temp +((fheight + 1) \* number) ;

}

}

**private** **int** getNumberOfLines() {

**return** (getHeight() / (fheight + 1)) - 1 ;

}

**protected** **void** paint(Graphics g) {

g.setColor(Constants.*BACKGROUND\_COLOUR*) ;

g.fillRect(0, 0, getWidth(), getHeight()) ;

g.setFont(font) ;

**if**(devicelist.size() > 0) {

**if**((devicelist.size() + 1) > getNumberOfLines()) {

size = getNumberOfLines() ;

}

**else** {

size = devicelist.size() ;

}

**if**(position < (devicelist.size() - 1) && position == ((size - 1) + count) &&

(devicelist.size() + 1) > getNumberOfLines()) {

count++ ;

}

**else** **if**(count > 0 && position == count && (devicelist.size() + 1) > getNumberOfLines()) {

count-- ;

}

**int** records = 0 ;

**for**(**int** i = count; i < (size + count); i++) {

**if**(position == i) {

g.setColor(Constants.*MENUITEM\_BOX\_COLOUR*) ;

g.fillRect(0, getPlace(i - count) + 5, getWidth(), fheight) ;

g.setColor(Constants.*TEXT\_COLOUR\_SELECTED*) ;

}

**else** **if** (g.getColor() == Constants.*TEXT\_COLOUR\_SELECTED* || g.getColor() == Constants.*BACKGROUND\_COLOUR*){

g.setColor(Constants.*TEXT\_COLOUR*) ;

}

BTDevice temp = (BTDevice)devicelist.elementAt(i) ;

String name = temp.getName() ;

**if**(name.equals("")) {

name = temp.getAddress() ;

}

**else** **if**(font.stringWidth(name + ": " + temp.getAddress()) < (getWidth() - 15)){

name = name + ": " + temp.getAddress() ;

}

g.drawString(name, wcenter, getPlace((i - count) + 1), Graphics.*HCENTER* | Graphics.*BASELINE*) ;

records = i ;

}

**if**(records < (devicelist.size() - 1) && (devicelist.size() > getNumberOfLines()) && position < (devicelist.size()- 2)) {

g.drawRegion(image, 0, 0, image.getWidth(), image.getHeight(), Sprite.*TRANS\_ROT180*,

(getWidth() - image.getWidth()) - 2, getPlace(((records - count) + 1)), Graphics.*LEFT* | Graphics.*TOP*);

}

**if**(records > (getNumberOfLines() - 1)) {

g.drawImage(image, (getWidth() - image.getWidth()) - 2, getPlace(0), Graphics.*LEFT* | Graphics.*TOP*) ;

}

}

}

**protected** **void** keyPressed(**int** keyCode) {

**switch**(keyCode) {

//UP

**case** -1:

**case** 50:

**if**(position > 0) {

position-- ;

}

**else** {

count = (devicelist.size() - size) ;

position = (devicelist.size() - 1) ;

}

**break** ;

//DOWN

**case** -2:

**case** 56:

**if**(position < (devicelist.size() - 1)) {

position++ ;

}

**else** {

position = 0 ;

count = 0 ;

}

**break** ;

//SELECT

**case** 53:

**case** -5:

commandControl() ;

**break** ;

}

repaint() ;

}

**private** **void** commandControl() {

**if**(position >= 0 && devicelist.size() > 0) {

loading.setName("Searching for service", Constants.*LOADING\_TIMEOUT\_UNDEFINED*, Constants.*CLASS\_REMOTECONTROLMENU*) ;

loading.startCount() ;

main.showLoadingRunning(loading) ;

rccont.searchRemoteControlService((BTDevice)devicelist.elementAt(position)) ;

}

}

**public** **void** commandAction(Command c, Displayable d) {

**if**(c == backCommand) {

position = 0;

main.showMainGui() ;

}

**else** **if**(c == infoCommand) {

**if**(position >= 0) {

ds.setCallingClass(Constants.*CLASS\_REMOTECONTROLMENU*) ;

ds.setDevice((BTDevice)devicelist.elementAt(position)) ;

main.showDeviceScreen(ds) ;

}

}

**else** **if**(c == connectCommand) {

commandControl() ;

}

}

}

#### RemoteControlOptions.java

**package** gui;

**import** appLogic.\* ;

**import** guiController.\* ;

**import** java.io.\* ;

**import** java.util.\* ;

**import** javax.microedition.lcdui.\* ;

**import** javax.microedition.lcdui.game.Sprite;

//Displays a list of all the available keys to custom map. There are a total of 18 keys that can be mapped.

**public** **class** RemoteControlOptions **extends** Canvas **implements** CommandListener {

**private** Main main ;

**private** RemoteControlKeyEdit rckeyedit ;

**private** RemoteControlCont rccont ;

**private** RemoteControlGui rcgui ;

**private** Loading loading ;

**private** Font font ;

**private** Image image ;

**private** **int** fheight ;

**private** **int** hcenter ;

**private** **int** position ;

**private** **int** selectedposition ;

**private** **int** size ;

**private** **int** count ;

**private** Vector keyActions ;

**public** String callingclass ;

**private** Command editCommand = **new** Command("Edit", Command.*OK*, 1) ;

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

**public** RemoteControlOptions() {

rccont = (GuiSingleton.*getGuiSingleton*()).getRemoteControlCont() ;

rcgui = (GuiSingleton.*getGuiSingleton*()).getRemoteControlGui() ;

loading = (GuiSingleton.*getGuiSingleton*()).getLoading() ;

rckeyedit = (GuiSingleton.*getGuiSingleton*()).getRemoteControlKeyEdit() ;

font = Font.*getFont*(Font.*FACE\_SYSTEM*, Font.*STYLE\_PLAIN*, Font.*SIZE\_SMALL*);

fheight = font.getHeight() ;

**try** {

image = Image.*createImage*(Constants.*PICTURE\_ARROW*) ;

}

**catch**(IOException io) {

}

hcenter = getHeight() / 2 ;

position = 0 ;

selectedposition = 0 ;

size = 0 ;

count = 0 ;

callingclass = "" ;

keyActions = **new** Vector() ;

addCommand(editCommand) ;

addCommand(backCommand) ;

setCommandListener(**this**) ;

}

**protected** **void** setMain(Main m) {

main = m ;

}

**public** **void** setKeyActions(Vector v) {

**if**(v != **null**) {

keyActions = v ;

}

**else** {

openRMS() ;

}

}

**public** **void** setCallingClass(String cc) {

callingclass = cc ;

}

//Opens the key action database

**public** **void** openRMS() {

loading.setName("Opening key mappings", Constants.*LOADING\_TIMEOUT\_UNDEFINED*, callingclass) ;

loading.startCount() ;

main.showLoadingRunning(loading) ;

rccont.openRMS(Constants.*CLASS\_REMOTECONTROLOPTIONS*) ;

repaint() ;

}

//Saves changes to the key actions

**public** **void** updateKeyActions(KeyAction ka) {

keyActions.setElementAt(ka, selectedposition) ;

rccont.saveKeyAction(ka, selectedposition) ;

}

**private** **int** getNumberOfLines() {

**return** (getHeight() / (fheight + 1)) ;

}

**private** **int** getPlace(**int** number) {

**int** temp = 0 ;

**int** listheight = (fheight + 1) \* ((Constants.*KEY\_NAMES*).length - 1) ;

**if**(listheight > getHeight()) {

**return** (fheight + 1) \* number ;

}

**else** {

temp = (fheight + 1) \* ((Constants.*KEY\_NAMES*).length - 1) ;

temp = temp / 2 ;

temp = hcenter - temp ;

**return** temp +((fheight + 1) \* number) ;

}

}

**public** **void** objectSaved(String event) {

Message m = **new** Message(main, event, Constants.*CLASS\_REMOTECONTROLOPTIONS*) ;

main.showMessageScreen(m) ;

}

**public** **void** objectsLoaded(String event) {

**if**(event.equals(Constants.*DEFAULT\_KEYACTIONS\_SAVED*) || event.equals(Constants.*KEYACTIONS\_LOADED*)) {

keyActions = rccont.getKeyActions() ;

loading.stoppCount(Constants.*CLASS\_REMOTECONTROLOPTIONS*) ;

}

**else** {

main.showRemoteControlMenu() ;

}

}

**protected** **void** paint(Graphics g) {

g.setFont(font) ;

g.setColor(Constants.*BACKGROUND\_COLOUR*) ;

g.fillRect(0, 0, getWidth(), getHeight()) ;

**int** indent = 0 ;

**for**(**int** a = 0; a < ((Constants.*KEY\_NAMES*).length - 1); a++) {

**if**(font.stringWidth((Constants.*KEY\_NAMES*)[a] + ": ") > indent) {

indent = font.stringWidth((Constants.*KEY\_NAMES*)[a]) ;

}

}

**if**(((Constants.*KEY\_NAMES*).length - 1) > getNumberOfLines()) {

size = getNumberOfLines() ;

}

**else** {

size = ((Constants.*KEY\_NAMES*).length - 1) ;

}

**if**(position < ((Constants.*KEY\_NAMES*).length - 2) && position == ((size - 1) + count) &&

((Constants.*KEY\_NAMES*).length - 1) > getNumberOfLines()) {

count++ ;

}

**else** **if**(count > 0 && position == count && ((Constants.*KEY\_NAMES*).length - 1) > getNumberOfLines()) {

count-- ;

}

**int** records = 0 ;

**for**(**int** i = count; i < (size + count); i++) {

**if**(position == i) {

g.setColor(Constants.*MENUITEM\_BOX\_COLOUR*) ;

g.fillRect(0, getPlace(i - count), getWidth(), fheight) ;

g.setColor(Constants.*TEXT\_COLOUR\_SELECTED*) ;

}

**else** **if** (g.getColor() == Constants.*TEXT\_COLOUR\_SELECTED* || g.getColor() == Constants.*BACKGROUND\_COLOUR*){

g.setColor(Constants.*TEXT\_COLOUR*) ;

}

g.drawString((Constants.*KEY\_NAMES*)[i] + ": ", 2, getPlace(((i - count))), Graphics.*TOP* | Graphics.*LEFT*) ;

KeyAction ka = ((KeyAction)keyActions.elementAt(i)) ;

**if**(ka.getKeys().size() > 0) {

String add = ka.getKey() ;

**if**((ka.getKeys()).size() > 1)

add = add + "+" ;

g.drawString(add, indent + 7, getPlace(((i - count))), Graphics.*TOP* | Graphics.*LEFT*) ;

}

records = i ;

}

**if**(records < ((Constants.*KEY\_NAMES*).length - 2) && ((Constants.*KEY\_NAMES*).length + 1) > getNumberOfLines() && position < ((Constants.*KEY\_NAMES*).length - 3)) {

g.drawRegion(image, 0, 0, image.getWidth(), image.getHeight(), Sprite.*TRANS\_ROT180*,

(getWidth() - image.getWidth()) - 2, getPlace(records - count), Graphics.*LEFT* | Graphics.*TOP*);

}

**if**(records > (getNumberOfLines() - 1)) {

g.drawImage(image, (getWidth() - image.getWidth()) - 2, getPlace(0), Graphics.*LEFT* | Graphics.*TOP*) ;

}

}

**protected** **void** keyPressed(**int** keyCode) {

**switch**(keyCode) {

//UP

**case** -1:

**case** 50:

**if**(position > 0)

position-- ;

**else** {

count = (((Constants.*KEY\_NAMES*).length - 1) - size) ;

position = ((Constants.*KEY\_NAMES*).length - 2) ;

}

**break** ;

//DOWN

**case** -2:

**case** 56:

**if**(position < ((Constants.*KEY\_NAMES*).length - 2))

position++ ;

**else** {

position = 0 ;

count = 0 ;

}

**break** ;

//SELECT

**case** 53:

**case** -5:

commandControl() ;

**break** ;

}

repaint() ;

}

**private** **void** commandControl() {

selectedposition = position ;

rckeyedit.setKeyAction((KeyAction)keyActions.elementAt(position)) ;

position = 0 ;

count = 0 ;

repaint() ;

main.showRemoteControlKeyEdit(rckeyedit) ;

}

**public** **void** commandAction(Command c, Displayable d) {

**if**(c == editCommand) {

commandControl() ;

}

**else** **if**(c == backCommand) {

position = 0 ;

count = 0 ;

**if**(callingclass.equals(Constants.*CLASS\_REMOTECONTROLGUI*))

main.showRemoteControlGui() ;

**else** **if**(callingclass.equals(Constants.*CLASS\_PROFILEADMIN*))

main.showProfileAdmin() ;

}

}

}

#### RemoteControlPointerSpeed.java

**package** gui;

**import** javax.microedition.lcdui.\* ;

//If the key action selected is a mouse directional type, this class is used to set the pointer (mouse) speed.

**public** **class** RemoteControlPointerSpeed **extends** Form **implements** CommandListener {

**private** RemoteControlKeyEdit rckeyedit ;

**private** TextField pointer ;

**private** Command okCommand = **new** Command("Ok", Command.*OK*, 1) ;

**private** Command clearCommand = **new** Command("Clear", Command.*SCREEN*, 2) ;

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

**public** RemoteControlPointerSpeed() {

**super**("Key options") ;

pointer = **new** TextField("Pointer speed", "", 4, TextField.*NUMERIC*) ;

append(pointer) ;

addCommand(okCommand) ;

addCommand(clearCommand) ;

addCommand(backCommand) ;

setCommandListener(**this**) ;

}

**public** **void** setRemoteControlKeyEdit(RemoteControlKeyEdit rcke) {

rckeyedit = rcke ;

}

**public** **void** setCurrentAmount(**int** amount) {

pointer.setString(Integer.*toString*(amount)) ;

}

**public** **void** commandAction(Command c, Displayable d) {

**if**(c == okCommand) {

String tmp = pointer.getString() ;

**int** amount = Integer.*parseInt*(tmp) ;

rckeyedit.setPointerSpeed(amount) ;

}

**else** **if**(c == backCommand) {

rckeyedit.setPointerSpeed(-1) ;

}

**else** **if**(c == clearCommand) {

pointer.setString("") ;

}

}

}

### A10.1.2 Package: guiController

#### ProfileAdminCont.java

**package** guiController;

**import** gui.\* ;

**import** appLogic.\* ;

**import** java.util.\* ;

//Presents an API to the user interface with the methods available from the domain layer.

**public** **class** ProfileAdminCont {

**private** ProfileAdmin padmin ;

**private** ProfileAdminLogic palogic ;

**public** ProfileAdminCont() {

palogic = (DomainSingleton.*getDomainSingleton*()).getProfileAdminLogic() ;

palogic.setProfileAdminCont(**this**) ;

}

**public** **void** setProfileAdmin(ProfileAdmin pa) {

padmin = pa ;

}

**public** **void** searchDevices(**int** dt) {

palogic.searchDevices(dt) ;

}

**public** **void** updateDevices(BTDevice btd, String saved) {

padmin.updateDevices(btd, saved) ;

}

**public** **void** storeDevice(BTDevice btd) {

palogic.storeDevice(btd) ;

}

**public** **void** deleteDevice(**int** position) {

palogic.deleteDevice(position) ;

}

**public** Vector getStoredDevices() {

**return** palogic.getStoredDevices() ;

}

**public** **void** searchCompleted() {

padmin.searchCompleted() ;

}

**public** **void** openRMS() {

palogic.openRMS() ;

}

**public** **void** closeRMS() {

palogic.closeRMS() ;

}

**public** **void** objectSaved(String event) {

padmin.objectSaved(event) ;

}

**public** ProfileAdminLogic getProfileAdminLogic() {

**return** palogic ;

}

}

#### RemoteControlCont.java

**package** guiController;

**import** gui.\* ;

**import** appLogic.\* ;

**import** java.util.\* ;

//Presents an API to the user interface with the methods available from the domain layer.

**public** **class** RemoteControlCont {

**private** RemoteControlMenu rcmenu ;

**private** **static** ProfileAdminCont *pacont* ;

**private** RemoteControlLogic rclogic ;

**private** RemoteControlGui rcgui ;

**private** RemoteControlOptions rcoptions ;

**private** String callingclass ;

**public** RemoteControlCont() {

*pacont* = (GuiSingleton.*getGuiSingleton*()).getProfileAdminCont() ;

rclogic = (DomainSingleton.*getDomainSingleton*()).getRemoteControlLogic() ;

rclogic.setRemoteControlCont(**this**) ;

}

**public** **void** setRemoteControlMenu(RemoteControlMenu rcm) {

rcmenu = rcm ;

}

**public** **void** setRemoteControlGui(RemoteControlGui rcg) {

rcgui = rcg ;

}

**public** **void** setRemoteControlOptions(RemoteControlOptions rco) {

rcoptions = rco ;

}

**public** **void** openRMS(String cc) {

callingclass = cc ;

rclogic.openRMS() ;

}

**public** **void** closeRMS() {

rclogic.closeRMS() ;

}

**public** **void** closeProfileAdminRMS() {

*pacont*.closeRMS() ;

}

**public** Vector getStoredDevices() {

**return** *pacont*.getStoredDevices() ;

}

**public** **void** searchRemoteControlService(BTDevice btd) {

rclogic.startSearch(btd) ;

}

**public** **void** connectedService(**boolean** status) {

rcmenu.connectedService(status) ;

}

**public** **void** closeConnections() {

rclogic.closeConnections() ;

}

**public** **boolean** setKeyAction(KeyAction ka) {

**return** rclogic.setKeyAction(ka) ;

}

**public** ProfileAdminLogic getProfileAdminLogic() {

**return** *pacont*.getProfileAdminLogic() ;

}

**public** **void** getSavedKeyActions(Vector v) {

rcgui.setKeyActions(v) ;

}

**public** Vector getKeyActions() {

**return** rclogic.getKeyActions() ;

}

**public** **void** objectSaved(String event) {

**if**(callingclass.equals(Constants.*CLASS\_REMOTECONTROLGUI*)) {

rcgui.objectSaved(event) ;

}

**else** **if**(callingclass.equals(Constants.*CLASS\_REMOTECONTROLOPTIONS*)){

rcoptions.objectsLoaded(event) ;

}

}

**public** **void** keyActionSaved(String event) {

rcoptions.objectSaved(event) ;

}

**public** **void** saveKeyAction(KeyAction ka, **int** position) {

rclogic.saveKeyAction(ka, position) ;

}

**public** **void** disconnect(String event) {

rcgui.disconnect(event) ;

}

**public** **void** serviceError(String event) {

rcmenu.serviceError(event) ;

}

**public** **void** endRemoteControl() {

rclogic.endRemoteControl() ;

}

}

### A10.1.3 Package: appLogic

#### BTDevice.java

**package** appLogic ;

**import** interfaces.\* ;

**import** java.io.\* ;

//Stores information about a Bluetooth device

**public** **class** BTDevice **implements** RMSSerial {

**private** String address ;

**private** String name ;

**private** **int** majorClass ;

**private** **int** minorClass ;

**private** **int** serviceClass ;

**public** BTDevice() {

address = **null** ;

name = **null** ;

majorClass = 0 ;

minorClass = 0 ;

serviceClass = 0 ;

}

**public** BTDevice(String a) {

address = a ;

}

//Get/Set methods

**public** **void** setAddress(String a) {

address = a ;

}

**public** **void** setName(String a) {

name = a ;

}

**public** **void** setMajorClass(**int** a) {

majorClass = a ;

}

**public** **void** setMinorClass(**int** a) {

minorClass = a ;

}

**public** **void** setServiceClass(**int** a) {

serviceClass = a ;

}

**public** String getAddress() {

**return** address ;

}

**public** String getName() {

**return** name ;

}

**public** **int** getMajorClass() {

**return** majorClass ;

}

**public** **int** getMinorClass() {

**return** minorClass ;

}

**public** **int** getServiceClass() {

**return** serviceClass ;

}

//Serializing methods, used to store objects in rms databases.

//The idea is taken from http://java.sun.com/developer/J2METechTips/2002/tt0226.html

//however the code is updated.

**public** **byte**[] serialize() **throws** IOException {

ByteArrayOutputStream bout = **new** ByteArrayOutputStream() ;

DataOutputStream dout = **new** DataOutputStream(bout) ;

dout.writeUTF(address) ;

dout.writeUTF(name) ;

dout.writeInt(majorClass) ;

dout.writeInt(minorClass) ;

dout.writeInt(serviceClass) ;

dout.flush() ;

**byte**[] data = bout.toByteArray() ;

bout.close() ;

dout.close() ;

**return** data ;

}

**public** **void** restore(**byte**[] data) **throws** IOException {

ByteArrayInputStream bin = **new** ByteArrayInputStream(data) ;

DataInputStream din = **new** DataInputStream(bin) ;

address = din.readUTF() ;

name = din.readUTF() ;

majorClass = din.readInt() ;

minorClass = din.readInt() ;

serviceClass = din.readInt() ;

din.close() ;

bin.close() ;

}

}

#### Constants.java

**package** appLogic;

//Contains all regular system constants. For example background colour and text colour.

**public** **class** Constants **extends** NetConstants {

//Platform type

**public** **static** String *PLATFORM* = "Java" ;

//Database operations

**public** **static** String *DBWRITE* = "DBWrite" ;

**public** **static** String *DBUPDATE* = "DBUpdate" ;

**public** **static** String *DBDELETE* = "DBDelete" ;

//Database names

**public** **static** String *DBBTDEVICE* = "DBBTDevice" ;

**public** **static** String *DBKEYACTION* = "DBKeyAction" ;

//Database messages

**public** **static** String *KEYACTION\_SAVED* = "Key action saved" ;

**public** **static** String *BTDEVICE\_SAVED* = "Device saved" ;

**public** **static** String *BTDEVICE\_DELETED* = "Device deleted" ;

**public** **static** String *BTDEVICE\_NEW* = "New BTDevice" ;

**public** **static** String *DEFAULT\_KEYACTIONS\_SAVED* = "Default key actions saved" ;

**public** **static** String *KEYACTIONS\_LOADED* = "Key actions loaded" ;

//Errors

**public** **static** String *ERROR\_IOEXCEPTION\_SAVE* = "IOException, problem serializing or restoring object" ;

**public** **static** String *ERROR\_DUPLICATE* = "Device already saved, duplicate object" ;

**public** **static** String *ERROR\_RECORDSTORENOTOPENEXCEPTION* = "RecordStoreNotOpenException, database not opened" ;

**public** **static** String *ERROR\_RECORDSTOREFULLEXCEPTION* = "RecordStoreFullException, no more available storage" ;

**public** **static** String *ERROR\_RECORDSTOREEXCEPTION* = "RecordStoreException" ;

**public** **static** String *ERROR\_BLUETOOTH\_OFF* = "Please turn on Bluetooth and restart the application" ;

**public** **static** String *ERROR\_KEYACTION\_WRONG\_INPUT* = "Invalid key action type" ;

**public** **static** String *ERROR\_REMOTECONTROL\_DISCONNETED* = "Disconneted from server" ;

**public** **static** String *ERROR\_SERVICE\_NOT\_FOUND* = "Service not found" ;

//Loading variables

**public** **static** String *LOADING\_TIMEOUT\_UNDEFINED* = "Undefined" ;

**public** **static** String *LOADING\_TIMEOUT\_DEFINED* = "Defined" ;

//Classes

**public** **static** String *CLASS\_BTDEVICE* = "BTDevice" ;

**public** **static** String *CLASS\_MAIN* = "Main" ;

**public** **static** String *CLASS\_PROFILEADMIN* = "ProfileAdmin" ;

**public** **static** String *CLASS\_REMOTECONTROLMENULOADING* = "RemoteControlMenuLoading" ;

**public** **static** String *CLASS\_REMOTECONTROLMENU* = "RemoteControlMenu" ;

**public** **static** String *CLASS\_INITIAL\_KEYACTION* = "InitialKeyAction" ;

**public** **static** String *CLASS\_KEYACTION* = "KA" ;

**public** **static** String *CLASS\_REMOTECONTROLGUI* = "RemoteControlGui" ;

**public** **static** String *CLASS\_REMOTECONTROLOPTIONS* = "RemoteControlOptions" ;

**public** **static** String *CLASS\_REMOTECONTROLERROR* = "RemoteControlError" ;

**public** **static** String *CLASS\_MESSAGE* = "Message" ;

//UUID

**public** **static** String *UUID\_REMOTECONTROL* = "9EF2AEC0AFA911dbABBD0800200C9A66" ;

//Device classes

**public** **static** **int** *MAJOR\_CLASS\_COMPUTER* = 256 ;

**public** **static** **int** *MINOR\_CLASS\_DESKTOP* = 4 ;

**public** **static** **int** *MINOR\_CLASS\_SERVER* = 8 ;

**public** **static** **int** *MINOR\_CLASS\_LAPTOP* = 12 ;

**public** **static** **int** *MINOR\_CLASS\_PDA* = 20 ;

**public** **static** **int** *MAJOR\_CLASS\_PHONE* = 512 ;

**public** **static** **int** *MINOR\_CLASS\_CELLULAR* = 4 ;

**public** **static** **int** *MINOR\_CLASS\_HOUSEHOLDPHONE* = 8 ;

**public** **static** **int** *MINOR\_CLASS\_SMARTPHONE* = 12 ;

**public** **static** **int** *MAJOR\_CLASS\_IMAGING* = 1536 ;

**public** **static** **int** *MINOR\_CLASS\_CAMERA* = 32 ;

**public** **static** **int** *MINOR\_CLASS\_SCANNER* = 64 ;

**public** **static** **int** *MINOR\_CLASS\_PRINTER* = 128 ;

**public** **static** **int** *MAJOR\_CLASS\_ALL* = 0 ;

//Pictures

**public** **static** String *PICTURE\_ARROW* = "/Pictures/up.png" ;

**public** **static** String *PICTURE\_INTRO* = "/Pictures/intro.png" ;

//Menu and text colours

**public** **static** **int** *TEXT\_COLOUR* = 0xFFFFFF ;

**public** **static** **int** *TEXT\_COLOUR\_SELECTED* = 0x000000 ;

**public** **static** **int** *BACKGROUND\_COLOUR* = 0x003EFF ;

**public** **static** **int** *MENUITEM\_BOX\_COLOUR* = 0xFFFFFF ;

**public** **static** **int** *KEYPRESSEDSCREEN\_BACKGROUND* = 0x000000 ;

**public** **static** **int** *KEYPRESSED\_TEXT\_COLOUR* = 0xFFFFFF ;

}

#### DomainSingleton.java

**package** appLogic;

**import** database.\* ;

**import** net.\* ;

//Ensures that there is only one instance of each domain and service class.

**public** **class** DomainSingleton {

**private** **static** DomainSingleton *singleton* = **null** ;

//appLogic classes

**private** ObjectSerializing serial = **null** ;

**private** RemoteControlLogic rclogic ;

**private** ProfileAdminLogic palogic ;

//database classes

**private** DBHandler db = **null** ;

**private** DBIntegration dbint = **null** ;

//net classes

**private** ProfileSearch profile = **null** ;

**private** RemoteControlCommunication rccom = **null** ;

**private** ServiceSearch service = **null** ;

//Private constructor to make sure it is only created once

**private** DomainSingleton() {

}

**public** **static** **synchronized** DomainSingleton getDomainSingleton() {

**if**(*singleton* == **null**) {

*singleton* = **new** DomainSingleton() ;

}

**return** *singleton* ;

}

**public** **synchronized** ObjectSerializing getObjectSerializing() {

**if**(serial == **null**) {

serial = **new** ObjectSerializing() ;

}

**return** serial ;

}

**public** **synchronized** ProfileAdminLogic getProfileAdminLogic() {

**if**(palogic == **null**) {

palogic = **new** ProfileAdminLogic() ;

}

**return** palogic ;

}

**public** **synchronized** RemoteControlLogic getRemoteControlLogic() {

**if**(rclogic == **null**) {

rclogic = **new** RemoteControlLogic() ;

}

**return** rclogic ;

}

**public** **synchronized** DBHandler getDBHandler() {

**if**(db == **null**) {

db = **new** DBHandler() ;

}

**return** db ;

}

**public** **synchronized** DBIntegration getDBIntegration() {

**if**(dbint == **null**) {

dbint = **new** DBIntegration() ;

}

**return** dbint ;

}

**public** **synchronized** ProfileSearch getProfileSearch() {

**if**(profile == **null**) {

profile = **new** ProfileSearch() ;

}

**return** profile ;

}

**public** **synchronized** RemoteControlCommunication getRemoteControlCommunication() {

**if**(rccom == **null**) {

rccom = **new** RemoteControlCommunication() ;

}

**return** rccom ;

}

**public** **synchronized** ServiceSearch getServiceSearch() {

**if**(service == **null**) {

service = **new** ServiceSearch() ;

}

**return** service ;

}

}

#### InitKeyActionDatabase.java

**package** appLogic;

**import** java.util.\* ;

//Contains the default key settings.

//It is loaded and stored the first time the application is started.

**public** **class** InitKeyActionDatabase {

**private** Vector defaultKeys ;

**public** InitKeyActionDatabase() {

defaultKeys = **new** Vector() ;

}

**public** Vector createDefaultKeyActions() {

KeyAction up = **new** KeyAction(Constants.*KEY\_NAME\_UP*,

Constants.*KEY\_ACTION\_TYPE\_PRESS\_RELEASE*, Constants.*KEY\_INITIAL\_MOUSE\_VALUE*,

Constants.*KEY\_INITIAL\_UP*) ;

KeyAction down = **new** KeyAction(Constants.*KEY\_NAME\_DOWN*,

Constants.*KEY\_ACTION\_TYPE\_PRESS\_RELEASE*, Constants.*KEY\_INITIAL\_MOUSE\_VALUE*,

Constants.*KEY\_INITIAL\_DOWN*) ;

KeyAction left = **new** KeyAction(Constants.*KEY\_NAME\_LEFT*,

Constants.*KEY\_ACTION\_TYPE\_PRESS\_RELEASE*, Constants.*KEY\_INITIAL\_MOUSE\_VALUE*,

Constants.*KEY\_INITIAL\_LEFT*) ;

KeyAction right = **new** KeyAction(Constants.*KEY\_NAME\_RIGHT*,

Constants.*KEY\_ACTION\_TYPE\_PRESS\_RELEASE*, Constants.*KEY\_INITIAL\_MOUSE\_VALUE*,

Constants.*KEY\_INITIAL\_RIGHT*) ;

KeyAction select = **new** KeyAction(Constants.*KEY\_NAME\_SELECT*,

Constants.*KEY\_ACTION\_TYPE\_PRESS\_RELEASE*, -1, Constants.*KEY\_INITIAL\_SELECT*) ;

KeyAction left\_softkey = **new** KeyAction(Constants.*KEY\_NAME\_LEFT\_SOFTKEY*,

Constants.*KEY\_ACTION\_TYPE\_SINGLE*, -1, Constants.*KEY\_INITIAL\_LEFTSOFTKEY*) ;

KeyAction right\_softkey = **new** KeyAction(Constants.*KEY\_NAME\_RIGHT\_SOFTKEY*,

Constants.*KEY\_ACTION\_TYPE\_SINGLE*, -1, Constants.*KEY\_INITIAL\_RIGHTSOFTKEY*) ;

KeyAction one = **new** KeyAction(Constants.*KEY\_NAME\_1*,

Constants.*KEY\_ACTION\_TYPE\_COMBINATION*, -1, Constants.*KEY\_INITIAL\_1*) ;

one.setKeyAction(KeyConstants.*KEY\_TAB*) ;

KeyAction two = **new** KeyAction(Constants.*KEY\_NAME\_2*,

Constants.*KEY\_ACTION\_TYPE\_COMBINATION*, -1, Constants.*KEY\_INITIAL\_2*) ;

two.setKeyAction(KeyConstants.*KEY\_D*) ;

KeyAction three = **new** KeyAction(Constants.*KEY\_NAME\_3*,

Constants.*KEY\_ACTION\_TYPE\_SINGLE*, -1, Constants.*KEY\_INITIAL\_3*) ;

KeyAction four = **new** KeyAction(Constants.*KEY\_NAME\_4*,

Constants.*KEY\_ACTION\_TYPE\_SINGLE*, -1, Constants.*KEY\_INITIAL\_4*) ;

KeyAction five = **new** KeyAction(Constants.*KEY\_NAME\_5*,

Constants.*KEY\_ACTION\_TYPE\_SINGLE*, -1, Constants.*KEY\_INITIAL\_5*) ;

KeyAction six = **new** KeyAction(Constants.*KEY\_NAME\_6*,

Constants.*KEY\_ACTION\_TYPE\_SINGLE*, -1, Constants.*KEY\_INITIAL\_6*) ;

KeyAction seven = **new** KeyAction(Constants.*KEY\_NAME\_7*,

Constants.*KEY\_ACTION\_TYPE\_SINGLE*, -1, Constants.*KEY\_INITIAL\_7*) ;

KeyAction eight = **new** KeyAction(Constants.*KEY\_NAME\_8*,

Constants.*KEY\_ACTION\_TYPE\_SINGLE*, -1, Constants.*KEY\_INITIAL\_8*) ;

KeyAction nine = **new** KeyAction(Constants.*KEY\_NAME\_9*,

Constants.*KEY\_ACTION\_TYPE\_SINGLE*, -1, Constants.*KEY\_INITIAL\_9*) ;

KeyAction zero = **new** KeyAction(Constants.*KEY\_NAME\_0*,

Constants.*KEY\_ACTION\_TYPE\_SINGLE*, -1, Constants.*KEY\_INITIAL\_0*) ;

KeyAction star = **new** KeyAction(Constants.*KEY\_NAME\_STAR*,

Constants.*KEY\_ACTION\_TYPE\_SINGLE*, -1, Constants.*KEY\_INITIAL\_STAR*) ;

KeyAction hash = **new** KeyAction(Constants.*KEY\_NAME\_HASH*,

Constants.*KEY\_ACTION\_TYPE\_SINGLE*, -1, Constants.*KEY\_INITIAL\_HASH*) ;

defaultKeys.addElement(up) ;

defaultKeys.addElement(down) ;

defaultKeys.addElement(left) ;

defaultKeys.addElement(right) ;

defaultKeys.addElement(select) ;

defaultKeys.addElement(left\_softkey) ;

defaultKeys.addElement(right\_softkey) ;

defaultKeys.addElement(one) ;

defaultKeys.addElement(two) ;

defaultKeys.addElement(three) ;

defaultKeys.addElement(four) ;

defaultKeys.addElement(five) ;

defaultKeys.addElement(six) ;

defaultKeys.addElement(seven) ;

defaultKeys.addElement(eight) ;

defaultKeys.addElement(nine) ;

defaultKeys.addElement(zero) ;

defaultKeys.addElement(star) ;

defaultKeys.addElement(hash) ;

**return** defaultKeys ;

}

}

#### KeyAction.java

**package** appLogic;

**import** interfaces.\* ;

**import** java.io.\* ;

**import** java.util.\* ;

//Stores the key action for a button on the phone

**public** **class** KeyAction **implements** RMSSerial {

//The name of the key

**private** String keyName ;

//The type of action, combination, progression or press/release

**private** String keyActionType ;

//The value, for determening the mouse movements

**private** **int** value ;

//The key actions

**private** Vector key ;

**public** KeyAction() {

key = **new** Vector() ;

}

**public** KeyAction(String n, String type, String s) {

keyName = n ;

keyActionType = type ;

key = **new** Vector() ;

key.addElement(s) ;

}

**public** KeyAction(String n, String type, **int** v, String s) {

keyName = n ;

keyActionType = type ;

value = v ;

key = **new** Vector() ;

key.addElement(s) ;

}

**public** **void** setKeyName(String s) {

keyName = s ;

}

**public** **void** setKeyAction(String s) {

key.addElement(s) ;

}

**public** **void** setKeyActionType(String type) {

keyActionType = type ;

}

**public** **void** setValue(**int** v) {

value = v ;

}

**public** String getKeyName() {

**return** keyName ;

}

**public** String getKey() {

**return** (String)key.elementAt(0) ;

}

**public** Vector getKeys() {

**return** key ;

}

**public** String getKeyActionType() {

**return** keyActionType ;

}

**public** **int** getValue() {

**return** value ;

}

**public** **void** removeKeyAt(**int** position) {

key.removeElementAt(position) ;

}

//Serializing methods, used to store objects in rms databases.

//The idea is taken from http://java.sun.com/developer/J2METechTips/2002/tt0226.html

//however the code is updated.

**public** **byte**[] serialize() **throws** IOException {

ByteArrayOutputStream bout = **new** ByteArrayOutputStream() ;

DataOutputStream dout = **new** DataOutputStream(bout) ;

dout.writeUTF(keyName) ;

dout.writeUTF(keyActionType) ;

dout.writeInt(value) ;

dout.writeInt(key.size()) ;

**for**(**int** i = 0; i < key.size(); i++) {

dout.writeUTF((String)key.elementAt(i)) ;

}

dout.flush() ;

**byte**[] data = bout.toByteArray() ;

bout.close() ;

dout.close() ;

**return** data ;

}

**public** **void** restore(**byte**[] data) **throws** IOException {

ByteArrayInputStream bin = **new** ByteArrayInputStream(data) ;

DataInputStream din = **new** DataInputStream(bin) ;

key = **new** Vector() ;

keyName = din.readUTF() ;

keyActionType = din.readUTF() ;

value = din.readInt() ;

**int** size = din.readInt() ;

**for**(**int** i = 0; i < size; i++) {

key.addElement(din.readUTF()) ;

}

din.close() ;

bin.close() ;

}

}

#### KeyConstants.java

**package** appLogic;

//Stores the keyboard actions that are possible to use in the application.

**public** **class** KeyConstants {

//Numbers

**public** **static** String *KEY\_0* = "0" ;

**public** **static** String *KEY\_1* = "1" ;

**public** **static** String *KEY\_2* = "2" ;

**public** **static** String *KEY\_3* = "3" ;

**public** **static** String *KEY\_4* = "4" ;

**public** **static** String *KEY\_5* = "5" ;

**public** **static** String *KEY\_6* = "6" ;

**public** **static** String *KEY\_7* = "7" ;

**public** **static** String *KEY\_8* = "8" ;

**public** **static** String *KEY\_9* = "9" ;

//Letters

**public** **static** String *KEY\_A* = "a" ;

**public** **static** String *KEY\_B* = "b" ;

**public** **static** String *KEY\_C* = "c" ;

**public** **static** String *KEY\_D* = "d" ;

**public** **static** String *KEY\_E* = "e" ;

**public** **static** String *KEY\_F* = "f" ;

**public** **static** String *KEY\_G* = "g" ;

**public** **static** String *KEY\_H* = "h" ;

**public** **static** String *KEY\_I* = "i" ;

**public** **static** String *KEY\_J* = "j" ;

**public** **static** String *KEY\_K* = "k" ;

**public** **static** String *KEY\_L* = "l" ;

**public** **static** String *KEY\_M* = "m" ;

**public** **static** String *KEY\_N* = "n" ;

**public** **static** String *KEY\_O* = "o" ;

**public** **static** String *KEY\_P* = "p" ;

**public** **static** String *KEY\_Q* = "q" ;

**public** **static** String *KEY\_R* = "r" ;

**public** **static** String *KEY\_S* = "s" ;

**public** **static** String *KEY\_T* = "t" ;

**public** **static** String *KEY\_U* = "u" ;

**public** **static** String *KEY\_V* = "v" ;

**public** **static** String *KEY\_W* = "w" ;

**public** **static** String *KEY\_X* = "x" ;

**public** **static** String *KEY\_Y* = "y" ;

**public** **static** String *KEY\_Z* = "z" ;

**public** **static** String *KEY\_Æ* = "æ" ;

**public** **static** String *KEY\_Ø* = "ø" ;

**public** **static** String *KEY\_Å* = "å" ;

//Keyboard directional

**public** **static** String *KEY\_KEYBOARDUP* = "Up key" ;

**public** **static** String *KEY\_KEYBOARDDOWN* = "Down key" ;

**public** **static** String *KEY\_KEYBOARDLEFT* = "Left key" ;

**public** **static** String *KEY\_KEYBOARDRIGHT* = "Right key" ;

//Mouse directional

**public** **static** String *KEY\_UP* = "Up" ;

**public** **static** String *KEY\_DOWN* = "Down" ;

**public** **static** String *KEY\_LEFT* = "Left" ;

**public** **static** String *KEY\_RIGHT* = "Right" ;

**public** **static** String *KEY\_PRESS* = "KeyPress" ;

**public** **static** String *KEY\_RELEASE* = "KeyRelease" ;

//Misc

**public** **static** String *KEY\_LEFTMOUSEBUTTON* = "lmouse" ;

**public** **static** String *KEY\_RIGHTMOUSEBUTTON* = "rmouse" ;

**public** **static** String *KEY\_CTRL* = "ctrl" ;

**public** **static** String *KEY\_ALT* = "alt" ;

**public** **static** String *KEY\_TAB* = "tab" ;

**public** **static** String *KEY\_ENTER* = "enter" ;

**public** **static** String *KEY\_SPACE* = "space" ;

**public** **static** String *KEY\_BACKSPACE* = "backspace";

**public** **static** String *KEY\_SHIFT* = "shift" ;

**public** **static** String *KEY\_ESC* = "esc" ;

**public** **static** String *KEY\_WINDOWS* = "windows" ;

//Array with all the variables

**public** **static** String[] *KEY\_VALUES* = {

*KEY\_0*,

*KEY\_1*,

*KEY\_2*,

*KEY\_3*,

*KEY\_4*,

*KEY\_5*,

*KEY\_6*,

*KEY\_7*,

*KEY\_8*,

*KEY\_9*,

*KEY\_A*,

*KEY\_B*,

*KEY\_C*,

*KEY\_D*,

*KEY\_E*,

*KEY\_F*,

*KEY\_G*,

*KEY\_H*,

*KEY\_I*,

*KEY\_J*,

*KEY\_K*,

*KEY\_L*,

*KEY\_M*,

*KEY\_N*,

*KEY\_O*,

*KEY\_P*,

*KEY\_Q*,

*KEY\_R*,

*KEY\_S*,

*KEY\_T*,

*KEY\_U*,

*KEY\_V*,

*KEY\_W*,

*KEY\_X*,

*KEY\_Y*,

*KEY\_Z*,

*KEY\_KEYBOARDUP*,

*KEY\_KEYBOARDDOWN*,

*KEY\_KEYBOARDLEFT*,

*KEY\_KEYBOARDRIGHT*,

*KEY\_UP*,

*KEY\_DOWN*,

*KEY\_LEFT*,

*KEY\_RIGHT*,

*KEY\_LEFTMOUSEBUTTON*,

*KEY\_RIGHTMOUSEBUTTON*,

*KEY\_CTRL*,

*KEY\_ALT*,

*KEY\_TAB*,

*KEY\_ENTER*,

*KEY\_SPACE*,

*KEY\_BACKSPACE*,

*KEY\_SHIFT*,

*KEY\_ESC*,

*KEY\_WINDOWS*

} ;

}

#### NetConstants.java

**package** appLogic;

//Stores constants that are used over the network.

**public** **class** NetConstants {

//RemoteControl status

**public** **static** String *REMOTECONTROL\_CONNECTED* = "Connected" ;

**public** **static** String *REMOTECONTROL\_DISCONNECT* = "Disconnect" ;

//Appliction type

**public** **static** String *APPLICATION\_REMOTECONTROL* = "RC" ;

//Key action types

**public** **static** String *KEY\_ACTION\_TYPE\_SINGLE* = "Single" ;

**public** **static** String *KEY\_ACTION\_TYPE\_PROGRESSION* = "Progression" ;

**public** **static** String *KEY\_ACTION\_TYPE\_COMBINATION* = "Combination" ;

**public** **static** String *KEY\_ACTION\_TYPE\_PRESS\_RELEASE* = "Press/Release" ;

**public** **static** String[] *KEY\_ACTION\_TYPES* = {

*KEY\_ACTION\_TYPE\_PROGRESSION*,

*KEY\_ACTION\_TYPE\_COMBINATION*,

*KEY\_ACTION\_TYPE\_SINGLE*,

*KEY\_ACTION\_TYPE\_PRESS\_RELEASE*

} ;

//Key names, the keys that are possible to map on the phone

**public** **static** String *KEY\_NAME\_UP* = "UP" ;

**public** **static** String *KEY\_NAME\_DOWN* = "DOWN" ;

**public** **static** String *KEY\_NAME\_LEFT* = "LEFT" ;

**public** **static** String *KEY\_NAME\_RIGHT* = "RIGHT" ;

**public** **static** String *KEY\_NAME\_SELECT* = "SELECT" ;

**public** **static** String *KEY\_NAME\_LEFT\_SOFTKEY* = "L KEY" ;

**public** **static** String *KEY\_NAME\_RIGHT\_SOFTKEY* = "R KEY" ;

**public** **static** String *KEY\_NAME\_1* = "1" ;

**public** **static** String *KEY\_NAME\_2* = "2" ;

**public** **static** String *KEY\_NAME\_3* = "3" ;

**public** **static** String *KEY\_NAME\_4* = "4" ;

**public** **static** String *KEY\_NAME\_5* = "5" ;

**public** **static** String *KEY\_NAME\_6* = "6" ;

**public** **static** String *KEY\_NAME\_7* = "7" ;

**public** **static** String *KEY\_NAME\_8* = "8" ;

**public** **static** String *KEY\_NAME\_9* = "9" ;

**public** **static** String *KEY\_NAME\_0* = "0" ;

**public** **static** String *KEY\_NAME\_STAR* = "\*" ;

**public** **static** String *KEY\_NAME\_HASH* = "#" ;

**public** **static** String[] *KEY\_NAMES* = {

*KEY\_NAME\_UP*,

*KEY\_NAME\_DOWN*,

*KEY\_NAME\_LEFT*,

*KEY\_NAME\_RIGHT*,

*KEY\_NAME\_SELECT*,

*KEY\_NAME\_LEFT\_SOFTKEY*,

*KEY\_NAME\_RIGHT\_SOFTKEY*,

*KEY\_NAME\_1*,

*KEY\_NAME\_2*,

*KEY\_NAME\_3*,

*KEY\_NAME\_4*,

*KEY\_NAME\_5*,

*KEY\_NAME\_6*,

*KEY\_NAME\_7*,

*KEY\_NAME\_8*,

*KEY\_NAME\_9*,

*KEY\_NAME\_0*,

*KEY\_NAME\_STAR*,

*KEY\_NAME\_HASH*

} ;

//The key codes for the directonal buttons and softkeys

**public** **static** **int** *KEY\_LEFTSOFTKEY* = -6 ;

**public** **static** **int** *KEY\_RIGHTSOFTKEY* = -7 ;

**public** **static** **int** *KEY\_POINTER\_UP* = -1 ;

**public** **static** **int** *KEY\_POINTER\_DOWN* = -2 ;

**public** **static** **int** *KEY\_POINTER\_LEFT* = -3 ;

**public** **static** **int** *KEY\_POINTER\_RIGHT* = -4 ;

**public** **static** **int** *KEY\_POINTER\_SELECT* = -5 ;

//The initial key values, used by InitialKeyActionDatabase class

**public** **static** String *KEY\_INITIAL\_UP* = KeyConstants.*KEY\_UP* ;

**public** **static** String *KEY\_INITIAL\_DOWN* = KeyConstants.*KEY\_DOWN* ;

**public** **static** String *KEY\_INITIAL\_LEFT* = KeyConstants.*KEY\_LEFT* ;

**public** **static** String *KEY\_INITIAL\_RIGHT* = KeyConstants.*KEY\_RIGHT* ;

**public** **static** String *KEY\_INITIAL\_SELECT* = KeyConstants.*KEY\_LEFTMOUSEBUTTON* ;

**public** **static** String *KEY\_INITIAL\_LEFTSOFTKEY* = KeyConstants.*KEY\_LEFTMOUSEBUTTON* ;

**public** **static** String *KEY\_INITIAL\_RIGHTSOFTKEY* = KeyConstants.*KEY\_RIGHTMOUSEBUTTON* ;

**public** **static** String *KEY\_INITIAL\_1* = KeyConstants.*KEY\_ALT* ;

**public** **static** String *KEY\_INITIAL\_2* = KeyConstants.*KEY\_WINDOWS* ;

**public** **static** String *KEY\_INITIAL\_3* = KeyConstants.*KEY\_ALT*;

**public** **static** String *KEY\_INITIAL\_4* = KeyConstants.*KEY\_SPACE* ;

**public** **static** String *KEY\_INITIAL\_5* = KeyConstants.*KEY\_KEYBOARDUP* ;

**public** **static** String *KEY\_INITIAL\_6* = KeyConstants.*KEY\_BACKSPACE* ;

**public** **static** String *KEY\_INITIAL\_7* = KeyConstants.*KEY\_KEYBOARDLEFT* ;

**public** **static** String *KEY\_INITIAL\_8* = KeyConstants.*KEY\_ENTER* ;

**public** **static** String *KEY\_INITIAL\_9* = KeyConstants.*KEY\_KEYBOARDRIGHT* ;

**public** **static** String *KEY\_INITIAL\_0* = KeyConstants.*KEY\_KEYBOARDDOWN* ;

**public** **static** String *KEY\_INITIAL\_STAR* = KeyConstants.*KEY\_ESC* ;

**public** **static** String *KEY\_INITIAL\_HASH* = "Exit" ;

**public** **static** **int** *KEY\_INITIAL\_MOUSE\_VALUE* = 6 ;

}

#### ObjectSerializing.java

**package** appLogic ;

**import** interfaces.\* ;

**import** java.io.\* ;

//A factory class. Receives an object and creates the correct class.

**public** **class** ObjectSerializing {

**public** ObjectSerializing() {

}

//Restores the object from byte

**public** Object restore(**byte**[] data) **throws** IOException {

ByteArrayInputStream bin = **new** ByteArrayInputStream(data) ;

DataInputStream din = **new** DataInputStream(bin) ;

**try** {

String type = din.readUTF() ;

**int** len = din.readInt() ;

**byte**[] b = **new** **byte**[len] ;

din.readFully(b) ;

Object o ;

**if**(type.equals(Constants.*CLASS\_BTDEVICE*)) {

o = **new** BTDevice() ;

((RMSSerial)o).restore(b) ;

**return** o ;

}

**else** **if**(type.equals(Constants.*CLASS\_KEYACTION*)) {

o = **new** KeyAction() ;

((RMSSerial)o).restore(b) ;

**return** o ;

}

**else** {

**return** **null** ;

}

}

**catch**(IOException io) {

**return** **null** ;

}

}

//Creates a byte array from an object. This makes it possible to send the object over the network and

//store it in the database.

**public** **byte**[] serialize(Object o, String type) **throws** IOException {

ByteArrayOutputStream bout = **new** ByteArrayOutputStream() ;

DataOutputStream dout = **new** DataOutputStream(bout) ;

**if**(o **instanceof** RMSSerial) {

dout.writeUTF(type) ;

**byte**[] data = ((RMSSerial)o).serialize() ;

dout.writeInt(data.length) ;

**if**(data.length > 0) {

dout.write(data) ;

}

}

dout.flush() ;

**byte**[] data2 = bout.toByteArray() ;

bout.close() ;

dout.close() ;

**return** data2 ;

}

}

#### ProfileAdminLogic.java

**package** appLogic;

**import** guiController.\* ;

**import** database.\* ;

**import** net.\* ;

**import** java.util.\* ;

**import** java.io.\* ;

//Contains the business logic for the profile administrator.

**public** **class** ProfileAdminLogic {

**private** ProfileAdminCont pacont ;

**private** ProfileSearch search ;

**private** DBHandler db ;

**private** ObjectSerializing serial ;

**private** DBIntegration dbint ;

**private** Vector devicesTemp ;

**private** Vector devices ;

**public** ProfileAdminLogic() {

search = (DomainSingleton.*getDomainSingleton*()).getProfileSearch() ;

search.setProfileAdminLogic(**this**) ;

dbint = (DomainSingleton.*getDomainSingleton*()).getDBIntegration() ;

dbint.setProfileAdminLogic(**this**) ;

db = (DomainSingleton.*getDomainSingleton*()).getDBHandler() ;

db.setDBIntegration(dbint) ;

serial = (DomainSingleton.*getDomainSingleton*()).getObjectSerializing() ;

devices = **new** Vector() ;

devicesTemp = **new** Vector() ;

}

**public** **void** setProfileAdminCont(ProfileAdminCont pac) {

pacont = pac ;

}

//Searches for devices, based upon the int value dt (device type)

**public** **void** searchDevices(**int** dt) {

devicesTemp = getStoredDevices() ;

search.setDeviceType(dt) ;

Thread t = **new** Thread(search) ;

t.start() ;

}

//When a new device is found this method is called. It sends updates with the BTDevice object

//and if it is a new or already saved object.

**public** **void** updateDevices(BTDevice btd) {

**boolean** saved = **false** ;

**for**(**int** i = 0; i < devicesTemp.size(); i++) {

BTDevice temp = (BTDevice)devicesTemp.elementAt(i) ;

**if**(temp.getAddress().equals(btd.getAddress())) {

saved = **true** ;

**break** ;

}

}

**if**(saved) {

pacont.updateDevices(btd, Constants.*BTDEVICE\_SAVED*) ;

}

**else** {

pacont.updateDevices(btd, Constants.*BTDEVICE\_NEW*) ;

}

}

//Saves a BTDevice in the database

**public** **void** storeDevice(BTDevice btd) {

devices = getStoredDevices() ;

**boolean** add = **true** ;

**for**(**int** i = 0; i < devices.size(); i++) {

BTDevice temp = (BTDevice)devices.elementAt(i) ;

**if**(temp.getAddress().equals(btd.getAddress())) {

add = **false** ;

**break** ;

}

}

**if**(add) {

**try** {

**byte**[] data = serial.serialize(btd, Constants.*CLASS\_BTDEVICE*) ;

db.write(data, Constants.*CLASS\_BTDEVICE*) ;

}

**catch**(IOException e) {

objectSaved(Constants.*ERROR\_IOEXCEPTION\_SAVE*) ;

}

}

**else** {

objectSaved(Constants.*ERROR\_DUPLICATE*) ;

}

}

//Deletes a device, based on the position.

**public** **void** deleteDevice(**int** position) {

db.delete(position, Constants.*CLASS\_BTDEVICE*) ;

}

//The message from the database, if the write/update/delete operation was completed.

**public** **void** objectSaved(String event) {

pacont.objectSaved(event) ;

}

//Returns the list of all the stored BTDevices

**public** Vector getStoredDevices() {

**return** db.read() ;

}

//Is called when the search is completed

**public** **void** searchCompleted() {

pacont.searchCompleted() ;

}

//Opens the database, tries to close any open connection first.

**public** **void** openRMS() {

db.closeRMS() ;

db.openRMS(Constants.*DBBTDEVICE*) ;

}

//Close database

**public** **void** closeRMS() {

db.closeRMS() ;

}

}

#### RemoteControlLogic.java

**package** appLogic;

**import** net.\* ;

**import** guiController.\* ;

**import** database.\* ;

**import** java.util.\* ;

**import** java.io.\* ;

**import** javax.bluetooth.UUID ;

//Contains the business logic for the remote control.

**public** **class** RemoteControlLogic {

**private** RemoteControlCont rccont ;

**private** ServiceSearch search ;

**private** RemoteControlCommunication rccom ;

**private** DBHandler db ;

**private** ObjectSerializing serial ;

**private** DBIntegration dbint ;

**private** **int** initialKeyNum ;

**private** Vector keyActions ;

**private** UUID uuid ;

**public** RemoteControlLogic() {

rccom = (DomainSingleton.*getDomainSingleton*()).getRemoteControlCommunication() ;

rccom.setRemoteControlLogic(**this**) ;

search = (DomainSingleton.*getDomainSingleton*()).getServiceSearch() ;

search.setRemoteControlCommunication(rccom) ;

db = (DomainSingleton.*getDomainSingleton*()).getDBHandler() ;

serial = (DomainSingleton.*getDomainSingleton*()).getObjectSerializing() ;

dbint = (DomainSingleton.*getDomainSingleton*()).getDBIntegration() ;

dbint.setRemoteControlLogic(**this**) ;

uuid = **new** UUID(Constants.*UUID\_REMOTECONTROL*, **false**) ;

initialKeyNum = 0 ;

keyActions = **new** Vector() ;

}

**public** **void** setRemoteControlCont(RemoteControlCont rcc) {

rccont = rcc ;

}

**public** **void** startSearch(BTDevice btd) {

search.startSearch(btd, uuid, Constants.*CLASS\_REMOTECONTROLMENU*) ;

}

//Send the KeyAction object over the network to the server.

//Returns true if the transfer is completed

**public** **boolean** setKeyAction(KeyAction ka) {

**try** {

**byte**[] data = serial.serialize(ka, Constants.*CLASS\_KEYACTION*) ;

**int** length = data.length ;

String type = Constants.*CLASS\_KEYACTION* ;

**return** rccom.setKeyAction(length, type, data) ;

}

**catch**(IOException io) {

**return** **false** ;

}

}

//Returns true if the connection is accepted by the server

**public** **void** connectedService(**boolean** status) {

rccont.connectedService(status) ;

}

//Returns the list of all the stored key actions from the databse

**public** Vector getKeyActions() {

Vector temp = **new** Vector() ;

Vector temp2 = **new** Vector() ;

temp = db.read() ;

**if**(temp.size() == 19) {

**for**(**int** i = 0; i < (Constants.*KEY\_NAMES*).length; i++) {

**for**(**int** i2 = 0; i2 < temp.size(); i2++) {

KeyAction ka = (KeyAction)temp.elementAt(i2) ;

**if**((Constants.*KEY\_NAMES*)[i].equals(ka.getKeyName())) {

temp2.addElement(ka) ;

**break** ;

}

}

}

}

**return** temp2 ;

}

//Closes the network connections

**public** **void** closeConnections() {

rccom.closeConnections() ;

}

//Returns the status message from the database about the initial key actions. If the were saved or not.

**public** **void** objectInitialKeysSaved(String event) {

initialKeyNum++ ;

**if**(initialKeyNum == 19) {

rccont.objectSaved(event) ;

initialKeyNum = 0 ;

}

}

//Is called if the key actions are loaded from the database

**private** **void** objectsLoaded(String event) {

rccont.objectSaved(event) ;

}

//The status message of the key action save operation

**public** **void** keyActionSaved(String event) {

rccont.keyActionSaved(event) ;

}

//Opens the database. Closes any open databases first.

//If the number of records is bigger than 0,

//the objects are loaded from the database.

//If the number of records is 0 the default key actions are saved.

**public** **void** openRMS() {

db.closeRMS() ;

db.openRMS(Constants.*DBKEYACTION*) ;

**if**(db.getNumberOfRecords() == 0) {

initDatabase() ;

}

**else** {

objectsLoaded(Constants.*KEYACTIONS\_LOADED*) ;

}

}

//Closes the database

**public** **void** closeRMS() {

db.closeRMS() ;

}

//Saves the default key actions

**private** **void** initDatabase() {

keyActions = **new** InitKeyActionDatabase().createDefaultKeyActions() ;

**try** {

**for**(**int** i = 0; i < keyActions.size(); i++) {

**byte**[] data = serial.serialize(((KeyAction)keyActions.elementAt(

i)), Constants.*CLASS\_KEYACTION*) ;

db.write(data, Constants.*CLASS\_INITIAL\_KEYACTION*) ;

}

}

**catch**(IOException io) {

}

}

//Updates the key actions

**public** **void** saveKeyAction(KeyAction ka, **int** position) {

db.read() ;

**try** {

**byte**[] data = serial.serialize(ka, Constants.*CLASS\_KEYACTION*) ;

db.update(data, (position + 1), Constants.*CLASS\_KEYACTION*) ;

}

**catch**(IOException io) {

}

}

//Is called if the application is disconnected from the server.

**public** **void** disconnect(String event) {

rccont.disconnect(event) ;

}

//Returns the error if the service is not found

**public** **void** serviceError(String event) {

rccont.serviceError(event) ;

}

//Disconnects from the server

**public** **void** endRemoteControl() {

rccom.endRemoteControl() ;

}

}

### A10.1.4 Package: net

#### ProfileSearch.java

**package** net;

**import** appLogic.\* ;

**import** javax.bluetooth.\* ;

**import** java.io.\* ;

//Contains the methods for the Bluetooth device search.

**public** **class** ProfileSearch **implements** Runnable, DiscoveryListener {

**private** ProfileAdminLogic palogic ;

**private** LocalDevice local ;

**private** DiscoveryAgent agent ;

**private** **int** deviceType ;

**private** **int** num ;

**public** ProfileSearch() { }

**public** **void** setProfileAdminLogic(ProfileAdminLogic pal) {

palogic = pal ;

}

**public** **void** setDeviceType(**int** dt) {

deviceType = dt ;

}

//Is called if a device is discovered.

**public** **void** deviceDiscovered(RemoteDevice rd, DeviceClass dc) {

**int** majorClass = dc.getMajorDeviceClass() ;

**if**(majorClass == deviceType || deviceType == 0) {

String address = rd.getBluetoothAddress() ;

**int** minorClass = dc.getMinorDeviceClass() ;

**int** serviceClass = dc.getServiceClasses() ;

String friendlyName = "" ;

**try** {

friendlyName = rd.getFriendlyName(**true**) ;

}

**catch**(IOException io) {

}

BTDevice btd = **new** BTDevice() ;

btd.setAddress(address) ;

btd.setName(friendlyName) ;

btd.setMajorClass(majorClass) ;

btd.setMinorClass(minorClass) ;

btd.setServiceClass(serviceClass) ;

**if**(num == 0) {

palogic.searchCompleted() ;

num++ ;

}

palogic.updateDevices(btd) ;

}

}

**public** **void** servicesDiscovered(**int** i, ServiceRecord[] sr) {

}

**public** **void** serviceSearchCompleted(**int** i, **int** i2) {

}

//Is called when the search is completed

**public** **void** inquiryCompleted(**int** i){

**if**(num == 0) {

palogic.searchCompleted() ;

num++ ;

}

}

//The search is done in a seperate thread, so that the rest of the application can still run.

**public** **void** run() {

**try** {

num = 0 ;

local = LocalDevice.*getLocalDevice*() ;

agent = local.getDiscoveryAgent() ;

agent.startInquiry(DiscoveryAgent.*GIAC*, **this**) ;

}

**catch**(BluetoothStateException bse) {

}

}

}

#### RemoteControlCommunication.java

**package** net;

**import** appLogic.\* ;

**import** java.io.\* ;

**import** javax.bluetooth.\* ;

**import** javax.microedition.io.\* ;

//Handles all the network communication for the remote control.

**public** **class** RemoteControlCommunication {

**private** RemoteControlLogic rclogic ;

**private** ServiceRecord[] sr ;

**private** StreamConnection con ;

**private** DataInputStream in ;

**private** DataOutputStream out ;

**public** RemoteControlCommunication() {

}

**public** **void** setRemoteControlLogic(RemoteControlLogic rcl) {

rclogic = rcl ;

}

**public** **void** setServiceRecord(ServiceRecord[] s) {

sr = s ;

}

//Sends the key action of the network.

**public** **synchronized** **boolean** setKeyAction(**int** length, String type, **byte**[] data) {

**try** {

out.writeUTF(type) ;

out.writeInt(length) ;

out.write(data) ;

out.flush() ;

**return** **true** ;

}

**catch**(IOException io) {

closeConnections() ;

rclogic.disconnect(Constants.*ERROR\_REMOTECONTROL\_DISCONNETED*) ;

**return** **false** ;

}

}

//Disconnects from the server

**public** **synchronized** **void** endRemoteControl() {

**try** {

out.writeUTF(Constants.*REMOTECONTROL\_DISCONNECT*) ;

out.flush() ;

}

**catch**(IOException io) {

}

**finally** {

closeConnections() ;

}

}

//Open all required connections.

**private** **void** openConnections() {

**if**(sr != **null**) {

**try** {

con = (StreamConnection)Connector.*open*(sr[0].getConnectionURL(

ServiceRecord.*AUTHENTICATE\_ENCRYPT*, **false**)) ;

in = con.openDataInputStream() ;

out = con.openDataOutputStream() ;

}

**catch**(IOException io) {

}

}

}

//Closes all open connections

**public** **void** closeConnections() {

**try** {

in.close() ;

out.close() ;

con.close() ;

}

**catch**(IOException io) {

}

}

//Is called when the service is found by the ServiceSearch class. It will first write to the server

//that this is the remote control application.

**public** **synchronized** **void** startCom() {

openConnections() ;

**try** {

String platform = Constants.*PLATFORM* + "<cr>" ;

**byte**[] pf = platform.getBytes() ;

out.write(pf) ;

out.flush();

String stat = in.readUTF() ;

**if**(stat.equals(Constants.*REMOTECONTROL\_CONNECTED*)) { out.writeUTF(Constants.*APPLICATION\_REMOTECONTROL*) ;

out.flush() ;

rclogic.connectedService(**true**) ;

}

**else**

**if**(stat.equals(Constants.*REMOTECONTROL\_DISCONNECT*)) {

rclogic.connectedService(**false**) ;

closeConnections() ;

}

}

**catch**(IOException io) {

rclogic.connectedService(**false**) ;

}

}

**public** **void** serviceError(String event) {

rclogic.serviceError(event) ;

}

}

#### ServiceSearch.java

**package** net;

**import** appLogic.\* ;

**import** javax.bluetooth.\* ;

**import** javax.bluetooth.UUID ;

//Contains the methods for the Bluetooth service search.

**public** **class** ServiceSearch **implements** DiscoveryListener, Runnable {

**private** BTDevice btd ;

**private** UUID uuid ;

**private** LocalDevice local ;

**private** DiscoveryAgent agent ;

**private** RemoteDevice rd ;

**private** String callerclass ;

**private** RemoteControlCommunication rccom ;

**public** ServiceSearch() {

}

**public** **void** setRemoteControlCommunication(RemoteControlCommunication rcc) {

rccom = rcc ;

}

**public** **synchronized** **void** startSearch(BTDevice b, UUID u, String cc) {

btd = b ;

uuid = u ;

callerclass = cc ;

rd = **new** RemoteDeviceInstance(btd.getAddress()) ;

**new** Thread(**this**).start() ;

}

**public** **void** deviceDiscovered(RemoteDevice rd, DeviceClass dc) {

}

**public** **void** inquiryCompleted(**int** i) {

}

**public** **void** serviceSearchCompleted(**int** transID, **int** respCode) {

**if**(respCode == *SERVICE\_SEARCH\_COMPLETED*) {

rccom.startCom() ;

}

**else** **if**(respCode == *SERVICE\_SEARCH\_NO\_RECORDS* || respCode == *SERVICE\_SEARCH\_DEVICE\_NOT\_REACHABLE* ||

respCode == *SERVICE\_SEARCH\_ERROR*) {

rccom.serviceError(Constants.*ERROR\_SERVICE\_NOT\_FOUND*) ;

}

}

**public** **void** servicesDiscovered(**int** transID, ServiceRecord[] sr) {

**if**(callerclass.equals(Constants.*CLASS\_REMOTECONTROLMENU*)) {

rccom.setServiceRecord(sr) ;

}

}

//Searches for a service based on the given UUID

**public** **void** run() {

**try** {

local = LocalDevice.*getLocalDevice*() ;

agent = local.getDiscoveryAgent() ;

agent.searchServices(**null**, **new** UUID[] { uuid }, rd, **this**) ;

}

**catch**(BluetoothStateException bse) {

}

}

//Creates a RemoteDevice based on the address

**private** **class** RemoteDeviceInstance **extends** RemoteDevice {

**public** RemoteDeviceInstance(String address) {

**super**(address) ;

}

}

}

### A10.1.5 Package: database

#### DBHandler.java

**package** database;

**import** interfaces.\* ;

**import** appLogic.\* ;

**import** java.io.\* ;

**import** java.util.\* ;

**import** javax.microedition.rms.\* ;

//Handles all operations on the database.

**public** **class** DBHandler {

**private** RecordStore rs ;

**private** ObjectSerializing serial ;

**private** DBIntegration dbint ;

**public** DBHandler() {

serial = (DomainSingleton.*getDomainSingleton*()).getObjectSerializing() ;

}

**public** **void** setDBIntegration(DBIntegration dbi) {

dbint = dbi ;

}

//Opens the database, based on the String variable

//The boolean variable means that the database should also be created if it does not already exist.

**public** **void** openRMS(String s) {

rs = **null** ;

**try** {

rs = RecordStore.*openRecordStore*(s, **true**) ;

}

**catch**(RecordStoreException rse) {

}

}

//Returns the number of records stored.

**public** **int** getNumberOfRecords() {

**try** {

**return** rs.getNumRecords() ;

}

**catch**(RecordStoreNotOpenException rsnoe) {

**return** -1 ;

}

}

//Closes the current database

**public** **void** closeRMS() {

**if**(rs != **null**) {

**try** {

rs.closeRecordStore() ;

}

**catch**(RecordStoreException rse) {

}

}

}

//Reads all the stored records

**public** Vector read() {

Vector objects = **new** Vector() ;

**try** {

RecordEnumeration re = rs.enumerateRecords(**null**, **null**, **true**) ;

**while**(re.hasNextElement()) {

**byte**[] data = re.nextRecord() ;

Object o = **new** Object() ;

**try** {

o = (RMSSerial)serial.restore(data) ;

objects.addElement(o) ;

}

**catch**(IOException io) {

}

}

}

**catch**(RecordStoreNotOpenException rsnoe) {

}

**catch**(InvalidRecordIDException irie) {

}

**catch**(RecordStoreException rse) {

}

**return** objects ;

}

//The method called from other objects to write a new records.

//It creates a new Thread that creates the record.

**public** **void** write(**byte**[] data, String obj) {

**try** {

**new** writeThread(data, -1, obj, Constants.*DBWRITE*).join() ;

}

**catch**(InterruptedException ie) {

ie.printStackTrace() ;

}

}

//Method called from other objects to update a record.

**public** **void** update(**byte**[] data, **int** position, String obj) {

**new** writeThread(data, position, obj, Constants.*DBUPDATE*) ;

}

//Method called from other objects to delete a record.

**public** **void** delete(**int** position, String obj) {

**new** writeThread(**null**, position, obj, Constants.*DBDELETE*) ;

}

//Called from the Thread class. Updates a record based on its position

**private** **synchronized** **void** privateUpdate(**byte**[] data, **int** position, String obj) {

**try** {

rs.setRecord(position, data, 0, data.length) ;

dbint.notifyStored(obj) ;

}

**catch**(RecordStoreNotOpenException rsnoe) {

dbint.notifyError(obj, Constants.*ERROR\_RECORDSTORENOTOPENEXCEPTION*) ;

}

**catch**(RecordStoreFullException rsfe) {

dbint.notifyError(obj, Constants.*ERROR\_RECORDSTOREFULLEXCEPTION*) ;

}

**catch**(RecordStoreException rse) {

dbint.notifyError(obj, Constants.*ERROR\_RECORDSTOREEXCEPTION*) ;

}

}

//Called from the Thread class. Writes a new record.

**private** **synchronized** **void** privateWrite(**byte**[] data, String obj) {

**try** {

rs.addRecord(data, 0, data.length) ;

dbint.notifyStored(obj) ;

}

**catch**(RecordStoreNotOpenException rsnoe) {

dbint.notifyError(obj, Constants.*ERROR\_RECORDSTORENOTOPENEXCEPTION*) ;

}

**catch**(RecordStoreFullException rsfe) {

dbint.notifyError(obj, Constants.*ERROR\_RECORDSTOREFULLEXCEPTION*) ;

}

**catch**(RecordStoreException rse) {

dbint.notifyError(obj, Constants.*ERROR\_RECORDSTOREEXCEPTION*) ;

}

}

//Called from the Thread class. Deletes a record based on its position.

**private** **synchronized** **void** privateDelete(**int** position, String obj) {

**boolean** deleted = **false** ;

**int** counter = getNumberOfRecords() - (position + 1) ;

**int** precount = 0 ;

**try** {

**for**(**int** i = 0; i < rs.getNextRecordID(); i++) {

**try** {

**byte**[] temp = rs.getRecord(i) ;

**if**(counter == precount) {

rs.deleteRecord(i) ;

dbint.notifyDeleted(obj) ;

deleted = **true** ;

**break** ;

}

**else** {

precount++ ;

}

}

**catch**(RecordStoreException rse) {

}

}

**if**(!deleted)

dbint.notifyError(obj, Constants.*ERROR\_RECORDSTOREEXCEPTION*) ;

}

**catch**(RecordStoreNotOpenException rsnoe) {

dbint.notifyError(obj, Constants.*ERROR\_RECORDSTORENOTOPENEXCEPTION*) ;

}

**catch**(RecordStoreFullException rsfe) {

dbint.notifyError(obj, Constants.*ERROR\_RECORDSTOREFULLEXCEPTION*) ;

}

**catch**(RecordStoreException rse) {

dbint.notifyError(obj, Constants.*ERROR\_RECORDSTOREEXCEPTION*) ;

}

}

//A new thread is created to do all the operations on the database. This is done because thesse operations

//can be time consuming and could possibly lock the entire a pplication if it is done in the main thread.

**private** **class** writeThread **extends** Thread {

**private** **byte**[] data ;

**private** String obj ;

**private** String optype ;

**private** **int** position ;

**public** writeThread(**byte**[] a, **int** p, String o, String type) {

data = a ;

position = p ;

obj = o ;

optype = type ;

start() ;

}

**public** **void** run() {

**if**(optype.equals(Constants.*DBWRITE*)) {

**try** {

privateWrite(data, obj) ;

}

**catch**(NullPointerException npe) {

npe.printStackTrace() ;

}

}

**else** **if**(optype.equals(Constants.*DBUPDATE*)) {

privateUpdate(data, position, obj) ;

}

**else** **if**(optype.equals(Constants.*DBDELETE*)) {

privateDelete(position, obj) ;

}

}

}

}

#### DBIntegration.java

**package** database;

**import** appLogic.\* ;

//Notifies the domain classes about the result of the database operation.

**public** **class** DBIntegration {

**private** ProfileAdminLogic palogic ;

**private** RemoteControlLogic rclogic ;

**public** DBIntegration() {}

**public** **void** setProfileAdminLogic(ProfileAdminLogic pal) {

palogic = pal ;

}

**public** **void** setRemoteControlLogic(RemoteControlLogic rcl) {

rclogic = rcl ;

}

//Is called if the object is stored

**public** **void** notifyStored(String objectName) {

**if**(objectName.equals(Constants.*CLASS\_BTDEVICE*)) {

palogic.objectSaved(Constants.*BTDEVICE\_SAVED*) ;

}

**else** **if**(objectName.equals(Constants.*CLASS\_INITIAL\_KEYACTION*)) {

rclogic.objectInitialKeysSaved(Constants.*DEFAULT\_KEYACTIONS\_SAVED*) ;

}

**else** **if**(objectName.equals(Constants.*CLASS\_KEYACTION*)) {

rclogic.keyActionSaved(Constants.*KEYACTION\_SAVED*) ;

}

}

//Is called if the object is deleted

**public** **void** notifyDeleted(String objectName) {

**if**(objectName.equals(Constants.*CLASS\_BTDEVICE*)) {

palogic.objectSaved(Constants.*BTDEVICE\_DELETED*) ;

}

}

//If an error occurs and the database operation could not be //completed, this method will be called.

**public** **void** notifyError(String objectName, String error) {

**if**(objectName.equals(Constants.*CLASS\_BTDEVICE*)) {

palogic.objectSaved(error) ;

}

**else** **if**(objectName.equals(Constants.*CLASS\_INITIAL\_KEYACTION*)) {

rclogic.objectInitialKeysSaved(error) ;

}

**else** **if**(objectName.equals(Constants.*CLASS\_KEYACTION*)) {

rclogic.keyActionSaved(error) ;

}

}

}

### A10.1.6 Package: interfaces

#### RMSSerial.java

**package** interfaces;

**import** java.io.\* ;

//Interface that is used by the DBHandler.

**public** **interface** RMSSerial {

**public** **byte**[] serialize() **throws** IOException ;

**public** **void** restore(**byte**[] data) **throws** IOException ; }

## A10.2 Windows Mobile

### A10.2.1 Package: Gui

#### CustomMapKeys.cs

using System;

using System.Collections.Generic;

using System.Collections;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using Microsoft.WindowsCE.Forms;

using System.Windows.Forms;

using BTShareMobile.appLogic;

using BTShareMobile.Controller;

using InTheHand.Windows.Forms;

namespace BTShareMobile.Gui

{

//Displays the custom map key options.

//This includes displaying a list of all the available keys to custom map.

//There are a total of 18 keys that can be mapped.

//It also displays a KeyAction object and presents the options the user has the edit it.

public partial class CustomMapKeys : Form

{

private Init init;

private ProfileAdminCont pacont;

private ArrayList menu, keys, keyConstants, keyTypes, tempKeyTypes;

private Boolean first;

private float menuHeight;

private int prevpos, position, count, numberOfLines, menunumber;

private KeyAction key;

private TextBox tb;

private Label label;

private delegate void setKeyActionDel(KeyAction ka);

private delegate void setKeyActionsDel(ArrayList keys);

public CustomMapKeys(Init i)

{

init = i;

pacont = GuiSingleton.GetProfileAdminCont();

pacont.setCustomMapKeys(this);

menu = new ArrayList();

menu.Add("Add key/pointer speed");

menu.Add("Remove key action");

menu.Add("Set action type");

keys = new ArrayList();

//Adds all the key options in an ArrayList (The options for what you can custom map a key to)

keyConstants = new ArrayList();

for (int a = 0; a < (Constants.keyNames).Length; a++)

{

keyConstants.Add((Constants.keyNames)[a]);

}

//Adds all the key types in an ArrayList

keyTypes = new ArrayList();

for (int a = 0; a < (Constants.keyTypes).Length; a++)

{

keyTypes.Add((Constants.keyTypes)[a]);

}

first = true;

//The label for the textbox, used on the pointer speed screen

label = new Label();

label.Text = "Pointer speed:";

label.Top = 10;

label.Left = 10;

label.Font = Constants.font;

this.Controls.Add(label);

label.Visible = false;

//The textbox in the pointer speed screen

tb = new TextBox();

tb.Top = label.Height + 10;

tb.Left = 10;

tb.MaxLength = 4;

this.Controls.Add(tb);

tb.Visible = false;

InputModeEditor.SetInputMode(tb, InputMode.Numeric);

menuHeight = 0;

position = 0;

count = 0;

numberOfLines = ScreenPlacement.getNumberOfLines(Height);

menunumber = 0;

BackColor = Constants.backgroundColour;

this.Owner = (GuiSingleton.GetProfileAdminGui(init));

InitializeComponent();

}

protected override void OnPaintBackground(PaintEventArgs e)

{

if (first)

base.OnPaintBackground(e);

}

public void setInit(Init i)

{

init = i;

}

//Called when all the KeyActions are returned from the database

public void setKeyActions(ArrayList list)

{

init.changeLoadingStatus();

Invoke(new setKeyActionsDel(setKeyActionsPrivate), new object[] { list });

}

//Invoked from the worker thread, stops the loading screen and shows the CustomMapKeys gui

private void setKeyActionsPrivate(ArrayList k)

{

keys = new ArrayList(k);

keys.RemoveAt(keys.Count - 1);

menuHeight = ScreenPlacement.getMenuHeight(keys.Count);

init.stopLoading();

}

//When the number of objects in the list that is shown on the screen is larger than the number

//of lines the screen has, the list is divided into the right parts that fits the screen.

//The counter object calculate what part of the list should be shown

private void setCounter(PaintEventArgs e, int size)

{

if (numberOfLines < size)

{

if ((position - count) == (numberOfLines - 1) && position != (size - 1))

{

if (!first)

count++;

first = true;

OnPaintBackground(e);

}

else if (count > 0 && (position - count) == 0)

{

if (!first)

count--;

first = true;

OnPaintBackground(e);

}

}

}

//Sets the right KeyActionTypes for the current KeyAction

private ArrayList setTypes()

{

tempKeyTypes = new ArrayList();

int c = (key.getKeys()).Count;

int start = 0;

int stop = c;

if (c > 1)

{

start = 2;

stop = 4;

}

else if (c == 1 && (key.getKey() == "Up" || key.getKey() == "Down" || key.getKey() == "Left" || key.getKey() == "Right" || key.getKey() == "lmouse"))

{

start = 0;

stop = 2;

}

else

{

start = 0;

stop = 1;

}

IEnumerator enumerator = keyTypes.GetEnumerator();

enumerator.MoveNext();

for(int i = 0; i < stop; i++)

{

if (i >= start)

{

string s = (string)enumerator.Current;

tempKeyTypes.Add(s);

}

enumerator.MoveNext();

}

return tempKeyTypes;

}

//Controls the KeyActionType, to see if it is correct when a KeyAction object has been change.

private void controlKeyType()

{

if ((key.getKeys()).Count > 1)

{

if (key.getKeyActionType() != Constants.keytype\_combination || key.getKeyActionType() != Constants.keytype\_progression)

{

key.setKeyActionType(Constants.keytype\_combination);

}

}

else if ((key.getKeys()).Count == 1)

{

if ((key.getKey() == "Up" || key.getKey() == "Down" || key.getKey() == "Left" || key.getKey() == "Right") && key.getKeyActionType() != Constants.keytype\_pressrelease)

{

key.setKeyActionType(Constants.keytype\_pressrelease);

}

else if (key.getKeyActionType() != Constants.keytype\_single || key.getKeyActionType() != Constants.keytype\_pressrelease)

{

key.setKeyActionType(Constants.keytype\_single);

}

}

}

//Paints the menu based on the menunumber.

//0 = list of the KeyActions

//1 = the menu for a selected KeyAction

//2 = add a new key

//3 = remove a key

//4 = change KeyActionType

//5 = change pointer speed

protected override void OnPaint(PaintEventArgs e)

{

if (menunumber == 0)

{

setCounter(e, keys.Count);

if (first)

{

ScreenPlacement.paintArrayList(e, keys, position, count, Height, Constants.class\_keyaction);

first = false;

}

else

{

ScreenPlacement.paintArrayListChange(e, keys, prevpos, position, count, Height, Constants.class\_keyaction);

}

}

else if (menunumber == 1)

{

if (first)

{

ScreenPlacement.paintKeyMenu(e, menu, key, Height);

first = false;

}

else

{

ScreenPlacement.painKeyMenuChange(e, menu, position, prevpos, Height);

}

}

else if (menunumber == 2)

{

setCounter(e, keyConstants.Count);

if (first)

{

ScreenPlacement.paintArrayList(e, keyConstants, position, count, Height, Constants.class\_keyconstant);

first = false;

}

else

{

ScreenPlacement.paintArrayListChange(e, keyConstants, prevpos, position, count, Height, Constants.class\_keyconstant);

}

}

else if (menunumber == 3)

{

setCounter(e, (key.getKeys()).Count);

if (first)

{

ScreenPlacement.paintArrayList(e, key.getKeys(), position, count, Height, Constants.class\_keyconstant);

first = false;

}

else

{

ScreenPlacement.paintArrayListChange(e, key.getKeys(), prevpos, position, count, Height, Constants.class\_keyconstant);

}

}

else if (menunumber == 4)

{

if (first)

{

ScreenPlacement.paintArrayList(e, setTypes(), position, count, Height, Constants.class\_keyconstant);

first = false;

}

else

{

ScreenPlacement.paintArrayListChange(e, setTypes(), prevpos, position, count, Height, Constants.class\_keyconstant);

}

}

else if (menunumber == 5)

{

tb.Text = Convert.ToString(key.getValue());

ScreenPlacement.paintPointerSpeed(tb, label);

}

}

//Left button

private void menuItem1\_Click(object sender, EventArgs e)

{

if (menunumber == 1)

{

first = true;

count = 0;

prevpos = 0;

position = 0;

ArrayList tmp = new ArrayList();

tmp.Add(pacont.saveKeyAction(key));

init.showMessage(tmp, this, false);

}

else

commandControl();

}

//Right button

private void menuItem2\_Click(object sender, EventArgs e)

{

prevpos = 0;

position = 0;

first = true;

count = 0;

if (menunumber == 0)

{

Close();

}

else if (menunumber == 1)

{

setEditButton();

menunumber = 0;

Invalidate();

}

else if (menunumber == 2)

{

setSaveButton();

menunumber = 1;

Invalidate();

}

else if (menunumber == 3)

{

setSaveButton();

menunumber = 1;

Invalidate();

}

else if (menunumber == 4)

{

setSaveButton();

menunumber = 1;

Invalidate();

}

else if (menunumber == 5)

{

tb.Visible = false;

label.Visible = false;

this.Focus();

setSaveButton();

menunumber = 1;

Invalidate();

}

}

//Controls all the commands based on the menunumber.

private void commandControl()

{

if (menunumber == 0)

{

if (keys.Count > 0)

{

int num = 0;

foreach (KeyAction ka in keys)

{

if (num == position)

{

setSaveButton();

menunumber = 1;

key = ka;

break;

}

num++;

}

}

}

else if (menunumber == 1)

{

if (position == 0)

{

if (key.getKey() != "Up" && key.getKey() != "Down" &&

key.getKey() != "Left" && key.getKey() != "Right")

{

menunumber = 2;

setAddButton();

}

else

{

menunumber = 5;

setOKButton();

}

}

else if (position == 1)

{

menunumber = 3;

setRemoveButton();

}

else if (position == 2)

{

menunumber = 4;

setAddButton();

}

}

else if (menunumber == 2)

{

int tmp = 0;

foreach (string s in keyConstants)

{

if (tmp == position)

{

key.setKeyAction(s);

break;

}

tmp++;

}

controlKeyType();

menunumber = 1;

setSaveButton();

}

else if (menunumber == 3)

{

(key.getKeys()).RemoveAt(position);

controlKeyType();

menunumber = 1;

setSaveButton();

}

else if (menunumber == 4)

{

int i = 0;

foreach (string s in tempKeyTypes)

{

if (i == position)

{

key.setKeyActionType(s);

break;

}

i++;

}

menunumber = 1;

setSaveButton();

}

else if (menunumber == 5)

{

menunumber = 1;

setSaveButton();

tb.Visible = false;

label.Visible = false;

this.Focus();

int value = Convert.ToInt32(tb.Text);

key.setValue(value);

string s = pacont.updateKeyActionValue(key);

ArrayList tmp = new ArrayList();

tmp.Add(s);

init.showMessage(tmp, this, false);

}

count = 0;

prevpos = 0;

position = 0;

first = true;

Invalidate();

}

//Registers when a key is pressed

private void Form1\_KeyDown(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Up)

{

if (menunumber == 0)

{

if (position > 0)

{

position--;

prevpos = position + 1;

}

else

{

position = (keys.Count - 1);

prevpos = 0;

count = keys.Count - numberOfLines;

first = true;

}

}

else if (menunumber == 1)

{

if (position > 0)

{

position--;

prevpos = position + 1;

}

else

{

position = (menu.Count - 1);

prevpos = 0;

}

}

else if (menunumber == 2)

{

if (position > 0)

{

position--;

prevpos = position + 1;

}

else

{

position = (keyConstants.Count - 1);

prevpos = 0;

count = keyConstants.Count - numberOfLines;

first = true;

}

}

else if (menunumber == 3)

{

if (position > 0)

{

position--;

prevpos = position + 1;

}

else

{

position = ((key.getKeys()).Count - 1);

prevpos = 0;

if (numberOfLines < (key.getKeys()).Count)

{

count = (key.getKeys()).Count - numberOfLines;

first = true;

}

}

}

else if (menunumber == 4)

{

if (position > 0)

{

position--;

prevpos = position + 1;

}

else

{

position = (tempKeyTypes.Count - 1);

prevpos = 0;

first = true;

}

}

}

else if (e.KeyCode == Keys.Down)

{

if (menunumber == 0)

{

if (position < (keys.Count - 1))

{

position++;

prevpos = position - 1;

}

else

{

position = 0;

prevpos = keys.Count - 1;

count = 0;

first = true;

}

}

else if (menunumber == 1)

{

if (position < (menu.Count - 1))

{

position++;

prevpos = position - 1;

}

else

{

position = 0;

prevpos = (menu.Count - 1);

}

}

else if (menunumber == 2)

{

if (position < (keyConstants.Count - 1))

{

position++;

prevpos = position - 1;

}

else

{

position = 0;

prevpos = keyConstants.Count - 1;

count = 0;

first = true;

}

}

else if (menunumber == 3)

{

if (position < ((key.getKeys()).Count - 1))

{

position++;

prevpos = position - 1;

}

else

{

position = 0;

prevpos = (key.getKeys()).Count - 1;

count = 0;

if (numberOfLines < (key.getKeys()).Count)

{

first = true;

}

}

}

else if (menunumber == 4)

{

if (position < (tempKeyTypes.Count - 1))

{

position++;

prevpos = position - 1;

}

else

{

position = 0;

prevpos = tempKeyTypes.Count - 1;

count = 0;

first = true;

}

}

}

else if (e.KeyCode == Keys.Enter)

{

commandControl();

}

Invalidate();

}

}

}

#### GuiSingleton.cs

using System;

using System.Collections.Generic;

using System.Text;

using System.Windows.Forms;

using BTShareMobile.Controller;

namespace BTShareMobile.Gui

{

//Ensures that there is only one instance of each user interface object (the only exception is the Message class).

public sealed class GuiSingleton

{

public static GuiSingleton guisingleton = null;

public static MainGui maingui = null;

public static ProfileAdminGui profileadmingui = null;

public static ProfileAdminCont profileadmincont = null;

public static Loading loading = null;

public static CustomMapKeys custommapkeys = null;

public static RemoteControlCont remotecontrolcont = null;

public static RemoteControlGui remotecontrolgui = null;

private static readonly object sync = new object();

private GuiSingleton()

{

}

public static GuiSingleton GetGuiSingleton()

{

lock (sync)

{

if (guisingleton == null)

{

guisingleton = new GuiSingleton();

}

}

return guisingleton;

}

public static MainGui GetMainGui()

{

lock (sync)

{

if (maingui == null)

{

maingui = new MainGui();

}

}

return maingui;

}

public static ProfileAdminGui GetProfileAdminGui(Init init)

{

lock (sync)

{

if (profileadmingui == null)

{

profileadmingui = new ProfileAdminGui(init);

}

}

return profileadmingui;

}

public static ProfileAdminCont GetProfileAdminCont()

{

lock (sync)

{

if (profileadmincont == null)

{

profileadmincont = new ProfileAdminCont();

}

}

return profileadmincont;

}

public static Loading GetLoading()

{

lock (sync)

{

if (loading == null)

{

loading = new Loading();

}

}

return loading;

}

public static CustomMapKeys GetCustomMapKeys(Init init)

{

lock (sync)

{

if (custommapkeys == null)

{

custommapkeys = new CustomMapKeys(init);

}

}

return custommapkeys;

}

public static RemoteControlCont GetRemoteControlCont()

{

lock (sync)

{

if (remotecontrolcont == null)

{

remotecontrolcont = new RemoteControlCont();

}

}

return remotecontrolcont;

}

public static RemoteControlGui GetRemoteControlGui(Init init)

{

lock (sync)

{

if (remotecontrolgui == null)

{

remotecontrolgui = new RemoteControlGui(init);

}

}

return remotecontrolgui;

}

}

}

#### Init.cs

using System;

using System.Collections;

using System.Collections.Generic;

using System.Windows.Forms;

using System.Threading;

using InTheHand.Net.Bluetooth;

using BTShareMobile.Gui;

using BTShareMobile.Controller;

using BTShareMobile.appLogic;

using BTShareMobile.net;

using BTShareMobile.DB;

namespace BTShareMobile.Gui

{

//Start the application and has responsibility for showing each user interface.

public class Init

{

private GuiSingleton guisingleton;

private MainGui maingui;

private RemoteControlCont remotecontrolcont;

private static Boolean change = false;

private static Boolean reset = false;

private static readonly object sync = new object();

//Turns the Bluetooth radio on, and then loads all the initial classes

public Init()

{

BluetoothRadio.PrimaryRadio.Mode = RadioMode.Connectable;

loadClasses();

}

//Creates the initial classes

private void loadClasses()

{

guisingleton = GuiSingleton.GetGuiSingleton();

maingui = GuiSingleton.GetMainGui();

maingui.setInit(this);

remotecontrolcont = GuiSingleton.GetRemoteControlCont();

remotecontrolcont.setInit(this);

remotecontrolcont.setMainGui(maingui);

runApp();

}

//Starts the application

private void runApp()

{

Application.Run(maingui);

}

//Closes all connections

public void exitApp()

{

remotecontrolcont.closeConnections();

maingui.Close();

Application.Exit();

}

//Shows MainGui

public void showMainGui()

{

maingui.Show();

}

//Shows ProfileAdminGui

public void showProfileAdminGui()

{

(GuiSingleton.GetProfileAdminGui(this)).ShowDialog();

}

//Shows CustomMapKeys

public void showCustomMapKeys()

{

(GuiSingleton.GetCustomMapKeys(this)).ShowDialog();

}

//Shows RemoteControlGui

public void showRemoteControlGui()

{

(GuiSingleton.GetRemoteControlGui(this)).Refresh();

(GuiSingleton.GetRemoteControlGui(this)).ShowDialog();

}

//Shows the message class. This is a simple form that gives the user information, like if

//the connection failed or if the KeyAction was saved correctly.

public void showMessage(ArrayList list, Form f, Boolean stop)

{

if(stop)

(GuiSingleton.GetLoading()).stopAll();

if (f != null)

{

new Message(list, f).ShowDialog();

}

else

{

new Message(list, maingui).ShowDialog();

}

}

//Sets the KeyActio objects in the RemoteControlGui

public void setKeysRemoteControlGui(ArrayList keys)

{

(GuiSingleton.GetRemoteControlGui(this)).setKeyActions(keys);

}

//Starts the loading screen. It will first sleep for 200 milliseconds, if the class is finished before this

//the loading screen is not shown. If the activity is not finished within this time the loading screen is shown.

public void startLoading(string cc, string lc)

{

Thread.Sleep(200);

lock (sync)

{

if (change)

{

changeResetStatus();

if (lc == Constants.class\_custommapkeys)

showCustomMapKeys();

else if (lc == Constants.class\_profileadmingui && cc == Constants.class\_profileadmingui)

{

}

else if(lc == Constants.class\_profileadmingui)

showProfileAdminGui();

}

else

{

(GuiSingleton.GetLoading()).setInit(this);

(GuiSingleton.GetLoading()).setClasses(cc, lc);

(GuiSingleton.GetLoading()).start();

(GuiSingleton.GetLoading()).ShowDialog();

}

}

}

//Stops the loadingscreen

public void stopLoading()

{

if(!reset)

(GuiSingleton.GetLoading()).stop();

}

//Cancels the loadingscreen

public void cancelLoading()

{

(GuiSingleton.GetLoading()).cancel();

}

//Disables the cancel button in the loading screen

public void noCancelOpt()

{

(GuiSingleton.GetLoading()).noCancelOpt();

}

//Resets the variables for the loading screen

public void resetChange()

{

change = false;

reset = false;

}

public void changeResetStatus()

{

reset = true;

}

//Called from the worker thread when the activity is finished

public void changeLoadingStatus()

{

change = true;

}

//Stops the database action

public void stopDBAction()

{

(GuiSingleton.GetProfileAdminCont()).stopDBAction();

}

//Stops the user clicking several times in a row, it disables the button when the action is in progress

public void changeMainGuiLoad()

{

maingui.changeLoad();

}

//Stops the connection to the server application

public void stopConnect()

{

(GuiSingleton.GetRemoteControlCont()).stopConnect();

}

[MTAThread]

static void Main()

{

new Init();

}

}

}

#### Loading.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

using System.Threading;

using BTShareMobile.appLogic;

namespace BTShareMobile.Gui

{

//Displays a loading screen for each task that takes some time to finish (like opening the database connection).

//It is used to show that the program is still running while

//a time consuming task is being executed in the background.

public partial class Loading : Form

{

private Init init;

private Boolean cont, first, call, cancelopt;

private int counter, timeout;

private float height, width;

private string callingclass, loadingclass;

private Thread t;

private delegate void finishLoading();

public Loading()

{

cont = true;

first = true;

call = true;

cancelopt = true;

counter = 1;

timeout = 0;

height = (Height / 2) - ((Constants.fontHeight / 2) + Constants.loadingbarheight);

width = ((Width / 2) - ((((Constants.loadingbarwidth) \* 3) + 20) / 2));

callingclass = "";

BackColor = Constants.backgroundColour;

Owner = GuiSingleton.GetMainGui();

InitializeComponent();

}

public void setInit(Init i)

{

init = i;

}

//Sets both the class that calls the loading object and the class that will be shown when the loading is finished

public void setClasses(string s, string s2)

{

callingclass = s;

loadingclass = s2;

}

//Creates a new thread that will update the user interface, so that it still changes while the task is

//done in the background

public void start()

{

cont = true;

counter = 1;

timeout = 0;

t = new Thread(new ThreadStart(start2));

t.Start();

}

//Sleets 300 milliseconds and then invokes on of the user interface changing methods

public void start2()

{

while (cont)

{

Thread.Sleep(300);

if (counter > 3)

{

counter = 1;

first = true;

}

else

{

counter++;

}

timeout++;

if (timeout == 100)

{

cont = false;

call = false;

}

Invoke(new finishLoading(update));

}

Invoke(new finishLoading(show));

}

//Updates the user interface

public void update()

{

if(cont)

Invalidate();

}

//Shows the correct user interface when the loading is finished

public void show()

{

if (call)

showLoadingclass();

else

showCallingClass();

}

//Stops the loading thread

public void stop()

{

cont = false;

}

//Aborts the loading thread, immediately closing the loading screen

public void stopAll()

{

if (timeout != 100)

{

try

{

t.Abort();

}

catch (ThreadAbortException)

{

}

cont = true;

first = true;

call = true;

cancelopt = true;

Close();

}

}

//Disables the cancel button

public void noCancelOpt()

{

cancelopt = false;

}

//Shows the classes that are supposed to be called after the loading is finished

private void showLoadingclass()

{

cont = true;

first = true;

cancelopt = true;

if (loadingclass == Constants.class\_profileadmingui && callingclass == Constants.class\_profileadmingui)

{

}

else if(loadingclass == Constants.class\_profileadmingui)

init.showProfileAdminGui();

else if (loadingclass == Constants.class\_custommapkeys)

init.showCustomMapKeys();

else if (loadingclass == Constants.class\_maingui)

init.showMainGui();

else if (loadingclass == Constants.class\_remotecontrolgui)

init.showRemoteControlGui();

Close();

}

//If something has gone wrong, the class that started the loading is shown

private void showCallingClass()

{

cont = true;

first = true;

call = true;

cancelopt = true;

if (callingclass == Constants.class\_profileadmingui)

{

init.stopDBAction();

}

else if (callingclass == Constants.class\_maingui)

{

if (loadingclass == Constants.class\_remotecontrolgui)

init.stopConnect();

}

Close();

}

protected override void OnPaintBackground(PaintEventArgs e)

{

if (first)

base.OnPaintBackground(e);

}

protected override void OnPaint(PaintEventArgs e)

{

if (first)

{

SizeF w = e.Graphics.MeasureString(Constants.loading\_text, Constants.font);

float w2 = Constants.width - (w.Width / 2);

e.Graphics.DrawString(Constants.loading\_text, Constants.font, Constants.text, w2, height);

first = false;

}

else

{

for (int i = 1; i < counter; i++)

{

if (cont)

{

e.Graphics.FillRectangle(Constants.text, new Rectangle((int)width + (((i - 1) \* Constants.loadingbarwidth) + ((i - 1) \* 10)),

(int)(height + Constants.fontHeight + 20), Constants.loadingbarwidth, Constants.loadingbarheight));

}

else

{

first = true;

break;

}

}

}

}

//Cancels the loading, and shows the user interface that called the loading screen

public void cancel()

{

call = false;

stop();

}

//Left button

private void menuItem1\_Click(object sender, EventArgs e)

{

if(cancelopt)

cancel();

}

}

}

#### MainGui.cs

using System;

using System.Collections;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

using InTheHand.Net.Sockets;

using BTShareMobile.appLogic;

using BTShareMobile.Controller;

namespace BTShareMobile.Gui

{

//The main menu.

public partial class MainGui : Form

{

private Init init;

private ArrayList menu, devices;

private float menuHeight;

private Boolean first, load;

private int position, prevpos, menunumber, count, numberOfLines;

private delegate void stopLoading();

private delegate void stopLoadRemoteControl();

private delegate void conFailed(string s);

public MainGui()

{

menu = new ArrayList();

menu.Add("Remote control");

menu.Add("Profile admin");

menu.Add("Exit");

devices = new ArrayList();

menuHeight = ScreenPlacement.getMenuHeight(menu.Count);

first = true;

position = 0;

menunumber = 0;

count = 0;

numberOfLines = ScreenPlacement.getNumberOfLines(Height);

load = false;

BackColor = Constants.backgroundColour;

InitializeComponent();

}

public void setInit(Init i)

{

init = i;

}

//Called from the worker thread, with information about the connection.

//If the connection is accepted the loading screen is stopped and RemoteControlGui is shown

//If the connection is rejected an infomation message is shown

public void connectionCompleted(string s)

{

if (s == Constants.connected)

{

init.noCancelOpt();

init.setKeysRemoteControlGui((GuiSingleton.GetRemoteControlCont()).getKeyActions());

init.changeLoadingStatus();

Invoke(new stopLoadRemoteControl(stopLoadingRemoteControl));

}

else

{

load = false;

position = 0;

first = true;

menunumber = 0;

init.changeLoadingStatus();

Invoke(new stopLoadRemoteControl(cancelLoading));

}

}

//Called from a worker thread if the connection fails

public void connectionFailed(string s)

{

Invoke(new conFailed(showMessage), new object[] { s });

}

//Shows an information message, for example if the connection has failed

private void showMessage(string s)

{

first = true;

ArrayList tmp = new ArrayList();

tmp.Add(s);

init.showMessage(tmp, this, true);

}

//Stops the loading screen

private void stopLoadingRemoteControl()

{

init.stopLoading();

}

//Cancels the loading screen

private void cancelLoading()

{

showOpenButton();

showExitButton();

init.cancelLoading();

}

//Returns all saved Bluetooth devices and adds them to an ArrayList

public void searchCompleted(BluetoothDeviceInfo[] dev)

{

if (dev.Length > 0)

{

devices.Clear();

foreach (BluetoothDeviceInfo bdi in dev)

{

devices.Add(bdi);

}

}

else

{

menunumber = 0;

}

init.changeLoadingStatus();

Invoke(new stopLoading(stop));

}

//Stops loading after the search for Bluetooth devices is complete

private void stop()

{

first = true;

Invalidate();

showConnectButton();

showBackButton();

init.stopLoading();

}

//Left button

private void menuItem1\_Click(object sender, EventArgs e)

{

if (!first)

{

if (menunumber == 0)

{

commandControl();

}

else if (!load)

{

load = true;

commandControl();

}

}

}

//Right button

private void menuItem2\_Click(object sender, EventArgs e)

{

if (menunumber == 0)

init.exitApp();

else if (menunumber == 1)

{

position = 0;

menunumber = 0;

first = true;

showOpenButton();

showExitButton();

Invalidate();

}

}

protected override void OnPaintBackground(PaintEventArgs e)

{

if (first)

base.OnPaintBackground(e);

}

//Paints the menu based on the menunumber

//0 = main menu

//1 = Bluetooth device menu for remote control connection

protected override void OnPaint(PaintEventArgs e)

{

if (menunumber == 0)

{

if (first)

{

ScreenPlacement.paintMenu(e, menu, position, Height);

first = false;

}

else

ScreenPlacement.paintMenuChange(e, menu, prevpos, position, Height);

}

else if (menunumber == 1)

{

if (numberOfLines < devices.Count)

{

if ((position - count) == (numberOfLines - 1) && position != (devices.Count - 1))

{

if (!first)

count++;

first = true;

OnPaintBackground(e);

}

else if (count > 0 && (position - count) == 0)

{

if (!first)

count--;

first = true;

OnPaintBackground(e);

}

}

if (first)

{

ScreenPlacement.paintArrayList(e, devices, position, count, Height, Constants.class\_bluetoothdeviceinfo);

first = false;

}

else

{

ScreenPlacement.paintArrayListChange(e, devices, prevpos, position, count, Height, Constants.class\_bluetoothdeviceinfo);

}

}

}

//Stops the button from being pressed several times in a row, load is true when an activity is running in

//the background

public void changeLoad()

{

load = false;

}

//Handles all the commands based on position and menunumber

private void commandControl()

{

first = true;

if (menunumber == 0)

{

if (position == 0)

{

position = 0;

menunumber = 1;

(GuiSingleton.GetProfileAdminCont()).startSearch(Constants.class\_maingui);

init.startLoading(Constants.class\_maingui, Constants.class\_maingui);

Invalidate();

}

else if (position == 1)

{

position = 0;

init.showProfileAdminGui();

}

else if (position == 2)

{

init.exitApp();

}

}

else if (menunumber == 1)

{

if (devices.Count > 0)

{

BluetoothDeviceInfo bdi = null;

IEnumerator enumerator = devices.GetEnumerator();

for (int i = 0; i < devices.Count; i++)

{

enumerator.MoveNext();

if (i == position)

{

bdi = (BluetoothDeviceInfo)enumerator.Current;

break;

}

}

if (bdi != null)

{

init.resetChange();

init.noCancelOpt();

load = false;

GuiSingleton.GetRemoteControlGui(init);

(GuiSingleton.GetRemoteControlCont()).connect(bdi);

init.startLoading(Constants.class\_maingui, Constants.class\_remotecontrolgui);

position = 0;

}

load = false;

}

else

{

load = false;

}

}

}

//Registers when a key is pressed

private void Form1\_KeyDown(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Up)

{

if (menunumber == 0)

{

if (position > 0)

{

position--;

prevpos = position + 1;

}

else

{

position = menu.Count - 1;

prevpos = 0;

}

}

else if (menunumber == 1)

{

if (position > 0)

{

position--;

prevpos = position + 1;

}

else

{

position = devices.Count - 1;

prevpos = 0;

if (numberOfLines < devices.Count)

{

count = devices.Count - numberOfLines;

first = true;

}

}

}

}

else if (e.KeyCode == Keys.Down)

{

if (menunumber == 0)

{

if (position < (menu.Count - 1))

{

position++;

prevpos = position - 1;

}

else

{

prevpos = menu.Count - 1;

position = 0;

}

}

else if (menunumber == 1)

{

if (position < (devices.Count - 1))

{

position++;

prevpos = position - 1;

}

else

{

prevpos = devices.Count - 1;

position = 0;

count = 0;

if (numberOfLines < devices.Count)

{

first = true;

}

}

}

}

else if (e.KeyCode == Keys.Enter)

{

commandControl();

}

Invalidate();

}

}

}

#### Message.cs

using System;

using System.Collections.Generic;

using System.Collections;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

using BTShareMobile.appLogic;

namespace BTShareMobile.Gui

{

//Displays an information message.

//This is for example used to let the user know that the KeyAction object is saved.

//This message object is created when needed.

public partial class Message : Form

{

private ArrayList list;

private Bitmap image;

public Message(ArrayList al, Form f)

{

string url = System.IO.Path.GetDirectoryName(System.Reflection.Assembly.GetExecutingAssembly().GetName().CodeBase);

image = new Bitmap(url + Constants.image\_info);

list = al;

BackColor = Color.Black;

Owner = f;

InitializeComponent();

}

protected override void OnPaint(PaintEventArgs e)

{

ScreenPlacement.paintMessage(e, list, Height, image);

}

//Left button

private void menuItem1\_Click(object sender, EventArgs e)

{

Close();

}

}

}

#### ProfileAdminGui.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Collections;

using System.Windows.Forms;

using System.Threading;

using BTShareMobile.appLogic;

using BTShareMobile.Controller;

using InTheHand.Net.Sockets;

namespace BTShareMobile.Gui

{

//Handles all the operations on the Bluetooth devices

//and also contains the menu for the Custom map keys option.

public partial class ProfileAdminGui : Form

{

private Init init;

private ProfileAdminCont pacont;

private ArrayList menu, searchmenu;

private Boolean first;

private int position, prevpos, numberOfLines, count, menunumber;

private float menuHeight;

public delegate void searchCompletedDel();

public ProfileAdminGui(Init i)

{

numberOfLines = ScreenPlacement.getNumberOfLines(Height);

init = i;

pacont = GuiSingleton.GetProfileAdminCont();

pacont.setProfileAdminGui(this);

menu = new ArrayList();

menu.Add("Custom map keys");

menu.Add("Open device list");

menu.Add("Back");

first = true;

menuHeight = ScreenPlacement.getMenuHeight(menu.Count);

position = 0 ;

BackColor = Constants.backgroundColour;

count = 0;

menunumber = 0;

searchmenu = new ArrayList();

this.Owner = (GuiSingleton.GetMainGui());

InitializeComponent();

}

public void setInit(Init i)

{

init = i;

}

//Called from worker thread when the search for Bluetooth devices is finished

public void searchCompleted(BluetoothDeviceInfo[] devices)

{

if (devices.Length > 0)

{

searchmenu.Clear();

foreach (BluetoothDeviceInfo bdi in devices)

{

searchmenu.Add(bdi);

}

}

else

{

menunumber = 0;

}

init.changeLoadingStatus();

Invoke(new searchCompletedDel(searchCompletedPrivate));

}

//The main thread is invoked to stop the loading screen

private void searchCompletedPrivate()

{

init.stopLoading();

}

protected override void OnPaintBackground(PaintEventArgs e)

{

if(first)

base.OnPaintBackground(e);

}

//Paints the user interface based on the menunumber

//0 = profile admin

//1 = list of Bluetooth devices

protected override void OnPaint(PaintEventArgs e)

{

if(menunumber == 0)

{

if (first)

{

ScreenPlacement.paintMenu(e, menu, position, Height);

first = false;

}

else

ScreenPlacement.paintMenuChange(e, menu, prevpos, position, Height);

}

else if (menunumber == 1)

{

if (numberOfLines < searchmenu.Count)

{

if ((position - count) == (numberOfLines - 1) && position != (searchmenu.Count - 1))

{

if (!first)

count++;

first = true;

OnPaintBackground(e);

}

else if (count > 0 && (position - count) == 0)

{

if (!first)

count--;

first = true;

OnPaintBackground(e);

}

}

if (first)

{

ScreenPlacement.paintArrayList(e, searchmenu, position, count, Height, Constants.class\_bluetoothdeviceinfo);

first = false;

}

else

{

ScreenPlacement.paintArrayListChange(e, searchmenu, prevpos, position, count, Height, Constants.class\_bluetoothdeviceinfo);

}

}

}

//Left button

private void menuItem1\_Click(object sender, EventArgs e)

{

commandControl();

Invalidate();

}

//Right button

private void menuItem2\_Click(object sender, EventArgs e)

{

if (menunumber == 0)

{

position = 2;

commandControl();

}

else if (menunumber == 1)

{

first = true;

position = 0;

menunumber = 0;

Invalidate();

}

}

private void commandControl()

{

first = true;

prevpos = 0;

if (menunumber == 0)

{

if (position == 0)

{

menunumber = 0;

position = 0;

init.resetChange();

GuiSingleton.GetCustomMapKeys(init);

pacont.getKeyActions();

init.startLoading(Constants.class\_profileadmingui, Constants.class\_custommapkeys);

}

else if (position == 1)

{

menunumber = 1;

position = 0;

init.resetChange();

pacont.startSearch(Constants.class\_profileadmingui);

init.startLoading(Constants.class\_profileadmingui, Constants.class\_profileadmingui);

}

else if (position == 2)

{

position = 0;

Close();

}

}

else if (menunumber == 1)

{

if (searchmenu.Count > 0)

{

ArrayList tmp = new ArrayList();

int counter = 0;

foreach (BluetoothDeviceInfo bdi in searchmenu)

{

if (counter == position)

{

tmp.Add("Name: " + bdi.DeviceName);

tmp.Add("Address: " + bdi.DeviceAddress);

tmp.Add("Class: " + ( bdi.ClassOfDevice).Device);

break;

}

counter++;

}

first = true;

init.showMessage(tmp, this, false);

}

}

}

//Registers when a key is pressed

private void Form1\_KeyDown(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Up)

{

if (menunumber == 0)

{

if (position > 0)

{

position--;

prevpos = position + 1;

}

else

{

prevpos = 0;

position = (menu.Count - 1);

}

}

else if (menunumber == 1)

{

if (position > 0)

{

position--;

prevpos = position + 1;

}

else

{

prevpos = 0;

position = (searchmenu.Count - 1);

if (numberOfLines < searchmenu.Count)

{

count = searchmenu.Count - numberOfLines;

first = true;

}

}

}

}

else if (e.KeyCode == Keys.Down)

{

if (menunumber == 0)

{

if (position < (menu.Count - 1))

{

position++;

prevpos = position - 1;

}

else

{

position = 0;

prevpos = menu.Count - 1;

}

}

else if (menunumber == 1)

{

if (position < (searchmenu.Count - 1))

{

position++;

prevpos = position - 1;

}

else

{

position = 0;

prevpos = searchmenu.Count - 1;

count = 0;

if (numberOfLines < searchmenu.Count)

{

first = true;

}

}

}

}

else if (e.KeyCode == Keys.Enter)

{

commandControl();

}

Invalidate();

}

}

}

#### RemoteControlGui.cs

using System;

using System.Collections;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Threading;

using System.Windows.Forms;

using BTShareMobile.appLogic;

using BTShareMobile.Controller;

namespace BTShareMobile.Gui

{

//Displays the remote control interface with a list of all the mapped keys.

//It also displays the action of the pressed key.

public partial class RemoteControlGui : Form

{

private Init init;

private RemoteControlCont rccont;

private KeyAction temp;

private Bitmap[] bitmapList;

private ArrayList keys;

private IEnumerator listreader;

private float[,] place;

private Boolean first, pressed, mousemovement, cont;

private int numOfLines;

private Thread thread;

private delegate void clearScreen();

public RemoteControlGui(Init i)

{

init = i;

rccont = GuiSingleton.GetRemoteControlCont();

rccont.setRemoteControlGui(this);

keys = new ArrayList();

first = true;

pressed = false;

mousemovement = false;

cont = true;

BackColor = Constants.backgroundColour;

this.Owner = (GuiSingleton.GetMainGui());

string url = System.IO.Path.GetDirectoryName(System.Reflection.Assembly.GetExecutingAssembly().GetName().CodeBase);

bitmapList = new Bitmap[4];

bitmapList[0] = new Bitmap(url + Constants.image\_up);

bitmapList[1] = new Bitmap(url + Constants.image\_down);

bitmapList[2] = new Bitmap(url + Constants.image\_left);

bitmapList[3] = new Bitmap(url + Constants.image\_right);

InitializeComponent();

}

public void setInit(Init i)

{

init = i;

}

public void setKeyActions(ArrayList list)

{

keys = list;

listreader = keys.GetEnumerator();

}

//Calculates the position for all the KeyActions that will be shown on the screen. This is only

//calculated once and stored in the float variable

private void calculatePlace(PaintEventArgs e)

{

if (keys.Count == 19)

{

place = new float[19, 3];

numOfLines = ScreenPlacement.getNumberOfLines(Height);

IEnumerator enumerator = keys.GetEnumerator();

enumerator.MoveNext();

SizeF tmp = new SizeF(0F, 0F);

float indent1 = 0;

float indent2 = 0;

for (int i = 0; i < numOfLines; i++)

{

KeyAction ka = (KeyAction)enumerator.Current;

tmp = e.Graphics.MeasureString((ka.getKeyActionName() + ":"), Constants.boldFont);

if (indent1 < tmp.Width)

{

indent1 = tmp.Width;

}

string keytemp = ka.getKey();

if ((ka.getKeys()).Count > 1)

keytemp += "+";

tmp = e.Graphics.MeasureString(keytemp, Constants.font);

float tmp2 = tmp.Width + indent1;

if (indent2 < tmp2)

{

indent2 = tmp2;

}

enumerator.MoveNext();

}

enumerator.Reset();

enumerator.MoveNext();

float indent3 = 0;

for (int i = 0; i < keys.Count; i++)

{

if (i >= numOfLines)

{

KeyAction ka = (KeyAction)enumerator.Current;

tmp = e.Graphics.MeasureString((ka.getKeyActionName() + ":"), Constants.boldFont);

if (indent3 < tmp.Width)

{

indent3 = tmp.Width;

}

}

enumerator.MoveNext();

}

for (int i = 0; i < numOfLines; i++)

{

place[i, 0] = ((Constants.fontHeight + 12) \* i);

place[i, 1] = 5;

place[i, 2] = (indent1 + 5);

}

for (int i = numOfLines; i < keys.Count; i++)

{

place[i, 0] = ((Constants.fontHeight + 12) \* (i - numOfLines));

place[i, 1] = (indent2 + 25);

place[i, 2] = (indent3 + 5);

}

}

else

{

init.changeLoadingStatus();

init.showMainGui();

}

}

protected override void OnPaintBackground(PaintEventArgs e)

{

if (first || !pressed)

base.OnPaintBackground(e);

}

//Paints the user interfaced based on if a key has been pushed or not

protected override void OnPaint(PaintEventArgs e)

{

if (first)

{

cont = true;

calculatePlace(e);

int counter = 0;

foreach (KeyAction ka in keys)

{

if (counter == 5)

{

menuItem1.Text = ka.getKey();

}

else if (counter == 6)

{

menuItem2.Text = ka.getKey();

break;

}

counter++;

}

first = false;

}

if(pressed)

ScreenPlacement.paintRemoteControlGuiKeyPressed(e, temp, Width, Height);

else

ScreenPlacement.paintRemoteControlGui(e, place, keys, numOfLines, bitmapList);

}

//Registers when a key is pressed

//First checks if the KeyAction is a mousemovement, then if it is #. The #-key is always exit.

//Sends the correct KeyAction across the network

private void Form1\_KeyDown(object sender, KeyEventArgs e)

{

string s = (e.KeyCode).ToString();

Boolean sent = true;

foreach (KeyAction ka in keys)

{

if (ka.getKeyActionID() == s)

{

if (ka.getKey() == "Up" || ka.getKey() == "Down" || ka.getKey() == "Left" ||

ka.getKey() == "Right")

{

if (!mousemovement)

{

ka.setKeyAction("KeyPress");

sent = rccont.setKeyAction(ka);

ka.removeKeyAt(1);

mousemovement = true;

}

else

{

break;

}

}

else if (ka.getKeyActionName() == "#")

{

rccont.endRemoteControl();

disconnect();

break;

}

else

{

sent = rccont.setKeyAction(ka);

}

temp = ka;

threadStart();

break;

}

}

if (!sent)

{

disconnect();

ArrayList al = new ArrayList();

al.Add("Disconnected");

init.showMessage(al, null, false);

}

}

//Disconnects from the server application

private void disconnect()

{

first = true;

pressed = false;

mousemovement = false;

cont = false;

if (thread != null)

thread.Abort();

Close();

}

//Starts the thread that controls the KeyPressed screen

public void threadStart()

{

Invalidate();

pressed = true;

if (thread != null)

{

thread.Abort();

}

thread = new Thread(new ThreadStart(count));

thread.Start();

}

//The tread sleeps for 2000 milliseconds and then clears the screen back to the list

private void count()

{

Thread.Sleep(2000);

if(cont)

Invoke(new clearScreen(clear));

}

//Repaints the list of all KeyActions on the screen

private void clear()

{

ScreenPlacement.changeResetKeyAction();

pressed = false;

Invalidate();

}

//Registers if a key has been released. This is only used when they key is a mousemovement and it

//has the Press/Release KeyActionType

private void Form1\_KeyUp(object sender, KeyEventArgs e)

{

string s = (e.KeyCode).ToString();

foreach (KeyAction ka in keys)

{

if (ka.getKeyActionID() == s)

{

if ((ka.getKey() == "Up" || ka.getKey() == "Down" || ka.getKey() == "Left" ||

ka.getKey() == "Right") && ka.getKeyActionType() == Constants.keytype\_pressrelease)

{

if (mousemovement)

{

ka.setKeyAction("KeyRelease");

rccont.setKeyAction(ka);

ka.removeKeyAt(1);

mousemovement = false;

}

}

break;

}

}

}

//Left button

private void menuItem1\_Click(object sender, EventArgs e)

{

int counter = 0;

foreach (KeyAction ka in keys)

{

if (counter == 5)

{

temp = ka;

rccont.setKeyAction(ka);

break;

}

counter++;

}

threadStart();

}

//Right button

private void menuItem2\_Click(object sender, EventArgs e)

{

int counter = 0;

foreach (KeyAction ka in keys)

{

if (counter == 6)

{

temp = ka;

rccont.setKeyAction(ka);

break;

}

counter++;

}

threadStart();

}

}

}

#### ScreenPlacement.cs

using System;

using System.Collections.Generic;

using System.Text;

using BTShareMobile.appLogic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Collections;

using System.Windows.Forms;

using InTheHand.Net.Sockets;

namespace BTShareMobile.Gui

{

//Contains all the methods for painting the different menus.

//Each user interface class will call ScreenPlacement with the PaintEventArgs variable

//and it will take care of the painting.

//This makes it possible to reuse the menus on several screens.

public static class ScreenPlacement

{

private static KeyAction temp;

private static TextBox tb = new TextBox();

//Returns the screen size of a menu

public static float getMenuHeight(int size)

{

float menuHeight = ((size \* (Constants.fontHeight + 4)) / 2);

return menuHeight;

}

//Returns the number of lines on the screen

public static int getNumberOfLines(float height)

{

int lines = (int)(height / (Constants.fontHeight + 12));

int rest = (int)(height % (Constants.fontHeight + 12));

if (rest > 2)

lines--;

return lines;

}

//Retuns the place for a row in a list

public static float getPlace(int number, int size, float menuHeight, float height)

{

float tmp = height - menuHeight;

tmp = tmp + (Constants.fontHeight \* number);

if (number > 0)

{

tmp = tmp + (number \* 10);

}

return tmp;

}

//Returns the width of a text

public static float getTextWidth(string s, Graphics g)

{

SizeF w = g.MeasureString(s, Constants.font);

float w2 = Constants.width - (w.Width / 2);

return w2;

}

//Paints the changed position on a menu after a key has been pushed

public static void paintMenuChange(PaintEventArgs e, ArrayList menu, int previousPosition,

int currentPosition, float height)

{

float menuHeight = getMenuHeight(menu.Count);

height = (height / 2) - (Constants.fontHeight / 2) - 2;

e.Graphics.FillRectangle(Constants.text, new Rectangle(0, ((int)(getPlace(currentPosition, menu.Count, menuHeight, height) - (Constants.fontHeight)) + Constants.fontHeight),

Screen.PrimaryScreen.Bounds.Width, (Constants.fontHeight) + 12));

IEnumerator num = menu.GetEnumerator();

num.MoveNext();

string prevText = "";

string curText = "";

for (int i = 0; i < menu.Count; i++)

{

String tmp = (String)num.Current;

if (i == previousPosition || i == currentPosition)

{

if (currentPosition == i)

{

curText = tmp;

}

else

{

prevText = tmp;

}

}

num.MoveNext();

}

e.Graphics.DrawString(curText, Constants.font, Constants.textSelected, getTextWidth(curText, e.Graphics), getPlace(currentPosition, menu.Count, menuHeight, height));

e.Graphics.FillRectangle(Constants.clearSelection, new Rectangle(0, ((int)(getPlace(previousPosition, menu.Count, menuHeight, height) - (Constants.fontHeight)) + Constants.fontHeight),

Screen.PrimaryScreen.Bounds.Width, (Constants.fontHeight) + 12));

e.Graphics.DrawString(prevText, Constants.font, Constants.text, getTextWidth(prevText, e.Graphics), getPlace(previousPosition, menu.Count, menuHeight, height));

}

//Paints the entire menu

public static void paintMenu(PaintEventArgs e, ArrayList menu, int position, float height)

{

float menuHeight = getMenuHeight(menu.Count);

height = (height / 2) - (Constants.fontHeight / 2) - 2;

e.Graphics.FillRectangle(Constants.text, new Rectangle(0, ((int)(getPlace(position, menu.Count, menuHeight, height) - (Constants.fontHeight)) + Constants.fontHeight),

Screen.PrimaryScreen.Bounds.Width, (Constants.fontHeight) + 12));

IEnumerator num = menu.GetEnumerator();

num.MoveNext();

for (int i = 0; i < menu.Count; i++)

{

String tmp = (String)num.Current;

if (position == i)

{

e.Graphics.DrawString(tmp, Constants.font, Constants.textSelected, getTextWidth(tmp, e.Graphics), getPlace(i, menu.Count, menuHeight, height));

}

else

{

e.Graphics.DrawString(tmp, Constants.font, Constants.text, getTextWidth(tmp, e.Graphics), getPlace(i, menu.Count, menuHeight, height));

}

num.MoveNext();

}

}

//Paints a list on the screen based on an ArrayList

public static void paintArrayList(PaintEventArgs e, ArrayList list, int position, int counter, float height, string cc)

{

int lines = getNumberOfLines(height);

float menuHeight = 0;

if (list.Count < lines)

{

menuHeight = getMenuHeight(list.Count);

}

else

{

menuHeight = lines;

}

e.Graphics.FillRectangle(Constants.text, new Rectangle(0, (5 + ((position - counter) \* (Constants.fontHeight + 12))),

Screen.PrimaryScreen.Bounds.Width, (Constants.fontHeight) + 12));

IEnumerator num = list.GetEnumerator();

num.MoveNext();

SizeF tmp = new SizeF(0F, 0F);

float indent = 0;

for (int i = 0; i < list.Count; i++)

{

string s = "";

if (cc == Constants.class\_keyaction)

{

KeyAction ka = (KeyAction)num.Current;

s = ka.getKeyActionName() + ":";

}

else if (cc == Constants.class\_bluetoothdeviceinfo)

{

BluetoothDeviceInfo bdi = (BluetoothDeviceInfo)num.Current;

s = bdi.DeviceName + ":";

}

else if (cc == Constants.class\_keyconstant)

{

s = (string)num.Current;

}

tmp = e.Graphics.MeasureString(s, Constants.boldFont);

if (indent < tmp.Width)

{

indent = tmp.Width;

}

num.MoveNext();

}

num.Reset();

num.MoveNext();

for (int i = 0; i < list.Count && i <= (lines + counter); i++)

{

string s1 = "";

string s2 = "";

if(cc == Constants.class\_keyaction)

{

KeyAction ka = (KeyAction)num.Current;

s1 = ka.getKeyActionName() + ":";

s2 = ka.getKey();

if((ka.getKeys()).Count > 1)

s2 += "+";

}

else if(cc == Constants.class\_bluetoothdeviceinfo)

{

BluetoothDeviceInfo bdi = (BluetoothDeviceInfo)num.Current;

s1 = bdi.DeviceName + ":";

s2 = Convert.ToString(bdi.DeviceAddress);

}

else if (cc == Constants.class\_keyconstant)

{

s1 = (string)num.Current;

s2 = "";

}

if (i >= counter)

{

if (position == i)

{

e.Graphics.DrawString(s1, Constants.boldFont, Constants.textSelected, 10, (5 + ((i - counter) \* (Constants.fontHeight + 12))));

e.Graphics.DrawString(s2, Constants.font, Constants.textSelected, (indent + 20), (5 + ((i - counter) \* (Constants.fontHeight + 12))));

}

else

{

e.Graphics.DrawString(s1, Constants.boldFont, Constants.text, 10, (5 + ((i - counter) \* (Constants.fontHeight + 12))));

e.Graphics.DrawString(s2, Constants.font, Constants.text, (indent + 20), (5 + ((i - counter) \* (Constants.fontHeight + 12))));

}

}

num.MoveNext();

}

}

//Paints the changes after a button has been pushed

public static void paintArrayListChange(PaintEventArgs e, ArrayList list, int previousPosition,

int currentPosition, int counter, float height, string cc)

{

int lines = getNumberOfLines(height);

float menuHeight = 0;

if (list.Count < lines)

{

menuHeight = getMenuHeight(list.Count);

}

else

{

menuHeight = lines;

}

height = (height / 2) - (Constants.fontHeight / 2) - 2;

e.Graphics.FillRectangle(Constants.text, new Rectangle(0, (5 + ((currentPosition - counter) \* (Constants.fontHeight + 12))),

Screen.PrimaryScreen.Bounds.Width, (Constants.fontHeight) + 12));

IEnumerator num = list.GetEnumerator();

num.MoveNext();

SizeF tmp = new SizeF(0F, 0F);

float indent = 0;

string curText1 = "";

string curText2 = "";

string prevText1 = "";

string prevText2 = "";

for (int i = counter; i < list.Count; i++)

{

string s = "";

if (cc == Constants.class\_keyaction)

{

KeyAction ka = (KeyAction)num.Current;

s = ka.getKeyActionName() + ":";

}

else if (cc == Constants.class\_bluetoothdeviceinfo)

{

BluetoothDeviceInfo bdi = (BluetoothDeviceInfo)num.Current;

s = bdi.DeviceName + ":";

}

else if (cc == Constants.class\_keyconstant)

{

s = (string)num.Current;

}

tmp = e.Graphics.MeasureString(s, Constants.font);

if (indent < tmp.Width)

{

indent = tmp.Width;

}

num.MoveNext();

}

num.Reset();

num.MoveNext();

for (int i = 0; i < list.Count && i <= (lines + counter); i++)

{

if (i >= counter)

{

if (i == currentPosition)

{

if (cc == Constants.class\_keyaction)

{

curText1 = ((KeyAction)num.Current).getKeyActionName() + ":";

curText2 = ((KeyAction)num.Current).getKey();

if ((((KeyAction)num.Current).getKeys()).Count > 1)

curText2 += "+";

}

else if (cc == Constants.class\_bluetoothdeviceinfo)

{

curText1 = ((BluetoothDeviceInfo)num.Current).DeviceName + ":";

curText2 = Convert.ToString(((BluetoothDeviceInfo)num.Current).DeviceAddress);

}

else if (cc == Constants.class\_keyconstant)

{

curText1 = (string)num.Current;

curText2 = "";

}

}

else if (i == previousPosition)

{

if (cc == Constants.class\_keyaction)

{

prevText1 = ((KeyAction)num.Current).getKeyActionName() + ":";

prevText2 = ((KeyAction)num.Current).getKey();

if ((((KeyAction)num.Current).getKeys()).Count > 1)

prevText2 += "+";

}

else if (cc == Constants.class\_bluetoothdeviceinfo)

{

prevText1 = ((BluetoothDeviceInfo)num.Current).DeviceName + ":";

prevText2 = Convert.ToString(((BluetoothDeviceInfo)num.Current).DeviceAddress);

}

else if (cc == Constants.class\_keyconstant)

{

prevText1 = (string)num.Current;

prevText2 = "";

}

}

}

num.MoveNext();

}

e.Graphics.FillRectangle(Constants.clearSelection, new Rectangle(0, (5 + ((previousPosition - counter) \* (Constants.fontHeight + 12))),

Screen.PrimaryScreen.Bounds.Width, (Constants.fontHeight) + 12));

e.Graphics.DrawString(prevText1, Constants.boldFont, Constants.text, 10, (5 + ((previousPosition - counter) \* (Constants.fontHeight + 12))));

e.Graphics.DrawString(prevText2, Constants.font, Constants.text, (indent + 20), (5 + ((previousPosition - counter) \* (Constants.fontHeight + 12))));

e.Graphics.FillRectangle(Constants.text, new Rectangle(0, (5 + ((currentPosition - counter) \* (Constants.fontHeight + 12))),

Screen.PrimaryScreen.Bounds.Width, (Constants.fontHeight) + 12));

e.Graphics.DrawString(curText1, Constants.boldFont, Constants.textSelected, 10, (5 + ((currentPosition - counter) \* (Constants.fontHeight + 12))));

e.Graphics.DrawString(curText2, Constants.font, Constants.textSelected, (indent + 20), (5 + ((currentPosition - counter) \* (Constants.fontHeight + 12))));

}

//Paints the RemoteControlGui

public static void paintRemoteControlGui(PaintEventArgs e, float[,] place, ArrayList keys, int numOfLines, Bitmap[] bitmapList)

{

IEnumerator enumerator = keys.GetEnumerator();

enumerator.MoveNext();

for (int i = 0; i < keys.Count; i++)

{

KeyAction ka = (KeyAction)enumerator.Current;

string keytemp = ka.getKey();

if ((ka.getKeys()).Count > 1)

keytemp += "+";

if (i < 4)

{

e.Graphics.DrawImage(bitmapList[i], (int)place[i, 1], (int)place[i, 0]);

}

else

{

e.Graphics.DrawString(ka.getKeyActionName() + ":", Constants.boldFont, Constants.text, place[i, 1], place[i, 0]);

}

e.Graphics.DrawString(keytemp, Constants.font, Constants.text, (place[i, 1] + place[i, 2]), place[i, 0]);

enumerator.MoveNext();

}

}

//Paints RemoteControlGui when a button has been pushed

public static void paintRemoteControlGuiKeyPressed(PaintEventArgs e, KeyAction ka, int width, int height)

{

if (temp == null || temp != ka)

{

temp = ka;

e.Graphics.FillRectangle(Constants.keyPressed, new Rectangle(0, 0, width, height));

int menuHeight = (height / 2) - (Constants.fontHeight);

menuHeight = menuHeight - (((ka.getKeys()).Count \* ((Constants.fontHeight \* 2) + 4)) / 2);

int counter = 0;

int place = 0;

SizeF w = new SizeF(0, 0);

float w2 = 0;

foreach (string key in ka.getKeys())

{

w = e.Graphics.MeasureString(key, Constants.largeFont);

w2 = Constants.width - (w.Width / 2);

place = menuHeight + (((Constants.fontHeight \* 2) + 10) \* counter);

e.Graphics.DrawString(key, Constants.largeFont, Constants.text, w2, place);

counter++;

}

}

}

public static void changeResetKeyAction()

{

temp = null;

}

//Paints the information screen

public static void paintMessage(PaintEventArgs e, ArrayList list, float height, Bitmap image)

{

float menuHeight = getMenuHeight(list.Count);

e.Graphics.DrawImage(image, (int)(Constants.width - (image.Width / 2)), (Constants.fontHeight \* 2));

height = (height / 2) - (Constants.fontHeight / 2) - 2;

int counter = 0;

foreach (string s in list)

{

e.Graphics.DrawString(s, Constants.boldFont, Constants.text, getTextWidth(s, e.Graphics), getPlace(counter, list.Count, menuHeight, height));

counter++;

}

}

//Paints the edit menu on a KeyAction

public static void paintKeyMenu(PaintEventArgs e, ArrayList menu, KeyAction ka, float height)

{

float menuHeight = getMenuHeight(menu.Count);

string type = "(" + ka.getKeyActionType() + ")";

SizeF w = e.Graphics.MeasureString(ka.getKeyActionName(), Constants.largeUnderlineFont);

SizeF size = e.Graphics.MeasureString(type, Constants.font);

float w2 = Constants.width - ((w.Width / 2) + (size.Width / 2));

float w3 = w2 + (w.Width + 5);

e.Graphics.DrawString(ka.getKeyActionName(), Constants.largeUnderlineFont, Constants.text, w2, 5);

e.Graphics.DrawString(type, Constants.font, Constants.text, w3, (Constants.fontHeight \* 2) + 5);

int counter = menu.Count;

e.Graphics.FillRectangle(Constants.text, new Rectangle(0, (int)(height - ((Constants.fontHeight + 10) \* counter)),

Screen.PrimaryScreen.Bounds.Width, (Constants.fontHeight) + 12));

IEnumerator enumerator = menu.GetEnumerator();

enumerator.MoveNext();

for (int i = 0; i < menu.Count; i++)

{

string s = (string)enumerator.Current;

w = e.Graphics.MeasureString(s, Constants.font);

w2 = Constants.width - (w.Width / 2);

if (i == 0)

e.Graphics.DrawString(s, Constants.font, Constants.textSelected, w2, (height - ((Constants.fontHeight + 10) \* (counter - i))));

else

e.Graphics.DrawString(s, Constants.font, Constants.text, w2, (height - ((Constants.fontHeight + 10) \* (counter - i))));

enumerator.MoveNext();

}

ArrayList keys = ka.getKeys();

float top = (Constants.fontHeight \* 2) + 30;

float bottom = ((Constants.fontHeight + 10) \* menu.Count);

int numberOfLines = getNumberOfLines((height - (top + bottom)));

int numberOfColumns = keys.Count / numberOfLines;

if ((keys.Count % numberOfLines) > 0)

numberOfColumns++;

float indent = 0;

string tmp = "";

foreach(string s2 in keys)

{

w = e.Graphics.MeasureString(s2, Constants.font);

w2 = w.Width + 10;

if(w2 > indent)

indent = w2;

}

float indent2 = (Constants.width -((indent \* (numberOfColumns - 1)) / 2));

enumerator = keys.GetEnumerator();

enumerator.MoveNext();

for (int i = 0; i < numberOfColumns; i++)

{

for (int a = (i \* numberOfLines); a < ((i + 1) \* numberOfLines) && a < keys.Count; a++)

{

tmp = (string)enumerator.Current;

e.Graphics.DrawString(tmp,Constants.font, Constants.text, indent2 + (i \* indent), (top + ((a - (i \* numberOfLines)) \* (Constants.fontHeight + 10))));

enumerator.MoveNext();

}

}

}

//Paints the edit menu on a KeyAction when a button has been pushed

public static void painKeyMenuChange(PaintEventArgs e, ArrayList menu, int currentposition, int previousposition, float height)

{

float menuHeight = getMenuHeight(menu.Count);

int counter = menu.Count;

e.Graphics.FillRectangle(Constants.clearSelection, new Rectangle(0, (int)(height - ((Constants.fontHeight + 10) \* (counter - previousposition))),

Screen.PrimaryScreen.Bounds.Width, (Constants.fontHeight) + 12));

e.Graphics.FillRectangle(Constants.text, new Rectangle(0, (int)(height - ((Constants.fontHeight + 10) \* (counter - currentposition))),

Screen.PrimaryScreen.Bounds.Width, (Constants.fontHeight) + 12));

IEnumerator enumerator = menu.GetEnumerator();

enumerator.MoveNext();

for(int i = 0; i < menu.Count; i++)

{

string s = (string)enumerator.Current;

if (i == currentposition)

{

e.Graphics.DrawString(s, Constants.font, Constants.textSelected, getTextWidth(s, e.Graphics), (height - ((Constants.fontHeight + 10) \* (counter - currentposition))));

}

else if (i == previousposition)

{

e.Graphics.DrawString(s, Constants.font, Constants.text, getTextWidth(s, e.Graphics), (height - ((Constants.fontHeight + 10) \* (counter - previousposition))));

}

enumerator.MoveNext();

}

}

//Shows the textbox for entering the pointer speed value

public static void paintPointerSpeed(TextBox tb, Label label)

{

tb.SelectAll();

tb.Visible = true;

label.Visible = true;

tb.Focus();

}

}

}

### A10.2.2 Package: Controller

#### ProfileAdminCont.cs

using System;

using System.Collections;

using System.Collections.Generic;

using System.Text;

using BTShareMobile.Gui;

using BTShareMobile.appLogic;

using InTheHand.Net.Sockets;

namespace BTShareMobile.Controller

{

//Presents an API to the user interface with the methods available from the domain layer.

public class ProfileAdminCont

{

private MainGui maingui;

private ProfileAdminGui profileadmingui;

private ProfileAdmin profileadmin;

private CustomMapKeys custommapkeys;

public ProfileAdminCont()

{

maingui = GuiSingleton.GetMainGui();

profileadmin = DomainSingleton.GetProfileAdmin(this);

}

public void setProfileAdminGui(ProfileAdminGui pa)

{

profileadmingui = pa;

}

public void setCustomMapKeys(CustomMapKeys cmk)

{

custommapkeys = cmk;

}

//Starts search for stored Bluetooth devices

public void startSearch(string cc)

{

profileadmin.startSearch(cc);

}

//Returns search to CustomMapKeys

public void searchCompletedCustom(BluetoothDeviceInfo[] bdi)

{

profileadmingui.searchCompleted(bdi);

}

//Returns search to MainGui

public void searchCompletedMain(BluetoothDeviceInfo[] bdi)

{

maingui.searchCompleted(bdi);

}

//Gets all stored KeyActions

public void getKeyActions()

{

profileadmin.getKeyActions();

}

//Returns all KeyAction objects

public void returnKeyActions(ArrayList keys)

{

custommapkeys.setKeyActions(keys);

}

//Stops the current database activity

public void stopDBAction()

{

profileadmin.stopDBAction();

}

//Saves the KeyAction in the database

public string saveKeyAction(KeyAction ka)

{

return profileadmin.saveKeyAction(ka);

}

//Updates a KeyAction

public string updateKeyActionValue(KeyAction ka)

{

return profileadmin.updateKeyActionValue(ka);

}

}

}

#### RemoteControlCont.cs

using System;

using System.Collections;

using System.Collections.Generic;

using System.Text;

using InTheHand.Net.Sockets;

using BTShareMobile.Gui;

using BTShareMobile.appLogic;

namespace BTShareMobile.Controller

{

//Presents an API to the user interface with the methods available from the domain layer.

public class RemoteControlCont

{

private Init init;

private RemoteControlGui rcgui;

private MainGui maingui;

private RemoteControl remotecontrol;

public RemoteControlCont()

{

remotecontrol = DomainSingleton.GetRemoteControl(this);

}

public void setInit(Init i)

{

init = i;

}

public void setRemoteControlGui(RemoteControlGui rcg)

{

rcgui = rcg;

}

public void setMainGui(MainGui mg)

{

maingui = mg;

}

//Connects to a server application

public void connect(BluetoothDeviceInfo bdi)

{

remotecontrol.connect(bdi);

}

//Returns the result of the connection, if it was successful or not

public void connectionCompleted(String s)

{

maingui.connectionCompleted(s);

}

//Stops the current connection

public void stopConnect()

{

remotecontrol.stopConnect();

}

//Returns all KeyAction objects stored in the database

public ArrayList getKeyActions()

{

return remotecontrol.getKeyActions();

}

//Disables the cancel button on the loading screen

public void noCancelOpt()

{

init.noCancelOpt();

}

//Sends a KeyAction object over the network

public Boolean setKeyAction(KeyAction ka)

{

return remotecontrol.setKeyAction(ka);

}

//Disconnects from the current connection

public void endRemoteControl()

{

remotecontrol.endRemoteControl();

}

//Returns if the connection has failed

public void connectionFailed(string s)

{

maingui.connectionFailed(s);

}

//Closes all open connections

public void closeConnections()

{

remotecontrol.closeConnections();

}

}

}

### A10.2.3 Package: appLogic

#### Constants.cs

using System;

using System.Collections;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

using BTShareMobile.Gui;

namespace BTShareMobile.appLogic

{

//Contains all the constants for the entire system.

//For example background colour and text colour.

public class Constants

{

//Font constants

public const int fontHeight = 10;

public static Font font = new Font("Arial", fontHeight, FontStyle.Regular);

public static Font largeFont = new Font("Arial", fontHeight \* 2, FontStyle.Bold);

public static Font largeUnderlineFont = new Font("Arial", fontHeight \* 2, FontStyle.Underline);

public static Font boldFont = new Font("Arial", fontHeight, FontStyle.Bold);

//Information messages

public const string error\_connection = "Connection failed";

public const string error\_keyactionsave = "Error saving KeyAction";

public const string error\_keyactionupdate = "Error updating KeyAction";

public const string msg\_keyactionsaved = "KeyAction saved";

public const string msn\_keyactionupdated = "KeyAction updated";

//Network constants

public const string platform = "Win";

public const string connected = "Connected";

public const string disconnected = "Disconnect";

public const string application = "RC";

//Loading constants

public const int loadingbarwidth = 15;

public const int loadingbarheight = 20;

public const string loading\_text = "Please wait";

//Class constants

public const string class\_profileadmingui = "ProfileAdminGui";

public const string class\_maingui = "MainGui";

public const string class\_loading = "Loading";

public const string class\_custommapkeys = "CustomMapKeys";

public const string class\_remotecontrolgui = "RemoteControlGui";

public const string class\_keyaction = "KA";

public const string class\_bluetoothdeviceinfo = "BluetoothDevice";

public const string class\_keyconstant = "KeyConstant";

//Database constant

public const string db\_pass = "oJmmS662lp";

//Image constants

public const string image\_up = "\\Images\\up.png";

public const string image\_down = "\\Images\\down.png";

public const string image\_left = "\\Images\\left.png";

public const string image\_right = "\\Images\\right.png";

public const string image\_info = "\\Images\\info.png";

//Screen constants

public static float width = (Screen.PrimaryScreen.Bounds.Width / 2);

//Colour constants

public static Color textColour = Color.White;

public static Color textSelectedColour = Color.Black;

public static Color selectedColour = Color.White;

public static Color backgroundColour = Color.Blue;

public static Color KeyPressedbackgroundColour = Color.Black;

public static SolidBrush text = new SolidBrush(textColour);

public static SolidBrush textSelected = new SolidBrush(textSelectedColour);

public static SolidBrush selected = new SolidBrush(selectedColour);

public static SolidBrush clearSelection = new SolidBrush(backgroundColour);

public static SolidBrush keyPressed = new SolidBrush(KeyPressedbackgroundColour);

public static Pen current = new Pen(selectedColour);

//Custom map keys constants

public const string keytype\_single = "Single";

public const string keytype\_pressrelease = "Press/Release";

public const string keytype\_progression = "Progression";

public const string keytype\_combination = "Combination";

public static string[] keyTypes = {

keytype\_single,

keytype\_pressrelease,

keytype\_progression,

keytype\_combination

};

public static string[] keyNames = {

"0",

"1",

"2",

"3",

"4",

"5",

"6",

"7",

"8",

"9",

"0",

"a",

"b",

"c",

"d",

"e",

"f",

"g",

"h",

"i",

"j",

"k",

"l",

"m",

"n",

"o",

"p",

"q",

"r",

"s",

"t",

"u",

"v",

"w",

"x",

"y",

"z",

"Up key",

"Down key",

"Left key",

"Right key",

"Up",

"Down",

"Left",

"Right",

"lmouse",

"rmouse",

"ctrl",

"alt",

"tab",

"enter",

"space",

"backspace",

"shift",

"esc",

"windows"

};

}

}

#### DomainSingleton.cs

using System;

using System.Collections.Generic;

using System.Text;

using BTShareMobile.net;

using BTShareMobile.DB;

using BTShareMobile.Controller;

namespace BTShareMobile.appLogic

{

//Ensures that there is only one instance of each domain class.

public sealed class DomainSingleton

{

public static DomainSingleton domainsingleton = null;

public static ProfileAdmin profileadmin = null;

public static DeviceSearch devicesearch = null;

public static DBHandler dbhandler = null;

public static RemoteControl remotecontrol = null;

public static Communication communication = null;

private static readonly object sync = new object();

private DomainSingleton()

{

}

public static DomainSingleton GetDomainSingleton()

{

lock (sync)

{

if (domainsingleton == null)

{

domainsingleton = new DomainSingleton();

}

}

return domainsingleton;

}

public static ProfileAdmin GetProfileAdmin(ProfileAdminCont pacont)

{

lock (sync)

{

if (profileadmin == null)

{

profileadmin = new ProfileAdmin(pacont);

}

}

return profileadmin;

}

public static DeviceSearch GetDeviceSearch()

{

lock (sync)

{

if (devicesearch == null)

{

devicesearch = new DeviceSearch();

}

}

return devicesearch;

}

public static DBHandler GetDBHandler()

{

lock (sync)

{

if (dbhandler == null)

{

dbhandler = new DBHandler();

}

}

return dbhandler;

}

public static RemoteControl GetRemoteControl(RemoteControlCont rcc)

{

lock (sync)

{

if (remotecontrol == null)

{

remotecontrol = new RemoteControl(rcc);

}

}

return remotecontrol;

}

public static Communication GetCommunication(RemoteControl rc)

{

lock (sync)

{

if (communication == null)

{

communication = new Communication(rc);

}

}

return communication;

}

}

}

#### KeyAction.cs

using System;

using System.Collections;

using System.Collections.Generic;

using System.Text;

namespace BTShareMobile.appLogic

{

//Stores the key action for a button on the phone.

public class KeyAction

{

private string KeyActionID, KeyName, KeyActionType;

private int Value;

private ArrayList Keys;

public KeyAction()

{

Keys = new ArrayList();

}

//Creates a new KeyAction based on the parameters sent

public KeyAction(string id, string name, string type, int val, ArrayList actions)

{

KeyActionID = id;

KeyName = name;

KeyActionType = type;

Value = val;

Keys = actions;

}

//Sets the KeyActionID

public void setKeyActionID(string i)

{

KeyActionID = i;

}

//Sets the KeyName

public void setKeyName(string name)

{

KeyName = name;

}

//Sets the KeyActionType

public void setKeyActionType(string type)

{

KeyActionType = type;

}

//Sets the value of the pointer speed

public void setValue(int v)

{

Value = v;

}

//Adds a row in the Key ArrayList

public void setKeyAction(string action)

{

Keys.Add(action);

}

//Returns the KeyActionID

public string getKeyActionID()

{

return KeyActionID;

}

//Returns the KeyName

public string getKeyActionName()

{

return KeyName;

}

//Returns the KeyActionType

public string getKeyActionType()

{

return KeyActionType;

}

//Returns the value of the pointer speed

public int getValue()

{

return Value;

}

//Removes a row in the Keys ArrayList at position

public void removeKeyAt(int position)

{

Keys.RemoveAt(position);

}

//Returns the first value of the Keys ArrayList

public string getKey()

{

if (Keys.Count > 0)

{

IEnumerator enu = Keys.GetEnumerator();

enu.MoveNext();

return (string)enu.Current;

}

else

return "";

}

//Returns the entire ArrayList

public ArrayList getKeys()

{

return Keys;

}

//Creates a string value of all the variables, each value is comma seperated

public string serialize()

{

string s = KeyName;

s += "," + KeyActionType;

s += "," + Value;

if(Keys.Count > 0)

{

IEnumerator enumerator = Keys.GetEnumerator();

enumerator.MoveNext();

for(int i = 0; i < Keys.Count; i++)

{

string key = (string)enumerator.Current;

s += "," + key;

enumerator.MoveNext();

}

}

return s;

}

}

}

#### ProfileAdmin.cs

using System;

using System.Collections;

using System.Collections.Generic;

using System.Text;

using BTShareMobile.Controller;

using BTShareMobile.net;

using InTheHand.Net.Sockets;

using System.Threading;

using BTShareMobile.DB;

using BTShareMobile.Gui;

namespace BTShareMobile.appLogic

{

//Contains the business logic for the profile administrator.

public class ProfileAdmin

{

private ProfileAdminCont pacont;

private DeviceSearch search;

private DBHandler dbhandler;

private string callingclass;

public ProfileAdmin(ProfileAdminCont pac)

{

search = DomainSingleton.GetDeviceSearch();

search.setProfileAdmin(this);

dbhandler = DomainSingleton.GetDBHandler();

dbhandler.setProfileAdmin(this);

pacont = pac;

}

public void setProfileAdminCont(ProfileAdminCont pac)

{

pacont = pac;

}

//Searches for stored bluetooth devices

public void startSearch(string cc)

{

callingclass = cc;

search.startSearch();

}

//Returns the bluetoothdevices found to the correct gui-class

public void searchCompleted(BluetoothDeviceInfo[] bdi)

{

if (callingclass != null)

{

if (callingclass == Constants.class\_profileadmingui)

pacont.searchCompletedCustom(bdi);

else if (callingclass == Constants.class\_maingui)

pacont.searchCompletedMain(bdi);

}

}

//Reads all the KeyAction objects stored in the database

public void getKeyActions()

{

dbhandler.read();

}

//Returns all KeyActions

public void returnKeyActions(ArrayList keys)

{

pacont.returnKeyActions(keys);

}

//Stops the current database action

public void stopDBAction()

{

dbhandler.stopDBAction();

}

//Saves the KeyAction in the database

public string saveKeyAction(KeyAction ka)

{

return dbhandler.saveKeyAction(ka);

}

//Updates the value of the KeyAction

public string updateKeyActionValue(KeyAction ka)

{

return dbhandler.updateKeyActionValue(ka);

}

}

}

#### QueryConstants.cs

using System;

using System.Collections.Generic;

using System.Text;

namespace BTShareMobile.appLogic

{

//Contains all the queries that are used in the system.

class QueryConstants

{

public const string selectkeyaction = "SELECT KeyActionID, KeyActionName, KeyActionType, Value FROM KeyAction";

public const string selectkey = "SELECT [Key] FROM Keys WHERE (FK\_KeysID = ";

public const string deletekey = "DELETE FROM Keys WHERE (FK\_KeysID = ";

public const string selectmaxid = "SELECT MAX(KeysID) AS M FROM Keys";

public const string insertintokey = "INSERT INTO Keys VALUES(";

public const string changevalue1 = "UPDATE KeyAction SET Value = ";

public const string changevalue2 = " WHERE (KeyActionID = ";

public const string changeactiontype1 = "UPDATE KeyAction Set KeyActionType = ";

public const string changeactiontype2 = " WHERE (KeyActionName = ";

}

}

#### RemoteControl.cs

using System;

using System.Collections;

using System.Collections.Generic;

using System.Text;

using System.Threading;

using InTheHand.Net.Sockets;

using BTShareMobile.DB;

using BTShareMobile.Controller;

using BTShareMobile.net;

namespace BTShareMobile.appLogic

{

//Contains the business logic for the remote control.

public class RemoteControl

{

private DBHandler dbhandler;

private RemoteControlCont rccont;

private Communication communication;

public RemoteControl(RemoteControlCont rcc)

{

rccont = rcc;

communication = DomainSingleton.GetCommunication(this);

dbhandler = DomainSingleton.GetDBHandler();

}

public void setRemoteControlCont(RemoteControlCont rcc)

{

rccont = rcc;

}

//Connection the a Bluetooth device

public void connect(BluetoothDeviceInfo bdi)

{

communication.connect(bdi);

}

//Returns if the connection is accepted or not

public void connectionCompleted(String s)

{

rccont.connectionCompleted(s);

}

//Stops the connection

public void stopConnect()

{

communication.stopConnect();

}

//Returns all the KeyActions

public ArrayList getKeyActions()

{

return dbhandler.getKeyActions();

}

//Disables the cancel button on the loading screen

public void noCancelOpt()

{

rccont.noCancelOpt();

}

//Sends the KeyAction object over the network and to the server application

public Boolean setKeyAction(KeyAction ka)

{

string keyaction = ka.serialize();

int l = keyaction.Length;

string length = Convert.ToString(l);

return communication.setKeyAction(length, Constants.class\_keyaction, keyaction);

}

//Disconnects from the current connected device

public void endRemoteControl()

{

communication.endRemoteContol();

}

//Returns if the connection fails

public void connectionFailed(string s)

{

rccont.connectionFailed(s);

}

//Closes all open connections

public void closeConnections()

{

communication.close();

}

}

}

### A10.2.4 Package: net

#### Communication.cs

using System;

using System.Collections.Generic;

using System.Text;

using System.IO;

using System.Threading;

using InTheHand.Net;

using InTheHand.Net.Sockets;

using InTheHand.Net.Bluetooth;

using System.Net.Sockets;

using BTShareMobile.appLogic;

namespace BTShareMobile.net

{

//Handles all the network communication for the remote control.

public class Communication

{

private RemoteControl remotecontrol;

private Thread thread;

private BluetoothDeviceInfo device;

private BluetoothClient client;

private BluetoothAddress btAddress;

private BluetoothEndPoint remoteEnd;

private NetworkStream stream;

private StreamWriter writer;

private StreamReader reader;

private Guid guid = new Guid("9EF2AEC0AFA911dbABBD0800200C9A66");

private Boolean send;

public Communication(RemoteControl rc)

{

remotecontrol = rc;

send = true;

}

public void setRemoteControl(RemoteControl rc)

{

remotecontrol = rc;

}

//Reads one byte at a time from the stream, maximum 12 bytes

public string readByte()

{

char[] a = new char[16];

reader.Read(a, 0, 16);

if (a[0] == null || a[0] == 0 || a[0] == 'C')

{

writer.Write(Constants.application + "<cr>");

writer.Flush();

return (Constants.connected);

}

else

return (Constants.disconnected);

}

//Stops the connection

public void stopConnect()

{

send = false;

close();

}

//Connects to the server application based on the BluetoothDeviceInfo

public void connect(BluetoothDeviceInfo bdi)

{

if (thread != null)

{

send = false;

thread.Abort();

}

send = true;

device = bdi;

thread = new Thread(new ThreadStart(connectPrivate));

thread.Start();

}

//The worker thread will connect in the background while the loading screen is shown

public void connectPrivate()

{

try

{

client = new BluetoothClient();

btAddress = device.DeviceAddress;

remoteEnd = new BluetoothEndPoint(btAddress, guid);

client.Connect(remoteEnd);

stream = client.GetStream();

writer = new StreamWriter(stream);

reader = new StreamReader(stream);

writer.Write(Constants.platform + "<cr>");

writer.Flush();

remotecontrol.noCancelOpt();

if (send)

{

string s = readByte();

if (s == Constants.disconnected)

close();

remotecontrol.connectionCompleted(s);

}

else

{

close();

send = true;

}

}

catch (IOException)

{

remotecontrol.connectionFailed(Constants.error\_connection + "\nIOException");

close();

}

catch (SocketException)

{

remotecontrol.connectionFailed(Constants.error\_connection + "\nSocketException");

close();

}

}

//Tries to close all open streams and connections

public void close()

{

try

{

client.Close();

writer.Close();

reader.Close();

stream.Close();

}

catch (IOException)

{

}

catch (SocketException)

{

}

catch (NullReferenceException)

{

}

}

//Sends a KeyAction object over the network to the server application

public Boolean setKeyAction(string length, string type, string data)

{

try

{

writer.Write(type + "<cr>");

writer.Write(length + "<cr>");

writer.Write(data);

writer.Flush();

return true;

}

catch (IOException)

{

close();

return false;

}

catch (ObjectDisposedException)

{

close();

return false;

}

}

//Disconnects from the server application

public void endRemoteContol()

{

try

{

writer.Write(Constants.disconnected + "<cr>");

writer.Flush();

}

catch (IOException)

{

}

finally

{

try

{

close();

}

catch (IOException)

{

}

}

}

}

}

#### DeviceSearch.cs

using System;

using System.Collections.Generic;

using System.Text;

using BTShareMobile.appLogic;

using InTheHand.Net.Sockets;

using System.Threading;

namespace BTShareMobile.net

{

//Contains the methods for the Bluetooth device search.

//For Windows Mobile only the devices already paired up with Windows will be listed.

public class DeviceSearch

{

private ProfileAdmin profileadmin;

private Thread thread;

private BluetoothDeviceInfo[] devices;

public DeviceSearch()

{

}

public void setProfileAdmin(ProfileAdmin pa)

{

profileadmin = pa;

}

//Creates a new thread that does the search for Bluetooth devices

public void startSearch()

{

thread = new Thread(new ThreadStart(threadSearch));

thread.Start();

}

//Returns a list of BluetoothDeviceInfo to ProfileAdmin

public void threadSearch()

{

BluetoothClient client = new BluetoothClient();

devices = client.DiscoverDevices(100, true, false, false);

profileadmin.searchCompleted(devices);

}

}

}

### A10.2.5 Package: DB

#### DBHandler.cs

using System;

using System.Collections;

using System.Collections.Generic;

using System.Text;

using System.Data.SqlServerCe;

using BTShareMobile.appLogic;

using System.Threading;

namespace BTShareMobile.DB

{

//Handles all operations on the database.

public class DBHandler

{

private ArrayList keys;

private ProfileAdmin profileadmin;

private SqlCeConnection connection;

private SqlCeCommand selectkeyaction;

private SqlCeCommand selectkey;

private SqlCeCommand selectmax;

private SqlCeCommand insert;

private SqlCeCommand delete;

private SqlCeCommand update;

private SqlCeResultSet resultset1;

private SqlCeResultSet resultset2;

private Thread t;

private static readonly object sync = new object();

private static Boolean cont;

public DBHandler()

{

cont = true;

keys = new ArrayList();

}

public void setProfileAdmin(ProfileAdmin pa)

{

profileadmin = pa;

}

public void stopDBAction()

{

lock (sync)

{

cont = false;

}

}

public Boolean getCont()

{

lock (sync)

{

return cont;

}

}

public void startDBAction()

{

lock (sync)

{

cont = true;

}

}

//Opens the database connection

private void openConnection()

{

lock (this)

{

if (connection == null)

{

string path = System.IO.Path.GetDirectoryName(System.Reflection.Assembly.GetExecutingAssembly().GetName().CodeBase);

connection = new SqlCeConnection("Data source = " + path + "\\Data\\KeyActionDB.sdf;" + " Password = " + Constants.db\_pass + ";");

connection.Open();

}

}

}

//Closes the database connection

public void closeConnection()

{

connection.Close();

}

//Returns all KeyAction objects stored in the database

public ArrayList getKeyActions()

{

if(keys.Count == 0)

readAll();

return keys;

}

//Used to read all the KeyAction objects stored in the database. Creates a new thread to handle the

//database work. This is done because this can take some time, at least the first time the connection is opened.

public void read()

{

if (keys.Count == 19)

profileadmin.returnKeyActions(keys);

else

{

t = new Thread(new ThreadStart(readThread));

t.Start();

}

}

//Checks if the KeyActions objects has been retrieved before, if not the are then found in the database.

//If they are already stored in the local variable they are simply sent.

private void readThread()

{

readAll();

if (getCont())

{

int counter = 0;

while(keys.Count != 19 && counter < 5)

{

readAll();

counter++;

}

profileadmin.returnKeyActions(keys);

}

else

startDBAction();

}

//The actual database method. It finds all the rows and creates the KeyAction objects from the data.

private void readAll()

{

if (connection == null)

openConnection();

selectkeyaction = new SqlCeCommand(QueryConstants.selectkeyaction, connection);

resultset1 = selectkeyaction.ExecuteResultSet(ResultSetOptions.Scrollable);

ArrayList tmp = new ArrayList();

if (resultset1.HasRows)

{

int idOrd = resultset1.GetOrdinal("KeyActionID");

int nameOrd = resultset1.GetOrdinal("KeyActionName");

int typeOrd = resultset1.GetOrdinal("KeyActionType");

int valOrd = resultset1.GetOrdinal("Value");

while (resultset1.Read())

{

string id = resultset1.GetString(idOrd);

string name = resultset1.GetString(nameOrd);

string type = resultset1.GetString(typeOrd);

int value = resultset1.GetInt32(valOrd);

selectkey = new SqlCeCommand((QueryConstants.selectkey + "'" + id + "')"), connection);

resultset2 = selectkey.ExecuteResultSet(ResultSetOptions.Scrollable);

ArrayList list = new ArrayList();

if (resultset2.HasRows)

{

while (resultset2.Read())

{

int key = resultset2.GetOrdinal("Key");

list.Add(resultset2.GetString(key));

}

}

tmp.Add(new KeyAction(id, name, type, value, list));

}

}

keys = tmp;

}

//Since the database always will contain 19 KeyActions (one for each key), this method starts with deleting

//the old value. It then continues with sacing the new and modified object into the database.

public string saveKeyAction(KeyAction ka)

{

try

{

if (connection == null)

openConnection();

delete = new SqlCeCommand((QueryConstants.deletekey + "'" + ka.getKeyActionID() + "')"), connection);

delete.ExecuteNonQuery();

selectmax = new SqlCeCommand(QueryConstants.selectmaxid, connection);

resultset1 = selectmax.ExecuteResultSet(ResultSetOptions.Scrollable);

long id = 0;

if (resultset1.HasRows)

{

resultset1.Read();

int i = resultset1.GetOrdinal("M");

id = resultset1.GetInt64(i);

}

id++;

ArrayList tempkeys = ka.getKeys();

foreach (string s in tempkeys)

{

insert = new SqlCeCommand((QueryConstants.insertintokey + id + ", '" + ka.getKeyActionID() + "', '" + s + "')"), connection);

int rows = insert.ExecuteNonQuery();

if (rows != 0)

id++;

}

update = new SqlCeCommand((QueryConstants.changeactiontype1 + "'" + ka.getKeyActionType() + "'" +

QueryConstants.changeactiontype2 + "'" + ka.getKeyActionName() + "')"), connection);

update.ExecuteNonQuery();

return Constants.msg\_keyactionsaved;

}

catch (Exception)

{

return Constants.error\_keyactionsave;

}

}

//Updates the value for the pointer speed

public string updateKeyActionValue(KeyAction ka)

{

try

{

update = new SqlCeCommand((QueryConstants.changevalue1 + ka.getValue() + QueryConstants.changevalue2 + "'" + ka.getKeyActionID() + "')"), connection);

int rows = update.ExecuteNonQuery();

if(rows == 1)

{

return Constants.msn\_keyactionupdated;

}

else

{

return Constants.error\_keyactionupdate;

}

}

catch(Exception)

{

return Constants.error\_keyactionupdate;

}

}

}

}

## A10.3 Server

### A10.3.1 Package: gui

#### Main.java

**package** gui;

**import** appLogic.\* ;

**import** guiController.\* ;

**import** javax.swing.\* ;

**import** javax.swing.table.DefaultTableModel;

**import** java.awt.\* ;

**import** java.awt.event.\* ;

**import** java.util.\* ;

**import** javax.bluetooth.\* ;

//Contains the user interface for the server application.

**public** **class** Main **extends** JFrame **implements** ActionListener {

**private** **static** **final** **long** *serialVersionUID* = 1L;

**private** Controller cont ;

**private** Vector<ConnectionObject> devices ;

**private** DefaultTableModel connectionDtm ;

**private** JTable connectionTable ;

**private** JTable keyTable ;

**private** JScrollPane connectionScroll ;

**private** JScrollPane keyScroll ;

**private** JPopupMenu popup ;

**private** JMenuItem disconnectmenu ;

**private** JMenuItem infomenu ;

**private** MouseListener listener ;

**private** JTextArea netCommunication ;

**private** JScrollPane netScroll ;

**private** Container container ;

**private** GridBagConstraints gbc ;

**private** JButton info ;

**private** JButton exit ;

**public** Main() {

**super**("BTShare server - " + Constants.*SERVER\_STATUS\_REGISTERING*) ;

cont = **new** Controller(**this**) ;

showGui() ;

cont.registerService() ;

}

**public** **void** serverStatus(String status) {

**if**(status.equals(Constants.*SERVER\_STATUS\_STARTED*)) {

setTitle("BTShare server - " + Constants.*SERVER\_STATUS\_STARTED*) ;

}

**else** {

JOptionPane.*showMessageDialog*(**this**, status, "Error", JOptionPane.*ERROR\_MESSAGE*) ;

System.*exit*(1) ;

}

}

**public** **boolean** updateConnection(ConnectionObject conobj) {

BTDevice btd = conobj.getBTDevice() ;

**int** answer = JOptionPane.*showConfirmDialog*(**this**, btd.getName() + ": " + btd.getAddress() +

"\nAllow connection?", "Incoming connection", JOptionPane.*YES\_NO\_OPTION*) ;

**if**(answer == JOptionPane.*YES\_OPTION*) {

devices.addElement(conobj) ;

connectionDtm.addRow(**new** Object[] { btd.getName(), btd.getAddress(), conobj.getID(), btd.getDevicePlatform() }) ;

**return** **true** ;

}

**else** {

**return** **false** ;

}

}

**public** **void** removeConnection(ConnectionObject conobj) {

**int** row = 0 ;

**long** id = conobj.getID() ;

**for**(**int** i = 0; i < devices.size(); i++) {

ConnectionObject temp = (ConnectionObject)devices.elementAt(i) ;

**if**(id == temp.getID()) {

row = i ;

**break** ;

}

}

**if**(row >= 0) {

devices.removeElementAt(row) ;

connectionDtm.removeRow(row) ;

}

}

**public** **void** updateKeyPressed(KeyAction ka) {

netCommunication.append("\nKey Name: " + ka.getKeyName() + " - Key action: " + ka.getKey() + " - Key type: " + ka.getKeyActionType()) ;

netCommunication.setCaretPosition(netCommunication.getText().length()) ;

}

**private** **void** showGui() {

netCommunication = **new** JTextArea() ;

netCommunication.setEditable(**false**) ;

netCommunication.append("Network communication:") ;

netScroll = **new** JScrollPane(netCommunication) ;

netScroll.setPreferredSize(**new** Dimension(netScroll.getPreferredSize().width, 75)) ;

connectionDtm = **new** DefaultTableModel() {

**private** **static** **final** **long** *serialVersionUID* = 1L;

**public** **boolean** isCellEditable(**int** row, **int** col) {

**return** **false** ;

}

};

connectionTable = **new** JTable(connectionDtm) ;

connectionDtm.addColumn("Device name") ;

connectionDtm.addColumn("Device address") ;

connectionDtm.addColumn("Connection ID") ;

connectionDtm.addColumn("Device platform") ;

devices = **new** Vector<ConnectionObject>() ;

container = getContentPane() ;

container.setLayout(**new** GridBagLayout()) ;

gbc = **new** GridBagConstraints() ;

gbc.fill = GridBagConstraints.*BOTH* ;

connectionScroll = **new** JScrollPane(connectionTable) ;

connectionScroll.setPreferredSize(**new** Dimension(connectionScroll.getPreferredSize().width, 150)) ;

keyScroll = **new** JScrollPane(keyTable) ;

keyScroll.setPreferredSize(**new** Dimension(connectionScroll.getPreferredSize().width, 75)) ;

gbc.insets = **new** Insets(2,2,2,2);

gbc.gridwidth = 3 ;

gbc.weightx = 2 ;

gbc.gridx = 0 ;

gbc.gridy = 0 ;

container.add(netScroll, gbc) ;

gbc.gridy = 1 ;

container.add(connectionScroll, gbc) ;

gbc.gridwidth = 1 ;

gbc.weightx = 1 ;

gbc.gridx = 0 ;

gbc.gridy = 2 ;

info = **new** JButton("Info") ;

container.add(info, gbc) ;

gbc.gridx = 1 ;

gbc.gridy = 2 ;

exit = **new** JButton("Exit") ;

container.add(exit, gbc) ;

info.addActionListener(**this**) ;

exit.addActionListener(**this**) ;

popup = **new** JPopupMenu() ;

disconnectmenu = **new** JMenuItem("Disconnect") ;

disconnectmenu.addActionListener(**this**) ;

infomenu = **new** JMenuItem("Device info") ;

infomenu.addActionListener(**this**) ;

popup.add(disconnectmenu) ;

popup.add(infomenu) ;

listener = **new** MouseMenuListener(popup) ;

connectionTable.addMouseListener(listener) ;

setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*) ;

setResizable(**false**) ;

pack();

center(**this**) ;

setVisible(**true**) ;

}

**private** **void** center(JFrame f) {

Dimension screenSize;

Dimension frameSize;

screenSize = Toolkit.*getDefaultToolkit*().getScreenSize();

frameSize = f.getSize();

f.setLocation((screenSize.width - frameSize.width) / 2,(screenSize.height - frameSize.height) / 2);

}

**public** **static** **void** main(String[] args) {

**try** {

LocalDevice local = LocalDevice.*getLocalDevice*() ;

**new** Main() ;

}

**catch**(BluetoothStateException bse) {

}

}

**public** **void** actionPerformed(ActionEvent ae) {

**int** selected = connectionTable.getSelectedRow() ;

**try** {

JButton tmp = (JButton)ae.getSource() ;

**if**(tmp == exit)

System.*exit*(0) ;

**else** **if**(tmp == info) {

**if**(selected != -1) {

ConnectionObject conobj = (ConnectionObject)devices.elementAt(selected) ;

BTDevice b = conobj.getBTDevice() ;

JOptionPane.*showMessageDialog*(**this**, "Device name: " + b.getName() + "\nDevice address: " + b.getAddress(),

"Device information", JOptionPane.*INFORMATION\_MESSAGE*) ;

}

}

}

**catch**(ClassCastException cce) {

**if**(selected != -1) {

**try** {

JMenuItem jmi = (JMenuItem)ae.getSource() ;

**if**(jmi == infomenu) {

ConnectionObject conobj = (ConnectionObject)devices.elementAt(selected) ;

BTDevice b = conobj.getBTDevice() ;

JOptionPane.*showMessageDialog*(**this**, "Device name: " + b.getName() + "\nDevice address: " + b.getAddress(),

"Device information", JOptionPane.*INFORMATION\_MESSAGE*) ;

}

**else** **if**(jmi == disconnectmenu) {

ConnectionObject conobj = (ConnectionObject)devices.elementAt(selected) ;

cont.disconnect(conobj) ;

removeConnection(conobj) ;

}

}

**catch**(ClassCastException cce2) {

}

}

}

}

**private** **class** MouseMenuListener **extends** MouseAdapter {

**private** JPopupMenu menu ;

**public** MouseMenuListener(JPopupMenu jpm) {

menu = jpm ;

}

**public** **void** mousePressed(MouseEvent me) {

popup.show(me.getComponent(), me.getX(), me.getY());

}

**public** **void** mouseReleased(MouseEvent me) {

popup.show(me.getComponent(), me.getX(), me.getY());

}

}

}

### A10.3.2 Package: guiController

#### Controller.java

**package** guiController;

**import** gui.\* ;

**import** net.\* ;

**import** appLogic.\* ;

**import** javax.microedition.io.\* ;

//Presents an API to the user interface with the methods available from the domain layer.

**public** **class** Controller {

**private** Main main ;

**private** RegisterService register ;

**private** CommunicationHandler handler ;

**public** Controller(Main m) {

main = m ;

register = **new** RegisterService(**this**) ;

handler = **new** CommunicationHandler(**this**) ;

}

**public** **void** registerService() {

register.startService() ;

}

**public** **void** serverStart(String status, StreamConnectionNotifier notifier) {

main.serverStatus(status) ;

**if**(notifier != **null**) {

handler.openConnect(notifier) ;

}

}

**public** **boolean** updateConnection(ConnectionObject conobj) {

**return** main.updateConnection(conobj) ;

}

**public** **void** removeConnection(ConnectionObject conobj) {

main.removeConnection(conobj) ;

}

**public** **void** updateKeyPressed(KeyAction ka) {

main.updateKeyPressed(ka) ;

}

**public** **void** disconnect(ConnectionObject conobj) {

handler.disconnect(conobj) ;

}

}

### A10.3.3 Package: appLogic

#### BTDevice.java

**package** appLogic ;

//Stores information about a Bluetooth device.

**public** **class** BTDevice {

**private** String address ;

**private** String name ;

**private** **int** majorClass ;

**private** **int** minorClass ;

**private** **int** serviceClass ;

**private** String platform;

**public** BTDevice() {

address = **null** ;

name = **null** ;

majorClass = 0 ;

minorClass = 0 ;

serviceClass = 0 ;

platform = **null**;

}

**public** BTDevice(String a) {

address = a ;

}

//Get/Set methods

**public** **void** setAddress(String a) {

address = a ;

}

**public** **void** setName(String a) {

name = a ;

}

**public** **void** setMajorClass(**int** a) {

majorClass = a ;

}

**public** **void** setMinorClass(**int** a) {

minorClass = a ;

}

**public** **void** setServiceClass(**int** a) {

serviceClass = a ;

}

**public** **void** setDevicePlatform(String s) {

platform = s;

}

**public** String getAddress() {

**return** address ;

}

**public** String getName() {

**return** name ;

}

**public** **int** getMajorClass() {

**return** majorClass ;

}

**public** **int** getMinorClass() {

**return** minorClass ;

}

**public** **int** getServiceClass() {

**return** serviceClass ;

}

**public** String getDevicePlatform() {

**return** platform ;

}

}

#### ConnectionObject.java

**package** appLogic;

//Stores information about a connection.

//It contains the connection id and the BTDevice object of the connection.

**public** **class** ConnectionObject {

**private** **long** id ;

**private** BTDevice btd ;

**public** ConnectionObject() {

id = 0 ;

btd = **null** ;

}

**public** ConnectionObject(**int** i, BTDevice b) {

id = i ;

btd = b ;

}

**public** **void** setID(**long** i) {

id = i ;

}

**public** **void** setBTDevice(BTDevice b) {

btd = b ;

}

**public** **long** getID() {

**return** id ;

}

**public** BTDevice getBTDevice() {

**return** btd ;

}

}

#### Constants.java

**package** appLogic;

//Contains all regular system constants. For example error messages and the UUID.

**public** **class** Constants **extends** NetConstants {

**public** **static** String *CLASS\_BTDEVICE* = "BTDevice" ;

**public** **static** String *CLASS\_KEYACTION* = "KA" ;

**public** **static** String *SERVER\_STATUS\_STARTED* = "Server started" ;

**public** **static** String *SERVER\_STATUS\_REGISTERING* = "Registering service" ;

**public** **static** String *SERVER\_STATUS\_ERROR* = "Server error" ;

**public** **static** String *UUID* = "9EF2AEC0AFA911dbABBD0800200C9A66" ;

**public** **static** **int** *SLEEP* = 30 ;

**public** **static** **int** *SWITCHPROGRAMS\_SLEEP* = 1500;

**public** **static** String *PLATFORM\_JAVA* = "Java" ;

**public** **static** String *PLATFORM\_WIN* = "Win" ;

}

#### KeyAction.java

**package** appLogic;

**import** java.util.\* ;

//Stores the key action for an action on the server computer.

**public** **class** KeyAction {

//The name of the key

**private** String keyName ;

//The type of action, combination or progression

**private** String keyActionType ;

//The value, for determening the mouse movements

**private** **int** value ;

//The key actions

**private** Vector<String> key ;

**public** KeyAction() {

key = **new** Vector<String>() ;

}

**public** KeyAction(String n, String type, String s) {

key = **new** Vector<String>() ;

keyName = n ;

keyActionType = type ;

key.addElement(s) ;

}

**public** KeyAction(String n, String type, **int** v, String s) {

keyName = n ;

keyActionType = type ;

value = v ;

key = **new** Vector<String>() ;

key.addElement(s) ;

}

**public** **void** setKeyName(String s) {

keyName = s ;

}

**public** **void** setKeyAction(String s) {

key.addElement(s) ;

}

**public** **void** setKeyActionType(String type) {

keyActionType = type ;

}

**public** **void** setValue(**int** v) {

value = v ;

}

**public** String getKeyName() {

**return** keyName ;

}

**public** String getKey() {

**return** (String)key.elementAt(0) ;

}

**public** Vector getKeys() {

**return** key ;

}

**public** String getKeyActionType() {

**return** keyActionType ;

}

**public** **int** getValue() {

**return** value ;

}

**public** **void** removeKeyAt(**int** position) {

key.removeElementAt(position) ;

}

}

#### KeyConstants.java

**package** appLogic;

//Stores the keyboard actions that are possible to use in the application.

**public** **class** KeyConstants {

**public** **static** String *KEY\_PRESS* = "KeyPress" ;

**public** **static** String *KEY\_RELEASE* = "KeyRelease" ;

//Numbers

**public** **static** String *KEY\_0* = "0" ;

**public** **static** String *KEY\_1* = "1" ;

**public** **static** String *KEY\_2* = "2" ;

**public** **static** String *KEY\_3* = "3" ;

**public** **static** String *KEY\_4* = "4" ;

**public** **static** String *KEY\_5* = "5" ;

**public** **static** String *KEY\_6* = "6" ;

**public** **static** String *KEY\_7* = "7" ;

**public** **static** String *KEY\_8* = "8" ;

**public** **static** String *KEY\_9* = "9" ;

//Letters

**public** **static** String *KEY\_A* = "a" ;

**public** **static** String *KEY\_B* = "b" ;

**public** **static** String *KEY\_C* = "c" ;

**public** **static** String *KEY\_D* = "d" ;

**public** **static** String *KEY\_E* = "e" ;

**public** **static** String *KEY\_F* = "f" ;

**public** **static** String *KEY\_G* = "g" ;

**public** **static** String *KEY\_H* = "h" ;

**public** **static** String *KEY\_I* = "i" ;

**public** **static** String *KEY\_J* = "j" ;

**public** **static** String *KEY\_K* = "k" ;

**public** **static** String *KEY\_L* = "l" ;

**public** **static** String *KEY\_M* = "m" ;

**public** **static** String *KEY\_N* = "n" ;

**public** **static** String *KEY\_O* = "o" ;

**public** **static** String *KEY\_P* = "p" ;

**public** **static** String *KEY\_Q* = "q" ;

**public** **static** String *KEY\_R* = "r" ;

**public** **static** String *KEY\_S* = "s" ;

**public** **static** String *KEY\_T* = "t" ;

**public** **static** String *KEY\_U* = "u" ;

**public** **static** String *KEY\_V* = "v" ;

**public** **static** String *KEY\_W* = "w" ;

**public** **static** String *KEY\_X* = "x" ;

**public** **static** String *KEY\_Y* = "y" ;

**public** **static** String *KEY\_Z* = "z" ;

**public** **static** String *KEY\_Æ* = "æ" ;

**public** **static** String *KEY\_Ø* = "ø" ;

**public** **static** String *KEY\_Å* = "å" ;

//Keyboard directional

**public** **static** String *KEY\_KEYBOARDUP* = "Up key" ;

**public** **static** String *KEY\_KEYBOARDDOWN* = "Down key" ;

**public** **static** String *KEY\_KEYBOARDLEFT* = "Left key" ;

**public** **static** String *KEY\_KEYBOARDRIGHT* = "Right key" ;

//Mouse directional

**public** **static** String *KEY\_UP* = "Up" ;

**public** **static** String *KEY\_DOWN* = "Down" ;

**public** **static** String *KEY\_LEFT* = "Left" ;

**public** **static** String *KEY\_RIGHT* = "Right" ;

//Misc

**public** **static** String *KEY\_LEFTMOUSEBUTTON* = "lmouse" ;

**public** **static** String *KEY\_RIGHTMOUSEBUTTON* = "rmouse" ;

**public** **static** String *KEY\_CTRL* = "ctrl" ;

**public** **static** String *KEY\_ALT* = "alt" ;

**public** **static** String *KEY\_TAB* = "tab" ;

**public** **static** String *KEY\_ENTER* = "enter" ;

**public** **static** String *KEY\_SPACE* = "space" ;

**public** **static** String *KEY\_BACKSPACE* = "backspace";

**public** **static** String *KEY\_SHIFT* = "shift" ;

**public** **static** String *KEY\_ESC* = "esc" ;

**public** **static** String *KEY\_WINDOWS* = "windows" ;

}

#### KeyMappings.java

**package** appLogic;

**import** java.util.\* ;

**import** java.awt.event.\* ;

//Contains a hashtable that combines all the key mappings to the key actions.

**public** **class** KeyMappings {

**private** Hashtable<String, Integer> keys ;

**public** KeyMappings() {

keys = **new** Hashtable<String, Integer>() ;

}

**public** Hashtable getKeys() {

keys.put(KeyConstants.*KEY\_0*, KeyEvent.*VK\_0*) ;

keys.put(KeyConstants.*KEY\_1*, KeyEvent.*VK\_1*) ;

keys.put(KeyConstants.*KEY\_2*, KeyEvent.*VK\_2*) ;

keys.put(KeyConstants.*KEY\_3*, KeyEvent.*VK\_3*) ;

keys.put(KeyConstants.*KEY\_4*, KeyEvent.*VK\_4*) ;

keys.put(KeyConstants.*KEY\_5*, KeyEvent.*VK\_5*) ;

keys.put(KeyConstants.*KEY\_6*, KeyEvent.*VK\_6*) ;

keys.put(KeyConstants.*KEY\_7*, KeyEvent.*VK\_7*) ;

keys.put(KeyConstants.*KEY\_8*, KeyEvent.*VK\_8*) ;

keys.put(KeyConstants.*KEY\_9*, KeyEvent.*VK\_9*) ;

keys.put(KeyConstants.*KEY\_A*, KeyEvent.*VK\_A*) ;

keys.put(KeyConstants.*KEY\_B*, KeyEvent.*VK\_B*) ;

keys.put(KeyConstants.*KEY\_C*, KeyEvent.*VK\_C*) ;

keys.put(KeyConstants.*KEY\_D*, KeyEvent.*VK\_D*) ;

keys.put(KeyConstants.*KEY\_E*, KeyEvent.*VK\_E*) ;

keys.put(KeyConstants.*KEY\_F*, KeyEvent.*VK\_F*) ;

keys.put(KeyConstants.*KEY\_G*, KeyEvent.*VK\_G*) ;

keys.put(KeyConstants.*KEY\_H*, KeyEvent.*VK\_H*) ;

keys.put(KeyConstants.*KEY\_I*, KeyEvent.*VK\_I*) ;

keys.put(KeyConstants.*KEY\_J*, KeyEvent.*VK\_J*) ;

keys.put(KeyConstants.*KEY\_K*, KeyEvent.*VK\_K*) ;

keys.put(KeyConstants.*KEY\_L*, KeyEvent.*VK\_L*) ;

keys.put(KeyConstants.*KEY\_M*, KeyEvent.*VK\_M*) ;

keys.put(KeyConstants.*KEY\_N*, KeyEvent.*VK\_N*) ;

keys.put(KeyConstants.*KEY\_O*, KeyEvent.*VK\_O*) ;

keys.put(KeyConstants.*KEY\_P*, KeyEvent.*VK\_P*) ;

keys.put(KeyConstants.*KEY\_Q*, KeyEvent.*VK\_Q*) ;

keys.put(KeyConstants.*KEY\_R*, KeyEvent.*VK\_R*) ;

keys.put(KeyConstants.*KEY\_S*, KeyEvent.*VK\_S*) ;

keys.put(KeyConstants.*KEY\_T*, KeyEvent.*VK\_T*) ;

keys.put(KeyConstants.*KEY\_U*, KeyEvent.*VK\_U*) ;

keys.put(KeyConstants.*KEY\_V*, KeyEvent.*VK\_V*) ;

keys.put(KeyConstants.*KEY\_W*, KeyEvent.*VK\_W*) ;

keys.put(KeyConstants.*KEY\_X*, KeyEvent.*VK\_X*) ;

keys.put(KeyConstants.*KEY\_Y*, KeyEvent.*VK\_Y*) ;

keys.put(KeyConstants.*KEY\_Z*, KeyEvent.*VK\_Z*) ;

keys.put(KeyConstants.*KEY\_KEYBOARDUP*, KeyEvent.*VK\_UP*) ;

keys.put(KeyConstants.*KEY\_KEYBOARDDOWN*, KeyEvent.*VK\_DOWN*) ;

keys.put(KeyConstants.*KEY\_KEYBOARDLEFT*, KeyEvent.*VK\_LEFT*) ;

keys.put(KeyConstants.*KEY\_KEYBOARDRIGHT*, KeyEvent.*VK\_RIGHT*) ;

keys.put(KeyConstants.*KEY\_UP*, -1) ;

keys.put(KeyConstants.*KEY\_DOWN*, -1) ;

keys.put(KeyConstants.*KEY\_LEFT*, -1) ;

keys.put(KeyConstants.*KEY\_RIGHT*, -1) ;

keys.put(KeyConstants.*KEY\_LEFTMOUSEBUTTON*, -1) ;

keys.put(KeyConstants.*KEY\_RIGHTMOUSEBUTTON*, -1) ;

keys.put(KeyConstants.*KEY\_CTRL*, KeyEvent.*VK\_CONTROL*) ;

keys.put(KeyConstants.*KEY\_ALT*, KeyEvent.*VK\_ALT*) ;

keys.put(KeyConstants.*KEY\_TAB*, KeyEvent.*VK\_TAB*) ;

keys.put(KeyConstants.*KEY\_ENTER*, KeyEvent.*VK\_ENTER*) ;

keys.put(KeyConstants.*KEY\_SPACE*, KeyEvent.*VK\_SPACE*) ;

keys.put(KeyConstants.*KEY\_BACKSPACE*, KeyEvent.*VK\_BACK\_SPACE*) ;

keys.put(KeyConstants.*KEY\_SHIFT*, KeyEvent.*VK\_SHIFT*) ;

keys.put(KeyConstants.*KEY\_ESC*, KeyEvent.*VK\_ESCAPE*) ;

keys.put(KeyConstants.*KEY\_WINDOWS*, KeyEvent.*VK\_WINDOWS*) ;

**return** keys ;

}

}

#### NetConstants.java

**package** appLogic;

//Stores constants that are used over the network.

**public** **class** NetConstants {

**public** **static** String *REMOTECONTROL\_CONNECTED* = "Connected" ;

**public** **static** String *REMOTECONTROL\_DISCONNECT* = "Disconnect" ;

**public** **static** String *APPLICATION\_REMOTECONTROL* = "RC" ;

//Key constants

**public** **static** String *KEY\_ACTION\_TYPE\_SINGLE* = "Single" ;

**public** **static** String *KEY\_ACTION\_TYPE\_PROGRESSION* = "Progression" ;

**public** **static** String *KEY\_ACTION\_TYPE\_COMBINATION* = "Combination" ;

**public** **static** String *KEY\_ACTION\_TYPE\_PRESS\_RELEASE* = "Press/Release" ;

**public** **static** String[] *KEY\_ACTION\_TYPES* = {

*KEY\_ACTION\_TYPE\_PROGRESSION*,

*KEY\_ACTION\_TYPE\_COMBINATION*,

*KEY\_ACTION\_TYPE\_SINGLE*,

*KEY\_ACTION\_TYPE\_PRESS\_RELEASE*

} ;

}

#### ObjectSerializing.java

**package** appLogic ;

**import** interfaces.\* ;

**import** java.io.\* ;

//Restores byte[] to the correct object.

**public** **class** ObjectSerializing {

**public** ObjectSerializing() {

}

**public** Object restore(**byte**[] data) **throws** IOException {

ByteArrayInputStream bin = **new** ByteArrayInputStream(data) ;

DataInputStream din = **new** DataInputStream(bin) ;

**try** {

String type = din.readUTF() ;

**int** len = din.readInt() ;

**byte**[] b = **new** **byte**[len] ;

din.readFully(b) ;

Object o ;

**if**(type.equals(Constants.*CLASS\_BTDEVICE*)) {

o = **new** BTDevice() ;

((RMSSerial)o).restore(b) ;

**return** o ;

}

**else** **if**(type.equals(Constants.*CLASS\_KEYACTION*)) {

o = **new** KeyAction() ;

((RMSSerial)o).restore(b) ;

**return** o ;

}

**else** {

**return** **null** ;

}

}

**catch**(IOException io) {

**return** **null** ;

}

}

}

### A10.3.4 Package: net

#### CommunicationHandler.java

**package** net;

**import** guiController.\* ;

**import** appLogic.\* ;

**import** winAppLogic.\* ;

**import** java.util.\* ;

**import** javax.bluetooth.\* ;

**import** javax.microedition.io.\* ;

**import** java.io.\* ;

//Registers all incoming connections and creates new threads for each connection.

**public** **class** CommunicationHandler {

**private** Controller cont ;

**private** Control control ;

**private** StreamConnectionNotifier notifier ;

**private** Hashtable<Long, CommunicationThread> threads ;

**private** Hashtable<Long, CommunicationThreadWin> threadsWin ;

**public** CommunicationHandler(Controller c) {

cont = c ;

control = **new** Control() ;

threads = **new** Hashtable<Long, CommunicationThread>() ;

threadsWin = **new** Hashtable<Long, CommunicationThreadWin>() ;

}

**public** **void** remoteControlCommunication(KeyAction ka) {

cont.updateKeyPressed(ka) ;

control.setKeyAction(ka) ;

}

**public** **synchronized** **void** openConnect(StreamConnectionNotifier n) {

notifier = n ;

**while**(**true**) {

**try** {

StreamConnection con = notifier.acceptAndOpen() ;

InputStream in = con.openInputStream() ;

StreamReader sr = **new** StreamReader(in);

String platform = sr.getString();

RemoteDevice rd = RemoteDevice.*getRemoteDevice*(con) ;

BTDevice btd = **new** BTDevice() ;

btd.setAddress(rd.getBluetoothAddress()) ;

btd.setName(rd.getFriendlyName(**false**)) ;

**if**(platform.equals(Constants.*PLATFORM\_JAVA*)) {

btd.setDevicePlatform(Constants.*PLATFORM\_JAVA*) ;

ConnectionObject conobj = **new** ConnectionObject() ;

conobj.setBTDevice(btd) ;

CommunicationThread ct = **new** CommunicationThread(con, conobj, **this**, cont) ;

threads.put(ct.getId(), ct) ;

}

**else** **if**(platform.equals(Constants.*PLATFORM\_WIN*)) {

btd.setDevicePlatform(Constants.*PLATFORM\_WIN*) ;

ConnectionObject conobj = **new** ConnectionObject() ;

conobj.setBTDevice(btd) ;

CommunicationThreadWin ctw = **new** CommunicationThreadWin(con, conobj, **this**, cont);

threadsWin.put(ctw.getId(), ctw);

}

}

**catch**(IOException io) {

**break** ;

}

}

}

**public** **void** disconnect(ConnectionObject conobj) {

**long** id = conobj.getID() ;

BTDevice btd = conobj.getBTDevice();

**if**((btd.getDevicePlatform()).equals(Constants.*PLATFORM\_JAVA*)) {

CommunicationThread ct = threads.get(id) ;

ct.disconnect() ;

threads.remove(id) ;

}

**else** **if**((btd.getDevicePlatform()).equals(Constants.*PLATFORM\_WIN*)) {

CommunicationThreadWin ctw = threadsWin.get(id) ;

ctw.disconnect() ;

threadsWin.remove(id) ;

}

}

}

#### CommunicationThread.java

**package** net;

**import** appLogic.\* ;

**import** guiController.\* ;

**import** java.io. \*;

**import** javax.microedition.io.\* ;

//A communication thread for a J2ME connection.

**public** **class** CommunicationThread **extends** Thread {

**private** CommunicationHandler handler ;

**private** ObjectSerializing serial ;

**private** Controller cont ;

**private** ConnectionObject conobj ;

**private** StreamConnection con ;

**private** DataInputStream in ;

**private** DataOutputStream out ;

**private** **boolean** connected ;

**public** CommunicationThread(StreamConnection sc, ConnectionObject co, CommunicationHandler ch, Controller c) {

con = sc ;

conobj = co ;

handler = ch ;

cont = c ;

connected = **true** ;

serial = **new** ObjectSerializing() ;

start();

}

**public** **void** disconnect() {

connected = **false** ;

}

**private** **void** openOutputStream() **throws** IOException {

out = con.openDataOutputStream() ;

}

**private** **void** openInputStream() **throws** IOException {

in = con.openDataInputStream() ;

}

**private** **void** closeConnections() **throws** IOException {

in.close() ;

out.close() ;

con.close() ;

handler.disconnect(conobj) ;

}

**private** **void** netCommunicationShutDown() {

**try** {

out.writeUTF(Constants.*REMOTECONTROL\_DISCONNECT*) ;

out.flush();

closeConnections() ;

}

**catch**(IOException io) {

}

}

**private** **void** netCommunication() {

**try** {

out.writeUTF(Constants.*REMOTECONTROL\_CONNECTED*) ;

out.flush();

String application = in.readUTF() ;

**if**(application.equals(Constants.*APPLICATION\_REMOTECONTROL*)) {

remoteControlCommunication() ;

}

}

**catch**(IOException io) {

}

}

**private** **void** remoteControlCommunication() {

**try** {

**while**(connected) {

String type = in.readUTF() ;

**if**(type.equals(Constants.*CLASS\_KEYACTION*) && connected) {

**int** length = in.readInt() ;

**byte**[] data = **new** **byte**[length] ;

in.readFully(data) ;

KeyAction ka = (KeyAction)serial.restore(data) ;

handler.remoteControlCommunication(ka) ;

}

**else** **if**(type.equals(Constants.*REMOTECONTROL\_DISCONNECT*)) {

cont.disconnect(conobj) ;

cont.removeConnection(conobj) ;

connected = **false** ;

**break** ;

}

}

closeConnections() ;

}

**catch**(IOException io) {

**try** {

closeConnections() ;

}

**catch**(IOException io2) {

}

}

}

**public** **void** run() {

**try** {

openInputStream() ;

openOutputStream() ;

conobj.setID(**this**.getId()) ;

**if**(cont.updateConnection(conobj)) {

netCommunication() ;

}

**else** {

netCommunicationShutDown() ;

}

}

**catch**(IOException io) {

}

}

}

#### Control.java

**package** net;

**import** appLogic.\* ;

**import** java.util.\* ;

**import** java.awt.\* ;

**import** java.awt.event.\* ;

//Transforms all received KeyAction objects to the correct action at the server computer.

**public** **class** Control {

**private** Robot robot ;

**private** KeyMappings km ;

**private** Hashtable keymap ;

**private** **int** pressrelease ;

**private** MouseMovementThread mmt ;

**private** SwitchProgramsThread spt;

**private** **boolean** interrupt, switchprograms ;

**public** Control() {

**try** {

robot = **new** Robot() ;

keymap = **new** Hashtable() ;

km = **new** KeyMappings() ;

keymap = km.getKeys() ;

pressrelease = 0 ;

}

**catch**(AWTException awte) {

}

interrupt = **false** ;

switchprograms = **false**;

}

**private** **synchronized** **void** setMouseAction(KeyAction ka) {

String key = ka.getKey() ;

**int** amount = ka.getValue() ;

PointerInfo pointer = MouseInfo.*getPointerInfo*() ;

Point location = pointer.getLocation() ;

**if**(key.equals(KeyConstants.*KEY\_UP*)) {

robot.mouseMove(location.x, location.y - amount) ;

}

**else** **if**(key.equals(KeyConstants.*KEY\_DOWN*)) {

robot.mouseMove(location.x, location.y + amount) ;

}

**else** **if**(key.equals(KeyConstants.*KEY\_LEFT*)) {

robot.mouseMove(location.x - amount, location.y) ;

}

**else** **if**(key.equals(KeyConstants.*KEY\_RIGHT*)) {

robot.mouseMove(location.x + amount, location.y) ;

}

**else** **if**(key.equals(KeyConstants.*KEY\_LEFTMOUSEBUTTON*)) {

robot.mousePress(InputEvent.*BUTTON1\_MASK*) ;

robot.mouseRelease(InputEvent.*BUTTON1\_MASK*) ;

}

**else** **if**(key.equals(KeyConstants.*KEY\_RIGHTMOUSEBUTTON*)) {

robot.mousePress(InputEvent.*BUTTON3\_MASK*) ;

robot.mouseRelease(InputEvent.*BUTTON3\_MASK*) ;

}

}

**private** **synchronized** **void** setMouseMovementPressRelease(KeyAction ka) {

Vector keys = ka.getKeys() ;

String button = (String)keys.elementAt(1) ;

**if**((mmt == **null** || !mmt.isAlive()) && button.equals(KeyConstants.*KEY\_PRESS*)) {

mmt = **new** MouseMovementThread(ka) ;

interrupt = **false** ;

}

**else** **if**(mmt.isAlive() && button.equals(KeyConstants.*KEY\_RELEASE*)){

mmt.interrupt() ;

interrupt = **true** ;

}

}

**private** **void** setMouseButtonPressRelease(KeyAction ka) {

String key = ka.getKey() ;

**if**(key.equals(KeyConstants.*KEY\_LEFTMOUSEBUTTON*)) {

**if**(pressrelease == 0) {

robot.mousePress(InputEvent.*BUTTON1\_MASK*) ;

pressrelease++ ;

}

**else** {

robot.mouseRelease(InputEvent.*BUTTON1\_MASK*) ;

pressrelease = 0 ;

}

}

}

**private** **void** keyPressSingle(**int** keycode) {

robot.keyPress(keycode) ;

robot.delay(20);

robot.keyRelease(keycode) ;

}

**private** **synchronized** **void** keyPressProgression(KeyAction ka) {

**if**((ka.getKeys()).size() > 1) {

Vector keys = ka.getKeys() ;

**for**(**int** i = 0; i < keys.size(); i++) {

**int** keycode = (Integer)keymap.get((String)keys.elementAt(i)) ;

robot.keyPress(keycode) ;

robot.delay(20);

robot.keyRelease(keycode) ;

}

}

}

**private** **synchronized** **void** keyPressCombination(KeyAction ka) {

**if** ((ka.getKeys()).size() > 1) {

Vector keys = ka.getKeys();

**for** (**int** i = 0; i < keys.size(); i++) {

**int** keycode = (Integer) keymap.get((String) keys.elementAt(i));

robot.keyPress(keycode);

}

robot.delay(20);

**for** (**int** i = (keys.size() - 1); i >= 0; i--) {

**int** keycode = (Integer) keymap.get((String) keys.elementAt(i));

robot.keyRelease(keycode);

}

}

}

**private** **synchronized** **void** switchPrograms(KeyAction ka) {

Vector v = ka.getKeys();

**int** keycode1 = (Integer)keymap.get((String)v.elementAt(0)) ;

**int** keycode2 = (Integer)keymap.get((String)v.elementAt(1)) ;

**if**(!switchprograms) {

robot.keyPress(keycode1) ;

robot.keyPress(keycode2) ;

**if**(spt !=**null** && spt.isAlive())

spt.interrupt();

spt = **new** SwitchProgramsThread(keycode1) ;

spt.start();

robot.delay(20) ;

robot.keyRelease(keycode2) ;

switchprograms = **true** ;

}

**else** {

**if**(spt !=**null** && spt.isAlive())

spt.interrupt();

spt = **new** SwitchProgramsThread(keycode1) ;

spt.start();

robot.keyPress(keycode2) ;

robot.delay(20) ;

robot.keyRelease(keycode2) ;

}

}

**protected** **synchronized** **void** setKeyAction(KeyAction ka) {

String key = ka.getKey() ;

**int** keycode = (Integer)keymap.get(key) ;

String keyActionType = ka.getKeyActionType() ;

**if**(keycode == -1) {

**if**(keyActionType.equals(Constants.*KEY\_ACTION\_TYPE\_SINGLE*)) {

setMouseAction(ka) ;

}

**else** **if**(keyActionType.equals(Constants.*KEY\_ACTION\_TYPE\_PRESS\_RELEASE*) && key.equals(KeyConstants.*KEY\_LEFTMOUSEBUTTON*)) {

setMouseButtonPressRelease(ka) ;

}

**else** **if**(keyActionType.equals(Constants.*KEY\_ACTION\_TYPE\_PRESS\_RELEASE*) && (key.equals(KeyConstants.*KEY\_UP* ) || key.equals(KeyConstants.*KEY\_DOWN*)

|| key.equals(KeyConstants.*KEY\_LEFT*) || key.equals(KeyConstants.*KEY\_RIGHT*))) {

setMouseMovementPressRelease(ka) ;

}

}

**else** {

**if**(keyActionType.equals(Constants.*KEY\_ACTION\_TYPE\_SINGLE*)) {

keyPressSingle(keycode) ;

}

**else** **if**(keyActionType.equals(Constants.*KEY\_ACTION\_TYPE\_PROGRESSION*)) {

keyPressProgression(ka) ;

}

**else** **if**(keyActionType.equals(Constants.*KEY\_ACTION\_TYPE\_COMBINATION*)) {

Vector v = ka.getKeys() ;

String s1 = "";

String s2 = "";

**if**(v.size() == 2) {

s1 = (String)v.elementAt(0) ;

s2 = (String)v.elementAt(1) ;

}

**if**(s1.equals(KeyConstants.*KEY\_ALT*) && s2.equals(KeyConstants.*KEY\_TAB*)) {

switchPrograms(ka) ;

}

**else**

keyPressCombination(ka) ;

}

}

}

**private** **class** MouseMovementThread **extends** Thread {

**private** KeyAction ka ;

**private** **int** counter ;

**public** MouseMovementThread(KeyAction k) {

ka = k ;

counter = 0 ;

start() ;

}

**public** **void** run() {

String key = ka.getKey() ;

**int** amount = ka.getValue() ;

**try** {

**while**(counter < 200 && !interrupt) {

PointerInfo pointer = MouseInfo.*getPointerInfo*() ;

Point location = pointer.getLocation() ;

**if**(key.equals(KeyConstants.*KEY\_UP*)) {

robot.mouseMove(location.x, location.y - amount) ;

}

**else** **if**(key.equals(KeyConstants.*KEY\_DOWN*)) {

robot.mouseMove(location.x, location.y + amount) ;

}

**else** **if**(key.equals(KeyConstants.*KEY\_LEFT*)) {

robot.mouseMove(location.x - amount, location.y) ;

}

**else** **if**(key.equals(KeyConstants.*KEY\_RIGHT*)) {

robot.mouseMove(location.x + amount, location.y) ;

}

Thread.*sleep*(Constants.*SLEEP*) ;

counter++ ;

}

}

**catch**(InterruptedException ie) {

interrupt = **false** ;

}

}

}

**private** **class** SwitchProgramsThread **extends** Thread {

**int** keycode;

**public** SwitchProgramsThread(**int** i) {

keycode = i;

}

**public** **void** run() {

**try** {

Thread.*sleep*(Constants.*SWITCHPROGRAMS\_SLEEP*) ;

robot.keyRelease(keycode) ;

switchprograms = **false**;

}

**catch**(InterruptedException ie) {

}

}

}

}

#### RegisterService.java

**package** net;

**import** appLogic.\* ;

**import** guiController.\* ;

**import** java.io.\* ;

**import** javax.bluetooth.\* ;

**import** javax.microedition.io.\* ;

//Registers the remote control service that

//makes it possible for clients to connect to the server application.

**public** **class** RegisterService **extends** Thread {

**private** Controller cont ;

**private** UUID uuid ;

**private** LocalDevice local ;

**private** StreamConnectionNotifier notifier ;

**private** ServiceRecord record ;

**private** DataElement data ;

**public** RegisterService(Controller c) {

cont = c ;

uuid = **new** UUID(Constants.*UUID*, **false**) ;

}

**public** **void** startService() {

Thread t = **new** Thread(**this**) ;

t.start() ;

}

**private** **synchronized** **void** createService() {

**try** {

local = LocalDevice.*getLocalDevice*() ;

notifier = (StreamConnectionNotifier)Connector.*open*("btspp://localhost:" + uuid.toString() + ";name:BTShare") ;

record = local.getRecord(notifier) ;

record.setAttributeValue(0x0008, **new** DataElement(DataElement.*U\_INT\_1*, 0xFF)) ;

data = **new** DataElement(DataElement.*DATSEQ*) ;

cont.serverStart(Constants.*SERVER\_STATUS\_STARTED*, notifier) ;

}

**catch**(BluetoothStateException bse) {

cont.serverStart(Constants.*SERVER\_STATUS\_ERROR* + ": " + bse.getMessage(), **null**) ;

}

**catch**(IOException io) {

cont.serverStart(Constants.*SERVER\_STATUS\_ERROR* + ": " + io.getMessage(), **null**) ;

}

}

**public** **void** run() {

createService() ;

}

}

### A10.3.5 Package: winAppLogic

#### CommunicationThreadWin.java

**package** winAppLogic;

**import** java.io.\*;

**import** javax.microedition.io.\*;

**import** net.\*;

**import** appLogic.\*;

**import** guiController.\*;

//A communication thread for a Windows Mobile connection.

**public** **class** CommunicationThreadWin **extends** Thread {

**private** CommunicationHandler handler ;

**private** Controller cont ;

**private** ConnectionObject conobj ;

**private** StreamReader reader;

**private** StreamConnection con ;

**private** DataOutputStream out ;

**private** **boolean** connected ;

**public** CommunicationThreadWin(StreamConnection sc, ConnectionObject co, CommunicationHandler ch, Controller c) {

con = sc ;

conobj = co ;

handler = ch ;

cont = c ;

connected = **true** ;

start();

}

**private** **void** openInputStream() **throws** IOException {

reader = **new** StreamReader(con.openInputStream());

}

**private** **void** openOutputStream() **throws** IOException {

out = con.openDataOutputStream() ;

}

**private** **void** closeConnections() **throws** IOException {

reader.close() ;

out.close() ;

con.close() ;

}

**private** **void** netCommunicationShutDown() {

**try** {

String s = Constants.*REMOTECONTROL\_DISCONNECT* + "<cr>";

**byte**[] data = s.getBytes();

out.write(data);

out.flush();

closeConnections() ;

}

**catch**(IOException io) {

}

}

**private** **void** netCommunication() {

**try** {

String s = Constants.*REMOTECONTROL\_CONNECTED* + "<cr>";

**byte**[] data = s.getBytes();

out.write(data);

out.flush();

String application = reader.getString();

**if**(application.equals(Constants.*APPLICATION\_REMOTECONTROL*)) {

remoteControlCommunication() ;

}

**else** {

disconnect();

}

}

**catch**(IOException io) {

disconnect();

}

}

**private** **void** remoteControlCommunication() {

**try** {

**while**(connected) {

String type = reader.getString();

**if**(type.equals(Constants.*CLASS\_KEYACTION*) && connected) {

String l = reader.getString() ;

**int** length = Integer.*parseInt*(l) ;

String data = reader.getByteArray(length) ;

handler.remoteControlCommunication(reader.getKeyAction(data)) ;

}

**else** **if**(type.equals(Constants.*REMOTECONTROL\_DISCONNECT*)) {

cont.disconnect(conobj) ;

cont.removeConnection(conobj) ;

connected = **false** ;

**break** ;

}

}

closeConnections() ;

}

**catch**(IOException io) {

disconnect();

**try** {

closeConnections() ;

}

**catch**(IOException io2) {

}

}

}

**public** **void** disconnect() {

connected = **false** ;

}

**public** **void** run() {

**try** {

openInputStream() ;

openOutputStream() ;

conobj.setID(**this**.getId()) ;

**if**(cont.updateConnection(conobj)) {

netCommunication() ;

}

**else** {

netCommunicationShutDown() ;

}

}

**catch**(IOException io) {

}

}

}

#### StreamReader.java

**package** winAppLogic;

**import** java.io.\* ;

**import** java.util.StringTokenizer;

**import** appLogic.KeyAction;

//Restores data sent from Windows Mobile to the correct objects or variables.

**public** **class** StreamReader {

**private** InputStream in ;

**private** StringTokenizer st ;

**public** StreamReader(InputStream is) {

in = is ;

}

**private** String readData() **throws** IOException {

**byte**[] a = **new** **byte**[1] ;

String s = "";

**int** counter = 0 ;

**while**(counter < 68) {

in.read(a) ;

String tmp = **new** String(a) ;

s = s + tmp ;

**if**(s.length() >= 5 && (s.substring(s.length() - 4)).equals("<cr>")) {

**break** ;

}

counter++;

}

s = s.substring(0, (s.length() - 4)) ;

**return** s;

}

**public** **int** getInt() **throws** IOException {

String tmp = readData();

**try**{

**int** i = Integer.*parseInt*(tmp);

**return** i;

}

**catch**(NumberFormatException nfe) {

**return** -1;

}

}

//A maximum of 64 bytes, used to find the length of the bytes sent

**public** String getString() **throws** IOException {

**return** readData();

}

//With the length know this is used to return the actual data

**public** String getByteArray(**int** length) **throws** IOException {

**byte**[] data = **new** **byte**[length] ;

in.read(data);

String s = **new** String(data);

**return** s ;

}

**public** **void** close() {

**try** {

in.close() ;

}

**catch**(IOException io) {

}

}

//Return the KeyAction object

**public** KeyAction getKeyAction(String s) {

st = **new** StringTokenizer(s, ",") ;

**int** i = st.countTokens();

**if**(i >= 4) {

KeyAction ka = **new** KeyAction();

ka.setKeyName(st.nextToken()) ;

ka.setKeyActionType(st.nextToken()) ;

ka.setValue(Integer.*parseInt*(st.nextToken())) ;

**while**(st.hasMoreTokens()) {

ka.setKeyAction(st.nextToken()) ;

}

**return** ka;

}

**return** **null**;

}

}

### A10.3.6 Package: interfaces

#### RMSSerial.java

**package** interfaces;

**import** java.io.\* ;

**public** **interface** RMSSerial {

**public** **byte**[] serialize() **throws** IOException ;

**public** **void** restore(**byte**[] data) **throws** IOException ;