Mobile Robotics

Powered by sourceforge

A special thanks to...
Ronny Sandova, Stevens Institute of Technology
Charlie Payne, Pacifik Neotek
Sourceforge

... And to our sponsors
Firepad, industrial strength mobile imaging
Bivalds Målia
Tandregleringen kronan

mobile**Robotics** by Niklas Bivald, Gabriel Lundmark, Johan Johanson and Thomas Erdelyi.

Project rapport: MobileRobotics

Abstract

Mobile Robotics is a project focusing on wireless communication with special focus on Bluetooth and IR. The basic idea is to create a Lego Robot which can be controlled by a regular cellphone assuming it has Bluetooth capabilities. The focus is sighted towards mobile communication and Lego as a visual help to demonstrate the results in a creative and interesting way.

Bluetooth and IR are two different methods to communicate wirelessly. IR stands for Infra Red and requires line-of-sight to work and Bluetooth is using the radio frequency 2.4 GHz and has a range between 10 and 100 meters.

Unfortunately the Lego Mindstorm system is using IR and not Bluetooth or similar radio communication. To solve this we had a cellphone connect to a Handheld device (PDA) attached on the robot and via Bluetooth send commands to it. The PDA will then translate the commands into IR and send it to the Lego RCX (A mini computer which controls the Lego). The RCX then executes it and the robot performs the action.

Table of contents

Introduction

Mobile Robotics is a project focusing on wireless communication with special focus on Bluetooth and IR. The basic idea is to create a Lego Robot which can be controlled by a regular cellphone assuming it has Bluetooth capabilities. The focus is sighted towards mobile communication and Lego as a visual help to demonstrate the results in a creative and interesting way.

Bluetooth and IR are two different methods to communicate wirelessly. IR stands for Infra Red and requires line-of-sight to work and Bluetooth is using the radio frequency 2.4 GHz and has a range between 10 and 100 meters.

Unfortunately the Lego Mindstorm system is using IR and not Bluetooth or a similar radio communication. This means that it has serious limitations of controllability since it requires line-of-sight. We intend to bypass this limitation.

In short the idea is to have a cellphone connect to a Handheld device (PDA) which is located on the robot and via Bluetooth send commands to it. The PDA will then interpret the user commands and forward them through IR to the Lego RCX (a processor which controls the Lego). The RCX then executes the immediate command and the robot performs the action.

We chose this subject due to a rising interest in the upcoming, new wireless technology and because it would be a real challenge. It could also be a chance for us to be able to provide documentation and real-life code to open source programmers since it’s quite difficult to find extensive applicable code examples using Java Micro Edition / Wireless Edition and the PalmOS c++ Bluetooth library. We thought it would be a way to contribute to the community; java, c++ and Lego.

As mentioned earlier there is not much information presently available on programming Bluetooth on the Java Micro Edition and the PalmOS platform - most of all the quality of the sources is varying. Therefore a great deal of time goes to research and trial-and-error. In the Lego community, including Lego Mindstorms (RCX), there is a great deal of good high quality sources.

Questions we had to get answered were a variety of protocols, standards, cross-platform reliability and wireless capabilities. To narrow down these questions we first discussed what standards we were interested in such as Bluetooth, IR and JSR-82[1] and how well these would cooperate when implemented in two different languages on two different platforms. We then look into all information we can find about the subject in forums, developer sites and books.

Most of all it is easy to get help and support in almost every open-source community.

We had knowledge of both java and c++ programming, but to program for, to us, completely different platforms than PC was a challenge.

Method

When we first started this project in October 2004, we had limited knowledge in Java and C++ but rather extensive knowledge in advanced Lego construction. Both programmers knew Java/C++ basics from the start but had no real experiences in the area of software for mobile equipment. Over the time, we’ve gained the experience needed to achieve our goal even though it has been rather testing from time to time.

[1] Bluetooth standard for the Java programming language

In the beginning of the project, we used the Sony Ericsson Development Kit to write and simulate our software, but as the complexity of the software grew and wireless communication was implemented, we had to use a real cellphone to test our software after each compilation. This way the test results are more accurate, but it slows down the process since it takes time to transfer and install the application each time you alter the code. As with the server development, we were lucky to have access to a Palm Pilot as well as a JSR-82 compatible cellphone for extensive testing. The PalmOS platform actually has some nice documentation and support for Bluetooth – you just got to find it. The PalmOS Developer Suite turns all knee-shaking PRC compilation and makefile managing into a cakewalk.

We had great help of volunteers such as Charlie Payne on Pacific Neotek and Ronny Sandova in Stevens Institute of Technology in USA (see the reference section). Without their help and the open source community, this project could not have been realized. Throughout the months we've spend countless hours reading up on official programming language documentation, forums and developer sites from all over the world.

We found out that most of our hours spent on debugging could be solved by simple answers and were caused by simply errors such as erroneously sized variables and typos.

The process of creating a physical product was eased a lot by our engineer's experience with Lego and technical expertise. Even more so, the electrician of the group has made some serious modifications to the hardware that we've used which obviously saved us some time. Almost equally helpful is the Lego Mindstorms community on the web, which has been a great source of inspiration for us.

Special thanks go out to Firepad for providing streaming video software. We've also had great help by our sponsors "Tandregleringen Kronan" and "Bivalds Målia".

Client - Server - Lego communication

The communication involves three parts: the cellphone (client), the handheld (server) and Alessa (the Lego Robot) and two connections: client -> server and server <-> Lego.

> *"Bluetooth and IR are two different methods to communicate wirelessly. IR stands for Infra Red and requires line-of-sight to work and Bluetooth is using the radio frequency 2.4 GHz and has a range between 10 and 100 meters."*

Client –> Server

Client –> Server is a Bluetooth connection and is a one-way communication. The client establishes a connection to the server and then sends commands such as Left, Right, Forward and backwards.

Server <-> Lego

Server <-> Lego is a limited two way IR link. When a command is received from the client the server software interprets it and sends a Lego compliant command through the IR link to the Lego RCX. The RCX then executes the command and the robot moves.

Client Software (Java J2ME, Cellphone)

This section is intended to give an overview of how the client software works. There are two sections. The first one is a theoretical overview which is intended for none-programmers and the second is for programmers who want to get an internal overview on method calls.

Over the months the program has evolved and one can say that it was birth the fifth January 2005 when the main file was first created. Two days later is was tested for the first time. Thanks to Mikael Lenneryd for letting me borrow his Sony Ericsson V800 (cellphone). A month later the code was reformatted and a couple of days after that Bluetooth code was implemented.

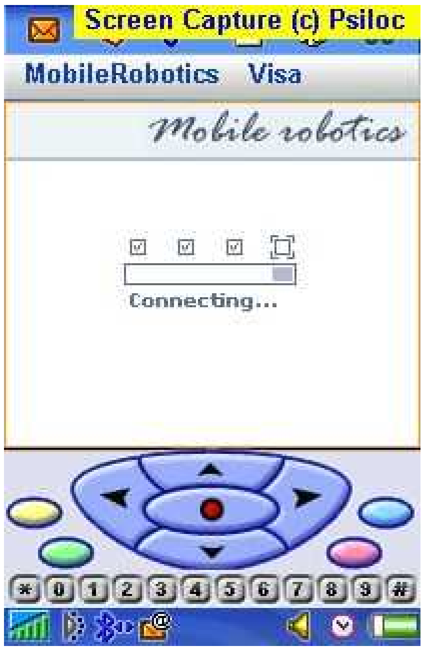
The main concern in the client software was the connection to the handheld. We spent many hours testing and rewriting that particular code-fragment (~500 lines) to finally establish an active connection (2005-04-08).

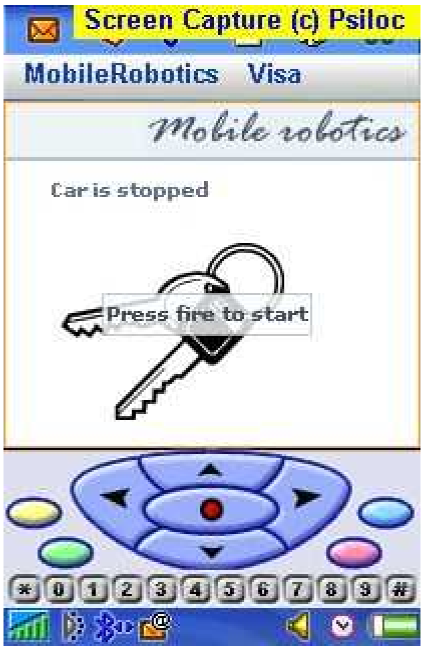
Section one

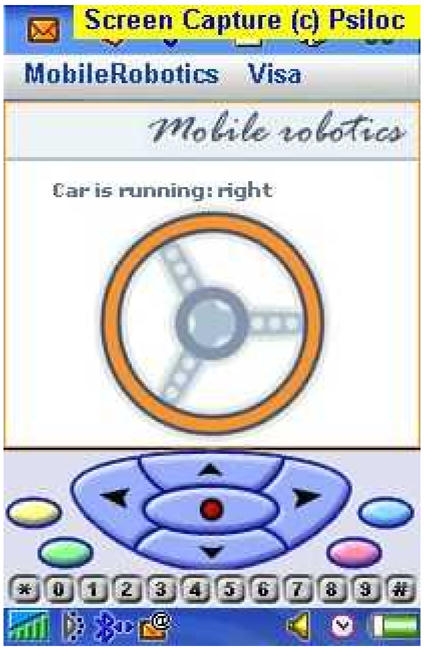
What the client actually does is to establish a Bluetooth connection to our handheld and send command based on users input. When we start the software we are shown a splash screen[2] which displays a loading bar [Image #1].

In the background Bluetooth searches all nearby devices and stores them in its memory. It then searches every device for a matching service[3]. If it finds our handheld it will attempt to establish a connection [Image #1]. When it does we will be taken to the control part from where we can turn on [Image #2] the robot and send commands [Image #3].

The handheld receives the command, looks up the matching IR signal and sends it to the handheld, more about this under Client-Server communication (5.2).







[2] A loading screen.
[3] A Bluetooth service is what kind of actions it may provide, such as file transfers.

Section two

This section is intended for programmers who can follow the method calls throughout the MIDlet.

```
/*********************************************************************************/

 - File created 2005-04-11 by Niklas Bivald, Client software programmer

This file will give you an idea of how the MIDlet work, calling each thread and
similar. See it as a intro and an overview.

Files:
- MobileRoboticsMidlet.java      : Main MIDlet
-  Bluetooth.java                : Handles bluetooth connections
- SplashScren                    : Error report and splash screen
-  ControllRobot.java            : Controlls our robot

What happens when the MIDlet is started, searches connects and sends a command.

 [MobileRoboticsMidlet]

   1 Starts [Bluetooth] Thread.

    2 Calls [Bluetooth].initiate() to open the bluetooth connection

              [Bluetooth.]initiate()

                2.1 Initiates [SplashScreen] - shows the loading bar (Moves the display
                focus to SplashScreen)

                2.2 Search for near bluetooth devices:

                      2.2.1 mobileRoboticsDeviceDiscovery() -  starts the discovery

                     2.2.1.2 deviceDiscovered() - Puts the discovered device in an array

                            2.2.1.3 inquiryCompleted() - Device discovery completed.

                2.3 Search devices for our service
```

```
            2.3.1 Loops our array with devices

            2.2.1.2 searchServices(); - Save the device with our services in a
            string

                   [Abort if no suitable device is found, fatalError()]

           2.4 Opens bluetooth connection - Connector.open(serviceUrl);

                   [Abort if there is an error opening the connection, fatalError()]

        2.5 Opens OutputStream - openOutputStream()

                   [Abort if there is an error opening the OutputStream,
                   fatalError()]

        2.6 Load images - loadImages()

         2.7 Stop splashscreen and return to [MobileRoboticsMidlet]

[We now got a bluetooth connection and we move on to ControllRobot]

3 Starts [ControllRobot] Thread.

4 Calls [ControllRobot].start()

         4.1 Starts our thread and set some values

        4.2 Moves over to run() - Loops a while-syntax as long as the
        application run. The while waits for user input by calling
                   input() and then updates the screen by using drawScreen(Graphic
                   g). Note: run() loops until the program exits.

                     About the run() loop. It got several modes, more about that
                     later on. The example below is for mode 1, controlling the robot
                     with the buttons.

                           4.2.1 Input() - Uses getKeyStates() to check for input. When
                           found the coresponding bluetooth command is sent out by
```

calling bluetooth.command(int command);

The stearing work like this:

If it is currently turning right, and you click left it will go into neutral. Next time you press it will turn left. Same thing when we travel forward. If you press backwards once it goes into neutral. When you press again it will go backwards.

4.2.1.1 [Bluetooth].command() - Starts by checking if we got a connection (connection != null) and that we got an outputStream (os != null). It then compares the sent command (int command) to the last sent command (int lastSentCommand) this is to make sure we doesn't send same commands over and over again.

It then converts the int into a byte (easier for the palm to recieve).

We then write the byte to the buffer by using os.write(cmd) and then we flush the buffer (os.flush()) to send our command. Method is void (returns nothing to input()).

These are our commands:

From Bluetooth.java

```
/*
 * Bluetooth commands and their function (int
 direction)
 *
 *  0) Power On
 *  1) Power Off
 *  2) Forward
*  3) Backward
*  4) Left
*  5) Right
*  6) Stop
*
*  For local use  only:
*  7) Switch (Switch modes (key to pointer))
*  8) Killed (It's powered off)
*/
```

4.2.2 drawScreen(Graphic g) - Updates the screen so that we can see that we are for example turning right.

4.3 About or recordAndRunAfterPointer - You toogle modes using the zero (0) button.

ControllRobot got three modes, 1 + 2, depending on the cellphone. Mode 1 is always the regular controllRobot with the wheel. Mode 2 is the about screen, this is shown for cellphones without pointer (touchscreen) Mode 2 (version 2) is for cellphones with touchscreen,

Let's demonstrate how it works if we switch to mode 1 to mode 2 (for touchscreens).

recordAndRunAfterPointer (not a method name): You can "record" a path by clicking multiple times on the cellphones screen. Then when you press the "Run path" button the robot will follow the directions. It is not precise but it will move in the similar direction.

Mode change: When you press the zero (0) button it is catched by the method keyPressed(int KeyCode)

4.3.1 keyPressed (int keyCode) - We start by checking if it's the right button: if(keyCode == 48) 48 represent zero (cross-platform (on all cellphones)).

If mode is "1" we send the command "6" so it will stop the robot before switching modes. we then set direction (so ControllRobot knows were the robot is heading) to seven as it represents "Switch modes" (as seen above).

After direction is changed and command is sent we set the mode to 2.

[If the direction is 2 (and we want to switch back) we send command 6, direction 6 and finaly mode 1]

Now mode is changed to 2. We then return to the run() loop.

4.4 Back to run() - Every time the loop runs it checks for what mode we are currently is:

if mode is 1: Read input from keys, input() as desribed above

if mode is 2: If we have touchScreen :2v2: Show the recordAndRunAfterPointer (se above) If we haven't:
:2 : Show about screen

Let's look at what each of them do:

4.4.1 mode 1: we've already went through it, input() 4.4.2 mode 2: If we have a pointer screen: -
recordAndRunAfterPointer is not a method name. It's just what we call the function internally.

This method does not use any while-loops to check for key-input. Instead it uses native methods that is called when someone presses the touchscreen.

However the run() loop is still going to update the screen: drawPointerScreen(Graphics g)

When someone clicks on the screen the method pointerPressed(int x,int y) is called (x and y beeing the cordinates).

4.4.2.1 pointerPressed(int x, int y) - First of all the method checks if the mode is 1, if so it means that someone has pressed the screen when it's another mode (steering with buttons). We then simply abort calling return;

Second of all it checks that we pressed inside the drawing area and if we have it set's the variable havePoints to true. This symbols that we have points/path saved. It also stores the x/y parameters in an
array called points[][]. At last it increases int nrpoints with 1.

If we haven't checked inside the drawing area it means that we either pressed someplace along the borders or on the buttons "Run path" or "Clear path".

If we have pressed run path method runPath() is called, If we have pressed clear path method clearPath()
is called.

4.4.2.2 clearPath() - It starts by looping the aray points[][] to null all cordinates. Then it sets havePoints to false and nrpoints to zero.

4.4.2.3 runPath() - Command not yet implented. '

4.4.3 mode 2: If you haven't got a pointer screen - About screen is shown, drawAboutScreen()

4.5 Shutdown - When ordered a shutdown [MobileRoboticsMidlet].exit() is called. This preforms the following tasks.

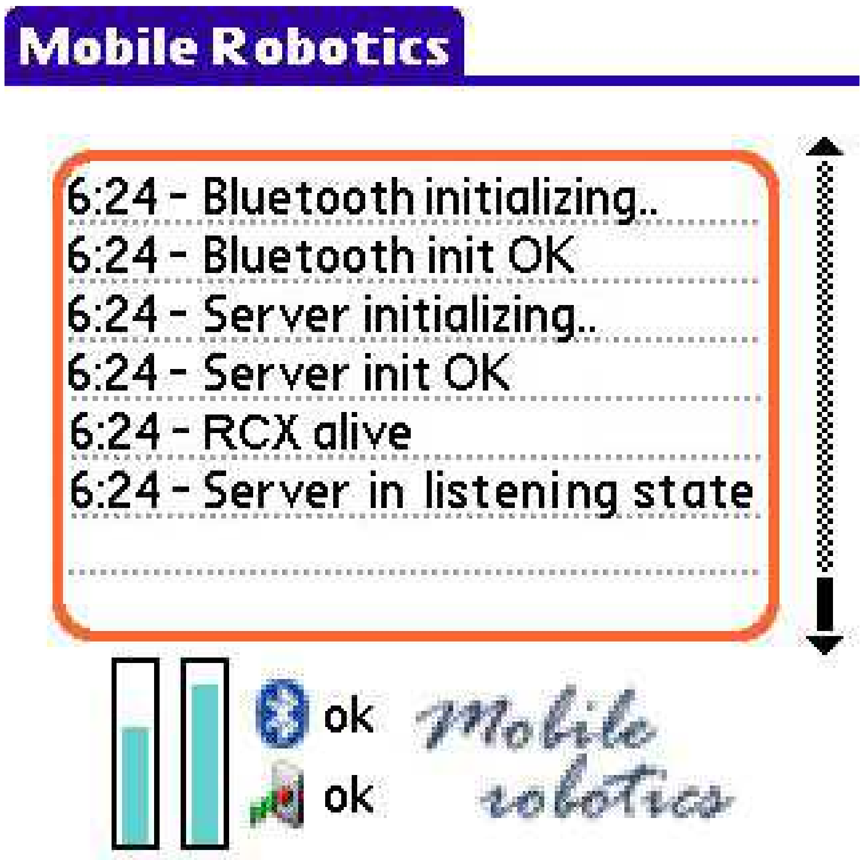
If the bluetooth thread is started and we got an connection and outputstream it will send the commands 6, 1 (stop, shutdown) and cleanUp (closes connections and such)

It then stops [ControllRobot] and exits the MIDlet.

```
/*************************************************************************************
```

Server Software (C++, Palm)

- File created 2005-04-14 by Johan Johanson, Server software programmer



```
6:24 - Bluetooth initializing..
6:24 - Bluetooth init OK
6:24 - Server initializing..
6:24 - Server init OK
6:24 - RCX alive
6:24 - Server in listening state
```

Overview of the Bluetooth & Infrared server application operable on the Palm Pilot PDA computer.

Introduction

This is a brief overview of the administration server application on the Palm Pilot. The application is structured like so:

Bluetooth server > **Main application** > **Lego infrared interface**

The main application acts like a centralized controller, receiving user commands from the client (cellphone) via Bluetooth and forwarding them through the IR transceiver to the Lego RCX computer, as RCX-interpretable opcode[4] IR samples.

Overview

This section provides a little more detailed information about the different parts of the application.

This whole application was written in C++ using the PalmOS Developer Suite.

The Bluetooth server

The Bluetooth wrapper server on the Palm Pilot consists of several classes that

[4] Operation code; an instruction that the Lego computer understands

form a stable base on which one could easily build new extensions for different uses (i.e other protocol[5] implementations, client implementations, etc):

```
----------------------------

BtBase
 + BtServerBase
    + BtRFCOMMServer
       + MobileRobotics

BtDevice
BtSocketPair
BtDataPacket
BtCustomEvent

----------------------------
```

BtBase is an abstract class with pure virtual methods providing the most basic functions for Bluetooth radio initialization among other things. It is from this class that one can derive server and client implementations.

BtServerBase is an example of an abstract (protocol-less) server implementation. This class provides expected basic server functions that must be implemented in derived, protocol-dependent implementations.

BtRFCOMMServer is a protocol-independent *socket*[6] implementation of BtServerBase. It allows for a Bluetooth server application through the RFCOMM serial protocol (stream-based) and the choice for our application. One can use this class as it is, as it automatically handles incoming clients with default functions. One should derive application-specific classes from this class and override the default methods for client handling etc.

MobileRobotics is our application-specific implementation of the BtRFCOMMServer class. It overrides default handling methods for incoming data among other things, and manages the infrared interface in order to forward incoming commands. The main application code in turn controls this class.

BtDevice encapsulates a single Bluetooth device (name, address, etc).

BtSocketPair simplifies server and client socket communications by pairing them.

BtDataPacket is used in data transfers, for a simple and effective way of encapsulating and sending any type of data.

BtCustomEvent is a custom application event used to communicate pending operations to the main application code.

---

[5] Set of rules used to define a data transmission method
[6] A socket through which network connections may be initiated

We are using a socket approach to our structure, and RFCOMM as the communications protocol as stated above. The server application flow looks something like this:

] Main application initializes Bluetooth server
] Bluetooth server classes initialize the Bluetooth radio and wait for result
] Main application receives radio initialization result and continues to start the server
] Server does maintenance; retrieves local device information and reports result
] Server continues to set up a socket that is used to listen for incoming client connections
] Application enters idle mode waiting for clients to connect
] When a client connects, the listening socket is paired up with the incoming connection if there is room
] Server puts inbound and outbound sockets in a list and opens a new listening socket

The infrared interface

The infrared interface to the Lego RCX is controlled by the MobileRobotics class, which also is a part of the Bluetooth server itself. The class used to control the ir transciever is only one:

RCXInterface
Here is some background information regarding the Lego infrared device and protocol:

RCXInterface uses the OmniRemote developer kit to emulate the Lego RCX consumer infrared transceiver (see reference section)

Ir settings used by the Lego RCX ir transciever: 2400 baud, 72/76kHz, odd parity, 1 stop bit, 8 data bits, invert data, LSB first

Basically this is how a Lego Protocol packet looks like:

0x55 0xff 0x00 O1 ~O1 O2 ~O2 .. On ~On S ~S

Where the first three bytes (the header) are constants The following n bytes are the **O**pcodes/instructions for the RCX to execute The last byte is the checksum, the sum of all the opcode bytes

NOTE: All opcodes are followed by their complement, as is the checksum

You may only send the same opcode packet twice in a row if you toggle the 0x08 bit in the opcode being sent (the RCX doesn't execute the same opcode twice in a row)

The RCX immediately echoes the sent packet, with the bytes backwards.

The OmniRemote library developer kit used is a set of functions provided in order to be able to emulate the correct Lego RCX consumer ir, which PalmOS natively does not. The RCXInterface class is a passive class, used by MobileRobotics to build ir packets, send packets and receive packets through the transciever.

The basic application flow for the whole system looks something like this:

[The Bluetooth connection retrieves a data packet through a client socket
] Code in the MobileRobotics class interprets the user command sent and forwards it to the Lego RCX

The RCX sensor programming

The Lego RCX comes with various sensors for one to control and read from. We use a touch sensor in the main steering system which triggers when the front wheels are centered. In such a case, the RCX will be instructed to shut off that particular engine controlling the steering to be able to perform a straight ahead steering. This program is written in NQC, a language written by Dave Baum, and then downloaded to the RCX and run. The program is active the whole session. Another program is available for manually overriding the power up / down function, in case of connection loss.

The GUI

The Palm Pilot server application GUI consists of one large text field in which various debugging information is shown such as Bluetooth status and clients connecting, as well as received commands. Battery level meters for both the Palm Pilot and the RCX are available and displayed on screen at all times. There are also two connection status icons available for the Blutooth and IR connection, respectively. They show "ok" when connection is established and "--" when there is an error with the connection.

Conclusion

The client/server connection between the cellphone and the Palm Pilot works actually quite nice, with a few unexplainable interrupts causing the connection to fail. This is probably because of an overload in data transmission.The RFCOMM protocol is credit based (this is specific to the Palm Pilot Bluetooth library) i.e you need one credit for each data transmission - both inbound and outbound. Replies from the RCX back to the Palm Pilot is unreliable, but plausible. We use it only when retrieving the RCX battery level in millivolts. Also, it has been concluded that when trying to receive an IR transmission with the OmniRemote Library when there is no active communication the Palm Pilot application freezes and needs a reset.

For full source code, se the appendix section and the end of this document.

Lego architecture

- File created 2005-04-14 by Gabriel Lundmark, Lego Builder

The original idea of our project was to design, construct and program a Mindstorms robot that would stream video and take commands from a mobile phone or handheld computer. Initially, the robot that we have come to refer to as Alessa, was intended to be a Mars vehicle. The very first designs were based around that theme and included several wheels instead of the ubiquitous set of four. However, the Mars prototype never left the drawing board, and we abandoned the idea in favor of a more RC car-like construction with 4-wheel drive. This model did get real, at least partially, and had more in common with its RC counterparts than we originally had imagined. Everything from its wheels and the accompanying suspension, to the transmission and differentials were all clones of your average RC car, and as such also very much like a real car. To make a long story short, it was flat out boring. Not only was the chassis finished within a couple of days – we had also very little to do with the purely technical aspects of the vehicle since most of it had been copied from ready-built Lego models and RC cars. Also, its transmission was far from perfect and left much to desire. It was painfully slow and involved worm gears that had a spongy effect on the vehicle's propulsion, much like the motors were connected with rubber bands. To top it off, we hadn't been very cautious in the building process, having missed out on important safety features as a slip wheel, which would prevent gears from cracking and motors from overheat if the vehicle would encounter an obstacle too tough for it to get over. The worm gears did provide good torque and much power, but the over-engineered gearing absorbed too much of it, leaving us with a rubber band vehicle.

Next model is not that easy to describe, since we had a lot to do coming up with a fresh design to replace the discarded RC car. Much of our time went into brainstorming and researching various aspects of building with Mindstorms, and what we created with the bricks during this period is not worth describing, apart from a few neat inventions that we decided to keep for the final version of Alessa. Among these is a pneumatic compressor that, whilst not necessary for the robot's core functions to work, is a very advanced technical solution that gives the robot the ability to raise and lower its wheels – in other words an adjustable suspension. This compressor can obviously connect to any pneumatic device and do pretty much anything with it, but because of limitations of the on-board computer (the RCX), we have no matters of controlling additional features due to the lack of channels. Steering, drive and suspension occupy the total of three channels. For the pneumatics to work, two motors are working two large pumps simultaneously in order to obtain enough air pressure for the pistons to extend and retract, which we'll cover in a second.

This was how the present Alessa was created – around a pneumatic compressor. The pneumatics turned out to be such a cool feature that we decided to base our product around it and integrate motors, transmissions and other technical features that we liked. Continuing to describe the pneumatics, we had to come up with a way of remotely controlling the airflow. The standard parts only allow for manual input and there are no means of controlling them electronically by default. After a few days of testing, we concluded that a worm-geared valve run by a separate motor would fit the bill. Hooked up to the RCX, it can be operated with on-board buttons, irDA or firmware. The compressor's undoubtedly coolest feature is its ability to automatically switch off when a desired level of pressure has been reached. This works explicitly using mechanics and no programming whatsoever is involved. A large pneumatic cylinder is hold shut tight by a spring-loaded mechanism. Connected to it is a power switch that is switched on when the cylinder is contracted. As pressure rises, the spring-mechanism slowly stretches and will eventually have reached to a point where the power switch cuts the power. Hence, the compressor stops to prevent the motors from stalling. When the pneumatic system is activated, provided it's used heavily enough, the compressor will begin pumping again and repeat the whole process continuously as needed.

With the compressor and valve working, our next assignment was to construct a suspension that, as aforementioned, would make use of pneumatics. Beginning with the front suspension, we ran into problems creating a steering mechanism that was sufficiently powerful enough to turn the large tires, yet soft enough not to crash the motor if the tires were to be obstructed and not be able to turn. This was going to be done not by geared rods, but by a vertical axle in between the wheels that would rotate the connected bars in order to turn the wheels. Connected to the axle is a gear that is driven by a worm gear, which subsequently is driven by a transmission belt that is precisely tense enough to turn the wheels, but will slip if strained too much. Also connected to the vertical axle is a touch sensor that provides the RCX with information about wheel positions and how to turn them. The result is a mechanism that works like a charm given the use of low-powered motors. Part of the challenge was to make the wheels able to travel up and down using the pneumatics and still be able to turn in both fully extended and fully contracted positions. We did it.

The front wheels, however, do not rotate by themselves as we decided to leave out any means of transmission. The reason for this was to let the front of Alessa concentrate solely on steering without having to bother with propulsion. Using this idea, the vehicle obviously has to be rear-wheel driven. Experiences from previous prototypes told us that long drive shafts and inaccurate worm gears were not the way to go, so we redesigned the whole system from scratch and ended up with two engines mounted behind the wheels connected through a vast array of gears, including slip wheels. The engines travel up and down with the wheels themselves when the suspension is activated, which eliminates the need for drive shafts. And since we have two motors, there is no need for a differential that would drain power.

All this technology is a real battery hog and to top it off, we have two high-powered lights composed of clustered LEDs that want a lot of electricity. These lights are mounted like on a typical 80s sport car, in a way that people refer to as frog eyes. When the valve is turned, these lights pop up along with the suspension. The pneumatics that power the light mechanism double as an automated power switch that will turn on or off the power for the LEDs. This power switch is hooked up to a 9V battery pack that also provides power to the compressor. That said, Alessa uses one single battery pack for its unique features. This excludes steering and propulsion, which is powered by the RCX, however both are equally important for the machine to work properly. Battery power is monitored by the on-board Tungsten T handheld of which main task is to – simply put - convert Bluetooth signals from a mobile phone to irDA utilized by the RCX. Please refer to the Client-Server Lego Communication section of this document for more information regarding the handheld and it's purpose.

Obviously, Alessa is nothing that has been built over-night. Although briefly described, all of its functions have been thoroughly tested, modified and rebuilt several times to achieve a near-perfect result. Also, the whole process of constructing a chassis large enough to carry all necessities aboard, yet rigid enough not to collapse, have been a complicated task despite not being covered in detail in this document. It is important to understand that this description of the creation of Alessa is scaled down to concentrate on the most vital parts, leaving single nuts and bolds uncovered.

Pneumatics, motors, steering systems, automated valves, pressure meters and pop-up lights aside, Alessa's perhaps most astonishing feature is its ability to stream and record video. A camera lens is mounted in the front between the lights, which is the perfect spot to look through. The lights will lit up the environment enough to see in completely black environments and the camera is mounted in a position that allows the viewer to get a good overview of the terrain ahead, but still being able to see the ground in order to navigate without a hassle. In the middle of Alessa, the camera's circuit board and the accompanying LCD display is mounted, and an antenna is to provide good reception is right nearby. The camera is connected to the very same power source as the LED lights, which means the camera and lights will be activated and deactivated at the same time. Please note that the power source is, as said earlier, not the RCX but an external 9V battery pack.

Regarding the vehicles design, not much effort was put into making Alessa prom queen of the year. Keeping deadline has been of outmost priority and cosmetic details have been left out due to the lack of time. The other reason is that we are quite fond of the bony and industrial look of the machine, seeing that it reveals the technically detailed innards for spectators to behold. In addition, leaving out unnecessary parts accommodates easy access for repairs and tuning.

Discussion

Overall we are more then satisfied with our results even though we do feel that some things could have been worked ever harder on, but time was against us. We feel for our work and could have continued to work, tune and search for more information a long period of time.

If we were to redo everything again we would do it the same way with only minor alterations. Following our project we have faced a great deal of problems and with a great deal of thinking, trying and doing research from forums, books and internet we have managed to solve them one by one. Due to our release of the entire project on open source websites and forums we can give back a great deal of shared knowledge to the community that has helped us evolve.

Overall we have learned and grown with time.

Results

Physical Result

The physical result is the easiest to see: Alessa, the Lego robot. It's design and implementation even though complicated is still visible by the eye. Those more deeply interested in its design and implementation can read the released documentation[7] on the subject. Alessa uses a complicated mechanic system powered by six engines and pneumatics.

Other results may not be as easy to read and understand the programming. The source code is heavily commented but does require intermediate knowledge about either java and / or c++. However we do provide an overview for both programmers and others, please consult section 5.2 or 5.3. There is also information about the Lego structure in section 5.4

Releases

The MobileRobotics project has several releases on Sourceforge[8]. Older versions of the source code and binaries can be downloaded if necessary to follow the development. We have two releases; the client (cellphone) and the server (handheld). Both of those include sub-divisions if you want to download the source or the binary. The difference is that you need the developer suite to be able to compile the source code. If you download the binary you get the pre-compiled[9] software. On the Source Forge page users can also submit bugs, questions and suggestions.

Documentation and code

As mentioned earlier under physical results, there are different types of documentation. Of course there is the source code itself which is commented and can be followed by a programmer. Then there is the more theoretical documentation on the Lego and the programming. This explains at a more theoretic level what happens.

Sourceforge

MobileRobotics project is hosted by SourceForge. Sourceforge is the largest Open Source community to this date. "SourceForge.net provides free hosting to Open Source software development projects. The concept of 'Open Source' promotes the benefits of collaborative development by ensuring that potential end-users are able to obtain and use software, and that the software may be improved and expanded to meet the needs of its users. Collaboration within the Open Source community (developers and end-users) promotes a higher standard of quality, and helps to ensure the long-term viability of both data and applications."

Releasing everything as Open Source is a way to contribute to other programmers and give back to the community.

Our project website can be found on http://mobilerobotics.sf.net

[7] See the appendix - Lego Structure
[8] A open-source community, please read part 6.4 - Sourceforge
[9] When you compile software you go from plain text to a code that the computer understands, machine code. If you download the source code you must have a developer kit to convert the code into machine code so you can run the application. If it is pre-compiled this is already done.

Our Sourceforge project page can be found on
https://sourceforge.net/projects/mobilerobotics/

Known bugs

Currently there are no known bugs in either software. However we do have a lack of feature in the Lego RCX (Lego computer). Settings of RCX is not saved when batteries gets removed. This means that if the RCX batteries fail settings and firmware[10] will be reset.

Conclusion

This is a project we could continue indefinitely adding even more commands and functions out of pure joy and interest. We are all amazed by the amount of information we've learned during the months in areas we knew nothing about and most of all, we made it. You could almost feel the tension leaving and the relief filling the room the first successful test-run.

It may sound like we are exaggerating but many months ago we set a goal high with only hope to be able to reach it. And now, with a few hours left to deadline everything works.

References

Most of the information we've used is available on the internet. Here are some we used in no particular order.

Project page containing full source and binaries: http://mobilerobotics.sf.net

Lego:
http://unite.com.au/~u11235a/lego/valve/

Client:
It is hard to specify a special source due to the fact that we've over the months used hundreds and hundreds of sites to gain general knowledge and to solve problems. We've searched many forums and blogs for information. Besides these we've used the javadoc for SonyEricsson's developer kit extensively.

- Programming
  http://www.onjava.com/pub/a/onjava/2001/03/15/java_palm.html
  http://today.java.net/pub/a/today/2004/07/27/bluetooth.html
  http://www.onjava.com/topics/java/Wireless_Java
  http://www.oreillynet.com/pub/a/wireless/2001/07/23/monitor.html
  http://java.sun.com/
  http://www.javaworld.com/javaworld/jw-01-2001/jw-0105-midp-p3.html
  http://www.imhotek.com/papers/WhatsinMIDP2_0_v1_0.pdf
  http://www.javaworld.com/javaworld/jw-01-2001/jw-0105-midp-p3.html
  http://www.microjava.com/articles/Bluetooth-jsr-82-training.pdf
  http://www.javaworld.com/javaworld/jw-01-2001/jw-0105-midp-p3.html
  http://wireless.klings.org/klings_jabwt_master_thesis.pdf
  http://developer.sonyericsson.com
  http://www.jasonlam604.com/articles_introduction_to_bluetooth_and_j2me_part2.ph
  p
  http://www.java201.com/resources/browse/116-2003-12.html
  http://www.gayanb.com/free_j2me_books.php
  http://www.microjava.com/articles/techtalk/game_api
  http://developers.sun.com/techtopics/mobility/midp/articles/smartticket/

[10] Firmware is the lowest level of software available in a device.

http://developers.sun.com/techtopics/mobility/midp/reference/techart/index.html
http://developers.sun.com/techtopics/mobility/midp/articles/midp2network/
http://www.javaworld.com/javaworld/jw-01-2001/jw-0105-midp-p3.html

- Hardware and companies / other:
  http://homepage.ntlworld.com/lawrie.griffiths/vision.html
  http://www.active-robots.com/products/accessories/legospares.shtml
  http://www.infosyncworld.com/news/n/1594.html
  http://www.qualcomm.com/
  http://www.firepad.com/catalog/
  http://www.oreillynet.com/pub/a/wireless/2001/07/23/monitor.html

Server
RCX Internals - http://graphics.stanford.edu/~kekoa/rcx/
Lego Mindstorms Internals - http://www.crynwr.com/lego-robotics/
OmniRemote - http://www.pacifikneotek.com
Lego IR Protocol in C++ - http://www.generation5.org/content/2001/rob08.asp
PalmOS developer - http://www.palmsource.com/palmos/
NQC - http://www.cs.uu.nl/people/markov/lego/
http://news.palmos.com/read/search/?forum=bluetooth-forum
http://flippinbits.com/twiki/bin/view/FAQ/WebHome

Appendices

About the Source Code

This document contains the source code of both server and client software, if you are interested in the binary part please use our binary download service on https://sourceforge.net/projects/mobilerobotics/

This revealed here is by all means not the complete application and will most certainly not compile as it is. All source code is free of use and is released under the GNU GPL license. Read more at http://www.gnu.org/licenses/gpl.txt. You can download the source code in its completeness at SourceForge (see reference section).

Appendix A - Client Software Source Code

- MobileRoboticsMidlet.java	: Main MIDlet
- Bluetooth.java	: Handles bluetooth connections
- SplashScren	: Error report and splash screen
- ControllRobot.java	: Controlls our robot

MobileRoboticsMidlet.java

```
/*************************************************************
 *
 * The Mobile Robotics Project
 * - mobilerobotics.sourceforge.net
 *
 * Coders in no particular order:
 *
 *  Niklas Bivald,    webmaster[a/t]bivald.com
 *  Johan Johansson,
 *                                *
 * Other team members, lego builders:
 *
 *  Gabriel Lundmark,
 *  Thomas Erdelyi,
 *
 * This file was created 2005-01-05
 *
 * Other, none important dates:
 * 2005-01-07: First time the program was tested on a real
 *        cell phone. Shout goes to Mikael Lenneryd for
 *        letting me use and abuse his v800
 *
 *         Thoughts: SE Simulator is slower then phone
 *
 * 2005-02-20: Shrinked and reformated the code (15% less)
 *
 * 2005-02-25: Code and graphics for everything but
 *            bluetooth communication is done.
 *
 * 2005-02-26: Bluetooth communication is done, in teory.
 *            Yet to be tested.
 *
 * 2005-04-08: First active bluetooth search (services
 *            and devices) and connection.
 *
 * Shout goes to:
```

```
*  - Mikael for putting up with me using his phone
*  - Those who could bare my questions and comments (usually Johan)
*  - Sun developers and their articles
*  - Java articles on the net
*
*  - J2ME Gaming book
*       URL: http://sourceforge.net/projects/j2megamingbook/
*
*  - Klings Bluetooth Master Thesis
*        URL: http://wireless.klings.org/klings_jabwt_master_thesis.pdf
*
*  - Jason Lam J2ME tutorials on Bluetooth,
*      http://www.jasonlam604.com/articles_introduction_to_bluetooth_and_j2me_part2.php
*    Perfect if you are new to Bluetooth.
*
 ************************************************************/

/*

    The runes say:
    Harald Christianized the Danes
    Harald controlled Denmark and Norway
    Harald thinks notebooks and cellular phones should communicate seamlessly


    And one last thing..
    "Beware of bugs in the above code; I have only proved it correct, not tried it. -
    Donald Knuth"

    PS: We have tried it I just like the quote.

 */


/*

 Visit http://mobilerobotics.sf.net for more information!

 Instructions to compile java midp2 software.
```

```
  1. Download and install Sony Ericsson J2ME SDK

     Register on developer.sonyericsson.com and log in,
    download "Sony Ericsson J2ME SDK"

    2. Start the Ktoolbar via the start menu:

    Start -> Program -> Sony Ericson -> J2ME SDK ->
    WTK2 -> Ktoolbar

  3. Copy the MobileRobotics folder to the
     C:\SonyEricsson\J2ME_SDK\PC_Emulation\WTK2\apps folder.

    If you want to put it on a regular phone do Projekt ->
    Package -> Create Package

  */

  /*

      As an intro I suggest reading overview.txt

   */

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

public class MobileRoboticsMidlet extends MIDlet {

  private Display display;
  private boolean btstatus;

  private Bluetooth blue               = null;
  private ControllRobot controllRobot = null;

  public void startApp()
       {
        display = Display.getDisplay(this);

        /*
```

```
 We start by initiating the blueooth thread. We pass on the valuse
  "this" in order for the thread to be able to call functions from this
  file and "display" so it can chooe what to display. We will call the
 bluetooth thread blue.

*/

System.err.println("Init: Bluetooth thread");

           blue   = new Bluetooth ( this, display );
Thread bluetooth = new Thread    ( blue          );
        bluetooth.start();


// Initiate bluetooth discovery
   System.err.println("Init: Bluetooth discovery");
  btstatus = blue.initiate();



   System.err.println("Inited: Bluetooth discovery, " + btstatus);

 if(btstatus == false)
  {
    exit();
  }

 // Since the btstatus returned true we have establised a bluetooth connection.
 // Therefor we continue with initating the controllRobot thread

  System.err.println("Init: controllRobot thread");

               controllRobot = new ControllRobot(this, blue);
               controllRobot.start();

 // We've initated the controllRobot, let's make so we actually see what we
 just
 // initated.
```

```
        display.setCurrent(controllRobot);
       }

public void setDisplayToControllRobot(Display display)
 {
        controllRobot.resetDirection();
    display.setCurrent( controllRobot );
 }


public Display getDisplay()
 {
   // Return our current Display
   return display;
 }


public void pauseApp()
 {
   // Pause application, exit software
   exit();
 }

public void destroyApp(boolean unconditional) {
  exit();
}

public void exit() {

  if(blue != null)
   {
    if(blue.connection !=null && blue.os != null)
     {
       blue.cleanUp();
     }
   }

  if(controllRobot != null)
   controllRobot.stop();
  System.gc();
  destroyApp(true);
  notifyDestroyed();
 }
}
```

Bluetooth.java

```
/***********************************************************
 *
 * The Mobile Robotics Project
 * - mobilerobotics.sourceforge.net
 *
 * This file was created 2005-01-07
 * More information in MobileRoboticsMidlet.java
 *
  ***********************************************************/

/*

  Bluetooth.java::v1.0::1
  This class main purpose is to initiate*, hold and handle
  the bluetooth communication to the hand-held.

      *: When initiating it will search every nearby devices, store it in a
         list (Array) and when all nearby devices are found search in the
         list (Array) for a device with our service.

  It also loads the SplashScreen upon initialization, Bluetooth:initiate(),
  so the user sees the SplashScreen and is told the current
  step and progress.

  Part of the Bluetooth Discovery and methods were inspired by Jason Lam,
  www.jasonlam604.com

  Please note, this is no sourcecode to learn Bluetooth from.
               this code is structured to fit our purpose.
               And it does. But if you are new to bluetooth,
               please check

               http://www.jasonlam604.com/articles_introduction_to_bluetooth_and_j2me_par
               t2.php

  However it might be a good example on how bluetooth may work in reality,
  not just on paper. I really have no more comments on this, except on thing.
  If you want to retrieve the bluetooth name of the device you use:

   "The remote device".getFriendlyName(true);

  This must be in a try {} and most importantly, it can NEVER be done while
  searching for devices. The logic part would be to insert it in
  the method deviceDiscovered(). This will throw an exception because it can only
  do one bluetooth thing at a time.

  You must put every found device in an array (wich would look something like):

    private RemoteDevice remoteDevices[] = new RemoteDevice[10];

  And then, after inquiryCompleted() you can just do a while loop to find out
  the device name and/or search for services. Whatever may suit you.

 */


import java.io.*;
import java.util.Timer;
import java.util.TimerTask;
import javax.microedition.io.*;
import javax.bluetooth.*;

import javax.microedition.lcdui.*;
import javax.microedition.lcdui.game.*;
import java.lang.Thread;
import java.lang.Runnable;

public class Bluetooth extends GameCanvas implements DiscoveryListener, Runnable {


  private MobileRoboticsMidlet mobileRoboticsMidlet;                    // The midlet
  private Display display;                                             // The display
  private String bterror = "";                                          // Errormsg.
  private SplashScreen splash;
  private boolean wait;
  private int direction;                                              // Hold the current
  direction. Used to keep
  tracks..


 // Fonts. Check J2ME Gaming book (see MobileRobticsMidlet.java for more information)
 static final Font boldFont    = Font.getFont(Font.FACE_MONOSPACE, Font.STYLE_BOLD, Font.
 SIZE_SMALL);
 static final Font regularFont = Font.getFont(Font.FACE_MONOSPACE, Font.STYLE_PLAIN,
 Font.SIZE_SMALL);

  /* Se the method initiate() for explanation */

  private int p1;
  private int p2;
  private int p3;
  private int p4;

  /* Images */
  public Image start;
  public Image stopped;
  public Image running;
  public Image logo;
  public Image Left;
  public Image right;
  public Image forward;
  public Image backward;
  public Image nowhere;
  public Image clear;
  public Image run;
  public Image about;
  public Sprite wheel;
  public Image Wheel;

  /* Bluetooth */
  private DiscoveryAgent discoveryAgent;
  private UUID[] uuidSet;
  private String serviceUrl     = null;
  private int lastSentCommand   = 0;
```

```
   OutputStream os = null;

   public static final UUID RFCOM_UUID = new UUID("716441D9D19641A980D8DCC753ADEC59",
   false);

       // Mobile Robotics is using UUID 0x716441D9D19641A980D8DCC753ADEC59, why?
       // The UUID was generated to fit our services name, it is random.
       // For more info read up on how to generate UUIDs.

   public StreamConnection connection;

   private int status;
   private String data;
   boolean discoveryWhile = false;
   boolean servicesWhile  = false;
   boolean tempBoolean    = false;


  public StreamConnectionNotifier notifier;

   // We will, later on, put every nearby Bluetooth device in an array.
   private RemoteDevice remoteDevices[] = new RemoteDevice[10];

   // Number of devices in the array
   private int nrRemoteDevices = 0;

  public Bluetooth(MobileRoboticsMidlet mobileMidlet, Display disp)
   {

     super(true);
     System.err.println(" Initied: Bluetooth");

     display              = disp;
     mobileRoboticsMidlet = mobileMidlet;

   }



 public void stop()
  {
   cleanUp();
  }


 public void start() {                 /* Not in use */                             }
 public void run()   {                /* Not in use */                             }


 /*

  Bluetooth::initiate()
  This method initiaties the Bluetooth connection and loads the splashScreen.
  The method is divided into two parts, the graphical part and the connection.

  The graphical part is the splashScreen (SplashScreen.java) wich shows the progress
  to the user upon starting the midlet. This is so that the users see's more than
  simply a blank screen. The splashScreen knows the current step due to the variables
  (int) p1, p2, p3, p4. P stands for part.

  pX variable is either 0, 1 or 2. 0 is none-initiated, 1 is active and 2 is finished.
  the variable will first have 0, turn to 1 when it is running and 2 when it is finished.

  The connection part is used to initiate, find and connect to the handheld (Palm).

  */


     public boolean initiate() {

       System.err.println(" Init: SplashScreen Thread");          // Initiating
       splashScreen (loading)
           Graphics g = getGraphics();

       p1 = 0;
       p2 = 0;
       p3 = 0;
       p4 = 0;

       SplashScreen splashScreen = new SplashScreen ( this );    // Initiating our
```

```
SplashScreen.
splashScreen.start();
splash = splashScreen;                                        // For other methods needing
splashscreen
display.setCurrent( splashScreen );




/* Part one, init, we check for the needed functions */
       p1 = 1;
     neededFunctions();
     p1 = 2;

/* Part two, start the device discovery               */
     p2 = 1;
      mobileRoboticsDeviceDiscovery();


      // We must use synchronized to be able to use wait();
      // Later on we call notifyAll() to continue

      while (discoveryWhile != true)
       {
          // We will wait for the device discovery to be finished
       }
     p2 = 2;

 /* Part three, we search for our service            */
     p3 = 1;
      try
       {
          /*
              Bluetooth name: "The remote device".getFriendlyName(true)

             Note: These values _must_ be retrieved after device discovered is
             finished. It will throw a
             bluetooth
              exception if it is used when searching for devices. Read comments on
             the top of this file for more
```

```
            info.
          */

          // Our UUID
          uuidSet    = new UUID[1];
          uuidSet[0] = RFCOM_UUID;

          // We need to search for services in every device in our array. Therefor,
          loop it.
          for (int i = 0; i < nrRemoteDevices; i++)
          {
              servicesWhile = false;
              int searchID  = discoveryAgent.searchServices(null,uuidSet,
              remoteDevices[i],this);

              while (servicesWhile != true)
               {
                  // Wait for service discovery to be finished
               }
              }
       }catch (Exception e)
       {
                  // We don't want the exceptions since it may come from any
                  bluetooth device.
                  // If it is from ours we will notice it due to no serviceUrl a
                  couple of lines later
       }

      p3 = 2;


/* Part four, open bluetooth connection and load images              */
      p4 = 1;
          if(serviceUrl != null)
           {
                        try {
                         /* Open bluetooth connection */
                           connection =
                            (StreamConnection)Connector.open(serviceUrl);
```

```
                    }catch (Exception exception)
                    {

                        bterror  = "Could not open a connection";
                        fatalError();
                    }

                    /* We want to open an OutputStream so we can actually send
                    data */
                    try {
                             os             = connection.openOutputStream();

                    }catch (Exception e)
                    {
                         bterror = "Could not open output stream";
                        fatalError();
                    }


               }else{

                   bterror  = "Could not find the hand held";
                  fatalError(); // Print error message
               }

             loadImages();
         p4 = 2;

         /* Just to make sure */
         if(connection == null)
           {
              return false;
           }

           // Stop the loading screen.
           splashScreen.stop();
           return true;

   } /*             End of init()                  */
```

```
/*
******************************************************************************************
*/





    /* Displays our fatalError(s) and waits for user input */
    public void fatalError()
     {
        splash.stop();
        display.setCurrent( splash );

        if(bterror == "")
         {
             bterror = "Unknown error";
         }

        // Loop error message until the user press fire
        while(splash.errorPressAnyKey())
         {
                       // Wait..
         }

                if(serviceUrl != null)
                 {

                      mobileRoboticsMidlet.setDisplayToControllRobot(display);

                    try{
                    if(connection != null)
                     {
                          connection.close();
                         connection = null;
                     }

                    if(os != null)
                     {
```

```
                    os.close();
                    os = null;
                 }
                } catch (Exception e)
                {
                    // Do nothing

                }
                           try {
                            /* Open bluetooth connection */
                              connection =
                               (StreamConnection)Connector.open(serviceUrl);

                        }catch (Exception exception)
                        {

                            bterror  = "Could not open a connection";
                            fatalError();
                        }

                        /* We want to open an OutputStream so we can actually send
                        data */
                        try {
                                os             = connection.openOutputStream();

                        }catch (Exception e)
                        {
                             bterror = "Could not open output stream";
                             fatalError();
                        }
                }else{
                   mobileRoboticsMidlet.exit();
                }

 }

/* Used for splashScreen */
public int p1() {   return p1;     }
public int p2() {   return p2;     }
public int p3() {   return p3;     }
public int p4() {    return p4;     }


/* Load our images and Sprite */
public void loadImages()
 {
        try { start       = Image.createImage ("/start.png");      } catch ( Exception e
        )  {  throw new
         RuntimeException ("Unable to load Image: "+e); }
        try { stopped     = Image.createImage ("/stopped.png");    } catch ( Exception e
        )  {  throw new
        RuntimeException ("Unable to load Image: "+e); }
        try { running     = Image.createImage ("/running.png");    } catch ( Exception e
        )  {  throw new
         RuntimeException ("Unable to load Image: "+e); }
        try { logo        = Image.createImage ("/logo.png");       } catch ( Exception e
        )  {  throw new
         RuntimeException ("Unable to load Image: "+e); }
        try { Left        = Image.createImage ("/left.png");       } catch ( Exception e
        )  {  throw new
        RuntimeException ("Unable to load Image: "+e); }
        try { right       = Image.createImage ("/right.png");      } catch ( Exception e
        )  {  throw new
         RuntimeException ("Unable to load Image: "+e); }
        try { forward    = Image.createImage ("/forward.png");     } catch ( Exception e
        )  {  throw new
         RuntimeException ("Unable to load Image: "+e); }
        try { backward    = Image.createImage ("/backwards.png"); } catch ( Exception e
        )  {  throw new
        RuntimeException ("Unable to load Image: "+e); }
        try { nowhere     = Image.createImage ("/nowhere.png");    } catch ( Exception e
        )  {  throw new
         RuntimeException ("Unable to load Image: "+e); }
        try { run        = Image.createImage ("/run.png");         } catch ( Exception e
        )  {  throw new
         RuntimeException ("Unable to load Image: "+e); }
        try { clear       = Image.createImage ("/clear.png");      } catch ( Exception e
        )  {  throw new
        RuntimeException ("Unable to load Image: "+e); }
        try { Wheel       = Image.createImage ("/wheel.png");      } catch ( Exception e
          )  {  throw new
           RuntimeException ("Unable to load Image: "+e); }
          try { about      = Image.createImage ("/about.png");      } catch ( Exception e
          )  {  throw new
           RuntimeException ("Unable to load Image: "+e); }
          wheel = new Sprite(Wheel, 118,118);
     }

    /* Used for splashScreen to retrieve our error message */
    public String bterror()
     {
      return bterror;
     }

    /* Check for our needed Functions */
    private void neededFunctions()
     {

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

       }catch (BluetoothStateException bse)
       {

           bterror  = "Bluetooth couldn't be initiated:\n " + bse;
          fatalError();
       }

     }

/*
*************************************************************************************
*/
/* Below this you will find the method used in init() to search for devices and service,
*/
/*  and to connect with them
*/
```

```
/* Start our bluetooth discovery */
private void mobileRoboticsDeviceDiscovery()
 {

   /* A regular deviceDiscovery class,
      but our getDiscoveryAgent has been moved to neededFunctions() */

   try {

       if(!discoveryAgent.startInquiry(DiscoveryAgent.GIAC, this))
         {
          bterror  = "Couldn't start Inquiry";
          fatalError();
         }

         } catch(BluetoothStateException bse)   {

          bterror  = "BluetoothStateException:\n " + bse;
          fatalError();
       }

    // Device Discovery ran
}




 public void deviceDiscovered(RemoteDevice btDevice, DeviceClass cod)
  {
    // Device was discovered
    // Adress: btDevice.getBluetoothAddress()
    nrRemoteDevices = nrRemoteDevices + 1;
    remoteDevices[nrRemoteDevices-1] = btDevice;
  }


  public void inquiryCompleted(int discType)
  {
        /*  Do nothing  */
    discoveryWhile = true;
  }


  public void servicesDiscovered(int transID, ServiceRecord[] servRecord) {

    // We only got one service, you can search for more though.
    // Services was discovered.

    //  (String)servRecord[i].getAttributeValue(0x100).getValue()

    for(int  i=0;i<servRecord.length;i++)
      {
          serviceUrl = servRecord[i].getConnectionURL(1, false);
      }
  }



  public void serviceSearchCompleted(int transID, int responseCode) {

    // Service search completed

    // If you are wondering why we don't use any errorchecking it is
    // due to the simply fact that they may come from any device
    // near us. If a unkown cellphone that is in the pocket of a
    // person near by throws a Service_search_error we don't want
    // to halt the search.

    servicesWhile = true;

  }



/*
***************************************************************************************
*/
```

```
public void command(int command)
 {

   /*
    * Bluetooth commands and their function
    *
    *  0) Power On
    *  1) Power Off
    *  2) Forward
    *  3) Backward
    *  4) Left
    *  5) Right
    *  6) Stop (forward/backward)
    *
    */


    /* If connection or os is null we don't have a working connection, then abort. */
    if(connection != null && os != null)
     {

         try {
                /*

                   It's easier for the palm to recieve it in byte format,
                   therefor we simply convert it from int to byte by using
                  the (byte)INT syntax.

                 */

                byte[] cmd          = new byte[1];
                cmd[0]              = (byte)command;

                // write to buffer
                os.write(cmd);

                // Send to buffer
                os.flush();

//                     try{ Thread.sleep(25); } catch (Exception e) { /* Do nothing */ }
```

```
                 }catch (Exception e)
             {
                bterror = "Could not send command";
                fatalError();
             }

    }else{
             bterror = "No connection and/or stream";
             fatalError();
    }
 }


public void cleanUp()
 {
          try {
           os.close();
            connection.close();
          } catch (IOException ioe) {
            System.err.println("Error Closing connection " + ioe);
        }
 }


}
```

ControllRobot.java

```
/************************************************************
 *
 * The Mobile Robotics Project
 * - mobilerobotics.sourceforge.net
 *
 * This file was created 2005-01-07
 * More information in MobileRoboticsMidlet.java
 *
  ************************************************************/

import javax.microedition.lcdui.*;
import javax.microedition.lcdui.game.*;
import java.lang.Thread;

public class ControllRobot extends GameCanvas implements Runnable {

 private boolean doPlay;                                       // Game Loop runs when isPlay is
                                                               //true

 private long delay;                                           // To give thread consistency
 private int width;                                            // To hold screen width
 private int height;                                           // To hold screen height
 private int direction  = 8;                                   // The direction the robot goes
 in
                                                               // (for display)..


 private int ydirection = 0;                                   // The last sent Y command
  (forward/backward/stop)
 private int xdirection = 0;                                   // The last send X command
  (left/right/stop)

 private Bluetooth bluetooth;                                     // Bluetooth class
 private MobileRoboticsMidlet mobileRoboticsMidlet;           // MobileRoboticsMidlet class
 private int mode;                                            // Mode (pointer or regular)
 private boolean pointer;                                     // Do we have a pointer?
 private boolean havePoints;                                  // Do we have points?
 private int boxy;

 private int left;                                             // The left (X) for our "is
running, stopped"
private int top;
private int[][] points;                                        // our points
private int     nrpoints;                                      // the nr of points..

private int[][] dpoints;                                       // A duplicate of points for when
running the path*
private int     dnrpoints;                                     // A duplocate of nr of points
for when running the path*

                                                               // *: When we run the path we
                                                               created we must be able
                                                               //    to see how many points we
                                                               have passed, and if there are any
                                                               left.
                                                               //    in short, we must have one
                                                               path to compare with and one to
                                                               //    store were we already been

  /*
   * Bluetooth commands and their function (int direction)
   *
   *  Car is running..
   *
   *  0) Power On
   *  1) Power Off
   *  2) Forward
   *  3) Backward
   *  4) Left
   *  5) Right
   *  6) Stop
   *
   *  For local use only:
   *  7) Switch (Switch modes (key to pointer))
   *  8) Killed (It's powered off)
   */



 // Constructor and initialization
 public ControllRobot(MobileRoboticsMidlet mobileMidlet, Bluetooth blue) {

  super(true);
  width     = getWidth();
  height    = getHeight();
  bluetooth = blue;
  mobileRoboticsMidlet = mobileMidlet;

 }

// Automatically start thread for game loop
public void start() {
  System.err.println(" Inited: ControllRobot");


  // Do we have pointers?
  pointer    = hasPointerEvents();

  // Loop
  doPlay     = true;

  // Software mode
  mode       = 1;

  // Start
  Thread t   = new Thread(this);
  t.start();

  // See bluetooth commands.
  direction  = 8;


  // If the display is to small we might have to change our values for
  // left and top margins due to otherwise overlapping images.

  if((width / 2 - 80) < 15)
    {
     left =    15;
    }else{
     left =    width / 2 - 80;
```

```
         }

   if((height / 2 - 50) < 70)
    {
     top =    65;
    }else{
     top =    height / 2 - 50;
    }

  // Boxes configureation
  boxy = height - 50;

  // We don't have any points stored
  havePoints = false;

  // Pointer array
  points = new int[10][2];

  // We don't have any points yet
  nrpoints = 0;


}

public boolean running()
 {
   return doPlay;
 }

public void stop()
  {
    doPlay = false;
  }


public void run() {
  Graphics g = getGraphics();

  while (doPlay == true) {
    // Two is for status/debug screen, one is for regular controllRobot
    if(mode == 1)
      {

          input();
          drawScreen(g);

        }else {

      if(pointer)
       {
         drawPointerScreen(g);
         }else{
         drawAboutScreen(g);
       }
      }
  }
}

 protected void keyPressed( int keyCode ) {

  // Press 0 to toggle modes
  if(keyCode == 48)
   {
    if(mode == 1)
     {

      bluetooth.command(6);
      direction = 7;
      mode      = 2;

     }else{

      if(havePoints == true)
       {
        clearPath();
       }

      bluetooth.command(6);
      direction = 6;
      mode      = 1;
     }

   }

 }

 public void pointerPressed(int x, int y)
  {
   // If we are in regular mode there is no reason for pointerevents, there for return.
   if(mode == 1)
    return;



   // If statement is true we have pressed the pointer in the draw area and not on a button
   if(y < boxy)
    {


    // We don't want anyone drawing lines that are going back and forth,
    // our robot should just go forward in a special pattern. Therefor we don't
    // count any pointerPress that is behind the previous dot.

    if((nrpoints > 0 ) && (y > points[nrpoints-1][1]))
      return;

    // You should be able to press a point outside of the drawing area.
    if(x < width / 2 - 89 || x > (width / 2 - 89 + 177) || y < 41 || y > (41 + boxy - 42))
     return;

    if(havePoints == false)
     {
      havePoints = true;
     }

    // Draw a line
    System.err.println("draw a line");

    points[nrpoints][0] = x;
    points[nrpoints][1] = y;
    nrpoints++;

  // Right button
  }else if((y > boxy) && (x > (width / 2)))
  {
    System.err.println("Run path");
    runPath();
  // Left button
  }else if((y > boxy) && (x < (width / 2)))
  {
    // Start clearing all path variables
    clearPath();
  }
 }

 private void clearPath()
  {
   System.err.println("Clear paths");

     for (int i=0; i < nrpoints; i++)
       {
           points[i][0] = 0;
           points[i][1] = 0;
       }

    points[nrpoints][0]  = 0;
    points[nrpoints][1]  = 0;

    havePoints   = false;
    nrpoints     = 0;
  }

public void runPath()
 {

 }
 public void resetDirection()
  {
    direction  = 6;
    ydirection = 6;
    xdirection = 6;
  }

// Method to Handle User Inputs
private void input() {
  int keyStates = getKeyStates();

  // Left
  if ((keyStates & LEFT_PRESSED) != 0)
   {
       if(xdirection == 4)
        {
            // Left
            direction = 4;
            bluetooth.command(4);


        }else if(xdirection == 5)
        {
            // We are turning right so let's go to neutral.
            bluetooth.command(4);

            // Depending on the ydirection we set the wheel to a certain angle.
            if(ydirection == 2 || ydirection == 3)
             {
                 direction  = ydirection;
             }else{
                 direction  = 6;
            }

           xdirection = 6;

        // If it's in neutral send left
        }else if(xdirection == 6)
        {
            bluetooth.command(4);
            xdirection = 4;
            direction  = 4;

        }else{ // Other

                    direction = 4;
                     bluetooth.command(4);
                     xdirection = 4;

        }
    }

  // Right
  if ((keyStates & RIGHT_PRESSED) !=0 )
   {


       if(xdirection == 5)
        {
            // Right
            direction = 5;
            bluetooth.command(5);

        }else if(xdirection == 4)
        {
            // We are turning right so let's go to neutral.
            bluetooth.command(5);

            // Depending on the ydirection we set the wheel to a certain angle.
            if(ydirection == 2 || ydirection == 3)
             {
                 direction  = ydirection;
             }else{
                 direction  = 6;
            }

           xdirection = 6;

        // If it's in neutral send right
```

```
      }else if(xdirection == 6)
      {

          bluetooth.command(5);
          xdirection = 5;
          direction  = 5;

      }else{ // Other
            direction  = 5;
            bluetooth.command(5);
            xdirection = 5;
      }


 }

// Up
if ((keyStates & UP_PRESSED) != 0)
 {
    if(ydirection == 2)
        {
          // Already going forward
          direction = 2;
      }else if(ydirection == 3)
      {
          bluetooth.command(6);
          ydirection = 6;
          direction  = 6;
      }else if(ydirection == 6)
      {
          bluetooth.command(2);
          ydirection = 2;
          direction  = 2;
      }else{
          bluetooth.command(2);
          ydirection = 2;
          direction  = 2;
      }

   }
```

```
// Down
if ((keyStates & DOWN_PRESSED) !=0)
 {
    if(ydirection == 3)
        {
          // Already going backwards
          direction = 3;
      }else if(ydirection == 2)
      {
          bluetooth.command(6);
          ydirection = 6;
          direction  = 6;

      }else if(ydirection == 6)
      {
          bluetooth.command(3);
          ydirection = 3;
          direction  = 3;
      }else{
          bluetooth.command(3);
          ydirection = 3;
          direction  = 3;
      }
}

// Fire
if ((keyStates & FIRE_PRESSED) !=0)
 {
   if(direction != 8)
    {

            // We don't know what we want to do here

    }else{
            direction = 6;
              bluetooth.command(0);
    }
 }
```

```
    keyStates = 0;
    try{ Thread.sleep(15); } catch (Exception e) { /* Do nothing */ }
}

// Method to Display Graphics
private void drawScreen(Graphics g) {
   g.setColor(0xffffff);
   g.fillRect(0, 0, getWidth(), getHeight());

    //border
   g.setColor(0xFC932C);
   g.fillRect(0, 0, width, height);

   g.setColor(0xffffff);
   g.fillRect(1, 1, width - 2, height - 2);

 // top

   g.setColor(0xEBF1F7);
   g.fillRect(1, 1, width - 2, 27);

   g.setColor(0xADC2D6);
   g.fillRect(1, 28, width - 2, 2);

   g.drawImage (bluetooth.logo, width - 138, 1, Graphics.TOP | Graphics.LEFT);

   if(direction == 8)
    {

           g.drawImage (bluetooth.start, left, top, Graphics.TOP | Graphics.LEFT);
           g.drawImage (bluetooth.stopped, left, 40, Graphics.TOP | Graphics.LEFT);
     }else{
           g.drawImage (bluetooth.running, left, 40, Graphics.TOP | Graphics.LEFT);

            if(direction == 4)
             {
               g.drawImage (bluetooth.Left, left + 82, 37, Graphics.TOP | Graphics.LEFT);
              bluetooth.wheel.setPosition(width / 2 - 59, height / 2 - 59 + 25);
              bluetooth.wheel.setFrame(0);
              bluetooth.wheel.paint(g);
```

```
          }

         if(direction == 5)
          {
           g.drawImage (bluetooth.right, left + 82, 37, Graphics.TOP | Graphics.LEFT);
           bluetooth.wheel.setPosition(width / 2 - 59, height / 2 - 59 + 25);
           bluetooth.wheel.setFrame(1);
           bluetooth.wheel.paint(g);
          }

         if(direction == 2)
          {
            g.drawImage (bluetooth.forward, left + 82, 37, Graphics.TOP | Graphics.LEFT);
           bluetooth.wheel.setPosition(width / 2 - 59, height / 2 - 59 + 25);
           bluetooth.wheel.setFrame(2);
           bluetooth.wheel.paint(g);
          }

         if(direction == 3)
          {
            g.drawImage (bluetooth.backward, left + 82, 37, Graphics.TOP | Graphics.LEFT);
           bluetooth.wheel.setPosition(width / 2 - 59, height / 2 - 59 + 25);
           bluetooth.wheel.setFrame(3);
           bluetooth.wheel.paint(g);
          }

         if(direction == 6)
          {
            g.drawImage (bluetooth.nowhere, left + 82, 37, Graphics.TOP | Graphics.LEFT);
           bluetooth.wheel.setPosition(width / 2 - 59, height / 2 - 59 + 25);
           bluetooth.wheel.setFrame(2);
           bluetooth.wheel.paint(g);
          }

        g.setColor(0x0000ff);
       }

 flushGraphics();
}
```

```
private void drawAboutScreen(Graphics g) {
 g.setColor(0xffffff);
 g.fillRect(0, 0, getWidth(), getHeight());

  //border
 g.setColor(0xFC932C);
 g.fillRect(0, 0, width, height);

 g.setColor(0xffffff);
 g.fillRect(1, 1, width - 2, height - 2);

// top

 g.setColor(0xEBF1F7);
 g.fillRect(1, 1, width - 2, 27);

 g.setColor(0xADC2D6);
 g.fillRect(1, 28, width - 2, 2);

 g.drawImage (bluetooth.logo, width - 138, 1, Graphics.TOP | Graphics.LEFT);
 g.drawImage (bluetooth.about, width / 2 - 59, height / 2 - 59 + 10, Graphics.TOP |
 Graphics.LEFT);
 flushGraphics();
}

private void drawPointerScreen(Graphics g) {
 g.setColor(0xffffff);
 g.fillRect(0, 0, getWidth(), getHeight());

  //border
 g.setColor(0xFC932C);
 g.fillRect(0, 0, width, height);

 g.setColor(0xffffff);
 g.fillRect(1, 1, width - 2, height - 2);

 g.setColor(0xFC932C);
 g.fillRect(width / 2 - 90, 40, 179, boxy - 40);

 g.setColor(0xDAE4ED);
 g.fillRect(width / 2 - 89, 41, 177, boxy - 42);

// top

 g.setColor(0xEBF1F7);
 g.fillRect(1, 1, width - 2, 27);

 g.setColor(0xADC2D6);
 g.fillRect(1, 28, width - 2, 2);

 g.drawImage (bluetooth.logo, width - 138, 1, Graphics.TOP | Graphics.LEFT);

  // Clear and run button
  g.drawImage (bluetooth.clear, width / 2 - 90, boxy + 10, Graphics.TOP | Graphics.LEFT);
  g.drawImage (bluetooth.run, width / 2 + 10, boxy + 10, Graphics.TOP | Graphics.LEFT);


   // Draws the line
   g.setColor(0x43506B);
   if(havePoints == true)
    {
      g.fillRect(points[0][0],points[0][1],2,2);

       for (int i=0; i < nrpoints; i++)
        {
          g.fillRect(points[i][0],points[i][1],2,2);

         if(i == 0)
          {
          }else{
          int tmp = i - 1;
             g.drawLine(points[tmp][0],points[tmp][1],points[i][0],points[i][1]);
         }
        }
  }

  flushGraphics();
}

private void drawPointerScreenRunning(Graphics g) {
 g.setColor(0xffffff);
 g.fillRect(0, 0, getWidth(), getHeight());

  //border
 g.setColor(0xFC932C);
 g.fillRect(0, 0, width, height);

 g.setColor(0xffffff);
 g.fillRect(1, 1, width - 2, height - 2);

 g.setColor(0xFC932C);
 g.fillRect(width / 2 - 90, 40, 179, boxy - 40);

 g.setColor(0xDAE4ED);
 g.fillRect(width / 2 - 89, 41, 177, boxy - 42);

// top

 g.setColor(0xEBF1F7);
 g.fillRect(1, 1, width - 2, 27);

 g.setColor(0xADC2D6);
 g.fillRect(1, 28, width - 2, 2);

 g.drawImage (bluetooth.logo, width - 138, 1, Graphics.TOP | Graphics.LEFT);

  // Clear and run button
  g.drawImage (bluetooth.clear, width / 2 - 90, boxy + 10, Graphics.TOP | Graphics.LEFT);
  g.drawImage (bluetooth.run, width / 2 + 10, boxy + 10, Graphics.TOP | Graphics.LEFT);


   // Draws the line
   g.setColor(0x43506B);
   if(havePoints == true)
    {
      g.fillRect(points[0][0],points[0][1],2,2);

       for (int i=0; i < nrpoints; i++)
        {
          g.fillRect(points[i][0],points[i][1],2,2);
```

```
         if(i == 0)
          {
          }else{
          int tmp = i - 1;
             g.drawLine(points[tmp][0],points[tmp][1],points[i][0],points[i][1]);
         }
        }
   }



    // Draws the second line
    g.setColor(0xFFFFFF);
    if(havePoints == true)
     {
       g.fillRect(dpoints[0][0],dpoints[0][1],2,2);

        for (int i=0; i < dnrpoints; i++)
         {
           g.fillRect(dpoints[i][0],dpoints[i][1],2,2);

         if(i == 0)
          {
          }else{
          int tmp = i - 1;
             g.drawLine(dpoints[tmp][0],dpoints[tmp][1],dpoints[i][0],dpoints[i][1]);
         }
        }
   }
   flushGraphics();
  }
 }
```

SplashScreen.java

```
/**********************************************************
 *
 * The Mobile Robotics Project
 * - mobilerobotics.sourceforge.net
 *
 * This file was created 2005-01-05
 * More information in MobileRoboticsMidlet.java
 *
 **********************************************************/

/*
  SplashScreen.java::v0.2::1
  This class shows the splashScreen upon loading and any
  error it might bring. There isn't that much of commentation
  in this class. And I strongly advice not looking at it
  when need of a code example. The class is loaded with pixel-
  referenses needed to draw the lines, boxes and such things
  only neccessary for our project. YOU WILL ONLY GET CONFUSED*.

  A box is not a box but four lines.

  In theory the only thing this class does is showing the
  loading bar. This is due to a loop in run() wich moves
  the bar one step to the right and updates (and loops this).

  * Many of these values are multiplications and fixed by
    trial and error. There is no logical explanation
    why some of them are as they are.

  If you want an example of drawCanvas use ControllRobot.java
  or check the J2ME Gaming book.

  Only thing that really is easy to change is the bar speed,
  change the (int) delay.The higher it is the slower is the
  speed.

  If you must look in this file, for example want to use the loading
  bar I suggest checking the line ~211 (a comment wich states
  where the bar starts and ends)
 */

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

public class SplashScreen extends GameCanvas implements Runnable {

 private boolean isPlay;            // Loading runs when isPlay is true
 private long delay;                 // This will make the thread stop for a short period of
 time every time the bar has moved. It is to controll the bars speed. Try lowering it and see
 what happens.
 private int currentX, currentY;          // To hold current position of the 'mobileRobotics'
 text
 private int width;                 // To hold screen width
 private int height;                // To hold screen height

 // Fonts. Check J2ME Gaming book (see MobileRobticsMidlet.java for more information)
 static final Font boldFont    = Font.getFont(Font.FACE_MONOSPACE, Font.STYLE_BOLD, Font.
 SIZE_SMALL);
 static final Font regularFont = Font.getFont(Font.FACE_MONOSPACE, Font.STYLE_PLAIN, Font.
 SIZE_SMALL);

 Bluetooth bluetooth;

 Image logo;
 Image init;
 Image conn;
 Image search;
 Image reg;
 Image error;

  // Constructor and initialization
 public SplashScreen(Bluetooth bluet) {

    super(true);

    width     = getWidth();
    height    = getHeight();
    delay     = 1;
    currentX  = - 26 ;
    bluetooth = bluet;
 }

// Automatically start thread for game loop
public void start() {

    System.err.println(" Inited: SplashScreen thread");
    // Starting self
    isPlay = true;
    Thread t = new Thread(this);
    t.start();
}

public void stop()
 {
    isPlay = false;
 }


public void run() {
    Graphics g = getGraphics();

      try { logo    = Image.createImage ("/logo.png");         } catch ( Exception e )  {  throw
      new RuntimeException ("Unable to load Image: "+e); }
      try { search  = Image.createImage ("/search.png");       } catch ( Exception e )  {  throw
      new RuntimeException ("Unable to load Image: "+e); }
      try { init    = Image.createImage ("/init.png");         } catch ( Exception e )  {  throw
      new RuntimeException ("Unable to load Image: "+e); }
      try { conn    = Image.createImage ("/connecting.png");   } catch ( Exception e )  {  throw
      new RuntimeException ("Unable to load Image: "+e); }
      try { reg     = Image.createImage ("/reg.png");          } catch ( Exception e )  {  throw
      new RuntimeException ("Unable to load Image: "+e); }

    while (isPlay == true) {

         if(currentX < 84)
               {
                currentX = currentX + 2;
               }else{
                currentX = - 26;
                }

        drawScreen(g);

        // This will make the thread stop for a short period of time every time the bar has
         moved. It is to controll the bars speed. Try setting 'delay' to 0 and see what happens.
        try { Thread.sleep(delay); }        catch (InterruptedException ie) {}
    }
}


public boolean errorPressAnyKey()
 {
  /*
    Method is called when there was an error in the bluetooth discovery.
    It will loop an error message until the users press fire.
  */

  Graphics g = getGraphics();

    drawErrorMessage(g);

       int keyStates = getKeyStates();

      if ((keyStates & FIRE_PRESSED) !=0)
       {
              return false;
       }

  return true;
 }



private void drawErrorMessage(Graphics g)  {

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

  //border
```

```
    g.setColor(0xFC932C);
    g.fillRect(0, 0, width, height);

    g.setColor(0xffffff);
    g.fillRect(1, 1, width - 2, height - 2);

    // top

    g.setColor(0xEBF1F7);
    g.fillRect(1, 1, width - 2, 27);

    g.setColor(0xADC2D6);
    g.fillRect(1, 28, width - 2, 2);

    g.drawImage (logo, width - 138, 1, Graphics.TOP | Graphics.LEFT);

    // Text

    g.setColor(0xFF0000);
    g.setFont(boldFont);
     g.drawString("There was an error:",5,40,Graphics.TOP|Graphics.LEFT);

    g.setColor(0x627293);
    g.setFont(regularFont);
      g.drawString(bluetooth.bterror(),5,55,Graphics.TOP|Graphics.LEFT);

    g.setColor(0x627293);
    g.setFont(regularFont);
     g.drawString("Press fire to continue.",5,65,Graphics.TOP|Graphics.LEFT);

    flushGraphics();
   }


  private void drawScreen(Graphics g) {

    /**********************************
     * Draws the splashScreen.
     **********************************/
```

```
//border
   g.setColor(0xFC932C);
   g.fillRect(0, 0, width, height);

   g.setColor(0xffffff);
   g.fillRect(1, 1, width - 2, height - 2);

// top

   g.setColor(0xEBF1F7);
   g.fillRect(1, 1, width - 2, 27);

   g.setColor(0xADC2D6);
   g.fillRect(1, 28, width - 2, 2);

//middle-box
   g.setColor(0x627293);
   g.fillRect(width / 2 - 44, height / 2 - 5, 86, 11);

   g.setColor(0xffffff);
   g.fillRect(width / 2 - 43, height / 2 - 4, 84, 9);

     g.drawImage (logo, width - 138, 1, Graphics.TOP | Graphics.LEFT);


   /* The moving bar, a set of wierd IF statements to make sure the
          moving bar doesn't cross over any borders and runs fine      */

   g.setColor( 0xB0B8C9 );

       if( (currentX + 26) > 84 )                           // If currentX + 26 is above 84
       the bar will run over the right border, this means we must reduce it's size
        {
      int width2 = (currentX +26) - 84;
            width2 = 26 - width2;

          g.fillRect(width / 2 - 44 + currentX,height / 2 - 3,width2,7);

        }else if( currentX < 1 ) {                            // If currentX is below 0 we must
        reduce our barsize otherwise it will be over the left border
```

```
             int width3 = 26 + currentX;

            if( (width / 2 - 44 + currentX + width3) < (width / 2 - 44) )
          {

      // If statement is true it means that it isn't even vissible, do nothing

          }else{


            // The bar is visible, but runs over the left border.
            // Therefor we calculate how much over the border it is,
            // and adds it to X and removes it from the width.

            int leftOver = (width / 2 - 43) - (width / 2 - 44 + currentX);
                g.fillRect(width / 2 - 43 + currentX + leftOver,height / 2 - 3,width3 -
                leftOver,7);

          }
    }else{

          // The bar is running freely crossing no borders
          g.fillRect(width / 2 - 44 + currentX,height / 2 - 3,26,7);
    }



//box1

     g.setColor(0x627293);
     g.fillRect(width / 2 - 41, height / 2 - 17, 8, 8);

     g.setColor(0xffffff);
     g.fillRect(width / 2 - 40, height / 2 - 16, 6, 6);



   int p1 = bluetooth.p1();
```

```
        if( p1 == 1 )
         {

          g.drawImage (init, width / 2 - 47, height / 2 + 6, Graphics.TOP | Graphics.LEFT);

          g.setColor(0x627293);
          // Draw the lines around the active box
          g.drawLine(width / 2 - 43, height / 2 - 18, width / 2 - 43, height / 2 - 16);
          g.drawLine(width / 2 - 43, height / 2 - 19, width / 2 - 40, height / 2 - 19);

          g.drawLine(width / 2 - 43, height / 2 - 11, width / 2 - 43, height / 2 - 9);
          g.drawLine(width / 2 - 43, height / 2 - 8 , width / 2 - 40, height / 2 - 8);

          g.drawLine(width / 2 - 43 + 11, height / 2 - 19, width / 2 - 43 + 11, height / 2 -
          16);
          g.drawLine(width / 2 - 43 + 8,  height / 2 - 19, width / 2 - 43 + 10, height / 2 -
          19);

          g.drawLine(width / 2 - 43 + 11, height / 2 - 11, width / 2 - 43 + 11, height / 2 - 9);
          g.drawLine(width / 2 - 43 + 8,  height / 2 - 8 , width / 2 - 43 + 11, height / 2 - 8);

         }else if ( p1 == 2 )
         {
          g.setColor(0x627293);
          g.drawLine(width / 2 - 39, height / 2 - 14, width / 2 - 38, height / 2 - 12);
          g.drawLine(width / 2 - 38, height / 2 - 12, width / 2 - 36, height / 2 - 15);
         }




//box2

    g.setColor(0x627293);
    g.fillRect(width / 2 - 41 + 24, height / 2 - 17, 8, 8);
    g.setColor(0xffffff);
    g.fillRect(width / 2 - 40 + 24, height / 2 - 16, 6, 6);

  int p2 = bluetooth.p2();

        if( p2 == 1 )
         {
          g.drawImage (search, width / 2 - 44, height / 2 + 6, Graphics.TOP | Graphics.LEFT);

          g.setColor(0x627293);
          // Draw the lines around the active box
          g.drawLine(width / 2 - 43 + 24, height / 2 - 18, width / 2 - 43 + 24, height / 2 -
          16);
          g.drawLine(width / 2 - 43 + 24, height / 2 - 19, width / 2 - 40 + 24, height / 2 -
          19);

          g.drawLine(width / 2 - 43 + 24, height / 2 - 11, width / 2 - 43 + 24, height / 2 - 9);
          g.drawLine(width / 2 - 43 + 24, height / 2 - 8 , width / 2 - 40 + 24, height / 2 - 8);

          g.drawLine(width / 2 - 43 + 11 + 24, height / 2 - 19, width / 2 - 43 + 11 + 24, height
          / 2 - 16);
          g.drawLine(width / 2 - 43 + 8 + 24,  height / 2 - 19, width / 2 - 43 + 10 + 24, height
          / 2 - 19);

          g.drawLine(width / 2 - 43 + 11 + 24, height / 2 - 11, width / 2 - 43 + 11 + 24, height
          / 2 - 9);
          g.drawLine(width / 2 - 43 + 8 + 24,  height / 2 - 8 , width / 2 - 43 + 11 + 24, height
          / 2 - 8);

         }else if ( p2 == 2 )
         {
          g.setColor(0x627293);
          g.drawLine(width / 2 - 39 + 24, height / 2 - 14, width / 2 - 38 + 24, height / 2 -
          12);
          g.drawLine(width / 2 - 38 + 24, height / 2 - 12, width / 2 - 36 + 24, height / 2 -
          15);
         }
```

```
//box3

     g.setColor(0x627293);
     g.fillRect(width / 2 - 41 + 24 + 24, height / 2 - 17, 8, 8);

     g.setColor(0xffffff);
     g.fillRect(width / 2 - 40 + 24 + 24, height / 2 - 16, 6, 6);

   int p3 = bluetooth.p3();

       if( p3 == 1 )
        {
       g.drawImage (reg, width / 2 - 44, height / 2 + 6, Graphics.TOP | Graphics.LEFT);

         g.setColor(0x627293);
         // Draw the lines around the active box
         g.drawLine(width / 2 - 43 + 24 + 24, height / 2 - 18, width / 2 - 43 + 24 + 24, height
         / 2 - 16);
         g.drawLine(width / 2 - 43 + 24 + 24, height / 2 - 19, width / 2 - 40 + 24 + 24, height
         / 2 - 19);

         g.drawLine(width / 2 - 43 + 24 + 24, height / 2 - 11, width / 2 - 43 + 24 + 24, height
         / 2 - 9);
         g.drawLine(width / 2 - 43 + 24 + 24, height / 2 - 8 , width / 2 - 40 + 24 + 24, height
         / 2 - 8);

         g.drawLine(width / 2 - 43 + 11 + 24 + 24, height / 2 - 19, width / 2 - 43 + 11 + 24 +
         24, height / 2 - 16);
         g.drawLine(width / 2 - 43 + 8 + 24 + 24,  height / 2 - 19, width / 2 - 43 + 10 + 24 +
         24, height / 2 - 19);

         g.drawLine(width / 2 - 43 + 11 + 24 + 24, height / 2 - 11, width / 2 - 43 + 11 + 24 +
         24, height / 2 - 9);
         g.drawLine(width / 2 - 43 + 8 + 24 + 24,  height / 2 - 8 , width / 2 - 43 + 11 + 24 +
         24, height / 2 - 8);

        }else if ( p2 == 2 )
        {
          g.setColor(0x627293);
          g.drawLine(width / 2 - 39 + 24 + 24, height / 2 - 14, width / 2 - 38 + 24 + 24, height
          / 2 - 12);
          g.drawLine(width / 2 - 38 + 24 + 24, height / 2 - 12, width / 2 - 36 + 24 + 24, height
          / 2 - 15);
        }

//box4


      g.setColor(0x627293);
      g.fillRect(width / 2 - 41 + 24 + 24 + 24, height / 2 - 17, 8, 8);

      g.setColor(0xffffff);
      g.fillRect(width / 2 - 40 + 24 + 24 + 24, height / 2 - 16, 6, 6);

    int p4 = bluetooth.p4();

        if( p4 == 1 )
         {
          g.drawImage (conn, width / 2 - 44, height / 2 + 6, Graphics.TOP | Graphics.LEFT);

          g.setColor(0x627293);
          // Draw the lines around the active box
          g.drawLine(width / 2 - 43 + 24 + 24 + 24, height / 2 - 18, width / 2 - 43 + 24 + 24 +
          24, height / 2 - 16);
          g.drawLine(width / 2 - 43 + 24 + 24 + 24, height / 2 - 19, width / 2 - 40 + 24 + 24 +
          24, height / 2 - 19);

          g.drawLine(width / 2 - 43 + 24 + 24 + 24, height / 2 - 11, width / 2 - 43 + 24 + 24 +
          24, height / 2 - 9);
          g.drawLine(width / 2 - 43 + 24 + 24 + 24, height / 2 - 8 , width / 2 - 40 + 24 + 24 +
          24, height / 2 - 8);

          g.drawLine(width / 2 - 43 + 11 + 24 + 24 + 24, height / 2 - 19, width / 2 - 43 + 11 +
          24 + 24 + 24, height / 2 - 16);
          g.drawLine(width / 2 - 43 + 8 + 24 + 24 + 24,  height / 2 - 19, width / 2 - 43 + 10 +
          24 + 24 + 24, height / 2 - 19);

          g.drawLine(width / 2 - 43 + 11 + 24 + 24 + 24, height / 2 - 11, width / 2 - 43 + 11 +
          24 + 24 + 24, height / 2 - 9);
          g.drawLine(width / 2 - 43 + 8 + 24 + 24 + 24,  height / 2 - 8 , width / 2 - 43 + 11 +
          24 + 24 + 24, height / 2 - 8);

         }else if ( p4 == 2 )
         {
          g.setColor(0x627293);
          g.drawLine(width / 2 - 39 + 24 + 24, height / 2 - 14, width / 2 - 38 + 24 + 24, height
          / 2 - 12);
          g.drawLine(width / 2 - 38 + 24 + 24, height / 2 - 12, width / 2 - 36 + 24 + 24, height
          / 2 - 15);
         }


    flushGraphics();
 }

}
```

Appendix B – Bluetooth server source code

In this section you may view *relevant* source code regarding our socketed RFCOMM Bluetooth server implementation. You can download all the source code and / or the already compiled executable over at SourceForge (open-source code community, see reference section). If you’re looking for the Lego RCX NCQ source code, please refer to the appendices section of this document.

```cpp
/*
 * Copyright (c) 2004-2005 Mobile Robotics
 * http://mobilerobotics.sf.net
 *
 * File: BtBase.cpp
 * Author: Johan Johanson
 * Date: January 16, 2005
 * Updated: February 26, 2005
 *
 * Description: See BtBase.h
 *
*/

#include "BtBase.h"
#include "BtCustomEvent.h"

const Int16 BtBase::InvalidSocket = 0xFFFF;
UInt16 BtBase::btRef = sysInvalidRefNum;
BtBase *BtBase::callbackRef = NULL;

BtBase::BtBase() : radioInitialized(false)
{

}

BtBase::~BtBase()
{

}

Boolean BtBase::initSystem()
{
	Err error = 0;
	UInt32 btVersion = 0;

	shutdownSystem();

	// look for BT compability and supported OS version
	if (FtrGet(btLibFeatureCreator, btLibFeatureVersion, &btVersion) != errNone)
		return false;
```

```
            // find the library, if not found, load it
            if (SysLibFind(btLibName, &btRef))
                        if (SysLibLoad(sysFileTLibrary, sysFileCBtLib, &btRef) != errNone)
                                    return false;

            // open the library
            if ((error = BtLibOpen(btRef, false)) != btLibErrNoError)
            {
                        btRef = sysInvalidRefNum;
                        return false;
            }

            // register Bluetooth event callback method
            if ((error = BtLibRegisterManagementNotification(btRef, managementCallback, 0)) != btLibErrNoError)
                        return false;

            // application must wait for radio to initialize
            // before performing bluetooth dependent tasks

            return true;
}

void BtBase::shutdownSystem()
{
            if (btRef != sysInvalidRefNum)
            {
                        BtLibUnregisterManagementNotification(btRef,  managementCallback);
                        BtLibClose(btRef);
                        btRef = sysInvalidRefNum;
            }
}

Boolean BtBase::init()
{
            Err error = 0;

            if (!radioInitialized || btRef == sysInvalidRefNum)
            {
                        debugPrint("Bluetooth not initialized");
                        return false;
            }

            // retrieve info about the local BT device
            if ((error = BtLibGetGeneralPreference(btRef, btLibPref_LocalClassOfDevice, (void
*)&localDevice.classOfDevice,

sizeof(localDevice.classOfDevice)))
                        != btLibErrNoError)
            {
                        debugPrint(btErrToStr(error));
                        return false;
            }

            if ((error = BtLibGetGeneralPreference(btRef, btLibPref_LocalDeviceAddress, (void *)&localDevice.address,

sizeof(localDevice.address)))
                        != btLibErrNoError)
            {
                        debugPrint(btErrToStr(error));
                        return false;
            }

            return true;
}

void BtBase::shutdown()
{
            if (btRef != sysInvalidRefNum)
                        disconnect();
}

void BtBase::setCallbackRef(BtBase *ref)
{
            callbackRef = ref;
}

Boolean BtBase::getRadioInitialized()
{
            return radioInitialized;
```

```
}

BtDevice *BtBase::getLocalDevice()
{
            return (btRef == sysInvalidRefNum) ? NULL : &localDevice;
}

void BtBase::btAddrToStr(BtLibDeviceAddressType *address, Char *buffer)
{
            if (btRef == sysInvalidRefNum || address == NULL || buffer != NULL)
                        return;

            Err error = 0;
            buffer = (Char *)MemPtrNew(18);

            MemSet(buffer, sizeof(buffer), 0);

            if ((error = BtLibAddrBtdToA(btRef, address, buffer, 18)) != btLibErrNoError)
            {
                        MemPtrFree(buffer);
                        return;
            }
}
```

```cpp
/*
 * Copyright (c) 2004-2005 Mobile Robotics
 * http://mobilerobotics.sf.net
 *
 * File: BtServerBase.cpp
 * Author: Johan Johanson
 * Date: January 24, 2005
 * Updated: February 26, 2005
 *
 * Description: See BtServerBase.h
 *
*/

#include "BtServerBase.h"

BtServerBase::BtServerBase() : BtBase(), connections(NULL), sdpHandle(0), maxNumClients(0),
                                                                                numConnectedClients(0),
activeListener(0)
{
          MemSet(serviceName, MaxLenService, 0);
          MemSet((void **)&sdpUuid, sizeof(BtLibSdpUuidType), 0);
}

BtServerBase::~BtServerBase()
{

}

Boolean BtServerBase::init(Char *serviceName, UInt16 maxNumClients)
{
          if (connections != NULL || serviceName == NULL || !maxNumClients)
                    return false;

          if (!BtBase::init())
                    return false;

          StrNCopy(this->serviceName, serviceName, MaxLenService);
          this->maxNumClients = maxNumClients;

          connections = (BtSocketPair *)MemPtrNew(sizeof(BtSocketPair) * maxNumClients);
```

```
		if (connections == NULL)
			return false;

		MemSet(connections, sizeof(BtSocketPair) * maxNumClients, 0);

		for (UInt16 i = 0; i < maxNumClients; i++)
			connections[i].local = connections[i].remote.socket = InvalidSocket;

		return true;
}

void BtServerBase::disconnect()
{
		// stop advertising service
		BtLibSdpServiceRecordStopAdvertising(btRef,  sdpHandle);
		BtLibSdpServiceRecordDestroy(btRef, sdpHandle);
		sdpHandle = 0;

		// disconnect all clients
		for (UInt16 i = 0; i < maxNumClients; i++)
			disconnect(i);

		if (connections != NULL)
		{
			MemPtrFree(connections);
			connections = NULL;
		}

		MemSet(serviceName, MaxLenService, 0);
		maxNumClients = numConnectedClients = activeListener = 0;
}

Boolean BtServerBase::disconnect(UInt16 client)
{
		if (connections == NULL || client >= maxNumClients || !numConnectedClients)
			return false;

		Err error = 0;
```

```
        // only try to disconnect an active connection
        if (connections[client].local != InvalidSocket || connections[client].remote.socket != InvalidSocket)
        {
                if ((error = BtLibSocketClose(btRef, connections[client].remote.socket)) != btLibErrNoError)
                        debugPrint(btErrToStr(error));

                if (client == activeListener)
                {
                        BtLibSdpServiceRecordStopAdvertising(btRef,  sdpHandle);
                        BtLibSdpServiceRecordDestroy(btRef, sdpHandle);
                        sdpHandle = 0;
                }

           if ((error = BtLibSocketClose(btRef, connections[client].local)) != btLibErrNoError)
                debugPrint(btErrToStr(error));

                MemSet(&connections[client], sizeof(BtSocketPair), 0);
                connections[client].local = connections[client].remote.socket = InvalidSocket;
                numConnectedClients--;
        }

        // connection inactive or already disconnected
        return true;
}

Boolean BtServerBase::sendPacket(BtDataPacket *packet, UInt16 receiver)
{
        if (packet == NULL || receiver >= maxNumClients)
                return false;

        Err error = 0;

        // send packet through client socket
        if ((error = BtLibSocketSend(btRef, connections[receiver].remote.socket, (UInt8 *)packet,
                                                                        sizeof(BtDataPacket) +
packet->length))
                != btLibErrPending)
        {
                debugPrint(btErrToStr(error));
                return false;
```

```
            }

            // send is asynchronous, result is pending
            return true;
}

void BtServerBase::authenticate(UInt16 link)
{
            Err error = 0;

            if (connections == NULL || link < 0 || link >= maxNumClients)
                        return;

            if ((error = BtLibLinkSetState(btRef, (BtLibDeviceAddressTypePtr)&connections[link].remote.address,

btLibLinkPref_Authenticated, NULL, 0)) != btLibErrNoError)
                        debugPrint(btErrToStr(error));
}

void BtServerBase::setAccessibility(UInt32 mode)
{
            Err error = 0;

            if ((error = BtLibSetGeneralPreference(btRef, btLibPref_CurrentAccessible, &mode, sizeof(UInt32))) !=
btLibErrNoError)
            {
                        debugPrint(btErrToStr(error));
                        return;
            }

            localDevice.accessibility = mode;
}

void BtServerBase::setEncryption(Boolean encryption, UInt16 link)
{
            Err error = 0;

            if (connections == NULL || link < 0 || link >= maxNumClients)
                        return;
            if  ((error  =  BtLibLinkSetState(btRef,  (BtLibDeviceAddressTypePtr)&connections[link].remote.address,
                                                                                                     btLibLinkPref_Encrypted,
(void  *)&encryption, sizeof(Boolean)))  != btLibErrNoError)
                        debugPrint(btErrToStr(error));
}

UInt16 BtServerBase::getMaxNumClients()
{
            return maxNumClients;
}

UInt16 BtServerBase::getNumConnectedClients()
{
            return numConnectedClients;
}

UInt16 BtServerBase::getActiveListener()
{
            return activeListener;
}

Boolean BtServerBase::getAuthenticated(UInt16 link)
{
            Err error = 0;
            Boolean authenticated = false;

            if (connections == NULL || link < 0 || link >= maxNumClients)
                        return false;

            if  ((error  =  BtLibLinkGetState(btRef,  (BtLibDeviceAddressTypePtr)&connections[link].remote.address,
                                                                                              btLibLinkPref_Authenticated,
(void  *)&authenticated, sizeof(Boolean)))
                        != btLibErrNoError)
                        debugPrint(btErrToStr(error));

            return authenticated;
}

Boolean BtServerBase::getEncryption(UInt16 link)
{
```

```
            Err error = 0;
            Boolean encryption = false;

            if (connections == NULL || link < 0 || link >= maxNumClients)
                        return false;

            if  ((error  =  BtLibLinkGetState(btRef,  (BtLibDeviceAddressTypePtr)&connections[link].remote.address,
                                                                                                    btLibLinkPref_Encrypted, (void
*)&encryption, sizeof(Boolean))) != btLibErrNoError)
                        debugPrint(btErrToStr(error));

            return encryption;
}

BtDevice  *BtServerBase::getClientDevice(UInt16  client)
{
            if (connections == NULL || client < 0 || client >= maxNumClients)
                        return NULL;

            return  &connections[client].remote;
}

Boolean  BtServerBase::onAuthenticationComplete(BtLibManagementEventType  *event)
{
            if (event->status == btLibErrNoError)
                        debugPrint("Link auth. successful");
            else if  (event->status == btLibMeStatusAuthenticateFailure)
                        debugPrint("Link auth. failed");

            return true;
```

```
/*
 * Copyright (c) 2004-2005 Mobile Robotics
 * http://mobilerobotics.sf.net
 *
 * File: BtRFCOMMServer.cpp
 * Author: Johan Johanson
 * Date: January 26, 2005
 * Updated: February 26, 2005
 *
 * Description: See BtRFCOMMServer.h
 *
*/

#include "BtRFCOMMServer.h"

BtRFCOMMServer::BtRFCOMMServer() : BtServerBase()
{

}

BtRFCOMMServer::~BtRFCOMMServer()
{

}

Boolean BtRFCOMMServer::init(Char *serviceName, UInt16 maxNumClients, Boolean advanceCredit)
{
	if (!BtServerBase::init(serviceName, maxNumClients))
		return false;

	advCredit = advanceCredit;

	// initialize service UUID
	// random generated
	// {716441D9-D196-41A9-80D8-DCC753ADEC59}
    BtLibSdpUuidInitialize(sdpUuid, "\x71\x64\x41\xD9\xD1\x96\x41\xA9\x80\xD8\xDC\xC7\x53\xAD\xEC\x59", 
btLibUuidSize128);

	return true;
}
```

```
Boolean BtRFCOMMServer::listen()
{
            // opens the next invalid socket in list for listening

            Err error = 0;
            UInt16 num = 0;

            for (num = 0; num < maxNumClients; )
                        if (connections[num].local == InvalidSocket && connections[num].remote.socket ==
InvalidSocket)
                                    break;
                        else
                                    num++;

            if (connections[num].local != InvalidSocket)
            {
                        debugPrint("No free listen sockets");
                        return false;
            }

            // create socket
            if ((error = BtLibSocketCreate(btRef, &connections[num].local, socketCallback, 0, btLibRfCommProtocol))
                        != btLibErrNoError)
            {
                        debugPrint(btErrToStr(error));
                        return false;
            }

            BtLibSocketListenInfoType info;
            MemSet(&info, sizeof(BtLibSocketListenInfoType), 0);

            info.data.RfComm.maxFrameSize = BT_RF_DEFAULT_FRAMESIZE;
            info.data.RfComm.advancedCredit = 0;

            // setup socket for listening
            if ((error = BtLibSocketListen(btRef, connections[num].local, &info)) != btLibErrNoError)
            {
                        debugPrint(btErrToStr(error));
                        return false;
            }

            BtLibSdpServiceRecordStopAdvertising(btRef, sdpHandle);
            BtLibSdpServiceRecordDestroy(btRef, sdpHandle);
            sdpHandle = 0;

            // create an SDP service record handle
            if ((error = BtLibSdpServiceRecordCreate(btRef, &sdpHandle)) != btLibErrNoError)
            {
                        debugPrint(btErrToStr(error));
                        disconnect(num);
                        return false;
            }

            // set SDP attributes
            if ((error = BtLibSdpServiceRecordSetAttributesForSocket(btRef, connections[num].local, &sdpUuid, 1,
serviceName,

                                                                  StrLen(serviceName), sdpHandle))
                        != btLibErrNoError)
            {
                        debugPrint(btErrToStr(error));
                        disconnect(num);
                        return false;
            }

            // start advertising service
            if ((error = BtLibSdpServiceRecordStartAdvertising(btRef, sdpHandle)) != btLibErrNoError)
            {
                        debugPrint(btErrToStr(error));
                        disconnect(num);
                        return false;
            }

            activeListener = num;

            return true;
}

Boolean BtRFCOMMServer::sendPacket(BtDataPacket *packet, UInt16 receiver)
```

```
{
            if (advCredit)
                        advanceCredit(1, receiver);

            return BtServerBase::sendPacket(packet, receiver);
}

void BtRFCOMMServer::advanceCredit(UInt8 credit, UInt16 link)
{
            Err error = 0;

            if (connections == NULL || link < 0 || link >= maxNumClients)
                        return;

            if ((error = BtLibSocketAdvanceCredit(btRef, connections[link].remote.socket, credit)) !=
btLibErrNoError)
                        debugPrint(btErrToStr(error));
}

Boolean BtRFCOMMServer::onConnectedInbound(BtLibSocketEventType *event)
{
            if (event->status != btLibErrNoError)
            {
                        debugPrint("Inbound connection denied");
                        return false;
            }

            // make an alias to avoid costly computing
            BtDevice *client = &connections[activeListener].remote;

            if (client->socket != InvalidSocket)
            {
                        debugPrint("Inbound socket already exists");
                        BtLibSocketClose(btRef, event->eventData.newSocket);
                        return false;
            }

            // save client socket with the listening socket
            client->socket = event->eventData.newSocket;
            // retrieve remote client information
            if (client->friendlyName.name != NULL)
                        MemSet(client->friendlyName.name, 32, 0);

            client->friendlyName.name = (UInt8 *)MemPtrNew(249);
            client->friendlyName.nameLength = 249;

            BtLibGetRemoteDeviceName(btRef,   (BtLibDeviceAddressTypePtr)&client->address, &client->friendlyName,
                                                                                      btLibRemoteOnly);


            Char msg[32];
            StrPrintF(msg, "Inbound link socket #%d", activeListener);
            debugPrint(msg);

            // begin listen on a new socket
            if (++numConnectedClients < maxNumClients)
                        listen();

            return true;
}

Boolean  BtRFCOMMServer::onConnectRequest(BtLibSocketEventType  *event)
{
            // accept/deny inbound connection
            // stop advertising on the corresponding listening socket
            if (numConnectedClients >= maxNumClients)
            {
                        BtLibSocketRespondToConnection(btRef, event->socket, false);
                        debugPrint("Inbound connection denied (max)");
                        return true;
            }

            BtLibSocketRespondToConnection(btRef, event->socket, true);
            BtLibSdpServiceRecordStopAdvertising(btRef, sdpHandle);

            // save address
            connections[activeListener].remote.address = event->eventData.requestingDevice;
            debugPrint("Connection inbound..");

            return true;
```

```
}

Boolean  BtRFCOMMServer::onDisconnected(BtLibSocketEventType *event)
{
        // find disconnecting socket
        for (UInt16 i = 0; i < maxNumClients; )
                if (connections[i].local != InvalidSocket && connections[i].remote.socket == InvalidSocket)
                {
                        disconnect(i);
                        Char msg[80];
                        StrPrintF(msg, "Socket #%d disconnected", i);
                }
                else
                        i++;

        return true;
}
```

```
/*
 * Copyright (c) 2004-2005 Mobile Robotics
 * http://mobilerobotics.sf.net
 *
 * File: MobileRobotics.cpp
 * Author: Johan Johanson
 * Date: February 05, 2005
 * Updated: April 15, 2005
 *
 * Description: See MobileRobotics.cpp
 *
*/

#include "MobileRobotics.h"
#include "resource.h"

MobileRobotics::MobileRobotics()  : BtRFCOMMServer(), busy(false)
{

}

MobileRobotics::~MobileRobotics()
{

}

void MobileRobotics::shutdown()
{
           rcxInterface.shutdown();
           BtRFCOMMServer::shutdown();
}

// called automatically from within class when the Bluetooth radio first is initialized
Boolean MobileRobotics::init(Char *serviceName, UInt16 maxNumClients, Boolean advanceCredit)
{
           if (!BtRFCOMMServer::init(serviceName, maxNumClients, advCredit))
                      return false;

           // initialize the RCX IR interface
           if (!rcxInterface.init())
```

```
        {
                debugPrint("Unable to initialize RCX interface");
                return false;
        }

        // see if the RCX is responding - is online and in range
        UInt16 len = 7;
        UInt8 pingPacket[7];
        MemSet((void *)&pingPacket, sizeof(pingPacket), 0);

        rcxInterface.buildPacket("\x10", 1, pingPacket);

        // send query and receive echo within 1 second
        if (!rcxInterface.sendPacket(pingPacket, sizeof(pingPacket)))
        {
                debugPrint("IR error");
                return false;
        }

        // this method clearly doesn't work declared in an extra code segment
        /*if (!rcxInterface.recvPacket(pingPacket, &len, 20))
        {
                debugPrint("Unable to find the RCX");
                return false;
        }*/

        OrLibStartRcvData(rcxInterface.getLibRef(), rcxInterface.getBaudRate());
        OrLibRcvData(rcxInterface.getLibRef(), pingPacket, &len, rcxInterface.getRecvFrequency(),
                                        rcxInterface.getDataBits(), rcxInterface.getParity(),
rcxInterface.isLsbFirst(), rcxInterface.isInvertData(), 10);
        OrLibEndRcvData(rcxInterface.getLibRef());

        if (len < 1)
        {
                debugPrint("RCX not found");
                return false;
        }

        debugPrint("RCX alive");
        WinDrawChars("ok", 2, 47 + 13, 141 + 13 / 2 - 5);
```

```
		return true;
}

void MobileRobotics::debugPrint(Char *str)
{
		// retrieve a ptr to debug field
		FieldPtr  field  =  (FieldPtr)FrmGetObjectPtr(FrmGetActiveForm(),  FrmGetObjectIndex(FrmGetActiveForm(),


		 MobileRoboticsDebugField));

		if (str != NULL && field != NULL)
		{
				// apply time stamp to message
				DateTimeType dateTime;
				Char date[timeStringLength + 3 + StrLen(str) + 1];

				TimSecondsToDateTime(TimGetSeconds(), &dateTime);
				TimeToAscii(dateTime.hour,  dateTime.minute,  tfColon24h,  date);

				StrCat(date, " - ");
				StrNCat(date, str, timeStringLength + StrLen(str) + 1);

				FldSetInsPtPosition(field,  FldGetTextLength(field));
				FldInsert(field, date, StrLen(date));
				FldInsert(field, "\n", 1);
		}
}

UInt16 MobileRobotics::getRCXBatteryLevel()
{
		busy = true;

    // send battery level request packet to the rcx
    UInt8 battPack[7];
    rcxInterface.buildPacket("\x30", 1, battPack);

    if (rcxInterface.sendPacket(battPack, sizeof(battPack)))
    {
```

```
                        UInt8  replyPack[9];
                        UInt16 len = 9;

                        // expect a reply..
                        OrLibStartRcvData(rcxInterface.getLibRef(), rcxInterface.getBaudRate());
                        OrLibRcvData(rcxInterface.getLibRef(), replyPack, &len, rcxInterface.getRecvFrequency(),
                                                        rcxInterface.getDataBits(), rcxInterface.getParity(),
rcxInterface.isLsbFirst(), rcxInterface.isInvertData(), 10);
                        OrLibEndRcvData(rcxInterface.getLibRef());

                        if (len > 0)
                        {
                                // calculate millivolt percentage
                                UInt16 mV = (replyPack[5] & 0xff) * 256;
                                mV += (replyPack[3] & 0xff);

                                busy = false;
                                return mV;
                        }
    }

    busy = false;

    return 0;
}

void MobileRobotics::update()
{
        if (timer == 0)
                return;

        if (TimGetSeconds() - timer >= 3)
        {
                UInt8  stopPacket[9];
                rcxInterface.buildPacket("\x21\x44", 2, stopPacket);

                if (!rcxInterface.sendPacket(stopPacket, sizeof(stopPacket)))
                        debugPrint("Unable to send command");

                timer = 0;
```

```
            }
}

Boolean  MobileRobotics::onConnectedInbound(BtLibSocketEventType  *event)
{
            if (!BtRFCOMMServer::onConnectedInbound(event))
                        return false;

            // since we're only expecting one client, use link #0
            // begin with 2 credits in order to be able to send/receive data
            advanceCredit(2, 0);
            //authenticate(0);
            //setEncryption(true, 0);

            Char msg[80];
            Char name[connections[0].remote.getFriendlyName()->nameLength];
            StrNCopy(name, (Char *)connections[0].remote.getFriendlyName()->name,
connections[0].remote.getFriendlyName()->nameLength);
            StrPrintF(msg, "Client connected: %s", name);

            debugPrint(msg);
            WinDrawChars("ok", 2, 47 + 13, 126 + 13 / 2 - 5);      // BT status

            return true;
}

Boolean MobileRobotics::onData(BtLibSocketEventType *event)
{
            if (event->eventData.data.data == NULL || busy)
                        return false;

            // we send simple integer data
            // NOTE: you should use BtDataPacket and cast the
            // data pointer to that
            UInt8 cmd = *event->eventData.data.data;

            // REMEMBER TO ADVANCE CREDITS FOR THE ACTIVE LINK
            // OR ELSE YOU WON'T BE ABLE TO SEND OR RECEIVE ANY MORE DATA
            // (saves you 3 hours of work debugging something completely irrelevant)
            // (yes it happened to me)
```

```
advanceCredit(1, 0);

/*
            interpret the command and send it to the RCX

            0) power on (the car, not the rcx)
            1) power off (the car, not the rcx)
            2) forward
            3) backward
            4) left
            5) right
            6) stop (only forward/backward engine)
*/

switch (cmd)
{
            case CMD_ON:
            {
                        debugPrint("command: power on");

                        UInt8 onPacket[9];
                        UInt8 dirPacket[9];
                        UInt8 beepPacket[9];

                        rcxInterface.buildPacket("\x21\x81", 2, onPacket);
                        rcxInterface.buildPacket("\xe1\x01", 2, dirPacket);
                        rcxInterface.buildPacket("\x51\x03", 2, beepPacket);

                        if (!rcxInterface.sendPacket(beepPacket, sizeof(beepPacket)))
                                    debugPrint("Unable to send command");

                        SysTaskDelay(10);

                        if (!rcxInterface.sendPacket(dirPacket, sizeof(dirPacket)))
                                    debugPrint("Unable to send command");

                        SysTaskDelay(10);

                        if (!rcxInterface.sendPacket(onPacket, sizeof(onPacket)))
                                    debugPrint("Unable to send command");
```

```
                // must wait a couple of seconds and then power it off again
                UInt32 start = TimGetSeconds();
                while (TimGetSeconds() - start < 7);

                UInt8 stopPacket[9];
                rcxInterface.buildPacket("\x21\x41", 2, stopPacket);

                if (!rcxInterface.sendPacket(stopPacket, sizeof(stopPacket)))
                        debugPrint("Unable to send command");

                break;
        }

        case CMD_OFF:
        {
                debugPrint("command: power off");

                UInt8 onPacket[9];
                UInt8 dirPacket[9];
                UInt8 beepPacket[9];

                rcxInterface.buildPacket("\x21\x81", 2, onPacket);
                rcxInterface.buildPacket("\xe1\x81", 2, dirPacket);
                rcxInterface.buildPacket("\x51\x02", 2, beepPacket);

                if (!rcxInterface.sendPacket(beepPacket, sizeof(beepPacket)))
                        debugPrint("Unable to send command");

                SysTaskDelay(10);

                if (!rcxInterface.sendPacket(dirPacket, sizeof(dirPacket)))
                        debugPrint("Unable to send command");

                SysTaskDelay(10);

                if (!rcxInterface.sendPacket(onPacket, sizeof(onPacket)))
                        debugPrint("Unable to send command");

                // must wait a couple of seconds and then power it off again
```

```
            UInt32 start = TimGetSeconds();
            while (TimGetSeconds() - start < 7);

            UInt8 stopPacket[9];
            rcxInterface.buildPacket("\x21\x41", 2, stopPacket);

            if (!rcxInterface.sendPacket(stopPacket, sizeof(stopPacket)))
                        debugPrint("Unable to send command");

            break;
}

case CMD_FORWARD:
{
            debugPrint("command: forward");

            UInt8 onPacket[9];
            UInt8 dirPacket[9];

            rcxInterface.buildPacket("\x21\x82", 2, onPacket);
            rcxInterface.buildPacket("\xe1\x02", 2, dirPacket);

            if (!rcxInterface.sendPacket(dirPacket, sizeof(dirPacket)))
                        debugPrint("Unable to send command");

            SysTaskDelay(10);

            if (!rcxInterface.sendPacket(onPacket, sizeof(onPacket)))
                        debugPrint("Unable to send command");

            break;
}

case CMD_BACKWARD:
{
            debugPrint("command: backward");

            UInt8 onPacket[9];
            UInt8 dirPacket[9];
```

```
                rcxInterface.buildPacket("\x21\x82", 2, onPacket);
                rcxInterface.buildPacket("\xe1\x82", 2, dirPacket);

                if (!rcxInterface.sendPacket(dirPacket, sizeof(dirPacket)))
                        debugPrint("Unable to send command");

                SysTaskDelay(10);

                if (!rcxInterface.sendPacket(onPacket, sizeof(onPacket)))
                        debugPrint("Unable to send command");

                break;
        }

        case CMD_LEFT:
        {
                debugPrint("command: left");

                UInt8 onPacket[9];
                UInt8 dirPacket[9];

                rcxInterface.buildPacket("\x21\x84", 2, onPacket);
                rcxInterface.buildPacket("\xe1\x84", 2, dirPacket);

                if (!rcxInterface.sendPacket(dirPacket, sizeof(dirPacket)))
                        debugPrint("Unable to send command");

                SysTaskDelay(10);

                if (!rcxInterface.sendPacket(onPacket, sizeof(onPacket)))
                        debugPrint("Unable to send command");

                // must wait a couple of seconds and then power steering off
                timer = TimGetSeconds();

                break;
        }

        case CMD_RIGHT:
        {
            debugPrint("command: right");

            UInt8 onPacket[9];
            UInt8 dirPacket[9];

            rcxInterface.buildPacket("\x21\x84", 2, onPacket);
            rcxInterface.buildPacket("\xe1\x04", 2, dirPacket);

            if (!rcxInterface.sendPacket(dirPacket, sizeof(dirPacket)))
                        debugPrint("Unable to send command");

            SysTaskDelay(10);

            if (!rcxInterface.sendPacket(onPacket, sizeof(onPacket)))
                        debugPrint("Unable to send command");

            // must wait a couple of seconds and then power it off again
            timer = TimGetSeconds();

            break;
}

case CMD_STOP:
{
            debugPrint("command: stop");

            UInt8 stopPacket[9];

            rcxInterface.buildPacket("\x21\x42", 2, stopPacket);

            if (!rcxInterface.sendPacket(stopPacket, sizeof(stopPacket)))
                        debugPrint("Unable to send command");

            break;
}

default:
{
            debugPrint("command: unknown");
            break;
```

```
                        }
                }

                return true;
}

Boolean  MobileRobotics::onDisconnected(BtLibSocketEventType  *event)
{
                // since we're only accepting 1 client, disconnect #0
                disconnect(0);
                debugPrint("Client disconnected");

                UInt8 stopPacket[9];
                UInt8 beepPacket[9];

                rcxInterface.buildPacket("\x21\x42", 2, stopPacket);
                rcxInterface.buildPacket("\x51\x02", 2, beepPacket);

                if (!rcxInterface.sendPacket(beepPacket, sizeof(beepPacket)))
                        debugPrint("Unable to send command");

                SysTaskDelay(10);

                // stop forward/backward engine
                if (!rcxInterface.sendPacket(stopPacket, sizeof(stopPacket)))
                        debugPrint("Unable to send command");

                WinEraseChars("ok", 2, 47 + 13, 126 + 13 / 2 - 5);
                WinDrawChars("--", 2, 47 + 13, 126 + 13 / 2 - 5);

                // go back to listening
                if (!listen())
                {
                        debugPrint("Unable to return to listening state");
                        return false;
                }

                debugPrint("Server in listening state");

                return true;
```

```
}
```

```
/*
 * Copyright (c) 2004-2005 Mobile Robotics
 * http://mobilerobotics.sf.net
 *
 * File: RCXInterface.cpp
 * Author: Johan Johanson
 * Date: February 11, 2005
 * Updated: February 11, 2005
 *
 * Description: See RCXInterface.h
 *
 */

 #include "RCXInterface.h"

 RCXInterface::RCXInterface() : orRef(sysInvalidRefNum),  sendFrequency(72000),  recvFrequency(76000),  baudRate(2400),
                                                                                                 parity(parity_odd),
dataBits(8), lsbFirst(true), invertData(true), internalIR(true),
                                                                                                 lastOpcode(0)
 {

 }

 RCXInterface::~RCXInterface()
 {
          shutdown();
 }

 // loads the OR library and initializes the IR transceiver
Boolean RCXInterface::init()
 {
          if (orRef != sysInvalidRefNum)
                   return false;

          UInt16 status = 0;

          // open / find library
          if (SysLibFind("ORLib.prc", &orRef))
                   if (SysLibLoad('libr', 'ORlb', &orRef))
                            return false;
```

```
            OrLibOpen(orRef);

            // identify IR hardware and setup the type of IR used
            OrLibDeviceType(orRef, NULL, &status, internalIR);

            // check OmniRemote software status
            if (status & STATUS_BETAEXPIRED || status & STATUS_PORTBLOCKED || !(status & 0xFF))
                        return false;

            return true;
 }

void RCXInterface::shutdown()
{
            UInt16 numApps;

            // close library
            OrLibClose(orRef, &numApps);

            // if we were the last app to close the lib, remove it from memory
            if (!numApps)
                        SysLibRemove(orRef);

            orRef = sysInvalidRefNum;
}

// builds a IR sendable          Lego protocol packet from a number of opcodes
Boolean RCXInterface::buildPacket(const Char *opcodes, UInt16 length, UInt8 *buffer)
{
            if (opcodes == NULL || buffer == NULL || !length)
                        return false;

            // fill in the header (0x55 0xff 0x00)
            buffer[0] = 0x55;
            buffer[1] = 0xff;
            buffer[2] = 0x00;

            UInt16 j = 3, sum = 0;
            Char op = 0;
            // insert the opcodes and their complement into buffer
            for (UInt16 i = 0; i < length; i++)
            {
                        op = opcodes[i];

                        // check if the first opcode in the sequence is the same as in the previous packet
                        if (i == 0)
                                    if (opcodes[i] == lastOpcode)
                                                op = (lastOpcode ^= 0x08);       // if so, toggle (XOR) its 0x08 bit so
that the RCX accepts it
                                    else
                                                lastOpcode = op;      // remember the last opcode

                        buffer[j++] = op;                                             // the opcode
                        buffer[j++] = (~(op)) & 0xff;    // its complement

                        // add to the sum
                        sum += op;
            }

            // insert the the checksum and its complement to the end of the packet
            buffer[j++] = sum & 0xff;
            buffer[j] = (~sum) & 0xff;

            return true;
}

Boolean RCXInterface::sendPacket(UInt8 *data, UInt16 length)
{
            if (orRef == sysInvalidRefNum || data == NULL)
                        return false;

            // fire up IR
            OrLibStartPlaySample(orRef, sendFrequency);

            // calculate the minimum size for the generated sample buffer
            // (1 / baud rate) * bits per frame * data length * 1024
            UInt16 bitsPerFrame = (parity == parity_none) ? 10 : 11;
            UInt16 minBuffLen = (UInt16)((1.0f / (float)baudRate) * bitsPerFrame * length * 1024);
            UInt8 sampleBuffer[minBuffLen + 16];
            Int16 sampleLen = sizeof(sampleBuffer);

            // generate a sample from the data packet
            OrLibGenSampleFromData(orRef, sampleBuffer, &sampleLen, data, length, baudRate, sendFrequency,
(ParityEnum)parity,
                                                                                 lsbFirst, invertData);

            // send sample
            if (OrLibSendData(orRef, sampleBuffer, sampleLen, sendFrequency))
            {
                        OrLibEndPlaySample(orRef);
                        return false;
            }

            // put IR to sleep
            OrLibEndPlaySample(orRef);

            // receive reply
            //if (!recvPacket(data, &length, 10))
            //          return false;

            return true;
}

Boolean RCXInterface::recvPacket(UInt8 *data, UInt16 *length, UInt16 timeout)
{
            if (orRef == sysInvalidRefNum || data == NULL || length == NULL)
                        return false;

            // start receiving data, fire up the IR
            if (OrLibStartRcvData(orRef, baudRate) == 0)
            {
                        OrLibEndRcvData(orRef);
                        return false;
            }

            // receive the data
            if (OrLibRcvData(orRef, data, length, recvFrequency, dataBits, (parity != parity_none), lsbFirst,
                                                  invertData, timeout))
          {
                    OrLibEndRcvData(orRef);

                    data = NULL;
                    *length = 0;

                    return false;
          }

          // done receiving data
          OrLibEndRcvData(orRef);

          return true;
}

void RCXInterface::setTransceiverMode(UInt16 mode)
{
          if (orRef != sysInvalidRefNum)
          {
                    internalIR = (Boolean)mode;
                    OrLibDeviceType(orRef, NULL, NULL, internalIR);
          }
}

UInt16 RCXInterface::getTransceiverMode()
{
          return (UInt16)internalIR;
}

UInt16 RCXInterface::getLibRef()
{
          return orRef;
}

UInt32 RCXInterface::getSendFrequency()
{
          return sendFrequency;
}
```

```
UInt32 RCXInterface::getRecvFrequency()
{
           return recvFrequency;
}

UInt32 RCXInterface::getBaudRate()
{
           return baudRate;
}

UInt32 RCXInterface::getParity()
{
           return parity;
}

UInt16 RCXInterface::getDataBits()
{
           return dataBits;
}

Boolean RCXInterface::isLsbFirst()
{
           return lsbFirst;
}

Boolean RCXInterface::isInvertData()
{
           return invertData;
```

Lego Source code

```
// prog_1.nqc
// by johan johanson 3e
// controls the main steering engine

task main()
{
     // assign our touch sensor
     SetSensor(SENSOR_2, SENSOR_TOUCH);

     while (true)
     {
           // if it's pressed, the wheels are in
           // their center position
           if (SENSOR_2 == 1)
           {
              // and that's where we want 'em
              Off(OUT_C);

              // now wait until they turn out of range
              // before checking again
              until (SENSOR_2 == 0);
           }
     }
}

// prog_2.nqc
// by johan johanson 3e
// manual override of engine 1 controlling car power
task main()
{
     On(OUT_A);
     Wait(700);
     Off(OUT_A);
}

// prog_3.nqc
// by johan johanson 3e
// manual override of engine 1 controlling car power
task main()
{
     OnRev(OUT_A);
     Wait(700);
     Off(OUT_A);
}
```