MASTER'S THESIS

Communication Between a MHP Set-top Box and Mobile Devices

by

T.J.H. Geelen

Supervisor: dr. J.J. Lukkien

January 2003
## Authorization

<table>
<thead>
<tr>
<th>Company, function</th>
<th>Name</th>
<th>Date</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>NotTheFly,</td>
<td>Joost Backus</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Managing director</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Document History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author</th>
<th>Description</th>
<th>Send list</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td></td>
<td>Tom Geelen</td>
<td>Initial version</td>
<td></td>
</tr>
</tbody>
</table>

This document is property of:
NotTheFly Component Solutions BV · Venlo · The Netherlands

P.O. Box 231  5900 AE  Venlo, The Netherlands
Goltzusstraat 41  5911 AT  Venlo, The Netherlands
Commercial Register Venlo no. 12042631

Tel.: +31 77 3200035  Fax.: +31 77 3200992  Email: info@NotTheFly.com

© Copyright NotTheFly Component Solutions

All rights are reserved. No part of this publication may be reproduced in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the copyright owner.

Alle rechten voorbehouden. Verveelvuldiging, geheel of gedeeltelijk, is niet toegestaan zonder schriftelijke toestemming van de auteursrechtthebbende.
Contents

1. Introduction ........................................................................................................ 5
   1.1 Abstract ........................................................................................................ 5
   1.2 Problem Definition ..................................................................................... 6
   1.3 Overview of this Document ........................................................................ 6
   1.4 Definitions, Acronyms and Abbreviations .................................................. 6

2. The Multimedia Home Platform ..................................................................... 8
   2.1 System overview ........................................................................................ 8
   2.2 Standalone MHP applications .................................................................. 9
   2.3 Connection of the STB with a mobile device ........................................... 10
   2.4 Connection of the STB with the broadcaster ............................................ 11

3. The mobile device ......................................................................................... 13
   3.1 Introduction ................................................................................................ 13
   3.2 Java applications on a mobile device ....................................................... 13
   3.3 IrDA Protocol Stack .................................................................................. 14
   3.4 Bluetooth protocol stack .......................................................................... 18
   3.5 OBEX Implementation ............................................................................. 19

4. Connection with the broadcaster .................................................................. 21
   4.1 The return channel via a mobile phone ..................................................... 21
   4.2 Sending SMS messages ............................................................................ 22

5. Experience ...................................................................................................... 23
   5.1 Overview of the work done ....................................................................... 23
   5.2 Building Xlets ........................................................................................... 24
      5.2.1 MHP in more detail ............................................................................ 24
      5.2.2 Building an Xlet using Alticonnector ................................................ 25
   5.3 Infrared Using Java on a Palm ................................................................ 27
   5.4 Protocol tunneling ..................................................................................... 28
   5.5 Infrared hardware ..................................................................................... 29
      5.5.1 Initial Configuration ......................................................................... 29
      5.5.2 Required Software .......................................................................... 30
      5.5.3 Modified Configuration ................................................................... 31
   5.6 Achievements ............................................................................................ 32

6. Conclusions .................................................................................................... 33
   6.1 Future of MHP ......................................................................................... 33
   6.2 Added value of Bluetooth ....................................................................... 33

7. References ...................................................................................................... 35

Appendix A: Java IrSockets Patch ....................................................................... 36

Appendix B: Design Documents .......................................................................... 37
   B.1 Fly_Weight ............................................................................................... 37
   B.2 Fly_IRForwarder ..................................................................................... 42
   B.3 OBEX Implementation ............................................................................ 43
   B.4 Sending objects from the STB: Fly.Download ....................................... 47
   B.5 Receiving objects on the STB: Fly_Image ............................................. 48
   B.6 Remote Procedure Call: Fly_RemoteScreen .......................................... 48
   B.7 Accessing the internet: Fly_GPRS .......................................................... 51
   B.8 Sending SMS messages: Fly_SMS ......................................................... 51

Appendix C: Source code .................................................................................. 52
   C.1 com.notthefly.mhp.Fly.Download ......................................................... 52
   C.2 com.notthefly.mhp.Fly_GPRS ................................................................. 56
   C.3 com.notthefly.mhp.Fly.Image ................................................................. 59
   C.4 com.notthefly.mhp.Fly_IRForwarder .................................................... 62
| C.5  | com.notthefly.mhp.Fly_SMS                                      | 64  |
| C.6  | com.notthefly.mhp.Fly_Weight                                   | 68  |
| C.7  | com.notthefly.mhp.remotescreen                                | 80  |
| C.8  | com.notthefly.mhp.locale                                      | 90  |
| C.9  | com.notthefly.mhp.util                                       | 92  |
| C.10 | com.notthefly.obex                                           | 105 |
1. Introduction

1.1 Abstract

This document is about joining the world of mobile (Java) applications and MHP set-top boxes (receivers for interactive digital television). Such a set-top box (STB) receives a digital data stream from a broadcaster that contains the audio and video. This data stream can also contain applications written in the Java programming language that can be run on the STB to provide interactivity. For a higher level of interactivity, an STB can have an internet connection, called return channel, via which it can send data back to the broadcaster. PDA’s and many new mobile phones can also run Java applications. These phones often have internet access via GPRS and/or UMTS technology. The possibilities for letting these devices communicate with an STB were investigated. An overview is shown in Figure 1-1. The first thing looked at was infrared communication. After successfully using that, we got our hands on some Bluetooth devices (which use short-distance radio transmission), so we could also investigate and use this technology.

![Diagram showing broadcaster, optional return channel, internet, GPRS, set-top box, and TV.]  

**Figure 1-1: System overview**

Our first goal was to send a file from the STB (that received it from the broadcaster) to a mobile phone. This file can be a ring tone, a logo, an MP3 file (for phones with built-in MP3-players), a text message, etc.

However, we found that there are also some other applications possible that are useful. One was the transmission of files the other way around, e.g. sending an image from a phone with a built-in camera to the STB (which can show it on the TV screen). Another was to let the STB send SMS messages to arbitrary numbers using a phone (which provides a good billing model for interactive TV, which it currently lacks).
We also implemented a return channel for the STB via a phone with GPRS technology (UMTS will probably also work, but is not yet available at the time of this writing).

1.2 Problem Definition

Investigate and validate the possibilities for creating and using a communication channel between a set-top box and a mobile device. These are the gray arrows in Figure 2.1. A more detailed description of the system is given in the next chapter. Not considered are solutions where the user is the link, where he/she simply enters data displayed on one device into the other. These solutions already exist (for example, voting during a show by sending a some code per SMS message with your mobile phone) and offer only small possibilities.

1.3 Overview of this Document

First a description of the system will be given. There are two main communication channels involved: one between the STB and the broadcaster and one between the STB and a mobile device. These two will be discussed in more technical depth in chapters 3 and 4.1. Chapter 5 describes the various experiences gained during this project. The report ends with some conclusions and ideas. The designs, implementations and sources of the various programs written are included as appendices.

1.4 Definitions, Acronyms and Abbreviations

API
Application Programming Interface

Binding
An association between two protocols (typically, a protocol and its carrier) that map the required service of the one protocol onto the provided service of the other one.

Carrier
A protocol or a piece of hardware in the role of providing a service that is used by another protocol.

CLDC
Connected Limited Device Configuration

DVB
Digital Video Broadcasting / www.dvb.org

GPRS
General Packet Radio Service

GSM
Global System for Mobile communication

HTTP
Hypertext Transfer Protocol

IP
Internet Protocol

IrDA
Infrared Data Association / www.irda.org

J2ME
Java 2 Micro Edition

JMF
Java Media Framework

JVM
Java Virtual Machine

LSAP
Link Service Access Point

MHP
Multimedia Home Platform / www.mhp.org

MDet
CLDC/MIDP Java application

MIDP
Mobile Information Device Profile

NotTheFly
NotTheFly Component Solutions

PDA
Personal Data Assistant

Protocol
A formal set of rules that dictates how information exchange as well as interaction between objects (can be devices, execution threads, etc.) should take place. These rules specify

1. the format of the messages exchanged;
2. a number of different protocol states and what messages are allowed to be sent in each state;

3. timing constraints and other non-functional properties, if any.

A protocol can be implemented directly in hardware. If not, the messages of a protocol are transported by another protocol, called the carrier. The functionality and properties (e.g. reliable transfer of data) that a protocol provides is called the provided service. The provided service is often expressed using an API for the protocol. A protocol requires services from its carrier. Any carrier providing these services can be used. The rules that specify how a protocol is mapped onto a carrier is called a binding.

Service

A functionality that is provided by a protocol to an application or higher protocol layer. This functionality includes internal state of the protocol and means to change this state as well as non-functional properties (e.g., timing characteristics). This makes a service comparable to an object in the Object Oriented Design paradigm.

A service is accessible in one of two ways:

- through an API
- through an underlying protocol

Silverbox Phillips Silverbox STB, part of the Philips MHP Software Development Kit.
SMS Short Message Service
STB Set-top box
TCP Transmission Control Protocol
UDP User Datagram Protocol
UMTS Universal Mobile Telecommunications System
WAP Wireless Application Protocol
Xlet MHP Java application
2. The Multimedia Home Platform

2.1 System overview

MHP, which stands for Multimedia Home Platform [1], specifies a platform for interactive digital television. It extends from the Digital Video Broadcasting (DVB) standard [2], which specifies digital television broadcasting. A digital television transmission is assembled by the broadcaster. Such a transmission generally consists of at least one (digital) audio and video stream. The broadcast can also include applications (called Xlets) that can be run on the receiver, the so-called set-top box (STB) (usually a separate box that sits on top of the TV, but this receiver can also be integrated in a TV). Application data for an Xlet can also be included in the broadcast stream. These Xlets can be controlled by the user using the remote control. This way interaction with the user is provided. The MHP also specifies an optional return channel that can be used so send data back to the broadcaster. This return channel uses the Internet as medium. The access to the Internet can be implemented using any available technology. When a modem is used, an Xlet can specify a phone number, so it can also dial in to a private network. Of course, the return channel can also be used to access other Internet applications besides the broadcaster’s (in this context, the application running at the broadcaster can be seen as a normal Internet application). A schematic overview is shown in Figure 2-1.

An increasing number of mobile devices (mostly PDA’s and cellular phones) can also run applications (called MIDlets). Almost all of these mobile phones also have the ability to set up an Internet connection, which can be used by a MIDlet or another device (e.g. a laptop). As stated in the introduction, a communication channel between the STB and a mobile device can provide interesting uses. How this communication link can be implemented will be discussed later. It will then also become clear whether there will be communication with an application or with a service provided by the underlying platform (or a combination).

![Figure 2-1: Schematic system overview](image-url)
2.2 Standalone MHP applications

MHP applications are written in the Java language [3]. They can draw objects on the TV screen, while leaving the broadcast video stream visible. This can be accomplished by displaying it in a smaller portion of the screen or by using transparency. The remote control has a few colored buttons that an Xlet can use. Other buttons, like the numbers, may only be used by an Xlet when it is obviously in the foreground (this is because when in the background, the user expects those keys to have their default behavior (e.g. switch to another channel). Application data (like images) can be retrieved from the broadcast stream, or, when connected, from the return channel. This return channel can also be used to transmit data to some internet server. To become acquainted with the MHP, a stand-alone Xlet was built that only interacts with the user. The Xlet lets the user calculate his/her Body Mass Index and other health indexes related to his/her weight. A version of this application for Java-enabled mobile phones already existed. Some screenshots of this Xlet can be found in Figure 2-2. There is no video stream visible in these screenshots, because we didn't have video streams available in our development setup.
2.3 Connection of the STB with a mobile device

A connection with a mobile device (the gray arrows in Figure 2-1) can be implemented in several ways. The most straightforward one is using a cable. However, not all devices have the option to connect a data cable for communication and every device uses a different cable that has to be bought separately in most cases. A more convenient way to connect the STB with a mobile device is using a wireless connection. For this there are two accepted standards: IrDA [12], which uses infrared light, and Bluetooth [13], which uses a short-range radio.

One use of a connection between the STB and a mobile device is file transfer. Files can be sent from the STB to the mobile device. These can for example be applications for the device, ring tones for phones, mp3 files for an mp3 player or just a text document. Files can also be sent from a mobile device to the STB. A nice application that uses this is sending images from a digital camera to the STB, which can show them on the TV.

Another use of the connection is data exchange between an Xlet on the STB and an application on a mobile device. In this way applications can use resources on the other device, for example, an application on a mobile phone could use the TV screen for its output.

Yet another use for the connection is to use it as a return channel. Most mobile phones can be used as a modem, so the STB can use it to create a return channel. This requires hardly any installation, so this would be a nice alternative to get a phone line (or other cable providing internet access) to the STB.
2.4 Connection of the STB with the broadcaster

The connection with the broadcaster is shown in Figure 2-1 as the arrows labeled "broadcast". The broadcast follows the DVB standard, which uses MPEG-2 [15] as basis. MPEG-2 allows the broadcast stream to be built up from multiple individual streams. These are not only video and audio streams, which are compressed using a lossy compression algorithm. The subtitling is sent in a separate stream (more streams can be used for different languages). This subtitling uses bitmaps and thus can also be used to display a channel logo. The most important stream for MHP is the one containing the file system, as this the stream used to deliver the Xlets and their data (although the return channel can also be used for this, if present). This file system is transmitted as an object carrousel and is called DSM-CC by the MPEG-2 standard, which stands for Digital Storage Media Command Control. This works by transmitting all files sequentially and when all files are transmitted, it starts again from the beginning. Some files may be sent multiple times during a single round so they can be accessed faster, but this results in each round taking more time, so other files take more time to access. These often slow access times can be compensated by using caching in the STB. The way this file system is transmitted is comparable with the way the Teletext pages in the Europe are transmitted nowadays.

This DSM-CC is also used to encapsulate UDP/IP packets (which are unidirectional of course) and includes service information (DVB-SI), which contains an electronic program guide. The various streams are shown in Figure 2-3. In this figure the parts that are accessible from an Xlet through some API are indicated.

![Diagram](image)

*Figure 2-3: The various parts of the MPEG-2 stream*

The broadcast stream can be transmitted via a cable network (DVB-C standard), using radio transmission over land (DVB-T standard) or via satellite (DVB-S standard) [2]. A mapping of the used standards onto the ISO/OSI model is shown in Figure 2-4. As can be seen, this a pure broadcasting system since there is no network layer. Individual receivers cannot be addressed using this broadcast. For this, a separate connection, called the return channel can be used. This return channel is a bi-directional link with some IP-based network, usually the internet, over which the broadcaster can be reached. Thus, so as far as the return channel is concerned, the broadcasters application can be seen as a regular internet application. Note that the data link and transport layers are not identical, they are just defined within the same DVB standards.
An example of an application making use of this connection is a voting application. An Xlet on the STB lets the user vote for something and then uses the return channel to send the vote to the application at the broadcaster that records the vote. The results of all votes so far can be broadcast again to everyone (as a continuous update).

<table>
<thead>
<tr>
<th>ISO/OSI Model</th>
<th>DVB</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 - Application</td>
<td>STB software &amp; Xlet</td>
</tr>
<tr>
<td>6 - Presentation</td>
<td>MPEG-2 (decoding)</td>
</tr>
<tr>
<td>5 - Session</td>
<td>MPEG-2</td>
</tr>
<tr>
<td>4 - Transport</td>
<td>DVB-C/DVB-T/DVB-S</td>
</tr>
<tr>
<td>3 - Network</td>
<td>Not used</td>
</tr>
<tr>
<td>2 - Data Link</td>
<td>DVB-C/DVB-T/DVB-S</td>
</tr>
<tr>
<td>1 - Physical</td>
<td>Cable/Ether/Satellite</td>
</tr>
</tbody>
</table>

*Figure 2-4: ISO/OSI model for DVB*
3. The mobile device

3.1 Introduction

This chapter will investigate the possibilities from the mobile device’s side to build a communication link with the STB (the gray arrows in Figure 2-1). It will look at the possible physical links and their accompanying protocol stacks, at the API’s available to use these stacks and at applications that use these API’s.

3.2 Java applications on a mobile device

An increasing number of mobile phones have support for Java applications that can be downloaded to the phone over the air. These applications use Java 2 Micro Edition (J2ME) [6], which is a stripped down version of the Java 2 environment. J2ME comes in two configurations: the Connected Device Configuration (CDC) and Connected Limited Device Configuration (CLDC). The latter is specially designed for devices with a small amount of memory available (typically less than 512 kB) and this is the configuration used on mobile phones. On top of these configurations profiles are defined that add to the core J2ME libraries. The Java enabled phones use the Mobile Information Device Profile (MIDP), which addresses issues such as user interface, persistence storage, networking, and application life cycle. Java applications using this CLDC/MIDP configuration are called MIDlets. Unfortunately, in the currently used version (1.0) the only support for connections with other devices is through HTTP connections in the internet domain. These HTTP connections may use a non-IP based network (such as WAP) and a gateway to access the internet. There are manufacturers that supply their own API with communication functionalities (gray arrows in Figure 3-1), but when a MIDlet uses such an API, it isn’t very portable anymore (the programs only work on phones from a specific manufacturer). At the time of writing only Siemens has a communications API that lets a MIDlet send bytes to a MIDlet on another phone using IrDA. So letting the STB communicate with a MIDlet on a phone is hardly an option (currently there is only one Siemens phone with Java and IrDA, the SL45i, and it requires the actual data sent in the packets to have the length prefixed, which is not standard).

![Figure 3-1: The mobile device](image)

Another solution is not to provide an API to the greatest common divisor of different phone hardware (and accompanying protocol stacks), but to provide an API to almost all (services provided by) available protocols. Access to complete protocol stacks is not necessary, but, looking at the ISO/OSI model, access...
to the services provided by the presentation layer should be provided (since the application represents the application layer, this is logical). In addition, access to the transport layers should be provided, so application specific can be used, without the need of tunneling (see paragraph 5.4). For portability reasons, there should be some rules about which protocols and API’s should be present. As seen in the MIDP 1.0 solution, the HTTP protocol can always be implemented, so an API to it should always exist, so applications have a way to make internet connections on any phone. Other rules can be that when a phone has infrared and/or Bluetooth, all infrared and/or Bluetooth protocols should be available. An overview is shown in Figure 3-2 (the gray arrows indicate optional API’s).

The latter solution is the more preferred option, because then the capabilities of a phone can be used to their full extent, while still providing the option to write portable applications.

Nevertheless, fortunately almost every phone with IrDA or Bluetooth provides standard services on it. They always provide access to their modem (if they have one) and most of the phones and PDA’s provide file transfer functionalities. We can use these standardized services instead of MIDlets for the implementation of the gray arrows in Figure 2-1.

Almost all PDA’s have an IrDA port, which is mainly used for object exchange (exchange business cards, send applications). Since it is possible to write native applications for all major PDA platforms, you can have low-level access to the infrared port so you can do virtually anything with it. However, since it takes a lot of time to master the native code development, this was not further investigated. We only looked at the possibilities to communicate using Java programs and using the object transfer services of PDA’s offered by their operating systems.

![Figure 3-2: Alternative design for a mobile device](image)

3.3 IrDA Protocol Stack

The IrDA standard [12] uses a protocol stack with several layers as shown in Figure 3-3. The PHY layer represents the physical layer, i.e. the infrared transceiver. Its interface to higher layers consists just of a method to send bytes and the delivery of received bytes. The API is shown in Figure 3-4. The Infrared Link Access Protocol (IrLAP) is a protocol that provides link access. It has an API (see Figure 3-5) that provides means to discover other IrDA devices, broadcast data (unreliable) and set up a connection with another device over which flow-controlled byte streams can be reliably sent in both directions. This connecting uses a client-server model.
The Infrared Link Management Protocol (IrLMP) layer provides multiplexing, so multiple channels can be used. Its API (see Figure 3-6) has roughly the same functionality as the one for IrLAP, but it allows creating multiple independent connections to the same device. When connecting, a number called the Link Service Access Point (LSAP) has to be given in addition to the address ID of the remote device, to be able to differentiate between the different connections. Only one connection is allowed for an address/LSAP combination. It is possible for a connection to claim exclusive use, so it can rely on IrLAP flow-control. Other connections, if they exist and the request for the exclusive connection was honored, will become blocked during this exclusive connection.
Figure 3-6: IrLMP API

The optional Tiny Transport Protocol (TinyTP) provides flow-control and segmentation-and-reassembly for the individual channels. This can be needed, since the only flow control in the lower layers is in the IrLAP layer and is for all connections. If data flow on one connection is dependent on the data flow in another connection, deadlock can occur.

This is somewhat strange: flow control is introduced by IrLAP and then removed by the addition of multiplexing in the IrLMP, so it needs to be added again by TinyTP. The reason that flow control is already introduced by IrLAP is to allow for a minimal stack implementation for protocols that don’t need multiplexing. For example, the COMM protocol can use an exclusive IrLMP connection (effectively bypassing the IrLMP) and also doesn’t need the TinyTP protocol. The only reason that IrLMP is used in that set up, is for the IAS.

The API of the TinyTP layer (shown in Figure 3-7) allows creating connections the same way as IrLMP, over which flow-controlled data can be reliably sent in both directions. It is also possible to send unreliable packets as with IrLMP.

Figure 3-7: TinyTP API

The Information Access Service (IAS) is a lookup service that provides service discovery. It consists of a database with records describing the services. This database is maintained by the device and may be static if the offered services never change. Each record has a unique ID and a class name, describing the

Error! No text of specified style in document. - Error! No text of specified style in document.

Version Error! No text of specified style in document.

© Copyright NotTheFly Component Solutions 9-12-2002

Page 16 of 128
type of service. Additional information for a service is stored in the so-called attributes. An attribute that is usually present is the LSAP at which the service is listening. The API (see Figure 3-8) consist of a method to retrieve a given attribute for all services with a given class name. There are some other methods specified to query the database, but these are not required in an implementation. The protocol uses IrLMP as carrier with the fixed LSAP 0.

<table>
<thead>
<tr>
<th>IAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>getValueBYClass(address:DeviceAddress, className:Text, attributeName:Text): List of [ObjectID, AttributeValue]</td>
</tr>
</tbody>
</table>

Figure 3-8: IAS API

On top of the TinyTP layer several application protocols are defined. The most important ones in this case are the COM and OBEX protocols. The COM protocol emulates serial and parallel ports and is used for example for printing and modem services. This protocol also has a mode in which it doesn't use the TinyTP protocol and runs directly on top of the IrLMP using an exclusive connection. The Object Exchange (OBEX) protocol is the protocol used for sending and receiving files and other data. It can be best described as a compact form of the HTTP protocol. Like HTTP, it uses a client-server model with the client sending requests to which the server replies with some response. The sending and receiving of data from a Java application on a phone or Palm PDA uses this OBEX protocol.

When an OBEX connection is initiated, the initiating device first starts a discovery. This means it broadcasts its address (ID), name and hint bits to any devices in range. A device that is in range will reply with its ID, name and hint bits. These hint bits describe the device, e.g. whether it is a computer, a PDA, etc., what functionality it has, e.g. printer, modem, etc., and what protocols it speaks, e.g. OBEX, COM, etc. This discovery is handled by the IrLAP layer. Next, a connection is made with the IAS to ask the address of the OBEX service. Next, the TinyTP connection can be started and the first OBEX command (which will be the OBEX connect command) is sent over it. Now the protocol runs until an OBEX disconnect command is sent, or the connection is closed on IrLMP level. A message sequence diagram is shown in Figure 3-9.

![Diagram showing OBEX connect via IrDA](image)

Figure 3-9: OBEX connect via IrDA

The OBEX protocol uses a command-response scheme: the client sends command to the server, which sends a response back for every command. There are six commands in OBEX: CONNECT, DISCONNECT, GET, PUT, SETPATH and ABORT. The use of the CONNECT and DISCONNECT commands are clear. The GET command is used to request an object, the PUT command to send an object, the SETPATH command...
changes the directory that objects are retrieved from/send to (when directories are supported by the implementation). An OBEX command message consists of a byte indicating the command, possibly some parameters (depending on the command) and a sequence of headers. There are four types of headers: a single byte, a four-byte quantity, an arbitrary length byte sequence and an arbitrary length Unicode string. There are a fixed number of header names, encoded in a single byte (which also indicates the header type). After this byte the header value follows. The most important headers are Name, Length, Body and End-Of-Body. The Body and End-Of-Body (which are both of the byte sequence type) contain the actual object data. Usually, a transfer is split over multiple messages and the End-Of-Body is used to indicate that it is the last piece of object data that is sent. The response messages consist of a single-byte response code (these are the same codes as used by the HTTP protocol), which is also followed by a number of headers. In case of a GET command, the requested object is sent back in these response headers. When a command is split over multiple messages, this is indicated using a special bit in the command (when there are no errors, these commands should be answered with a “100 Continue” response. When a response is split over multiple messages, this is indicated by sending the “100 Continue” response code for all but the last message.

For full control over IrDA from an application, such as listing names of devices that are discovered, it has to access multiple protocol layers directly, so in that case it is not really a stack. However, it can also be used as a real stack. For example, on an OBEX connect the OBEX layer queries the IAS, which in turn does an IrLMP connect. The IrLMP then does a discovery and connects using IrLAP.

### 3.4 Bluetooth protocol stack

The Bluetooth functionalities and protocol stack are very similar to that for IrDA. The protocol stack for Bluetooth [13] is (partially) shown in Figure 3-10. SDP is the Service Discovery Protocol and is similar to the IAS in the IrDA stack. RFCOMM provides serial cable emulation and is as such comparable to the IrCOMM layer in the IrDA stack. However, the RFCOMM is also used as transport layer for other protocols, such as the OBEX protocol. This OBEX protocol is the same protocol as in the IrDA stack.

<table>
<thead>
<tr>
<th>OBEX</th>
<th>RFCOMM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDP</td>
<td>Logical Link Control and Adaptation Protocol (L2CAP)</td>
</tr>
<tr>
<td></td>
<td>Host Controller Interface (HCI)</td>
</tr>
<tr>
<td></td>
<td>Link Manager Protocol (LMP)</td>
</tr>
<tr>
<td></td>
<td>Baseband [Link Controller (LC)]</td>
</tr>
<tr>
<td></td>
<td>Bluetooth Radio</td>
</tr>
</tbody>
</table>

*Figure 3-10: Bluetooth Protocol Stack*

Bluetooth has authentication and authorization methods available in the lower layers. This is to prevent connections to a device without the owner’s knowledge and/or permission. With IrDA, this isn’t necessary since the communicating devices have to be close together (at most 1m according to the specification) and require a line of sight, so it’s hardly possible to set up an IrDA connection without the user’s knowledge.

A larger difference with IrDA is the way in which services are presented. IrDA has a database in which services are registered. An entry in this database does not indicate much more than the availability of a certain protocol and how to connect to it. The specification of the OBEX protocol (and probably other protocols as well) provide quite some features that allow it to be used for many different scenarios. Most of these features are optional however. This leads to lots of interoperability problems. To avoid this, Bluetooth has specified so-called profiles. A profile specifies required protocols and services for a specific scenario. Profiles can be extended, as shown in Figure 3-11. The File Transfer Profile, for example extends the Generic Object Exchange Profile, which defines the OBEX protocol and its basic requirements. The File Transfer Profile extends this by specifying exactly how files and directories should be transferred or
deleted, specifying how an OBEX server should respond when an operation isn’t allowed or valid and requiring folder listing capabilities (also exactly specified).

<table>
<thead>
<tr>
<th>Dial-Up Networking Profile</th>
<th>Fax Profile</th>
<th>Headset Profile</th>
<th>LAN Access Profile</th>
<th>Generic Object Exchange Profile</th>
<th>Service Discovery Application Profile</th>
<th>Cordless Telephony Profile</th>
<th>Intercom Profile</th>
<th>TCS-BIN Based Profiles</th>
</tr>
</thead>
</table>

Serial Port Profile

Generic Access Profile

**Figure 3-11: Bluetooth Profiles**

### 3.5 OBEX Implementation

Since the MHP standard has no options to communicate with other devices (besides the return channel), we had to have the software on STB modified to allow using the serial port on it (which is normally used for connecting a modem) from an Xlet. To this port we then connect an infrared or Bluetooth device. We chose to use intelligent devices that have the necessary protocol stacks implemented to save development time. Eventually these separate devices will not be needed and the necessary hard- and software will be either built-in or sold as add-on modules. The devices we use can run transparently the serial port emulation (IrCOMM for IrDA, Serial Port Profile for Bluetooth). They can also do OBEX transport: connect to OBEX services and just send the OBEX messages to the devices' serial port without having any knowledge about the protocol itself. This means that the implementation for the OBEX protocol must be in the STB. All other (lower) layers are taken care of by the intelligent devices. Since the implementation must run on the STB, it has to be written in Java. It can make use of Java’s InputStream and OutputStream classes to read and write data to the serial port (and thus, the intelligent device) and supplies object send and receive services to Xlets (actually the OBEX implementation will be a part of an Xlet, since it is not possible to install separate components on the STB). For the client-side of the protocol an interface using connect, disconnect, send (put in OBEX terminology) and receive (get) methods is supplied. For the server-side of the protocol, a callback system is used to inform the Xlet of connects, disconnects, put-requests and get-request. See Figure 3-12. The object data is read or written through an InputStream and OutputStream respectively. The exact protocol requirements are specified in [11] and details about the implementation can be found in appendix B.3.
Figure 3.12: Place of the OBEX implementation
4. Connection with the broadcaster

4.1 The return channel via a mobile phone

A mobile phone can also be used to pass on data to the internet. Using this the STB can access the internet via the phone instead of (or in addition to) directly using a conventional cable, DSL or modem connection (see Figure 2-1). Most newer phones have GPRS support. GPRS provides packet switched data transfer (like IP). This allows for higher data transfer speeds than GSM-data (which sends data over an ordinary GSM connection) and allows users to be always online. Users only pay for the packets (i.e. data volume) they send and receive, not for the duration of the connection.

GPRS internet access from the STB is achieved by connecting to the GPRS "modem" that is built-in in every GPRS phone. It is not a modem in the sense that it does modulation and demodulation, but from the application point of view, it behaves as a conventional modem. After setting up the connection by dialing a special phone number, a PPP session is started to allow TCP/IP packets to be transmitted. To send TCP/IP data over GPRS no special protocol like PPP is needed, so unlike the normal telephone line case where the other end of the connection runs a PPP server, the PPP server is run on the phone. When a STB wants to use such a GPRS connection all it needs is access to the GPRS modem using its serial port and just start a connection like if it were just a regular modem. In MHP, a dial API is included that lets an Xlet specify the dial string that will be sent to the modem. Then the Xlet can use the standard java.net API for internet access.

![Diagram of GPRS Return channel using Bluetooth](image)

Figure 4-1: GPRS Return channel using Bluetooth

Some phones simply provide a RS232 data cable that can be used to access the modem. Other options are the use of IrDA (uses IrCOMM protocol) or Bluetooth (uses Dial-Up Networking Profile). For these to work with a standard STB, an intelligent device is needed that transparently "converts" RS232 to IrDA or Bluetooth. These devices exist, for example the ACTISYS ACT-IR1005-L series for IrDA and the Stollmann BlueRS+ for Bluetooth. For Bluetooth the various protocols used in the communication are shown in Figure
4.1. For IrDA, the situation is similar. The Xlet doesn’t have to do much special; it can just use the regular return channel API provided by the MHP. The only thing that makes it different from a regular Xlet is that it should request to let the GPRS modem dial the special phone number for GPRS access (and supply a username and password for the connection if required).

4.2 Sending SMS messages

The link with a phone via IrDA or Bluetooth can also be used to send SMS messages to arbitrary numbers (but the most useful would be some SMS-server behind some premium rate number). For this, a connection is needed with the modem in a phone. Just some special commands have to be sent to send an SMS message (it uses an extension to the standard AT-command set for modems). This is accomplished in the same way as for the GPRS return channel. However, since the MHP return channel API doesn’t let you send your own commands to the modem (only the phone number and credentials for the PPP connection can be specified by an Xlet), we need the direct serial port access modification, so we have full control over what is sent to the modem.
5. Experience

5.1 Overview of the work done

The first thing I did was reading about the MHP. A more detailed description of Xlets and their requirements can be found in paragraph 5.2.1. To get some hands-on experience with MHP I tried to port a simple existing MIDlet for the mobile phone to the MHP using a program called AlticComposer, which lets you design Xlets without the need to program in Java yourself. More about this is in paragraph 5.2.2.

We didn’t have a STB at that time, so I then started looking at infrared communication. We ordered an intelligent infrared device that connects to a serial port. I tried to send bytes using Infrared between PalmOS based PDA’s (I had a Palm V and a Handspring Visor available), a mobile phone with infrared (we used the Siemens SL45i, since it has a (rather limited) Java API for infrared) and the intelligent device. More information about programming for PalmOS devices is in paragraph 5.3. Both the Siemens phone and the Palm PDA’s use the OBEX protocol to send data using infrared. The choice for using this protocol is not ideal as is explained in paragraph 5.4. Our intelligent device can transport OBEX messages, but does not interpret them. So for communication with the intelligent device I wrote an implementation of the OBEX protocol in Java. Using this, communication with either the phone or the PDA and the intelligent device (connected to a PC) worked, but only for sending data to the PC. This was because the intelligent device can not initiate OBEX connections, it can only accept them. Sending bytes from the PDA to the phone or vice versa didn’t work at all, because these two use different headers in their OBEX request and won’t recognize each others messages (the intended use was communication between two equal devices). Sending bytes between the Palm V to the Handspring Visor worked fine, as could be expected.

Next, we got a Silverbox STB from Philips, which was still under development. The MHP implementation on that STB wasn’t completely finished and contained some bugs. From time to time newer versions of the STB software became available. To get acquainted with MHP programming, I ported the same MIDlet I also made in AlticComposer to an Xlet. For this, most of the code was rewritten from scratch. Since we cannot build our own broadcast streams (this requires very expensive hardware), a special development setup is used to test Xlets on the Silverbox: the STB starts a PPP session with a PC to which it is connected using a null-modem cable. Using the HTTP protocol it loads Xlets and their data files. A telnet connection can also be made to the box over this connection to which debug output is sent. Other TCP and/or UDP connections can be made by an Xlet using the standard java.net package (which is part of return channel API).

Then it was time to make an Xlet that can communicate with the Siemens phone and Palm PDA. Since MHP does not support access to a STB’s serial port, I had to use a PC between the infrared device and the STB. We had also exchanged our intelligent device with a dumb device (which only implements the IrPHY layer), since the Intelligent device cannot initiate OBEX connections. The first goal was to build an Xlet that can send a file that it received from the broadcaster via infrared. Receiving is done by the standard file transfer functionality built into the operating systems of the SL45i and Palm PDA’s. More about this can be found in paragraph 5.5.

When sending files from the STB to mobile devices worked, a new phone had become available: the Nokia 7650. This phone has a built-in camera, and IrDA and Bluetooth communication possibilities (but all of this is unfortunately not accessible from a MIDlet on the phone). I developed a new application for the STB that can receive files from mobile devices, in this case images taken by the phone’s camera, which it shows on the TV screen.

Also the Bluetooth connectivity was investigated. Since Bluetooth uses the same OBEX protocol, it should not be much work to get the same applications working using Bluetooth to communicate. We found an intelligent Bluetooth device that was similar to the intelligent IrDA device: it connects to the serial port, provides transparent serial cable replacement (using the Bluetooth Serial Port Profile) and can transport OBEX messages. This device can, unlike the IrDA device, initiate OBEX connections. Using this device I managed to get the existing applications working with Bluetooth.

Then a new version of the STB software became available that included modem support (for the return channel). I started experimenting with GPRS internet connections using the phone’s modem. The connection to this modem was via the intelligent Bluetooth device. This connection worked, but getting it
to work with the STB gave some problems, due to serial port speed settings. I could get it to work, but
only at a fixed speed of 1200 baud.
We then got another version of the STB software, which allows for access to the serial port. This version
was custom made for us, after we demonstrated our applications (which used a PC in between). I only had
to make small modifications to make the Xlets use the API for the serial port.

5.2 Building Xlets

5.2.1 MHP in more detail

Applications for a STB are small compiled Java programs called Xlets. The lifecycle of such an Xlet is
managed by an application manager. This is shown in Figure 5-1. The implementation may choose to use a
single Java Virtual Machine (JVM) to run all Xlets and the application manager in, or to start a separate
JVM for each Xlet. When an Xlet is instantiated, a method of the Xlet is called to initialize it. It is then in
the paused state. In this state it does nothing, but is ready to start. When an Xlet needs to be started, its
start method is called. When started, an Xlet can be paused again. It must then release any scarce
resource (e.g. the TV screen or the MPEG decoder) it was using. An Xlet can be destroyed at any time,
regardless of its state. When an Xlet is destroyed a method in it is called that should release all resources
the Xlet was using and that can do some other clean up as well.

![Figure 5-1: Xlets in MHP](image)

The input and output possibilities available to an Xlet for communicating with other devices are quite
limited. Of course it has access to the user events from the remote control (by registering event handlers)
and to the television broadcast (i.e. the audio and video streams). These streams can be played using the
Java Media Framework (JMF) [14]. Besides these, an Xlet also has access to a broadcast file system, which
takes the form of an object carousel. This works in a similar way as the broadcasting of Teletext pages in
the Europe. The contents of the files in file system are sequentially broadcast and after all files are
transmitted, it is repeated from the beginning. Obviously, a STB cannot write to this file system (but it is
not a read-only file system because the broadcaster can write to it). There is however also persistent
storage available for Xlets on the STB itself. Another communication option are the smart cards used for
conditional access (CA). An Xlet can do much with these, but since MHP 1.1 Xlets may access other non-CA
smart cards in the card reader through the OpenCard Framework API. However this API is optional so it is
not always available.

For the platform Java is chosen, because the various STB’s can use completely different processors and
architecture. The platform is based on PersonalJava [4], which is a stripped down version of the Java
platform specially designed for small devices with limited resources. Most of the API’s is taken from already
existing API’s. The main part consists of Suns JavaTV API [5], complemented with the HAVi TV user
interface API [7] and the DAVIC specification [8]. Furthermore, for the configuration files that are used in the standard (e.g. application description files) the XML standard [9] is used and some other existing API's are adopted, for example the optional support to access smart cards (other than those used for conditional access) is done via the OpenCard API [10].

The most useful communication possibility is the return channel. An API is provided for internet- and other TCP/IP based connections. A connection can be always connected, or require to be explicitly set up. In the latter case, it allows the Xlet to specify a phone number to dial to when the connection is made using a modem. This return channel functionality is however not always available, simply because an STB doesn’t support it, or because the STB is not connected to a modem or network. For these bi-directional return channel connections the full java.net API is available. An overview of the available API's is shown in Figure 5-2.

![Diagram of API connections](image)

Legend:
- I/O Resource
- Required API
- Optional API

*Figure 5-2: Input and output available to an Xlet*

### 5.2.2 Building an Xlet using AltiComposer

AltiCast has made a program to develop Xlets without the need to program in Java. The program is intended to be used primarily by designers, not programmers. You just use drag and drop, property sheets and a JavaScript like scripting language to create slideshow-like applications. When you want to do more than that, it isn't very useful as you will soon reach the limitations of the scripting language. However, it is possible to write additional components in Java that can be used in the program (this uses JavaBean technology [16]). The program can generate a completely packaged Xlet, consisting of Java class files and the used resource files, such as images.

As an exercise, I started to make an application using AltiComposer, based on the Fly_Weight Java program that exists for mobile phones. This was to get better acquainted with designing MHP applications.
In addition, this should give insight into the usability of such a program, because such a program can be very useful, since for most interactive applications, not much programming is needed, and the quality of such an application is heavily dependent on its graphical design.

When using the program I soon encountered the first problem: I couldn’t find a way to react to key-press events from the remote control (especially the number keys). The arrow and enter keys are handled automatically by the code generated by AltiComposer for navigation. Therefore, I drew a keypad on the screen using buttons for the numerical input (not the best solution, since a user will expect to be able to use the number keys on his remote control).

The next problem was that there are no global variables. I first tried to use properties of hidden elements, but these get reset when moving from one so-called scene to the next. A possible workaround for this could be to use only one scene, and write a huge script that rearranges the components in the scene and sets their visibility. However, this would make this simple and small application already quite complex. Therefore, as a solution I tried to write an invisible JavaBean component to store variables. This component is very simple, it just contains a java.awt.HashTable and public methods to store and retrieve values from/to it. However, I couldn’t get this (or any other) JavaBean component to work. After e-mailing with AltiCast it appeared to be a known bug in their software, and after patching it, I could make it work with a little effort.

The next problem is that when you enter a new scene, most variables are initialized to constant values set in the application. Among them is the text attribute of a text label. However, in the application there is a scene where I want to have the text in the label dynamically generated depending on previously stored variables. Since there is only one button in that scene, I worked around it by putting the code that generates the text into the FocusGained event of that button.

The next problem I encountered was floating point math. The scripting language supports simple floating-point operations, but not raising power to a non-integer exponent, which I needed. To solve the problem I decided to use the jscience.net fixed point library, which was also used in the Fly_Weight program. To be able to use it wrote a wrapper-component for that class.

Finally, I tried to write a component that can generate key events for the key on the remote control. However, I didn’t succeed in getting it to work in the emulator supplied with the program.

Unfortunately, the resulting Xlet that is generated is quite large, so it takes a long time to load on a real set-top box and after loading, it showed nothing, although it worked fine in AltiCast’s emulator.

Concluding I can say that I got a better idea of the possibilities and limitations of Xlets. The program can be useful for designers (if the resulting Xlets work on a real STB), although some functionality is lacking. Most of this can be added through self-written components, as I did. These components should be included with the package (or be downloadable from AltiCast’s website), since creating these components is not a task for the intended audience (designers). Strange thing about the program is that you need the scripting language all the time, even for simple things like going from one scene to another. Moreover, the way of entering code is irritating, since you can’t just type it in, you must use the mouse to select statements from a list. This is probably done for the people that can hardly program so they don’t have to worry about syntax, but there should have been also an option to just type it in.
5.3 Infrared Using Java on a Palm

There are various Java Virtual Machines (JVMs) available for the PalmOS platform that the Palm PDAs use. Most of them, unfortunately, haven’t support for the IR port. Next follows a list of Java-for-Palm solutions that were investigated.

<table>
<thead>
<tr>
<th>Name</th>
<th>URL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SuperWaba</td>
<td><a href="http://www.superwaba.org/">http://www.superwaba.org/</a></td>
<td>JVM with nice API and a lot functionality. It provides low-level access to the IR port and IrCOMM access. IrOBEX is not supported however.</td>
</tr>
<tr>
<td>Sun’s KVM</td>
<td><a href="http://webdev.apl.jhu.edu/~rbe/kvm/index.html">http://webdev.apl.jhu.edu/~rbe/kvm/index.html</a></td>
<td>This KVM isn’t supported anymore by Sun, but is has IrOBEX support through a send method and a receive callback method.</td>
</tr>
<tr>
<td>Sun’s J2ME CLDC/MIDP</td>
<td><a href="http://java.sun.com/products/midp4palm/">http://java.sun.com/products/midp4palm/</a></td>
<td>This is Sun’s official JVM for the Palm (and other small devices, like phones). There is no IR port support in it.</td>
</tr>
<tr>
<td>Jump</td>
<td><a href="http://sourceforge.net/projects/jump">http://sourceforge.net/projects/jump</a></td>
<td>Compiles Java applications into native code. API consists mainly of one big class with methods for native OS calls. Doesn’t have the methods for the native OBEX calls.</td>
</tr>
</tbody>
</table>

Figure 5-3: Screenshots of the resulting Xlet in Alticast’s emulator
Table 5-1: JVM’s for PalmOS

We decided to use Sun’s KVM, since it is the only solution that has direct IrOBEX. For SuperWaba an entire IrDA stack must be written to be able to use IrOBEX. I didn’t take much time to look into the complex PalmOS calls that Jump uses, but it seems that there is no support for OBEX. Maybe low-level access is possible, but then also a complete IrDA stack has to be written.

Using the KVM a small application was written that can send and receive a few bytes to and from another Palm (or any other device that uses IrOBEX and uses the same headers to describe the packets.

5.4 Protocol tunneling

The IrDA protocol stack can be mapped onto the layers defined by the ISO/OSI networking model, as shown in Figure 5-4. The Bluetooth protocol stack is similar to the IrDA stack and also supports the OBEX protocol (which uses Bluetooth’s RFCOMM protocol as carrier). With our demo set-up, we used a PC to forward TinyTP data to TCP/IP data and vice versa. These protocols are at the same OSI layer (layer 4, the transport layer), so this didn’t complicate the architectural design very much.

When something else than files has to be transferred, things become somewhat ugly. When applications want to exchange data over infrared the OBEX protocol shouldn’t be used, but the IrCOMM protocol or some other application specific protocol that uses the TinyTP layer as carrier. However, the JVM we used on a Palm V and the IrDA API that Siemens provides for its Java phones only allow sending bytes using OBEX. When sending to a device of the same type this choice isn’t noticed, since it will work as expected, it just sends the bytes as requested (but if only a few bytes are sent each time, the performance is bad because of the protocol overhead).

<table>
<thead>
<tr>
<th>ISO/OSI Model</th>
<th>IrDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - Physical</td>
<td>PHY</td>
</tr>
<tr>
<td>2 - Data</td>
<td>IrLAP</td>
</tr>
<tr>
<td>3 - Network</td>
<td>IrLMP</td>
</tr>
<tr>
<td>4 - Transport</td>
<td>TinyTP</td>
</tr>
<tr>
<td>5 - Session</td>
<td></td>
</tr>
<tr>
<td>6 - Presentation</td>
<td></td>
</tr>
<tr>
<td>7 - Application</td>
<td>File transfer</td>
</tr>
</tbody>
</table>

Figure 5-4: Mapping of the IrDA stack to the ISO/OSI model

That the choice of providing send and receive methods using the OBEX protocol is not ideal becomes clear when trying to communicate with another device. For example, sending something from a Siemens phone to a Palm or vice versa doesn’t work at all, since the API is too restrictive: an application has no influence on the OBEX headers used (except for the header containing the actual data, of course). Because of this they don’t recognize the objects of each other as they expect specific header values (Siemens for example wants the object name to be "java.jd"). In fact, these devices use the OBEX protocol as a transport layer, for which it is not intended. What happens here is so-called protocol tunneling: a protocol uses a carrier that is (according to the OSI model) at the same or higher layer. In our case, an application level protocol (defined by the application) uses another application level protocol (IrOBEX) as carrier.
tunneling is not elegant, it is usually a hack, which is used when no transport layer is available, but some application protocol is.
An example is when being behind a firewall that blocks general TCP/IP traffic, but allows HTTP traffic to pass through. Then a HTTP tunnel with another host outside of the firewall can be set up and various protocols can be run over this HTTP tunnel (i.e. use the HTTP as carrier). Protocol tunneling is also used when one wants to use existing technology, but in which some functionality is missing in some layers. An example of this are the protocols for Virtual Private Networking that tunnel IP packets in other IP packets, where the former are encrypted before they are encapsulated in the latter.
Note that not every protocol can be tunnelled; this is because of the services that a protocol requires from it carrier. For example, if the protocol to be tunnelled uses bi-directional communication, the carrier protocol must also be bi-directional. And if a protocol is client-server based (i.e. using a request-response scheme), it can only act as carrier for client-server based protocols, since the server side cannot start sending data. However, if the protocol to be tunnelled has no strict timing restrictions, this can be worked around by letting the client poll the server at regular intervals to check if it wants to send something (which can then be sent back as reply to the poll). Also timing requirements and other non-functional requirements (if any) have to be fulfilled, for example a protocol that requires a certain throughput (like streaming audio or video) might not work on a protocol that can only reply with one-byte response codes, where a request has to be done for each byte that must be received. For some tunnel applications the binding can become quite complex, resulting in an extra "layer" that is inserted between the protocols to add the missing services and other requirements.

5.5 Infrared hardware

5.5.1 Initial Configuration
Infrared communication with a Java program on the Siemens SL45i or on a Palm PDA is only possible using the IrOBEX protocol. You don't have any control over the protocol from Java; you just have a send method to send an array of bytes and a callback function to receive an array of bytes. These bytes are sent as an OBEX object. The intelligent IrDA adapter we had, the ACT-IR1005-L, has an IrDA protocol stack built in. This device has an RS232 interface and can transparently use the IrCOMM protocol, which simulates an RS232 interface over IR. It can act as carrier for the OBEX protocol (so OBEX messages are sent and received at the serial port). So in this case you have to be able to have control over what bytes exactly are sent out of this port. Since we want to establish a communications channel with a MHP set-top-box, the serial port of this box has to be controlled. This set-up is shown in Figure 5-5. Unfortunately, using the serial port directly is not supported within the MHP standard and it hasn't built-in IrDA either. At that time, we didn't have the direct serial port access extension to MHP from Philips for our Silverbox, so we had to try another way to at least be able to build a demo. The box does support TCP/IP over the debug connection on the serial port, which on its own isn't usable for IrOBEX communication. However, we can use a PC in between as a forwarder, forwarding OBEX messages that arrive via TCP/IP to the serial port and vice versa. This way, we can build the demo set-up that is shown in Figure 5-6.

![Figure 5-5: Intended configuration](image-url)
5.5.2 Required Software

The protocol stack for infrared communication is shown in Figure 5-7. The gray boxes indicate the parts that are already provided by the hardware and/or operating system software. When using IrOBEX with the IR100S-L, you send and receive OBEX messages over the serial port. Therefore, an implementation of the OBEX protocol is needed. Since this implementation should eventually run on the STB, it should be written in Java.
5.5.3 Modified Configuration

No new hardware comes without its problems. The intelligent adapter we had, the ACT-IR100S-L V1.00 had a bug, we found out. This bug prevented to send OBEX messages back (the responses to the commands). We got a new ACT-IR100S-L V1.01, which worked ok. However (this was in the specs), it can only act as a secondary device, which means it cannot initiate IR connections.

With IrDA, one device has the primary, active role and one or more others a secondary, passive role. It is the primary device that does the discovery and then connects to one or more discovered devices. Therefore, the primary device fulfills a client role, while the secondary devices has a server role. It is preferred that the STB initiates the transfer for our application, so it should be the primary device (which is not possible using the ACT-IR100S-L).

When a (primary) device has made a connection to an OBEX server on another (secondary) device, data can be exchanged in both directions, but it is controlled by the client (primary device) that does either a data-push or a data-pull. In our situation on both the phone and PDA, we can do only a data-push (there is no API for data-pull). To receive data, the other device has to do a data-push. This means it is required that STB has the primary role and does a data-push, but since this is not possible with our ACT-IR100S-L, we need to find a workaround.

The ACT-IR100S can play both primary and secondary roles, but this version unfortunately has no support for IrOBEX. As a solution to this, we use a simple, non-intelligent IR adapter (the ACT-IR220+) on the serial port of the PC, with the IrDA stack as driver software on the PC.

The resulting protocol stack situation is shown in Figure 5-8. Comparing this with Figure 5-7 it becomes clear that the same software can be used for this set-up.

For this set-up to work successfully with all devices, a small change had to be made to the C code that connects the Java API to the Linux IrDA driver (patch included in Appendix A). During discovery a device sends a set of hint bits that indicate the type of device (e.g. Computer, PDA) and the possibilities (e.g. Modem, Fax, IrCOMM). Among these bits is also a bit to indicate OBEX support, but this bit is not turned on automatically when you register an OBEX server with the stack (this registering consists of starting to listen at an TinyTP TTPSAP and add this to the IAS database). Some devices, like the Palm, won't even try

![Diagram](image)

Figure 5-8: Set-up with ACT-IR220(+).
to connect if this bit isn't set, so the code was modified to turn this bit on when registering a server. Since there were some troubles identifying the name of the service that is being registered in the code, this bit is turned on for all servers, whether they are OBEX servers or not. We won't be running any other servers than OBEX servers, so in our setup this is not a problem.

5.6 Achievements

This paragraph shows what kinds of communications between the STB and mobile devices were achieved. There are working demo applications for sending and receiving files via Bluetooth and IrDA using the OBEX protocol. The IrDA demo's use the set-up with the PC as described in paragraph 5.5.3. There is also a working demo that uses a GPRS return channel (via Bluetooth). Also an application that can send arbitrary SMS messages from the STB (also using a Bluetooth connection) was made. The performance for the GPRS return channel is very bad, because the STB only wants to talk to the "modem" at 1200 baud. This results in a download speed of about 3 kB/min (for fetching an image from an internet server). For the OBEX transfers, the speeds are shown in Table 5-2. For these the serial port can be used at maximum speed (115200 baud). The speeds achieved vary, depending on the maximum speed a device supports and the speed that the STB and mobile device negotiate for the wireless link (this speed will be lower when there is interference from the environment). Another parameter that influences the speed of OBEX transfers, is the maximum OBEX packet size that is used. Larger values result greater throughput, but it is restricted by the size of the buffers used in the infrared hardware. The cause that sending to the STB is much slower is that the devices use relatively small OBEX packet size. This means that more packets have to be transmitted, resulting in more overhead.

<table>
<thead>
<tr>
<th>Type of transfer</th>
<th>Approx. transfer speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>From STB to Nokia 7650, via Bluetooth</td>
<td>3.5 kB/s</td>
</tr>
<tr>
<td>From STB to Nokia 7650, via IrDA</td>
<td>2.6 kB/s</td>
</tr>
<tr>
<td>From STB to Handspring Visor, via IrDA</td>
<td>4.1 kB/s</td>
</tr>
<tr>
<td>From Nokia 7650 to STB, via Bluetooth</td>
<td>1 kB/s</td>
</tr>
<tr>
<td>From Nokia 7650 to STB, via IrDA</td>
<td>1 kB/s</td>
</tr>
<tr>
<td>From Handspring Visor to STB, via IrDA</td>
<td>1 kB/s</td>
</tr>
</tbody>
</table>

*Table 5-2: OBEX transfer speeds*
6. Conclusions

6.1 Future of MHP

Although I do not think many consumers will want to pay for interactive television, MHP can become a success, because consumers have to buy a set-top box when the broadcasters decide to use digital broadcasting only. Then MHP would be a good choice, since it’s a widely spread and open standard in enhanced digital broadcasting, supported by most big companies.

However, even when MHP is common, the popularity of interactive MHP applications stands or falls with good quality content. The average “couch potato” won’t use complex applications or go through applications with too many steps. The interface and user experience should be roughly the same for all MHP applications. For this purpose some basic rules have been put into the MHP standard. For example, when the input from the remote control goes to an application, that application must occupy at least a certain percentage of the screen to make this clear.

Another problem that could stand interactive applications in the way is the lack of a useful business model. Applications are just delivered with the audio and video data, at no extra charge. So why should one develop MHP applications if it doesn’t generate a revenue? Useful can be advertising applications that give detailed information about some products in the hope that the consumer will buy them. Nevertheless, the consumer won’t accept just interactive commercials; he’ll want interactive shows as well. The latter can make a channel more attractive so advertisers would pay more. However, I don’t expect this to compensate for the development costs.

The only useful business model is to let the STB dial in to a premium rate phone number. This is however only possible when a phone line (and modem, if not built in to the STB) is connected to the STB.

If, however, MHP is extended with IrDA and/or Bluetooth support (or serial port access, but then the consumer has to buy an IrDA/Bluetooth add-on module) we can use our connectivity solutions to create a personal return channel for everybody watching TV via their mobile phones. This not only greatly enhances the possibilities (everybody can vote simultaneously on a poll for example, and not just the person with the remote), but also allows users to be charged for services (by means of GPRS traffic or sending an SMS message).

6.2 Added value of Bluetooth

Bluetooth provides cable replacement, just like IrDA. However, without the line of sight requirement of IrDA and the much wider range (10 to 50 meters), it is a lot more convenient than IrDA.

Although the use of profiles as discussed in paragraph 3.4 should reduce the interoperability problems, they unfortunately still exist. This is because some profiles have some overlap, so in some cases it isn’t clear which profile to use. For example, when you try to send an image via Bluetooth with the Nokia 7650 phone, it will try to use the Object Push Profile. This profile is mainly used to send business cards, calendar items, messages, and notes to other devices, but any object may be sent. However, most devices don’t accept arbitrary objects on its Object Push server. For this application the use of the File Transfer Profile was also a (better) option.

Bluetooth offers more than just cable replacement. Since there is no need to physically do something to connect devices (e.g. plug in a cable, or point infrared eyes to each other), devices can set up spontaneous connections. This spontaneous networking allows for a new range of applications. For
example, a coffee machine can display a message on the TV screen stating that the coffee is ready. This can be a convenient application, but nobody will lay cables in their house for this. A television will become no more than a display and a set-top box will become just a central digital receiver and storage of different types of information (radio, TV, internet, photos, CD's).
7. References

Appendix A: Java IrSockets Patch

This is the OBEX hint bit patch for Java IrSockets-1.1. Apply by running "patch -p0 < patch.txt" in the Java IrSockets directory (contents of the patch.txt file follow).

```c
--- src/IrSocketImpl.c  Tue Jan 30 23:15:10 2001
+++ src/IrSocketImpl.c  Thu Mar 14 13:29:00 2002
@@ -176,7 +176,22 @@
   *(JNIEnv *env, jclass cls, jobject fdobj, jint backlog);
 {
   int fd, res;
+  struct sockaddr_irda addr;
+  int hints;
+  fd = fdValue(env, fdobj);
+  
+  /* Set OBEX hint bit for OBEX servers -- added by Tom Geelen */
+  if (addr.sir_family != AF_IRDA) (*env)->ThrowNew(env, (*env)->FindClass(env, "java/io/IOException"), "Got crap!");
+  /*
+   * See setsockopt(fd, SOL_IRLMP, IRLMP_HINTS_SET, &hints, sizeof hints)!
+   * (hints & HINT_EXTENSIONS) | (HINT_IRMP << 8)
+   * if (setsockopt(fd, SOL_IRLMP, IRLMP_HINTS_SET, &hints, sizeof hints))
+   * (*env)->ThrowNew(env, (*env)->FindClass(env, "java/io/IOException"), strerror(errno));
+   */
   res = listen(fd, backlog);
   if (res == -1)
      (*env)->ThrowNew(env, (*env)->FindClass(env, "java/io/IOException"), strerror(errno));
```
Appendix B: Design Documents

B.1 Fly_Weight

B.1.1 Design of Fly_Weight

The design for the Fly_Weight Xlet roughly follows that of the Fly_Weight MIDlet. The classes used for localization (Localisation, LanguageNotSupportedException and Properties) can be ported from the MIDlet versions. Also the generic SplashScreen and AboutScreen classes will be used, although most of the code will have to be rewritten, since the user interface API’s for TV applications are completely different than those for mobile phones. For displaying the forms on the TV, a container class is used to get a similar look for all the screens. The SplashScreen uses an interface SplashListener, which should be implemented by a class in the application using it (Fly_Weight_Xlet in the case of this application). There are two methods in this interface: one is called to load the rest of the application classes and data and the other is to signal that the user left the splash screen and that the rest of the application should be started. The loading by the splash screen is done so the rest of the application can run fast, without having to wait for data to load. The continue button on the splash screen is only made available when the load method has returned and thus all needed data for the application is loaded.

The design of the application logic is different from the MIDlet. Instead of a separate class for each thing that can be calculated, a single class is used with static methods to calculate the various values. These methods return an advice string that is displayed by the MenuScreen class when the corresponding menu item is selected. The InputScreen class displays the form and can show the alerts when the entered values are not in range. An overview of all the classes is shown in Figure 7-1. For clarity reasons the uses of the Localisation class are not displayed. This class is used by every class that uses text that can be shown on the TV screen.
B.1.2 Implementation of Fly_Weight

B.1.2.1 Overview

The actual implementation follows the design. An input field class and a radio group class had to be implemented, because the ones on the STB we have were broken. Also a class, HaviUtils, with some static methods was written for convenience. This class contains methods to create buttons and labels with default values for many parameters and a text wrap function, since the MHP class that should be able to do that didn't exist in our STB (the middleware installed it then was quite an early version).

A class diagram is shown in Figure 7-2, with details for each package in Figure 7-3, Figure 7-4 and Figure 7-5. As in the design diagram, uses of the Localisation class are not shown for clarity reasons. For the same reason the uses of the HaviUtils class are not shown.
Figure 7-2: Fly_Weight implementation classes
Figure 7-3: Details of package com.notthefly.mhp.Fly_Weight

Figure 7-4: Details of package com.notthefly.mhp.locale
B.1.2.2 Description of the General Purpose Classes

First, the following classes from the MIDP implementation were ported:

- com.notthefly.mobile.locale.*: localization support.

These classes were put into the package com.notthefly.mhp instead of com.notthefly.mobile.

Furthermore, the following additional classes were created:
- `com.notthefly.mhp.util.FlyContainer`: This extends `org.havi.ui.IContainer` (which is an extended `java.awt.IContainer`) and is used to give all screens the same look. This class paints a standard border around the container and displays a background image.

- `com.notthefly.mhp.util.FlyRadioGroup`: This class implements a radio group, since the HAVi alternatives in the current Philips SDK do not work. It supports a horizontal and vertical layout.

- `com.notthefly.mhp.util.FlyLineEdit`: This class implements an input field for text. This class was made because the current `HSingleLineEntry` in the Philips SDK is unusable (you cannot read out the contents). The FlyLineEdit can be set to numerical mode, in which only numbers can be entered and leading zeroes are stripped.

- `com.notthefly.mhp.util.HaviUtils`: This class contains some static methods to make the code simpler. It has functions to generate standard buttons and text labels and a (simple) text wrapping function.

In addition, an interface `com.notthefly.mhp.util.SplashListener` was created, which should be implemented by a class that wants to be notified of SplashScreen "events".

The SplashScreen class is almost a complete rewrite. It tries to display a complete splash screen as soon as possible and then starts to load the rest of the application, so the application can react quickly. It has the continue button disabled at the start and makes it available when the loading is complete. Loading is accomplished by calling a method `loadApplication()` in the SplashListener that was supplied when the SplashScreen was created. It loads the AboutScreen by itself and makes the "about" button available when this is done. When the user selects the "continue" button, the SplashListener is notified by calling `splashDone()`.

B.1.2.3 Description of the Fly_Weight Classes

The actual application is `com.notthefly.xlets.Fly_Weight.Fly_Weight_Xlet`. This class implements the SplashListener interface. It first creates a SplashScreen object and loads the rest of the application in its `loadApplication()` method. This is done by instantiating all other classes needed and calling their `init()` method to turn load inSplashScreen and files they need. After receiving notification that the SplashScreen is done, it starts the InputScreen class (by calling its `start()` method).

The InputScreen class displays an input form to gather the needed values (the gender, height and weight), using a FlyRadioGroup and two numerical FlyInputLines. When the ok button is pressed, the entered values are checked, and if any of them are too small or too large, an alert is displayed. This is done by putting replacing the FlyContainer that is contained in the HScene object that is displayed. When the ok button is pressed, the previous container is put back, displaying the input form again. When all values are ok, the next class, MenuScreen, is started.

The MenuScreen displays a menu, displaying the four available calculators. The items are presented as a radio group, with buttons for ok, back and exit. For this purpose, the items are marked with a colored circle. When an item is selected, the resulting advice is requested from the Calculator class and then displayed using the same technique as the alerts in the InputScreen.

All buttons are colored and can also be activated by pressing the corresponding colored key on the remote control.

B.2 Fly_IRForwarder

B.2.1 Introduction

This is a PC application that forwards TCP/IP traffic to the IR or RS232 port and vice versa. It was needed for the programs that wanted to use IR or Bluetooth when there was no access to the STB's serial port.
available. As a work-around, the traffic was sent through a PC using a TCP/IP connection (which runs over the PPP link the STB also uses to load Xlets and to send debug output).

B.2.2 Design
The program consists of two main threads: one that starts listening for incoming TCP/IP connections and one that listens for incoming IR or serial connections. Command-line options control which of these threads should be started and whether IR or RS232 should be used. When an incoming connection is received, it tries to connect to the other side (TCP/IP or IR/RS232). It then starts two separate threads, implemented in the inner class Forwarder, that do the actual forwarding. One thread reads data from the TCP socket and writes it to the infrared/serial socket and the other does it the other way around. No new connections are accepted as long as there is an existing connection.

B.2.3 Implementation
For the infrared communication the hp.infrared.* Java API is used. For this API currently only an implementation for Linux exists (using the Linux IrDA drivers). For socket connections the API uses TinyTP connections. OBEX also uses a TinyTP connection, so with an own OBEX implementation we can use this API to communicate using IrOBEX. However, in a lower layer the possibilities of a device are exposed to other devices using so-called hint bits. Whether a device supports OBEX is one of the things that can be expressed with these bits. When the OBEX bit is not turned on, some devices don’t even try to connect to an OBEX server, even though it is present. Therefore, the native C code of the API implementation had to be modified, which consists mainly of calls to Linux kernel functions. Only a single call at one place needed to be added to turn the OBEX hint bit on when registering a server. A patch file for the Java_IrSockets-v1.1 can be found in Appendix A.

The first version of the application read one byte at a time from the sockets, and wrote one byte at a time. This was not only terribly slow; it also didn’t work correctly when transferring a file from the Set-Top Box to a Palm V. It is unknown why it didn’t work correctly, but it probably was because of the slow speed, because the data sent and received by the IRForwarder program seemed just ok. When the code was modified to read and write multiple bytes at a time, everything worked fine (and fast).

B.3 OBEX Implementation

B.3.1 Introduction
This is an implementation of the OBEX 1.2 protocol, as specified by the Infrared Data Association (IrDA). This is the protocol designed for infrared file or object transfer. This implementation is needed because the intelligent IrDA and Bluetooth devices only provide OBEX transport, i.e. they pass through the OBEX data but do not interpret it.

Fortunately, the OBEX protocol is quite simple; it is some sort of compressed HTTP. The protocol is connection based. First a CONNECT packet is sent, to which an appropriate connection response packet is sent. During this phase the maximum allowed packet size is negotiated. Next data can be sent or received using PUT and GET commands. All commands can have a number of headers that describe the object, request or response. In particular, the actual object data sent using GET and PUT are also header values. Other header types are for example the name, type, size and description of the object and the number of packets for this object. For the headers, there are four different types of data supported: Unicode text, a byte sequence, a single byte and a four-byte quantity. The objects can be split over multiple packets (which will happen most of the time since packets are usually only a few kilobytes in size).

B.3.2 Design
The implementation uses several classes, as shown in Figure 7-6. The use of the MalformedOBEXException class is not shown in this diagram for clarity reasons.
These classes represent the following:

- **OBEXHeader**: Represents an OBEX header. Its public decodeHeaders method can be used to parse OBEX header data and returns a list of OBEXHeaders.
- **OBEXUnicodeHeader**: Represents an OBEX header containing a Unicode string.
- **OBEXByteSequenceHeader**: Represents an OBEX header containing a byte sequence.
- **OBEXByteHeader**: Represents an OBEX header containing a single byte.
- **OBEXIntHeader**: Represents an OBEX header containing a four-byte quantity.
- **OBEXCommand**: Represents an OBEX request (sent from client to server). This object contains a (possibly empty) list of OBEXHeader objects.
- **OBEXResponse**: Represents an OBEX response (sent from server to client). This object contains a (possibly empty) list of OBEXHeader objects.
- **OBEXApp**: A Java program that uses the OBEXServer and OBEXClient classes to send and/or receive files.
- **MalformedObexPacketException**: An exception that is thrown whenever the parsing of a packet fails.
- **OBEXServer**: This class represents the server side of an OBEX connection.
- **OBEXClient**: This class represents the client side of an OBEX connection.
- **OBEXServerListener**: This interface is used by the OBEXServer class. A class implementing this interface should supply input and output streams for sending and receiving objects. It is also notified by the OBEXServer of the start and end of connections, transmission progress and communication errors.
B.3.3 Implementation

B.3.3.1 Class diagram

Figure 7-7: Classes for com.notthefly.obex implementation

B.3.3.2 SomeSocket and SomeServerSocket

Because the implementation is used for different media, it uses an interface type for server sockets and the general (client) sockets used by server sockets. The interface is similar to that of java.net.Socket and java.net.ServerSocket, since at first only TCP/IP sockets (java.net.*) and IrDA sockets (hp.infrared.*) were used. Later serial RS232 communication was added (using javax.comm.* API). These connections can also be modeled as sockets. The RS232ServerSockets acceptConnection doesn’t block, but directly returns an RS232Socket for the port it is “listening” on. A better implementation could check the RS232 status lines, but in our case this was not needed. The intelligent Bluetooth device we have can be controlled using a modem AT-command set, and for this purpose a ModemServerSocket was implemented. This uses an arbitrary socket type, which it will open. It then waits for an incoming connection by waiting for the text ‘RING’ to appear. Then it sends back the answer command ‘ATA’ and waits for a ‘CONNECT’ response. It then returns this opened socket. For this to work more than once, the SomeSocket interface was extended with a reopen method, that opens the socket again with the same parameters as was specified in its constructor.

B.3.3.3 OBEXHeader and its subclasses

The OBEXHeader class is an abstract class, with methods to get and the raw data and the header type and methods to get a string representation of the header. The method that gives a string value of the value part is an abstract method. The class also contains a static method that parses a byte sequence and returns a vector of OBEXHeader objects that were represented by that byte sequence.
In the OBEX protocol there are four different value types defined for headers, which are represented by the four subclasses of OBEXHeader. These subclasses have in addition to the OBEXHeader class a (class specific) method to get the header value.

B.3.3.4 OBEXCommand and OBEXResponse
The OBEXCommand class has a constructor with multiple arguments to set all fields and a constructor with only a byte array as argument. The latter parses the supplied data, interpreting it as an OBEX command packet. It calls the static parseHeaders method in OBEXHeader to parse the headers. Further it contains the expected methods to get the various fields (the opcode, whether more packets follow, the headers, the maximum packet size (only for a connect command) and the raw bytes for the command) and a method to get a string representation.

The OBEXResponse class works in the same way as OBEXHeader class.

B.3.3.5 MalformedOBEXPacketException
This class is a subclass of java.lang.Exception. This exception is thrown when illegal values are encountered while parsing raw data.

B.3.3.6 OBEXClient
The OBEXClient class offers an abstraction for the client side of the protocol. It keeps track of the connection state and deals with splitting up large data blocks into smaller fragments so they fit in an OBEX packet. It is constructed using an InputStream and an OutputStream as parameters, which define the channel over which the OBEX protocol is used. The rest of the interface consists of connect, sendObject, receiveObject and disconnect methods.

The sendObject and receiveObject methods should be supplied respectively with an InputStream or OutputStream from/to which the object is read/written.

B.3.3.7 OBEXServer and OBEXServerListener
The OBEXServer class offers an abstraction for the server side of the protocol. It keeps track of the connection state and deals with splitting up large data blocks into smaller fragments so they fit in an OBEX packet. It needs a SomeServerSocket to listen for incoming connections.

The OBEXServer is implemented as a separate thread that handles the incoming connections. For GET and PUT requests, the server calls a method in the OBEXServerListener that was passed as argument in the constructor to get an Output- or InputStream to write the to or read from. The OBEXServerListener interface also contains a method that is called by the OBEXServer for notification purposes. This method is called when a device connects or disconnects, each time a GET or PUT data packet is received and at the end of a GET or PUT operation.

During server operation, IOExceptions and MalformedOBEXPacketExceptions can be thrown. These are also passed back to the OBEXServerListener.

B.3.3.8 OBEXApp
The OBEXApp class is a Java program using the OBEXServer and OBEXClient classes. Depending on the command line arguments it has a different behavior, as shown in Table 7-1.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Program behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>server</td>
<td>Starts a server that will save received objects to the current directory, using the Name-header for the filename.</td>
</tr>
<tr>
<td>client</td>
<td>Start a client and send a test message to a Palm KJava program.</td>
</tr>
<tr>
<td>client2</td>
<td>Start a client and send a test message to a Siemens MIDP Java program.</td>
</tr>
<tr>
<td>send filename</td>
<td>Start a client and send the file with name filename.</td>
</tr>
</tbody>
</table>

Table 7-1: OBEXApp command line arguments
B.4 Sending objects from the STB: Fly\_Download

B.4.1 Introduction
Fly\_Download is an application that sends a file from the STB to a mobile device using the OBEX protocol.

B.4.2 Design
The Fly\_Download program is very simple, since it will just show a screen with an option to start a download. For this download a `com.notthefly.obex.OBEXClient` object will be used. A sequence diagram for a download is shown in Figure 7-8: Sequence diagram for Fly\_Download.

![Sequence diagram for Fly\_Download](image)

**Figure 7-8: Sequence diagram for Fly\_Download**
B.4.3 Implementation

The implementation consists of two classes, the main class `Fly_Download_Xlet`, which contains most of the code, and a class `ProgressBar`, which is a progress bar object used by `Fly_Download_Xlet` to show the download progress. The name of the file that should be sent is read from a configuration file. The program also makes use of the `HaviUtils` and `FlyContainer` classes from the `com.notthefly.mhp.util` package.

B.5 Receiving objects on the STB: `Fly_Image`

B.5.1 Introduction

`Fly_Image` is an application that can receive images via OBEX and display them on the TV screen.

B.5.2 Design

This is a very simple application, since most of the functionality is in the `com.notthefly.obex.OBEXServer` class. Such a server will be started, and then the application will simply wait for incoming files. If it can be successfully decoded as an image (by using the `createImage` method from `javax.awt.Toolkit`), it will be shown.

B.5.3 Implementation

The implementation simply follows the design. There are two classes, one is the main Xlet that shows a splash screen and starts the OBEX server. The other class implements the `OBEXServerListener` and handles all callbacks from the server. GET-requests are rejected and for PUT-requests a `ByteArrayOutputStream` is supplied. When a PUT transfer finishes, the byte array from the `ByteArrayOutputStream` is passed to `java.awt.Toolkit.createImage`. If successful, this image is displayed centered on the entire screen.

B.6 Remote Procedure Call: `Fly_RemoteScreen`

B.6.1 Introduction

`Fly_RemoteScreen` is a client-server application to give small devices the possibility to use the TV screen as an output device. A problem is that according to the specification J2ME MIDP applications cannot use Bluetooth or IrDA. There exists a vendor-specific API however.

B.6.2 Design

B.6.2.1 Remote Procedure Calls

For displaying something from a mobile device on the TV screen, some data has to be sent from the mobile device to the STB, for which the OBEX protocol must be used, since it is the only available protocol for Java applications on a mobile phone (only Siemens phones have IrDA support from Java at the moment of writing). Some ways to do the communication were investigated.

First, the possibility to just extend a few classes used for rendering, and write stubs for them on the client side that signals a class on the server side was investigated. This server-side class then calls this method in the real class on the server. However, in the AWT rendering is done by so-called implementing classes, which cannot be accessed without writing an almost complete AWT implementation. The `javax.awt.Graphics` class is not used for the rendering of components like labels and buttons. Besides, the `javax.awt.Graphics` objects that are used for rendering are created by the implementation.

The next method investigated was Java's support for remote procedure calls, or Remote Method Invocation (RMI) as it is called by Java. The RMI classes are an optional part in Personal Java, which is the
Java that runs on the STB. However, unfortunately there is no RMI support in CLDC/MIDP Java, which runs on the mobile devices.

The final solution was to write some RMI-like protocol, using a socket connection. A set of commands was defined to:

- Initialize, i.e. create a screen with specified size to use
- Create new objects
- Invoke a method in an object
- Invoke a static method in a class
- Cleanup and terminate the connection.

The newly created objects and the results of the method calls can be stored in variables, which are stored in a Hashtable. These variables can be used in method invocations, as can constant strings, integers and booleans.

For the object creation and method invocations the java.lang.Class and java.lang.reflect.* classes are used.

The drawback of this method is that all required classes should be available at the STB side by forehand. In addition, things like images cannot be sent over the link, but support for that can be built if needed.

**B.6.2.2 Protocol**

The protocol uses java.io.DataInputstream and java.io.DataOutputstream to send integers, strings and booleans. This communication is synchronous. The server side continually waits for commands. When it receives a command it is processed and the result (ok or error) is sent back to the client. There are no time constraints; the server just sends the response as soon as it has finished the clients request.

The format of a client request is as follows. The first value is an integer specifying the command:

- 0. initialize
- 1. new object
- 2. method
- 3. static method
- 4. quit

Next follow the parameters for the command; these vary depending on the command:

- 0. <int> x, <int> y, <int> w, <int> h
  - x: desired x position of the screen
  - y: desired y position of the screen
  - w: desired width of the screen
  - h: desired height of the screen

- 1. <string> var, <string> classname, <int> numargs, numargs times: [<string> "var", <string> type, <string> name or <string> "java.lang.String", <string> value or <string> "int", <string> value or <string> "boolean", <int> value]
  - var: the variable to store the new object in
  - classname: the class name of the class to instantiate
  - numargs: the number of arguments that follow
  - type: the type of the variable, as expected by the method
  - name: the name of the variable to use as argument
  - value: the value for a constant expression: a string, an int or boolean.
booleans are indicated with an int: 0 for false, 1 for true.

2. `<string> var, <string> methodname, <int> numargs, `numargs` times: [ <string> "var", <string> type, <string> name or <string> "java.lang.String", <string> value or <string> "int", <int> value or <string> "boolean", <int> value], <string> resultvar
   var: variable of the object of which to invoke the method
   methodname: the name of the method to invoke
   resultvar: the variable to store the return value of the method, or "void" to
   discard the value.
   for the other parameters, see 1.

3. `<string> classname, <string> methodname, <int> numargs, `numargs` times: [ <string> "var", <string> type, <string> name or <string> "java.lang.String", <string> value or <string> "int", <int> value or <string> "boolean", <int> value], <string> resultvar
   classname: the class of which a method should be called
   for the other parameters, see 1.

4. (no parameters)

The response sent back by the server consists of one or more strings. Possible responses are:
"ok": Everything went ok.
"error": Something went wrong. A second string follows with an error message.

B.6.3 Implementation

The class diagram for Fly_RemoteScreen is shown in Figure 7-9.

![Class diagram for Fly_RemoteScreen](image)

Figure 7-9: Class diagram for Fly_RemoteScreen

The server side is formed by the Fly_RemoteScreen_Xlet, Server and Screen classes. The
Fly_RemoteScreen_Xlet creates a Server class. This server class takes care of the communication, for
which it uses an OBEXServer class. The handling of the commands and the drawing on the screen is
handled by the Screen class.
The client side consists of a RemoteScreenClient class, which is a MIDlet or Palm Java application. This application can use the methods provided by the Client class to do some remote drawing. When an error occurs (the server response was not "ok"), a RemoteScreenException is thrown. The client class contains example code to draw a rectangle and some text in different colors, which is used by the current RemoteScreenClient classes.

For now, the application doesn’t work correctly, because of problems with the infrared connection of the Siemens phone used. It works when the transmission is done in one direction, i.e. no server responses are sent back (all commands are assumed to have no errors by the client). The Palm client also doesn’t work; it crashes for some unknown reason.

B.7 Accessing the internet: Fly_GPRS

B.7.1 Introduction

This is a simple application to demonstrate the use of a return channel via GPRS. It will continuously fetch a (web cam) image from a web server and display it.

B.7.2 Design

The design is very simple since the application can just use the MHP return channel API and java.net package. First the connection should be set up using the return channel API, dialing a special number for GPRS access. Next, a loop fetches and displays the images using the java.net API.

B.7.3 Implementation

Implementation is done using one class, following the above design.

B.8 Sending SMS messages: Fly_SMS

B.8.1 Introduction

This is a simple application to demonstrate the ability to send SMS messages. It will show a form in which a mobile phone number and a message can be entered that can then be sent by SMS.

B.8.2 Design

The design is simple since the application just has to send a few commands via the serial port of the STB to the phone’s modem (connected transparently via an intelligent IrDA or Bluetooth device), where some of the data comes from the form. For this form the com.notthefly.mhp.util package is used (see the design of Fly_Weight).

B.8.3 Implementation

Implementation is done using one class, following the above design. It uses a send method to send a command to the modem and a waitFor method to wait for a specific response from the modem.
Appendix C: Source code

C.1 com.notethefly.mhp.Fly_Download

C.1.1 Fly_Download_Xlet.java

```java
package com.notethefly.mhp.Fly_Download;

import java.tv.xlet.*;
import java.io.*;
import java.awt.*;
import java.awt.event.KeyEvent;
import com.notethefly.mhp.util.*;
import com.notethefly.mhp.locale.*;
import org.havi.ui.*;
import org.havi.ui.event.*;
import org.apache.xml.DVBUIColor;
import org.apache.xml.event.*;
import com.notethefly.mhp.xlet.
import com.pupilphilmappserial.SerialSerialConnection;

/**
 * Description: Fly_Download Xlet -- main class
 * @author Tom Geelen
 * @version 5.2: Fly_Download_Xlet.java v 1.4 2002/03/28 10:34:49 Tom Geelen
 * @copyright: NotTheFly Component Solutions BV
 */
public class Fly_Download_Xlet implements Xlet, Runnable, OBEXEventListener {
    final static int PORT = 5913;
    final static int PACKER_SIZE = 128;
    final static String BT_ADDRESS = "0002E1C4A07";
    final static int BT_CHANNEL = 9;
    String filename;

    /** The Xlet context */
    XletContext xletContext = null;
    /** Is the application already started? */
    boolean isStarted = false;

    /** The scene on which all screens are displayed */
    XScene scene = null;
    static Localization locale = Localization.getLocalization();
    static FlyContainer container;
    static Image background = null;
    static Image background2 = null;
    static Image logo = null;
    static ProgressBar progressBar = null;
    static boolean destroyed = false;
    static boolean isConnected = false;
    // static Socket sock = null;
    static SerialConnection sock = null;
    Thread thread;
    static FileInputStream fis;
    static int fileLength;
    UserEventListener listener = null;

    /** 
     * Initializes the Xlet. It creates a new XScene and creates a SplashScreen.
     
     * @public void init(JXletContext context) throws XletStateChangeException 
     * @xletcontxt = context
     */
    // Load filename to send from a file.
    try {
        fis = new FileInputStream("Fly_Download\download");
        System.out.println("File not found: Fly_Download\download");
        exitXlet();
        return;
    }
    catch (FileNotFoundException e) {
        System.out.println("File not found: Fly_Download\download");
        exitXlet();
        return;
    }
    try {
        if (filename = new BufferedReader(new InputStreamReader(fis))).readLine();
        System.out.println("Error reading Fly_Download\download: " + e.getMessage());
        exitXlet();
    }
    catch (IOException e) {
        System.out.println("Error reading Fly_Download\download: " + e.getMessage());
        exitXlet();
    }
    try {
        fis.close();
    }
    catch (IOException e) {
        System.out.println("Error reading Fly_Download\download: " + e.getMessage());
        exitXlet();
    }
    try {
        fis = new FileInputStream(filename);
    }
    ```

Error! No text of specified style in document.
Error! No text of specified style in document.
Error! No text of specified style in document.
Version Error! No text of specified style in document.

© Copyright NotTheFly Component Solutions 9-12-2002 Page 52 of 128
 */
public void pauseXlet() {  
    System.out.println("Fly_Download: pauseXlet()");
    scene.setVisible(false);
}

/**
 * Starts the Xlet. If paused, just unpause, else show message.
 */
public void startXlet() {  
    System.out.println("Fly_Download: startXlet()");
    if (!isStarted) {  
        GridBaglayout gridBag = new GridBagLayout();
        GridBagConstraints constraints = new GridBagConstraints();
        HStaticText label;
        MediaTracker tracker = new MediaTracker(scene);
        background = Toolkit.getDefaultToolkit().getImage("Fly_Download/background-400x360.png");
        background2 = Toolkit.getDefaultToolkit().getImage("Fly_Download/background-520x48.png");
        logo = Toolkit.getDefaultToolkit().getImage("Fly_Download/logo.png");
        tracker.addImage(background, 1);
        tracker.addImage(background2, 2);
        tracker.addImage(logo, 1);
        try {  
            tracker.waitForAll();  
        } catch (InterruptedException e) {  
            container = new FlyContainer(MESSAGE_WIDTH, MESSAGE_HEIGHT, background);
            container.setLayout(new FlowLayout());
            subContainer = new HContainer();
            subContainer.setLayout(new FlowLayout());
            subContainer.add(new JLabel(Rep.localizationMessage("message.quit", Color.Yellow, 20)));
            gridBag.setConstraints(subContainer, constraints);
            gridBag.add(subContainer);
            subContainer = new HContainer();
            subContainer.setLayout(new FlowLayout());
            subContainer.add(new JLabel(Rep.localizationMessage("message.quit", Color.Yellow, 20)));
            gridBag.setConstraints(subContainer, constraints);
            gridBag.add(subContainer);
        }
        scene.setVisible(true);
    }
    else {  
        scene.repaint();
        System.out.println("Fly_Download: resumeXlet()");
    }
    isStarted = true;
    }
case HRcEvent.VK_COLORED_KEY_1:
    case HRcEvent.VK_F2:
        EntityManager.getInstance().removeEventListener(userEventListener);
        scene.setScale(100, 500, STATUS_WIDTH, STATUS_HEIGHT);
        background = null;
        respawn(true);
        thread.start();
        break;
    }
}
EntityManager.getInstance().addUserEventListener(userEventListener, keyEvents);
}
scene.setVisible(true);
}

public void run() {
    byte[] buffer;
    try {
        sock = new Socket("10.1.1.110", PORT);
        sock = new Socket("10.1.1.110", PORT);
        in = new InputStreamReader(sock.getInputStream());
        out = sock.getOutputStream();
    } catch (IOException e) {
        System.out.println("Setting up communication failed!");
        e.printStackTrace();
        return;
    }
    try {
        sock.setSoTimeout(0); // No timeout
    } catch (SocketException e) {
    }
    try {
        // Start BT connection.
        send("AT+br=0", "BT_CHANNEL = \\
        waitfor("OK");
        send("ATD" + BT_ADDRESS + \\
        waitfor("CONNECT" + \\
        BConnected = true;
        // Start OBEX client
        OBEXclient objex = new OBEXClient(in, out);
        objex.connect();
        scene.remove(container);
        container = new FlyContainer(STATUS_WIDTH, STATUS_HEIGHT, background2);
        container.setScale(0, 0, STATUS_WIDTH, STATUS_HEIGHT);
        container.setLayout(new FlowLayout());
        container.add(new Label("filename + " + filelength + " bytes": ",
                Color.yellow, 20));
        progressBar = new ProgressBar();
        container.add(progressBar);
        scene.add(container);
        container.validate();
        scene.validate();
        container.repaint();
        scene.repaint();
        objex.sendObject(fis, filename, filelength, this);
        objex.disconnect();
        message = locale.getMessage("status.success");
        progressBar = null;
    } catch (IOException e) {
        if (!destroyed)
            message = locale.getMessage("status.ioerror");
            System.out.println("---1/0 Error---");
        e.printStackTrace();
        return;
    } catch (MalformedURLException e) {
        message = locale.getMessage("status.protocolerror");
        System.out.println("---OBEX protocol error---");
        e.printStackTrace();
        return;
    } finally {
        if (sock != null) {
            try {
                sock.close();
            } catch (IOException e) {
            }
        } sock = null;
    }
    synchronized(this) {
        try {
            wait(3000);
        } catch (InterruptedException e) {
        }
    }
    if (!destroyed) {
        exitList();
    }

    /** Helper function that displays a new message. * 
    * @param msg The message to display.*
    */
    private void message(String msg) {
        scene.remove(container);
        container = new FlyContainer(STATUS_WIDTH, STATUS_HEIGHT, background2);
        container.setScale(0, 0, STATUS_WIDTH, STATUS_HEIGHT);
        container.setLayout(new FlowLayout());
        container.add(new JLabel("filename + " + filelength + " bytes": ",
                Color.yellow, 20));
    }
}

Version Error! No text of specified style in document.
container.repaint();
scene.add(container);
scene.validate();
scene.validate()
container.repaint();
scene.repaint();
/**
 * Writes a string to a stream.
 * @param out The stream to write to.
 * @param str The string to write.
 * @throws IOException when an I/O error occurs.
 */
private void send(String str) throws IOException {
    System.out.println("Sending \"\" + str + \"\"\n");
    out.write(str.getBytes());
    out.flush();
}
/**
 * Waits for a specific string to appear on a stream.
 * @param str The string to look for.
 * @throws IOException when an I/O error occurs.
 */
private void waitFor(String str) throws IOException {
    System.out.println("Waiting for \"\" + str + \"\"\n");
    byte[] text = str.getBytes();
    int pos = 0;
    while (pos &lt; text.length) {
        b = in.read();
        if (b == text[pos]) {
            pos++;
        } else if (b == text[0]) {
            pos = 1;
        } else {
            pos = 0;
        }
    }
    System.out.println("Just saw \"\" + str + \"\"");
}
/**
 * Callback from com.notthefly.obex.Client. This callback updates
 * the progress bar.
 * @param bytesRead The number of bytes sent so far.
 */
public void clientProgress(int bytesRead) {
    progressBar.setProgress((double) bytesRead / fileLength * 100);
}
/**
 * Destroys the Xlet. Disposes all unused resources,
 * Xlet is always destroyed.
 * @throws IOException Never thrown, Xlet is always destroyed.
 */
public void destroyXlet(boolean unconditional) throws IOException {
    System.out.println("Fly_DoWn: destroyXlet()!");
    if (userEventListener != null) {
        EventManager.getInstance().removeUserEventListener(userEventListener);
    }
    if (scene != null) {
        scene.setVisible(false);
        scene.remove(container);
        container = null;
        RSceneFactory.getInstance().dispose(scene);
    }
    if (fis != null) {
        try {
            fis.close();
        } catch (IOException e) {
        }
    }
    if (background != null) {
        background.flush();
    }
    if (background2 != null) {
        background2.flush();
    }
    if (logo != null) {
        logo.flush();
    }
    if (sock != null) {
        if (!isVisible) {
            try {
                Thread.sleep(1100);
                send("++*
    Thread.sleep(1100);
                waitFor("OK\n    send("ATN\n    waitFor("OK\n    in.close();
    out.close();
    } catch (IOException e) {
        } catch (InterruptedException e) {
    }
    try {
        sock.close();
    } catch (IOException e) {
        sock = null;
    }
    }
    if (thread != null) {
        thread.interrupt();
        try {
            thread.join();
        } catch (InterruptedException e) {
    }
    }
}
C.1.2 ProgressBar.java

```java
class ProgressBar extends Component {
    double percent = 0.0;
    public ProgressBar() {
        setBounds(0, 0, 202, 26);
    }
    public void setPercentage(double p) {
        repaint();
        percent = p;
    }
    public Dimension getPreferredSize() {
        return new Dimension(202, 26);
    }
    public void paint(Graphics g) {
        Color green = new Color(0, 127, 0);
        g.setColor(Color.yellow);
        g.drawArc(0, 0, 202, 26);
        g.setColor(green);
        g.fillRect(1, 1, (int) (2 * percent), 24);
        g.setColor(Color.white);
        g.setFont(newFont("arial", Font.BOLD, 22));
        g.drawString(((int) percent) + "," + (((int) percent * 10) % 10) + " ", 72, 23);
    }
}
```

C.2 com.notthefly.mhp.Fly_GPRS

C.2.1 Fly_GPRS_Xlet.java

```java
import java.tv.xlet.*;
import java.io.*;
import java.net.*;
import java.awt.*;
import java.awt.event.KeyEvent;
import com.notthefly.mhp.util.*;
import org.havi.ui.*;
import org.dvb.net.*;
import org.dvb.resources.*;
import com.notthefly.obex.RealInputOutputStream;
import com.philips.mhp.xml.serialization;

/**
 * description: Fly的形象 -- main class
 * author: Tom Oliver
 * version: 2.0: Fly_GPRS.Xlet.java v 1.1 2000/11/20 10:30:18 tong Exp $
 * copyright: NotTheFly Component Solutions BV
 */
public class Fly_GPRS_Xlet implements Xlet, Runnable, ResourceClient, ConnectionListener {
    final static String BR_ADDRESS = '0002B1C4A07';
    final static int BT_CHANNEL = 1;
    /** The Xlet context */
    XletContext xletContext = null;
    /** Is the application already started? */
    boolean isStarted = false;
    static boolean BTconnected = false;
    /** The scene on which all screens are displayed */
    static HScene scene = null;
    static FlyContainer container;
    Thread thread;
    byte[] imageData;
    Image image = null;
    HSStaticAnimation anim = null;
    /** Width of the Xlet */
    static final int W = 300;
    /** Height of the Xlet */
    static final int H = 200;
    static ConnectionBCInterfacecroft = null;
    /**
     * Initializes the Xlet. It creates a new HScene and the main thread.
     */
```
public void initLet(XletContext context) throws XletStateException { 
XletContext = context;
HSceneTemplate template = new HSceneTemplate();
template.setPreference(HSceneTemplate.SCENE_PIXEL_DIMENSION, new Dimension(W, H),
    HSceneTemplate.PREFERRED);
    template.setPreference(HSceneTemplate.SCENE_PIXEL_LOCATION, new Point(175, 200),
    HSceneTemplate.PREFERRED);
    scene = HSceneFactory.getScene(this,template);
if (scene == null) System.out.println("Fly.SPRS: Oops... I didn't get a HScene...");
    scene.setBounds(175, 200, W, H);
    // workaround, scenefactory doesn't seem to work okay...
    scene.setLayout(null);
    container = new FlyContainer(W, H, 
    Toolkit.getDefaultToolkit().getImage("background-300x200.png"));
    container.setBounds(0, 0, W, H);
    container.setLayout(new FlowLayout());
    container.add(HButils.makeLabel("\"NotTheFly\"\" NFRS Return channel in Test App", 
    Color.yellow, 20));
    container.validate();
    thread = new Thread(this);
}
/**
 * Pauses the Xlet.
 */
public void pauseXlet() { 
    System.out.println("Fly.Image: pauseXlet()";
    scene.setVisible(false);
}
/**
 * Start the Xlet. If paused, just unpaese, else start the program thread.
 */
public void startXlet() { 
    System.out.println("Fly.Image: startXlet()";
if (scene != null) scene.setVisible(true);
    if (!isStarted) 
    isStarted = true;
            thread.start();
}
/* Main thread. Displays splash for 3 sec. and starts the server. */
public void run() { 
    //show splash
    scene.add(container);
    scene.validate();
    scene.repaint();
    //wait 3 sec and remove the splash
    synchronized(this) { 
    try {
        wait(3000); 
    } catch (InterruptedException e) { }
    }
    scene.remove(container);
    scene.repaint();
    scene.setVisible(false);
    try {
    /*
     //Start BT connection
    SerialConnection conB = new SerialConnection("RS232");
    InputStream inB = conB.getInputStream();
    OutputStream outB = conB.getOutputStream();
    send(outB, AT *"rscle" * "BT\ CHANNEL" + "\n");
    waitFor(inB, "OK");
    send(outB, AT + BT_ADDRESS + "\n");
    waitFor(inB, CONNECT + "\n");
    BTConnected = true;
    inB.close();
    outB.close();
    conB.close();
    System.out.println("Bluetooth connection initialized");
    */
    RCIConnectionManager mgr = RCIConnectionManager.getInstance();
    RCIConnection rcifs[] = mgr.getConnections();
    for (int i = 0; i < rcifs.length; i++) {
    if (rcifs[i].instanceof(RCInterfaceConnection) { 
    rcif = (RCInterfaceConnection) rcifs[i];
    break;
    }
    if (rcif == null) { 
    System.out.println("No Connection RC if's found...");
    } else { 
    rcif.addConnectionListener(this);
    rcif.reserve(this);null);
    try 
    rcif.setTarget(new ConnectionParameters("\"999", \"libertel\", \"libertel", 
    InetAddress.getByName("131.155.2.3")));
    catch (IncompleteTargetException e) { 
    e.printStackTrace();
    }
    System.out.println("Connecting...");
    rcif.connect();
    /*
     System.out.println("Connecting... (going to sleep for 1 secs)";
     synchronized (this) { 
    try 
    wait(1000); 
    } catch (InterruptedException e) { 
    }
    */
    while (true) {
Page 57 of 128
System.out.println("Retrieving image...");
URLConnection con = new URL("http://ottoroom.tv/webcam/webcam2.jpg").openConnection();
con.connect();
//OutputSteam out2 = con.getOutputStream();
InputStream in = con.getInputStream();
int length = con.getContentLength();
ImageData = new byte[length];
for (int i = 0; i < length; i++) {
    System.out.println("received " + i + " of " + length + " bytes");
}
System.out.println("Received done.");
in.close();
showImage();
/*
 * Socket sock = new Socket("www.notthefly.com", 80);
 * Printwriter out = new Printwriter(sock.getOutputStream);
 * BufferedReader in = new BufferedReader(new InputStreamReader(sock.getInputStream()));
 * out.println("GET / HTTP/1.0
in");
 * out.flush();
 * String line = in.readLine();
 * while (line != null) {
 *     System.out.println(line);
 *     line = in.readLine();
 * }*/
System.out.println("Closing socket connection.");
in.close();
out.close();
sock.close();
System.out.println("Closing internet connection.");
}
crc5.disconnect();
//
//*/
catch (IOException e) {
    System.out.println("IO Exception ": e.printStackTrace());
}
catch (FileNotFoundException e) {
    System.out.println("Stupid STB doesn't give me permission ": e.printStackTrace());
}
scene.setVisible(false);
}
/**
 * Writes a string to a stream.
 * @param out The stream to write to.
 * @param str The string to write.
 * @throws IOException When an I/O error occurs.
 */
private void send(OutputStream out, String str) throws IOException {
    System.out.println("Sending ": str);
    out.write(str.getBytes());
    out.flush();
}
/**
 * Waits for a specific string to appear on a stream.
 * @param in The stream to read.
 * @param str The string to look for.
 * @throws IOException When an I/O error occurs.
 */
private void waitForInputStreamIn(String stream, String str) throws IOException {
    System.out.println("Waiting for ": str);
    byte[] text = str.getBytes();
    int pos = 0;
    while (pos < text.length) {
        if (b == text[pos]) {
            pos++;
        } else if (b == text[0]) {
            pos = 1;
        } else {
            pos = 0;
        }
    }
    System.out.println("Just saw ": str);
}
/**
 * Displays the image in <code>imageData</code> centered on the screen.
 */
public void showImage() {
    if (image == null) {
        System.out.println("Cleaning up previous image");
        SceneManager.removeAnim();
        image.flush();
    }
    ImageMethod = Toolkit.getDefaultToolkit().createImage(imageData);
    anim = new StaticAnimation(ImageMethod, 10, 0, 0);
    MediaTracker tracker = new MediaTracker(anim);
    tracker.addImage(image, 0);
    try {
        tracker.waitForAll();
    } catch (InterruptedException e) {
    }
    if (tracker.isErrorAny()) {
        System.out.println("Error loading image");
        image.flush();
        image = null;
    } else {
        int width = image.getWidth(null);
        int height = image.getHeight(null);
        System.out.println("Image loaded. Dimensions: "+ width + " x " + height);
        scene.setLayout(FixedLayout(920 - width / 2, (755 - height) / 2, width, height));
        scene.setVisible(true);
        validDate();
    }
```java
scene.setVisible(true);
scene.repaint();
}
/**
 * Destroys the Xlet. Disposes all used resources.
 * NB: If no unconditional UncaughtExceptionHandler then this exception may be ignored.
 * @return XletStateChangeException Never thrown, Xlet is always destroyed.
 */
public void destroyXlet(boolean unconditional) throws XletStateChangeException {
    if (scene == null) {
        scene.setVisible(false);
        SceneFactory.getInstance().dispose(scene);
    }
    if (ocrif != null) {
        try {
            ocrif.disconnect();
        } catch (PermissionDeniedException e) {
            ocrif.release();
        }
    }
    if (PFconnected) {
        try {
            SerialConnection con = new SerialConnection("RS232");
            InputStream in = con.getInputStream();
            OutputStream out = con.getOutputStream();
            Thread.sleep(1100);
            send(out, "+++*");
            Thread.sleep(1100);
            waitFor(in, "OK");
            send(out, "AT\r\n");
            waitFor(in, "OK");
            in.close();
            out.close();
        } catch (IOException e) {
            catch (InterruptedException e) {
        }
    }
}
/**
 * quits the xlet */
public void exitXlet() {
    try {
        destroyXlet(true);
        } catch (XletStateChangeException e) {
            } throw e;
    }
    } catch (XletStateChangeException e) {
        xletContext.notifyDestroyed();
    }
    //====================================================================
    // ResourceClient Interface
    //====================================================================

    public boolean requestRelease(ResourceProxy proxy, Object requestData) {
        System.out.println("Fly_GPRS_Xlet.requestRelease called");
        return true;
    }
    public void release(ResourceProxy proxy) {
        System.out.println("Fly_GPRS_Xlet.release called");
    }
    public void notifyRelease(ResourceProxy proxy) {
        System.out.println("Fly_GPRS_Xlet.notifyRelease called");
    }
    //====================================================================
    // ConnectionListener interface
    //====================================================================

    public void connectionChanged(ConnectionChangeEvent e) {
        System.out.println("connectionChanged called, Event: "+ e);
        if (e instanceof ConnectionFailedEvent) {
            exitXlet();
        }
    }
}

C.3 com.notethefly.mhp.Fly_Image

C.3.1 Fly_Image_Xlet.java

package com.notethefly.mhp.Fly_Image;

import javax.tv.xlet.*;
import java.io.*;
//import java.net.*;
import java.awt.*;
import java.awt.event.KeyEvent;
import com.notethefly.mhp.util.*;
import org.havu.ui.*;
import com.notethefly.obex.*;
/**
 * description: Fly_Image Xlet -- main class
 * @author: Tom Geelen
 * @version: 0.3; Fly_Image_Xlet.java v 1.3 2002/08/08 12:53:02 tong Exp $
 * copyright: NotTheFly Component Solutions BV
 * public class Fly_Image_Xlet implements Xlet, Runnable {
 * final static int PORT = 913;
 */
```
/* The Xlet context */
XletContext xletContext = null;
/* Is the application already started? */
boolean isStarted = false;
/* The scene on which all screens are displayed */
static HScene scene = null;
static FlyContainer container;
Thread thread;
/* Width of the Xlet */
static final int W = 300;
/* Height of the Xlet */
static final int H = 200;
OBEXServer server = null;

/* Initializes the Xlet. It creates a new HScene and the main thread. */
public void initXlet(XletContext context) throws XletStateChangeException {
    xletContext = context;
    HSceneTemplate template = new HSceneTemplate();
    template.setPreference(HSceneTemplate.SCENE_PIXEL_DIMENSION, new Dimension(W, H),
                           HSceneTemplate.PREFERRED);
    template.setPreference(HSceneTemplate.SCENE_PIXEL_LOCATION, new Point(175, 200),
                           HSceneTemplate.PREFERRED);
    scene = HSceneFactory.getInstance().getBestScene(template);
    if (scene == null) System.out.println("FlyImage: Oops... I didn't get a HScene...");
    scene.setLayout(null);
    container = new FlyContainer(W, H, Toolkit.getDefaultToolkit().getImage("background-300x200.png");
    container.setBounds(0, 0, W, H);
    container.setLayout(new FlowLayout());
    container.add(NavitUtil.makeLabel("inNotTheFly\'s\nimage display\nfor mobile devices", Color.yellow, 20));
    container.validate();
    thread = new Thread(this);
}

/* Pauses the Xlet. */
public void pauseXlet() {
    System.out.println("FlyImage: pauseXlet()");
    scene.setVisible(false);
}

/* Starts the Xlet. If paused, just unpause, else start the program thread. */
public void startXlet() {
    System.out.println("FlyImage: startXlet()");
    if (scene != null) scene.setVisible(true);
    if (!isStarted) {
        isStarted = true;
        thread.start();
    }
}

/* Main thread. Displays splash for 3 sec. and starts the server. */
public void run() {
    // Show splash
    scene.setVisible(true);
    scene.repaint();
    // Start the server.
    try {
        // server = new com.notthefly.obex.OBEXServer(new TCPServerSocket(PORT),
        // new OBEXListener(scene));
        // server = new com.notthefly.obex.OBEXServer(new ModemServerSocket() {something});
        // server = new com.notthefly.obex.OBEXServer(srvSock, new OBEXListener(scene));
        // catch (IOException e) {
        //     System.out.println("Server creation failed: \" + e.getMessage());
        // }
        // wait 3 sec and remove the splash
        synchronized(this) {
            try {
                wait(3000);
            } catch (InterruptedException e) {}
        }
        scene.remove(container);
        scene.setVisible(false);
    } catch (Exception e) {
        e.printStackTrace();
        // Destroy the Xlet. Disposes all used resources.
        // @param unconditional Ignored. Xlet is always destroyed.
        // @throws XletStateChangeException Never thrown. Xlet is always destroyed.
        public void destroyXlet(boolean unconditional) throws XletStateChangeException {
            if (scene != null) {
                scene.setVisible(false);
                HSceneFactory.getInstance().dispose(scene);
            }
            if (server != null) {
                server.kill();
                server = null;
            }
        }
    }
}
/* quits the xlet */
public void exitXlet() {

Error! No text of specified style in document.
Error! No text of specified style in document.
Error! No text of specified style in document.

Version Error! No text of specified style in document.

© Copyright NotTheFly Component Solutions 9-12-2002
C.3.2 OBEXListener.java

```java
package com.notthefly.mhp.fly_image;
import com.notthefly.obex.*;
import java.io.*;
import java.awt.*;
import org.havli.ui.*;
/**
 * Description: Fly_Image Xlet -- main class
 * Author: Tom Geelen
 * Copyright: NotTheFly Component Solutions BV
 */
public class OBEXListener implements OBEXServerListener {
    private static final byte END_IN = 0x0;
    private static final byte END_OUT = 0x1;
    private static final byte END_END = 0x2;
    private static final byte END_OOB = 0x3;
    private static final byte END_OOOB = END_OOB | 0x80;
    private static final byte END_ENDO = END_END | 0x80;
    private static final byte END_OOOOB = END_OOB | 0x80 | 0x80;
    private static final byte END_OOOOOB = END_OOB | 0x80 | 0x80 | 0x80;
    private static final byte END_OOOOOOOB = END_OOB | 0x80 | 0x80 | 0x80 | 0x80;
    private static final byte END_OOOOOOOOOB = END_OOB | 0x80 | 0x80 | 0x80 | 0x80 | 0x80;
    private static final byte END_OOOOOOOOOOOB = END_OOB | 0x80 | 0x80 | 0x80 | 0x80 | 0x80 | 0x80;
    private static final byte END_OOOOOOOOOOOOOB = END_OOB | 0x80 | 0x80 | 0x80 | 0x80 | 0x80 | 0x80 | 0x80;
    public void serverConnected(int type, int data) {
        System.out.println("Connecting...");
    }
    public void serverDisconnected(int type, int data) {
        System.out.println("Disconnected...");
    }
    public void serverMessage(ByteArray message, int type, int data) {
        System.out.println("Message received...");
    }
    public void serverException(Exception e) {
        e.printStackTrace();
    }
    public void serverError(Exception e) {
        e.printStackTrace();
    }
    public void serverChannelOpenFailed(Exception e) {
        e.printStackTrace();
    }
    public void inputInputStream(InputStream in) {
        byte[] buffer = new byte[1024];
        int bytesRead = in.read(buffer);
        if (bytesRead >= 0) {
            String line = new String(buffer, 0, bytesRead);
            System.out.println(line);
        }
    }
    public void inputOutputStream(OutputStream out) {
        try {
            out.write(buffer, 0, bytesRead);
            out.flush();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    public void outputInputStream(InputStream in) {
        try {
            byte[] buffer = new byte[1024];
            int bytesRead = in.read(buffer);
            if (bytesRead >= 0) {
                String line = new String(buffer, 0, bytesRead);
                System.out.println(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    public void outputOutputStream(OutputStream out) {
        try {
            byte[] buffer = new byte[1024];
            int bytesWritten = out.write(buffer);
            if (bytesWritten >= 0) {
                String line = new String(buffer, 0, bytesWritten);
                System.out.println(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    public void serverRequestClientReady() {
        System.out.println("Server ready...");
    }
    public void serverRequestSendingReady() {
        System.out.println("Server sending ready...");
    }
    public void serverRequestEnding() {
        System.out.println("Server ending...");
    }
    public void serverRequestEndingFailed(Exception e) {
        e.printStackTrace();
    }
    public void serverRequestComplete() {
        System.out.println("Server complete...");
    }
    public void serverRequestError(Exception e) {
        e.printStackTrace();
    }
    public void serverRequestChannelOpenFailed(Exception e) {
        e.printStackTrace();
    }
    public void serverRequestChannelClosed() {
        System.out.println("Server channel closed...");
    }
    public void serverRequestChannelClosedFailed(Exception e) {
        e.printStackTrace();
    }
    public void serverRequestBytesRead(int bytesRead) {
        System.out.println("Bytes read: ", bytesRead);
    }
    public void serverRequestBytesWritten(int bytesWritten) {
        System.out.println("Bytes written: ", bytesWritten);
    }
    public void serverRequestLargeMessage(int largeMessageSize) {
        System.out.println("Large message size: ", largeMessageSize);
    }
    public void serverRequestUnlikelyType(int type) {
        System.out.println("Unlikely type: ", type);
    }
    public void serverRequestUnlikelyData(int data) {
        System.out.println("Unlikely data: ", data);
    }
    public void serverRequestUnlikelyCounter(int counter) {
        System.out.println("Unlikely counter: ", counter);
    }
    public void serverRequestUnlikelyMessage(int messageId) {
        System.out.println("Unlikely message: ", messageId);
    }
    public void serverRequestUnlikelyConnection() {
        System.out.println("Unlikely connection...");
    }
    public void serverRequestUnlikelyTransaction(String messageId)
```
C.4 com.notthefly.mhp.Fly_IRForwarder

C.4.1 Fly_IRForwarder.java

```java
package com.notthefly.mhp.Fly_IRForwarder;

import java.net.*;
import java.io.*;
//import hp.infrared.*;
import com.notthefly.obex.*; //for Socket classes

* description:  TCP <-> IR forwarding program.
* Author: Tom Geelen
* copyright: NotTheFly Component Solutions BV

public class Fly_IRForwarder implements Runnable {
  private final static int PORT = 5513;
  private static int medium;
  private static int mode;

  private final static int IR = 0;
  private final static int RS232 = 1;
  private final static int CLIENT = 2;
  public static void main(String[] args) {
    if (args.length == 2) {
      if (args[0].toLowerCase().equals("ir")) {
        medium = IR;
      } else if (args[0].toLowerCase().equals("rs232")) {
        medium = RS232;
      } else {
        medium = -1;
      }
      if (args[1].toLowerCase().equals("server")) {
        mode = SERVER;
      } else if (args[0].toLowerCase().equals("client")) {
        mode = CLIENT;
      } else if (args[0].toLowerCase().equals("both")) {
        System.out.println("Warning: Client won't work correctly on RS232!");
      }
      mode = SERVER | CLIENT;
    } else {
      mode = -1;
    }
  }

  if (args.length != 2 || medium == -1 || mode == -1) {
    System.out.println("Invalid arguments. Usage: Fly_IRForwarder <medium> <mode>\n\t<medium>: IR or RS232, <mode>: SERVER or CLIENT or BOTH\n");
    System.exit(1);
  }

  if ((mode & CLIENT) != 0) {
    new Thread(new Fly_IRForwarder()).start(); //TCP listen thread
  }
  if ((mode & SERVER) != 0) {
    new Fly_IRForwarder().run(); //IR listen thread
  }

  private SomeServerSocket createSrvSocket() throws IOException {
    switch (medium) {
    case RS232: return new RS232ServerSocket(new RS232Socket("/dev/ttyS0"));
    case IR: return new IRDaServerSocket("OBEX");
    default: return null;
    }

  private SomeSocket createSocket() throws IOException {
    switch (medium) {
    case RS232: return new RS232Socket("/dev/ttyS0");
    case IR: return new IRDaSocket("ANY", "OBEX");
    default: return null;
    }

  }

  /**
   * Thread that listens for IR connections
   */
  void irListen() {
    SomeSocket irSock = null;
    SomeServerSocket irSrvSock = null;
    //IRSocket irSock = null;
    //IRServerSocket irSrvSock = null;
    Socket netSock = null;
    Forwarder ntfSock = null;
    Forwarder inet = null;

    try {
      //IRSyncSock = new IrsyncSock("OBEX");
      irSrvSock = createSrvSocket();
      irSock = catch(IException e) {
        System.out.println("Could not start IRDAS server.");
      }
    }
```
System.out.println(e.getMessage());
} System.exit(1);

System.out.println("IR Server started, registered OBEX service");
//Run forever, handle only one connection simultaneously.
while (true) {
  //Wait for an incoming connection
  try {
    irSock = irSvrSock.acceptConnection();
    catch (IOException e) {
      System.out.println("I/O Error: " + e.getMessage());
      System.exit(1);
    }

    //Try to connect TCP
    System.out.println("Received incoming connection, trying to connect to TCP...");
    boolean connected = false;
    while (!connected) {
      try {
        netSock = new Socket("10.1.1.192", PORT);
        connected = true;
      } catch (IOException e) {
        connected = false;
        System.out.println("TCP could not connect: " + e.getMessage());
        // Wait a while before attempting a new connection
        try {
          Thread.sleep(1000);
        } catch (InterruptedException ie) {
        }
      }
    }
    System.out.println("TCP connection established, starting to forward.");

    //Create forwarding threads
    try {
      netToFIR = new Forwarder(netSock.getInputStream(), irSock.getOutputStream());
      irToFIR = new Forwarder(irSock.getInputStream(), netSock.getOutputStream());
    } catch (IOException e) {
      System.out.println("I/O Error: " + e.getMessage());
      System.exit(1);
    }
    netToFIR.start();
    irToFIR.start();

    //Wait for forwarding threads to finish
    try {
      netToFIR.join();
      catch (InterruptedException e) {
      }
      irToFIR.join();
      catch (InterruptedException e) {
      }
    }

    //Threads are done, so clean up
    try {
      irSock.close();
      netSock.close();
    } catch (IOException e) {
    }
    try {
      netSock = null;
      irSock = null;
    } catch (IOException e) {
      irSock = null;
      netSock = null;
    }
    System.out.println("Connections closed.");
  }

  // Thread that listens for TCP connections
  public void run() {
    try {
      netSvrSock = new ServerSocket(PORT);
    } catch (IOException e) {
      System.out.println("Could not start server on port " + PORT + ":");
      System.out.println(e.getMessage());
      System.exit(1);
    }

    System.out.println("Server started, listening at port " + PORT);
    //Run forever, handle only one connection simultaneously.
    while (true) {
      //Wait for an incoming connection
      try {
        netSock = netSvrSock.accept();
      } catch (IOException e) {
        System.out.println("I/O Error: " + e.getMessage());
        System.exit(1);
      }

      //Try to connect IR
      System.out.println("Received incoming connection, trying to connect to IR...");
      boolean connected = false;
      while (!connected) {
        try {
          irSock = createSocket();
          irSock = new IRSocket("ANY", "OBEX");
          connected = true;
        } catch (IOException e) {
          connected = false;
          System.out.println("IR could not connect: " + e.getMessage());
          // Wait a while before attempting a new connection
        }
      }
    }
  }
} // Run forever, handle only one connection simultaneously.
try {
    Thread.sleep(2000);
    catch (InterruptedException ie) {}
}
System.out.println("IR connection established, starting to forward.");
//Create forwarding threads
try {
    netToIr = new Forwarder(netSock.getInputStream(), irSock.getOutputStream());
    irToNet = new Forwarder(irSock.getInputStream(), netSock.getOutputStream());
    catch (IOException e) {
        System.out.println("I/O Error: " + e.getMessage());
        System.exit(1);
    }
    netToIr.start();
    irToNet.start();
//Wait for forwarding threads to finish
    try {
        netToIr.join();
            catch (InterruptedException e) {}
    try {
        irToNet.join();
            catch (InterruptedException e) {}
//Threads are done, so clean up
        try {
            irSock.close();
                catch (IOException e) {}
        try {
            netSock.close();
                catch (IOException e) {}
            irSock = null;
            netSock = null;
            System.out.println("Connections closed.");
        }
    }
/**
     * Thread that just copies bytes from an input stream to an output stream.
     */
    private class Forwarder extends Thread {
        InputStream in;
        OutputStream out;
        /**
         * Constructs a forwarder that copies bytes from an InputStream to an
         * OutputStream.
         * @param in The InputStream to read from.
         * @param out The OutputStream to write to.
         */
        public Forwarder(InputStream in, OutputStream out) {
            this.in = in;
            this.out = out;
        }
        /** Main thread that does the copying. **/
        public void run() {
            int n;
            byte buffer[] = new byte[4096];
            try {
                while (true) {
                    n = in.read(buffer, 0, 4096);
                    if (n == -1) { // End of stream, close sockets
                        throw new IOException("End of stream");
                    }
                    out.write(buffer, 0, n);
                }
                catch (IOException e) {
                    try {
                        in.close();
                    } catch (IOException ioe) {
                        try {
                            out.close();
                            catch (IOException ioe) {
                                return;
                            }
                        }
                    }
                }
            }
        }
    }
C.5 com.notthefly.mhp.Fly_SMS
C.5.1 Fly_SMS_Xlet.java
package com.notthefly.mhp.Fly_SMS;

import java.awt.*;
import java.io.*;
import java.util.*;
import javax.microedition.io.*;
import javax.microedition.io.util.*;
import com.notthefly.mhp.util.*;
import com.notthefly.mhp.locale.*;
import org.hsi.u1i.*;
import org.hsi.u1i.u1_event.*;
import org.hsi.u1i.U1RColor;

import org.dub.event. *
import com.notthefly.core. *
import com.phillips.hspw.serial. SerialConnection;

/**
* description: Fly_Download Xlet -- main class
* @author: Tom Geelen
* @version: $Id: Fly_SMS_Xlet.java,v 1.1 2002/08/28 10:40:16 tong Exp $
* copyright: NotTheFly Component Solutions BV
*/
public class Fly_SMS_Xlet implements Xlet, Runnable {

    final static String BT_ADDRESS = "0000EE1C4D00";
    final static int BT_CHANNEL = 1;

    /** The Xlet context */
    XletContext xletContext = null;
    /** Is the application already started? */
    boolean isStarted = false;
    static boolean BConnected = false;
    static InputStream in = null;
    static OutputStream out = null;

    /** The scene on which all screens are displayed */
    HScene scene = null;
    static FlyContainer container;
    static Image background = null;
    static Image background2 = null;
    static boolean destroyed = false;
    static Socket sock = null;
    static SerialConnection sock = null;
    Thread thread;
    UserEventListener userEventListener = null;
    static InputStream phoneInput;
    static InputStream messageInput;
    static InputLine messageInput;
    static final int STATUS_WIDTH = 520;
    static final int STATUS_HEIGHT = 48;
    static final int MESSAGE_WIDTH = 408;
    static final int MESSAGE_HEIGHT = 308;

    /**
     * Initializes the Xlet. It creates a new HScene and creates a
     * SplashScreen.
     */
    public void initXlet(XletContext context) throws XletContextException {
        XletContext context = context;
        HSceneTemplate template = new HSceneTemplate();
        template.setPreference(HSceneTemplate.SCENE_PREFERRED, new Dimension(MESSAGE_WIDTH, MESSAGE_HEIGHT),
                  HSceneTemplate.PREFERRED);
        scene = HSceneFactory.getInstance().getBestScene(template);
        if (scene == null) System.out.println("Fly_Download: Oops... I didn't get a HScene...");
        scene.setLayout(null);
        container = null;
        thread = new Thread(this);
    }

    /**
     * Pauses the Xlet.
     */
    public void pauseXlet() {
        System.out.println("Fly_Download: pauseXlet()!");
        scene.setVisible(false);
    }

    /**
     * Starts the Xlet. If paused, just unpauses, else show message.
     */
    public void startXlet() {
        System.out.println("Fly_Download: startXlet()!");
        if (isStarted) {
            isStarted = true;
            GridBagLayout gridBag = new GridBagLayout();
            GridBagConstraints constraints = new GridBagConstraints();
            MediaTracker tracker = new MediaTracker(scene);
            background = Toolkit.getDefaultToolkit().getImage("Fly_Download/background-400x300.png");
            background2 = Toolkit.getDefaultToolkit().getImage("Fly_Download/background-520x40.png");
            tracker.addImage(background, 1);
            tracker.addImage(background2, 2);
            try {
                tracker.waitForAll();
            } catch (InterruptedException e) {
                } catch (InterruptedException e) {
        }
        container = new FlyContainer(MESSAGE_WIDTH, MESSAGE_HEIGHT, background);
        container.setLayout(gridBag);
        container.add(new JLabel("");
        container.add(new JLabel("");
        container.add(new JLabel("");
        container.add(new JLabel("");
        container.add(new JLabel("");
        container.add(new JLabel("");
        container.add(new JLabel("");
        container.add(new JLabel("");
        container.add(new JLabel("");
        container.add(new JLabel("");
        container.add(new JLabel("");
        container.add(new JLabel("");
        container.add(new JLabel("");
        } catch (InterruptedException e) {
        }

    } catch (InterruptedException e) {
        }

    } catch (InterruptedException e) {
        }

    } catch (InterruptedException e) {
        }

}
label = HaviUtil.makeLabel("Enter message", Color.yellow, 22);
gridBag.setConstraints(label, constraints);
container.add(label);
messageInput = new FlyInputLine(36, false);
gridBag.setConstraints(messageInput, constraints);
container.add(messageInput);
messageInput2 = new FlyInputLine(36, false);
gridBag.setConstraints(messageInput2, constraints);
container.add(messageInput2);

HContainer subContainer = new HContainer();
subContainer.setLayout(new FlowLayout());
subContainer.add(HaviUtil.makeLabel("Press the", Color.yellow, 20));
subContainer.add(HaviUtil.makeLabel("GREEN", Color.green, 24));
subContainer.add(HaviUtil.makeLabel("button to send SMS", Color.yellow, 20));
gridBag.setConstraints(subContainer, constraints);
container.add(subContainer);

subContainer = new HContainer();
subContainer.setLayout(new FlowLayout());
subContainer.add(HaviUtil.makeLabel("or the", Color.yellow, 20));
subContainer.add(HaviUtil.makeLabel("RED", Color.red, 24));
subContainer.add(HaviUtil.makeLabel("button to quit", Color.yellow, 20));
gridBag.setConstraints(subContainer, constraints);
container.add(subContainer);

scene.addComponent(container);
phoneInput.requestFocus();
scene.repaint();

thread = new Thread(this);
UserEventRepository keyEvents = new UserEventRepository("Fly_Download");
keyEvents.addKeyListener(KeyEvent.VK_PERMISSIONKEY_0);
keyEvents.addKeyListener(KeyEvent.VK_PERMISSIONKEY_1);
keyEvents.addKeyListener(KeyEvent.VK_PERMISSIONKEY_2);
keyEvents.addKeyListener(KeyEvent.VK_PERMISSIONKEY_3);
userEventListener = new UserEventListener() {
    public void userEventReceived(UserEvent e) {
        switch (e.getType()) {
        case HCREvent.VK_PERMISSIONKEY_0:
            // do something
            break;
        case HCREvent.VK_PERMISSIONKEY_1:
            // do something
            break;
        case HCREvent.VK_PERMISSIONKEY_2:
            // do something
            break;
        case HCREvent.VK_PERMISSIONKEY_3:
            // do something
            break;
        }
    }
    EventManager.getInstance().addUserEventListener(userEventListener);
    scene.setVisible(true);
}

public void run() {
    try {
        byte[] buffer;
        try {
            sock = new Socket("10.1.1.130", PORT);
            sock = new SerialConnection("RS232");
            in = new RealInputStream(socket.getInputStream());
            out = sock.getOutputStream();
        } catch (IOException e) {
            System.out.println("Setting up communication failed!");
            e.printStackTrace();
            return;
        }
        try {
            sock.setSoTimeout(0); // No timeout
        } catch (SocketException e) {
        }
        try {
            // Start BT connection
            send("AT+CRECN = BT_CHANNEL + "r\n");
            waitFor("OK");
            send("AT+ BT_ADDRESS + "r\n");
            waitFor("CONNECT+r\n");
            if (true) {
                // Send SMS
                send("AT+smsr="r");
                waitFor("OK");
                send("AT+CMSGF="r");
                waitFor("OK");
                send("AT+CMSGS="r + " phoneInput.getContents() + ","r");
                waitFor("OK");
                send(messageInput1.getContents() = ","r");
                waitFor("OK");
                send(messageInput2.getContents() = ","r");
                waitFor("OK");
                send(messageInput2.getContents() = ","r");
                waitFor("OK");
            }
        }
    }

Version Error! No text of specified style in document.
Page 66 of 128
© Copyright NotTheFly Component Solutions 9-12-2002
waitFor('OK\r\n');  
} //waitFor('OK');  
message('SMS sent!');  
} catch (IOException e) {  
if (!destroyed) {  
message('I/O error.');  
System.out.println('---I/O Error---');  
e.printStackTrace();  
}  
return;  
} finally {  
if (sock != null) {  
if (!connected) {  
try {  
Thread.sleep(1100);  
send('+++');  
Thread.sleep(1100);  
waitFor('OK');  
send('ABN\r\n');  
waitFor('OK');  
} catch (IOException e) {  
} catch (InterruptedException e) {  
}  
try {  
in.close();  
catch (IOException e) {  
}  
try {  
out.close();  
catch (IOException e) {  
}  
try {  
sock.close();  
catch (IOException e) {  
sock = null;  
connected = false;  
}  
}  
synchronized(this) {  
try {  
wait(3000);  
catch (InterruptedException e) {  
}  
}  
}  
/**  
* Writes a string to a stream.  
* @param out The stream to write to.  
* @param str The string to write.  
* @throws IOException When an I/O error occurs.  
*/  
private void send(String str) throws IOException {  
System.out.println("Sending \"" + str + "\"");  
out.write(str.getBytes());  
out.flush();  
}  
/**  
* Waits for a specific string to appear on a stream.  
* @param in The stream to read.  
* @param str The string to look for.  
* @throws IOException When an I/O error occurs.  
*/  
private void waitFor(String str) throws IOException {  
System.out.println("Waiting for \"" + str + "\"");  
byte[] text = str.getBytes();  
int pos = 0, b = text.length();  
while (pos < text.length) {  
//System.out.println("Read: \"" + b + "\"");  
if (b == text[pos]) {  
pos++;  
} else if (b == text[0]) {  
pos = 2;  
} else {  
pos = 0;  
}  
System.out.println("Just saw \"" + str + "\".");  
}  
/**  
* Helper function that displays a new message.  
* @param msg The message to display.  
*/  
private void message(String msg) {  
scene.remove(container);  
container = new FlyContainer(STATUS_WIDTH, STATUS_HEIGHT, background2);  
container.setBounds(0, 0, STATUS_WIDTH, STATUS_HEIGHT);  
container.setLayout(new FlowLayout());  
container.add(new Util.makeLabel(msg, Color.yellow, 20));  
container.repaint();  
scope.add(container);  
container.validate();  
scene.validate();  
container.repaint();  
scene.repaint();  
}  
/**  
* Destroys the Xlet. Disposes all used resources.  
* @throws XletException Ignored. Xlet is always destroyed.  
*/  
public void destroy() {  
}
public void destroyXlet(boolean unconditional) throws XletStateChangeException {
    System.out.println("Fly_SMS: destroyXlet()");
    destroyed = true;
    if (userEventListener != null) {
        EventManager.getInstance().removeUserEventListener(userEventListener);
    }
    if (scene != null) {
        scene.setVisible(false);
        scene.remove(container);
        container = null;
        HSceneFactory.getInstance().dispose(scene);
    }
    if (background != null) {
        background.flush();
    }
    if (background2 != null) {
        background2.flush();
    }
    if (sock != null) {
        if (!Connected) {
            try {
                Thread.sleep(1100);
                send("+++!");
                Thread.sleep(1100);
                waitFor("OK");
                send("* Ok
                ");
                waitFor("OK");
                in.close();
                out.close();
            } catch (IOException e) {
                catch (InterruptedException e) {
                }
            }
        }
    }
    try {
        sock.getInputStream().close();
    } catch (IOException e) {
    }
    try {
        sock.getOutputStream().close();
    } catch (IOException e) {
    }
    sock = null;
    //since we closed the socket, any blocking reads in the thread will fail and
    //execution of it will terminate.
    if (thread != null) {
        thread.interrupt();
        try {
            thread.join();
            catch (InterruptedException e) {
        }
    }
}

public void exitXlet() {
    try {
        destroyXlet(true);
    } catch (XletStateChangeException e) {
        xletContext.notifyDestroyed();
    }
}

C.6 com.notthefly.mhp.Fly_Weight

C.6.1 Fly_Weight_Xlet.java

package com.notthefly.mhp.Fly_Weight;
import java.tv.xlet.*;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Point;
import com.notthefly.mhp.util.*/;
import org.havi.ui.*/;
import org.dbv.ui.DBCColor;
/**
 * description: Fly_Weight Xlet -- main class
 * author: Tom Beeler
 * @version: Std: Fly_Weight_Xlet.java,v 1.3 2002/07/23 10:47:50 toms Exp $
 * copyright: NotTheFly Component Solutions BV
 */
public class Fly_Weight_Xlet implements Xlet, SplashListener {
    /**
     * The splash screen */
    SplashScreen splash = null;
    /** The data input screen */
    InputScreen inputScreen = null;
    /** The menu screen */
    MenuScreen menuScreen = null;
    /** The Xlet context */
    XletContext xletContext = null;
    /** Is the application already started? */
    boolean isStarted = false;
    /** The scene on which all screens are displayed */
    HScene scene = null;
    /** Width of the Xlet */
    static final int W = 420;


Version Error! No text of specified style in document.

© Copyright NotTheFly Component Solutions 9-12-2002
public void initXlet(XletContext context) throws XletStateException {  
  XletContext = context;  
  HSceneTemplate template = new HSceneTemplate();  
  template.setPreference(HSceneTemplate.SCENE_PIXEL_DIMENSION, new Dimension(W, H));  
  template.setPreference(HSceneTemplate.SCENE_PIXEL_LOCATION, new Point(100, 20));  
  scene = HSceneFactory.getInstance().getScene(template);  
  splash = new SplashScreen(this, scene);  
}

/** Pauses the Xlet.  
 * 
 */
public void pauseXlet() {  
  System.out.println("Fly_Weight: pauseXlet()");  
  scene.setVisible(false);  
}

/** * Starts the Xlet. If paused, just unpaase, else 'run' the SplashScreen.  
 */
public void startXlet() {  
  System.out.println("Fly_Weight: startXlet()");  
  if (!isStarted) isStarted = true;  
  splash.init();  
  scene.setVisible(true);  
}

/** * Destroys the Xlet. Disposes all used resources.  
 * Param unconditional: (Always destroyed, unconditional. Xlet is always destroyed.  
 * Throws XletStateChangeException Never thrown. Xlet is always destroyed.  
 */
public void destroyXlet(boolean unconditional) throws XletStateChangeException {  
  if (scene == null) scene.setVisible(false);  
  splash = null;  
  inputScreen = null;  
  menuScreen = null;  
  if (scene != null) HSceneFactory.getInstance().dispose(scene);  
}

/** * Loads the entire application. Called by SplashScreen.  
 */
public void loadApplication() {  
  System.out.println("loadApplication() start");  
  inputScreen = new InputScreen(this, scene);  
  inputScreen.init();  
  menuScreen = new MenuScreen(this, scene);  
  menuScreen.init();  
  System.out.println("loadApplication() done");  
}

/** * Starts the InputScreen. Called by SplashScreen when it's done.  
 */
public void splashDone() {  
  System.out.println("splashDone called");  
  splash = null;  
  System.out.println("starting InputScreen");  
  inputScreen.start();  
  System.out.println("started InputScreen");  
}

/** * Starts the menuScreen. Called by InputScreen when it's done.  
 */
public void inputDone(int gender, int weight, int height) {  
  menuScreen.start(gender, weight, height);  
}

public void exitXlet() {  
  try {  
    destroyXlet(true);  
  } catch (XletStateChangeException e) {}  
  XletContext.notifyDestroyed();  
}

C.6.2 InputScreen.java

package com.notTheFly.mhp.Fly_Weight;
import com.notTheFly.mhp.locale.*;
import com.notTheFly.mhp.util.*;
import java.io.*;
import java.awt.*;

Version Error! No text of specified style in document.

© Copyright NotTheFly Component Solutions 9-12-2002
import java.awt.event.*;
import org.havi.ui.*;
import org.havi.ui.event.*;
import org.dvb.event.*;
import org.dvb.ui.DVBColor;

/**
 * description: InputScreen. Get the necessary input from the user.
 * The localized language file needs to be:
 * <br>
 * &lt;html&gt;
 * &lt;button onclick="&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&n

Error! No text of specified style in document. - Error! No text of specified style in document.
Error! No text of specified style in document.
/** Sets the visibility of the form
 * @param flag <code>true</code> for visible, <code>false</code> for
 * invisible.
 */
public void setVisible(boolean flag) {
  if (scene != null)
    scene.setVisible(flag);
}

/** Initializes this object, i.e. load all necessary objects/data and set up
 * the screen.
 */
public void init() {
  // Start loading images (for the alerts & ok button)
  ok1 = Toolkit.getDefaultToolkit().getImage("ok1.png");
  ok2 = Toolkit.getDefaultToolkit().getImage("ok2.png");
  images[0] = Toolkit.getDefaultToolkit().getImage("fly_diet_alert_too_light.png");
  ModalTracker tracker = new ModalTracker(inputcontainer);
  tracker.addImage(ok1, 1);
  tracker.addImage(ok2, 3);
  for (int i = 0; i < 4; i++) tracker.addImage(images[i], 2);

  // Create constraints object
  GridBagConstraints constraints = new GridBagConstraints();
  constraints.gridx = GridBagConstraints.REMAINDER;
  constraints.gridy = GridBagConstraints.REMAINDER;

  // Input screen initialization
  label =北汽UIUtil.makeLabelLocale.getMessage("dataform.name").Color.yellow, 30);
  inputGridBag.setConstraints(label, constraints);
  inputcontainer.add(label);
  label =北汽UIUtil.makeLabelLocale.getMessage("dataform.enter!") + " \n ", Color.white, 22);
  inputGridBag.setConstraints(label, constraints);
  inputcontainer.add(label);
  label =北汽UIUtil.makeLabelLocale.getMessage("dataform.sex").Color.white, 22);
  inputGridBag.setConstraints(label, constraints);
  inputcontainer.add(label);
  gender = new FlyRadioGroup()
    .new String[] {北汽.getLocalizedString("dataform.gender").
                  locale.getLocalizedString("dataform.gender")
                  true
    .getConstraints().gridwidth = GridBagConstraints.REMAINDER;
  inputGridBag.setConstraints(gender, constraints);
  inputcontainer.add(gender);
  weightInput =北汽UIUtil.makeLabelLocale.getMessage("dataform.weight").Color.white, 22);
  constraints.gridwidth = GridBagConstraints.REMAINDER;
  inputGridBag.setConstraints(weightInput, constraints);
  inputcontainer.add(weightInput);
  heightInput =北汽UIUtil.makeLabelLocale.getMessage("dataform.height").Color.white, 22);
  constraints.gridwidth = GridBagConstraints.REMAINDER;
  inputGridBag.setConstraints(heightInput, constraints);
  inputcontainer.add(heightInput);
  ck =北汽佈GraphicButton(ck2, ok1, ok1);
  gender.setFocusTraversalVisible(null, null, null, null);
  weightInput.setFocusTraversalVisible(null, null, null, null);
  heightInput.setFocusTraversalVisible(null, null, null, null);
  ok.setFocusTraversalVisible(null, null, null, null);
  constraints.fill = GridBagConstraints.BOTH;
  constraints.gridwidth = GridBagConstraints.REMAINDER;
  inputGridBag.setConstraints(ok, constraints);
  inputcontainer.add(ok);
  ck.addActionListener(this);
  inputcontainer.validate();
  inputcontainer.repaint();
  // set up key listener
  keyEvents.addKey(北汽佈KeyEvent.VK_COLOURED_KEY_1);
  keyEvents.addKey(北汽佈KeyEvent.VK_P2);
  keyListener =北汽佈UserEventListener()
    .public void keyPressed(KeyEvent e) {
      switch (e.getKeyCode())
      {
      case KeyEvent.VK_0:
      case KeyEvent.VK_9:
        break;
      default:
        break;
      }
    }
    inputContainer.setVisible(true);
    inputContainer.repaint();
}
case HReEvent.VK_COLORED_KEY_1:
    case KeyEvent.VK_F2:
        if (displayingAlert) {
            displayingAlert = false;
            scene.remove(alertContainer);
            alertContainer = null;
            scene.add(inputContainer);
            scene.validate();
            ok.requestFocus();
            scene.repaint();
        } else {  
            actionPerformed(new ActionEvent(this, ActionEvent.ACTION_PERFORMED, "OK"));
        }
        break;
    }
    }
    })
    // Wait until all images are loaded
    try {
        tracker.waitForAll();
        catch (InterruptedException e) {}  
    }
/**
 * Displays this screen (and start interaction with it). (Elink init) should
 * have been called before calling this method.
 */
public void start() {
    scene.add(inputContainer);
    scene.setVisible(true);
    scene.repaint();
    gender.requestFocus();
    EventManager.getInstance().addUserEventListener(keyListener, keyEvents);
    }
/**
 * Show an alert.
 */
    *param caption The caption for the alert.
    *param text The text to display in the alert.
    *param img The image to show with the alert, or <code>null</code> for
    * no image.
    */
private void alert(String caption, String text, Image img) {
    displayingAlert = true;
    alertContainer = new HContainer(W, H);
    alertContainer.setBounds(0, 0, W, H);
    GridBagLayout alertGridBag = new GridBagLayout();
    alertContainer.setLayout(alertGridBag);
    try {
        alertContainer.setBackground(new HFlatMatte(0.6f));
        catch (HMatteException e) {
            System.out.println(e);
    }
    // Create constraints object
    GridBagConstraints constraints = new GridBagConstraints();
    constraints.gridwidth = GridBagConstraints.REMAINDER;
    // Alert screen initialization.
    HStaticText label;  
    label = HUtilUtils.makeLabel(caption, Color.yellow, 30);
    alertGridBag.setConstraints(label, constraints);
    alertContainer.add(label);
    if (img != null) {
        constraints.gridwidth = GridBagConstraints.REMAINDER;
        HStaticAnimation staticanim = new HStaticAnimation(new Image[] {img}, 10, 0, 0);
        alertGridBag.setConstraints(staticanim, constraints);
        alertContainer.add(staticanim);
    }
    constraints.gridwidth = GridBagConstraints.REMAINDER;
    label = HUtilUtils.makeLabel(HUtilUtils.wrap(text, 25), Color.white, 22);
    alertGridBag.setConstraints(label, constraints);
    alertContainer.add(label);
    NGraphicButton button = new NGraphicButton(ok2, ok1, ok1);
    constraints.fill = GridBagConstraints.BOTH;
    constraints.gridwidth = GridBagConstraints.REMAINDER;
    alertGridBag.setConstraints(button, constraints);
    alertContainer.add(button);
    button.addActionListener();
    new NActionListener() {
        /**
         * Handle the OK button presses on the advice screen.
         */
        public void actionPerformed(ActionEvent e) {
            displayingAlert = false;
            scene.remove(alertContainer);
            alertContainer = null;
            scene.add(inputContainer);
            scene.validate();
            ok.requestFocus();
            scene.repaint();
        }
    });
    alertContainer.validate();
    alertContainer.setVisible(true);
    alertContainer.repaint();
    scene.remove(inputContainer);
scene.add(alertContainer);
scene.validate();
scene.repaint();
button.requestFocus();
}
/**
 * Check if user input is in range, and if not display an alert.
 * @param weight The entered weight.
 * @param height The entered height.
 * @return <code>true</code> if all input was ok, <code>false</code> otherwise.
 */
private boolean checkInput(int weight, int height) {
    if (weight < MIN_WEIGHT) {
        // Too light
        alert(localize.getLocalizedString("weightless.name"),
             localize.getLocalizedString("weightless.advice"),
             images[8]);
        return false;
    } else if (weight > MAX_WEIGHT) {
        // Too heavy
        alert(localize.getLocalizedString("tooveavy.name"),
             localize.getLocalizedString("tooveavy.advice"),
             images[8]);
        return false;
    } else if (height < MIN_HEIGHT) {
        // Too small
        alert(localize.getLocalizedString("toosmall.name"),
             localize.getLocalizedString("toosmall.advice"),
             images[8]);
        return false;
    } else if (height > MAX_HEIGHT) {
        // Too big
        alert(localize.getLocalizedString("toobig.name"),
             localize.getLocalizedString("toobig.advice"),
             images[8]);
        return false;
    }
    return true;
}
/**
 * Handle commands on the input screen.
 */
public void actionPerformed(ActionEvent e) {
    if (g == null) // gender, weight and height
        g = gender.getSelected();
    try {
        w = Integer.parseInt(weightInput.getText());
        catch (NumberFormatException nfe) {
            w = 0;
            try {
                h = Integer.parseInt(heightInput.getText());
                catch (NumberFormatException nfe) {
                    h = 0;
                }
            }
            if (checkInput(w, h)) {
                // All input OK
                EventManager.getInstance().removeUserEventListener(keyListener);
                scene.setVisible(false);
                scene.remove(inputContainer);
                xlet.inputDone(g, w, h);
            }
        }
    }
}

C.6.3 MenuScreen.java

package com.notthefly.mxp.Fly_Weight;
import com.notthefly.mxp.locale;,
import com.notthefly.mxp.util.*;
import java.awt.*;
import java.awt.event.*;
import org.dvb.event.*;
import org.havi.ui.*;
import org.havi.ui.event.*;
/**
 * description: MenuScreen. Displays a menu with calculators, and when
 * a selection is made, it displays the appropriate advice.
 * The localized language file needs:<br>
 * &lt;menu: BNInbsp;&nbsp;&nbsp;&nbsp;&nbsp; &nbsp; "Body Mass Index" &nbsp; text
 * &lt;menu: RSAInbsp;&nbsp;&nbsp;&nbsp;&nbsp; &nbsp; "Body Surface Area" &nbsp; text
 * &lt;menu: LWInbsp;&nbsp;&nbsp;&nbsp;&nbsp; &nbsp; "Lean Body Weight" &nbsp; text
 * &lt;menu: IBWInbsp;&nbsp;&nbsp;&nbsp;&nbsp; &nbsp; "Ideal Body Weight" &nbsp; text
 * &lt;menu: BMII&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; &nbsp; "BMI advice texts
 * &lt;menu: RSA&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; &nbsp; "Body Surface Area advice texts
 * &lt;menu: LW&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; &nbsp; "Lean Body Weight advice texts
 * &lt;menu: IBW&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; &nbsp; "Ideal Body Weight advice texts
 *<author Tom Geelen
 * &lt;version 1: MenuScreen.java, v 1.2 2002/05/31 12:45:20 tomh Exp $
/** Localisation instance to safe a lot of typing */
private Localisation localse = Localisation.getLocalisation();
/** The "back" button */
private HBorderButton back = null;
/** The "exit" button */
private HBorderButton exit = null;
/** The "ok" button */
private HBorderButton ok = null;
/** ok button image [focused] */
private Image ok;
/** ok button image [unfocused] */
private Image ok2;
/** The Fly_Weight_Xlet */
private Fly_Weight_Xlet xlet = null;
/** The HScene object on which the splash screen is displayed */
private HScene scene = null;
/** The container which contains all menu screen components */
private FlyContainer menucontainer = null;
/** GridBagLayout object */
private GridBagLayout menuGridBag = null;
/** The container which contains all advice screen components */
private FlyContainer advicecontainer = null;
/** The menu-buttons */
private FlyRadioButtonGroup menu;
/** The event repository for the colored key events */
private UserEventRepository keyEvents;
/** The listener for the colored key events */
private UserEventHandler keylistener;
/** Is the advice screen displayed? */
private boolean displayingAdvice = false;
/** Is the HScene focussed? */
private boolean hasFocus = false;
/** The weight entered by the user */
private int weight;
/** The height entered by the user */
private int height;
/** The gender entered by the user */
private int gender;
/** The last made menu choice */
private int choice;
/** Width of the Xlet */
static final int W = 420;
/** Height of the Xlet */
static final int H = 510;

public MenuScreen(Fly_Weight_Xlet xlet, HScene scene)
{
  this.xlet = xlet;
  this.scene = scene;
  menucontainer = new FlyContainer(W, H);
  menucontainer.setBounds(0, 0, W, H);
  menuGridBag = new GridBagLayout();
  menucontainer.setLayout(menuGridBag);
  try {
    menucontainer.setMatte(new HPaintMatte(0.6f));
    catch (H MatteException e) {
      System.out.println(e);
    }
  }
}

/**
 * Initializes this object, i.e. load all necessary objects/data and set up
 * the screen.
 */
public void init()
{
  // Create constraints object
  GridBagConstraints constraints = new GridBagConstraints();
  constraints.gridx = GridBagConstraints.REMAINDER;
  constraints.gridy = GridBagConstraints.REMAINDER;
  ok = Toolkit.getDefaultToolkit().getImage("ok.png");
  ok2 = Toolkit.getDefaultToolkit().getImage("ok2.png");
  image ok = Toolkit.getDefaultToolkit().getImage("back1.png");
  image ok2 = Toolkit.getDefaultToolkit().getImage("back2.png");
  image exit1 = Toolkit.getDefaultToolkit().getImage("exit1.png");
  image exit2 = Toolkit.getDefaultToolkit().getImage("exit2.png");
  MediaTracker mediaTracker = new MediaTracker(menucontainer);
  medialocator.addImage(ok, 1);
  medialocator.addImage(ok2, 1);
  medialocator.addImage(back1, 1);
  medialocator.addImage(back2, 1);
  medialocator.addImage(exit1, 1);
  medialocator.addImage(exit2, 1);
  HStaticText label = HGuiUtils.makeLabel(localse.getString("menu.name"),
                                          0);
  label.setLocation(120, 30);
menuGridBag.setConstraints(label, constraints);
menucontainer.add(label);

menu = new FlyradioGroup;
new String[] { locales.getLocalizedString(‘menu.BMI’),
locales.getLocalizedString(‘menu.BSA’),
locales.getLocalizedString(‘menu.LBM’),
locales.getLocalizedString(‘menu.IBM’),
}, false);
constraints.gridwidth = GridBagConstraints.RELATIVE;
constraints.gridheight = 3;
constraints.gridx = 0;
constraints.gridy = 1;
menuGridBag.setConstraints(menu, constraints);
menucontainer.add(menu);

/*

menu[0] = HAvUI3.makeButton(locales.getLocalizedString(‘menu.BMI’));
menu[1] = HAvUI3.makeButton(locales.getLocalizedString(‘menu.BSA’));
menu[3] = HAvUI3.makeButton(locales.getLocalizedString(‘menu.IBM’));
constraints.fill = GridBagConstraints.HORIZONTAL;
for(int i = 0; i < 4; i++) {
constraints.gridwidth = GridBagConstraints.RELATIVE;
ball = new Ball(i);
menuGridBag.setConstraints(ball, constraints);
menucontainer.add(ball);
constraints.gridwidth = GridBagConstraints.REMAINDER;
constraints.gridx = 1;
constraints.gridy = 1;
menuGridBag.setConstraints(exit, constraints);
menucontainer.add(exit);
constraints.gridy = 2;
menuGridBag.setConstraints(ok, constraints);
menucontainer.add(ok);
constraints.gridy = 3;
menuGridBag.setConstraints(back, constraints);
menucontainer.add(back);
}

*/

//Create some space between the buttons
label = HAvUI3.makeLabel(‘ ’, Color.black, 16);
menuGridBag.setConstraints(label, constraints);
menucontainer.add(label);

/*

exit = new HGraphicButton(exit2, exit1, exit1);
ok = new HGraphicButton(ok2, ok1, ok1);
back = new HGraphicButton(back2, back1, back1);
constraints.gridwidth = GridBagConstraints.REMAINDER;
constraints.gridheight = 1;
constraints.gridx = 3;
constraints.gridy = 1;
menuGridBag.setConstraints(exit, constraints);
menucontainer.add(exit);
constraints.gridy = 2;
menuGridBag.setConstraints(ok, constraints);
menucontainer.add(ok);
constraints.gridy = 3;
menuGridBag.setConstraints(back, constraints);
menucontainer.add(back);
*/

//Focus traversal

/*

menu[0].setFocusTraversal(null, menu[1], null, null);
menu[1].setFocusTraversal(menu[0], menu[2], null, null);
menu[2].setFocusTraversal(menu[1], menu[3], null, null);
menu[3].setFocusTraversal(menu[2], exit, null, null);
back.setFocusTraversal(menu[3], null, null, exit);
exit.setFocusTraversal(menu[3], null, back, null);
*/

menu.setFocusTraversal(null, null, ok, ok);
exit.setFocusTraversal(null, ok, menu, menu);
back.setFocusTraversal(ok, null, menu, menu);
ok.setFocusTraversal(exit, back, menu, menu);

exit.addActionListener(this);
back.addActionListener(this);
ok.addActionListener(this);

menucontainer.validate();
menucontainer.setVisible(true);
menucontainer.repaint();

*/

//Set up keyevent listener
keyEvents = new KeyEventRepository(“com.notthefly.xlets.Fly_Weight.MenuScreen”);
keyEvents.addKey(HKeyEvent.VK_COLORED_KEY_0);
keyEvents.addKey(HKeyEvent.VK_F1);
keyEvents.addKey(HKeyEvent.VK_COLORED_KEY_1);
keyEvents.addKey(HKeyEvent.VK_F2);
keyEvents.addKey(HKeyEvent.VK_COLORED_KEY_2);
keyEvents.addKey(HKeyEvent.VK_F3);
keyListener = new UserEventListener()
{
    public void userEventReceived(UserEvent e)
    {
        if (e.getKeyType() == KeyEvent.KEY_PRESSED) {
            switch (e.getKeyCode()) {
                case HKeyEvent.VK_COLORED_KEY_0:
                    keyEvent.VK_F1;
                    if ((KeyEvent) e.getKeyCode()) {
                        EventManager.getInstance().removeUserEventListener(keyListener);
                        menuContainer = null;
                        xlet.exitXlet();
                    } break;
                case HKeyEvent.VK_COLORED_KEY_1:
                    keyEvent.VK_F2;
                    if ((KeyEvent) e.getKeyCode()) {
                        EventManager.getInstance().removeUserEventListener(keyListener);
                        menuContainer = null;
                    } break;
                case HKeyEvent.VK_COLORED_KEY_2:
                    keyEvent.VK_F3;
                    if ((KeyEvent) e.getKeyCode()) {
                        EventManager.getInstance().removeUserEventListener(keyListener);
                        menuContainer = null;
                    } break;
            }
        }
    }
}

// Error! No text of specified style in document.
// Error! No text of specified style in document.
// Version Error! No text of specified style in document.

© Copyright NotTheFly Component Solutions 9-12-2002
Calculator.getBNAAdvice(height, weight));
break;
case 2:
    advice(Translator.getInstance().getLocalizedString("menu.BNA"),
    Calculator.getBNAAdvice(gender, height, weight));
    break;
case 3:
    advice(Translator.getInstance().getLocalizedString("menu.BNA"),
    Calculator.getBNAAdvice(gender, height, weight));
    break;
} else {
    displayAdvice = false;
    scene.remove(adviceContainer);
    adviceContainer = null;
    scene.add(menuContainer);
    scene.validate();
    //menu[choice].requestFocus();
    menu.requestFocus();
    scene.repaint();
    break;
}
if (displayingAdvice) {
    EventManager.getInstance().removeEventListener(keyListener);
    scene.setVisible(false);
    scene.remove(menuContainer);
    xlet.splashDone();
    break;
}
}

//FocusListener
scene.addFocusListener(
    new FocusListener() {
        public void focusGained(FocusEvent focusEvent) {
            hasFocus = true;
        }
        public void focusLost(FocusEvent focusEvent) {
            hasFocus = false;
        }
    });
// Wait for images to load
try {
    mediastorage.waitForAll();
} catch (InterruptedException ex) {}
}

public void start(int gender, int height, int weight) {
    this.gender = gender;
    this.height = height;
    this.weight = weight;
    EventManager.getInstance().addUserEventListener(keyListener, keyEvents);
    scene.add(menuContainer);
    scene.validate();
    scene.setVisible(true);
    scene.repaint();
    //menu[0].requestFocus();
    menu.requestFocus();
}

/**
 * Show an advice.
 *
 * @param caption The caption for the advice.
 * @param text The advice text to display.
 */
private void advice(String caption, String msg) {
    displayAdvice = true;
    adviceContainer = new Container(W, H);
    adviceContainer.setOpaque(true);
    adviceContainer.setLayout(new BorderLayout());
    adviceContainer.add(adviceLabel);
    try {
        adviceLabel.setAdviser(new HFlatMatte(0.6f));
    } catch (HFlatException e) {
        System.out.println(e);
    }
    GridBagConstraints constraints = new GridBagConstraints();
    constraints.gridWidth = GridBagConstraints.REMAINDER;
    constraints.gridHeight = GridBagConstraints.REMAINDER;
    label = HUtil.makeLabel(caption, Color.yellow, 30);
    adviceGridBag.setConstraints(label, constraints);
    adviceContainer.add(label);
    label = HUtil.makeLabel(msg, Color.white, 22);
    adviceGridBag.setConstraints(label, constraints);
    adviceContainer.add(label);
    HGraphicButton button = new HGraphicButton(ok2, ok1, ok1);
    adviceGridBag.setConstraints(button, constraints);
    adviceContainer.add(button);
    button.addActionListener(
        new ActionListener() {
            public void actionPerformed(ActionEvent actionEvent) {
                scene.remove(adviceContainer);
                adviceContainer = null;
                scene.add(menuContainer);
                scene.validate();
            }
        });
}

C.6.4 Calculator.java

package com.notthefly.mhp.Fly_Weight;
import com.notthefly.mhp.locale.*;

public class Ball extends Component
{
    private int key;

    public Dimension getPreferredSize()
    {
        return new Dimension(20, 20);
    }

    public void paint(Graphics g)
    {
        switch(key)
        {
            case 0:
                g.setColor(Color.red);
                break;
            case 1:
                g.setColor(Color.green);
                break;
            case 2:
                g.setColor(Color.yellow);
                break;
            case 3:
                g.setColor(Color.blue);
                break;
            default:
                g.setColor(Color.black);
                break;
        }
        g.fillOval(0, 0, 20, 20);
    }

    public Ball(int i)
    {
        key = i;
        setBounds(0, 0, 20, 20);
    }
}

/**
 * Class that displays a colored circle (to indicate which colored key can be used).
 */

Error! No text of specified style in document.

Error! No text of specified style in document.

Error! No text of specified style in document.

Version Error! No text of specified style in document.

© Copyright NotTheFly Component Solutions 9-12-2002
import com.notthefly.nhp.util.HavUIUtils;
import java.lang.Math;

/**
 * description: Class containing the various calculators.
 * The localized language file needs `<br>
 * <li>NMI.kgpersqm
 * <li>NMI.info
 * <li>BSA.squarem
 * <li>BSA.info
 * <li>BSA.male
 * <li>BSA.female
 * <li>EBW.kg
 * <li>EBW.lbw
 * <li>EBW.abw
 * <li>EBW.dlf
 * <li>EBW.info
 * <li>LBW.leanweight
 * <li>LBW.muscles
 * <li>LBW.bones
 * <li>LBW.cortons
 * <li>LBW.ligaments
 * <li>LBW.water
 * <li>LBW.fatweight
 * <li>LBW.fat
 * <li>LBW.organs
 *
 * @author Tom Geelen
 * @version FID: Calculator.java, v 1.2 2002/05/03 12:45:20 tong Exp $
 * copyright: NotTheFly Component Solutions BV
 */

public class Calculator {

/**
 * Get a string representation of a double, with only 1 decimal.
 * @param d The double to convert.
 * @return A string representation of <code>d</code> with 1 decimal.
 */
private static String doubleToString(double d) {
    // Negative numbers should not occur with normal values, but can occur
    // when nonsense was entered. With this check we prevent results such as
    // "-5.8".
    if (d < 0) return "0.0";
    return ((int) d) + "." + (((int) (d * 10)) % 10);
}

/**
 * Gives Body Mass Index (BMI) advice.
 * The BMI is calculated from the weight and the height with the
 * following formula:
 * $BMI \text{ = weight (kg)} / \text{height (m)}^2$
 * @param weight The body weight.
 * @param height The body height.
 * @return A string containing the advice.
 */
public static String getBMIAdvice(int weight, int height) {
    double BMI = weight / (height * height);
    String advice = doubleToString(BMI) + "\n";
    advice += HavUIUtils.getLocalizedString("BMI.kgpersqm") + "\n";
    advice += HavUIUtils.getLocalizedString("BMI.info") + "\n";
    return advice;
}

/**
 * Gives Body Surface Area (BSA) advice.
 * The BSA is calculated from the weight and the height with the
 * following formula (Dubois):
 * $BSA = 0.20247 \times (\text{height (m)}^{0.725}) / \text{weight (kg)}^{0.425}$
 * STRANGE: Had to use the following formula to get the expected results....
 * @param weight The body weight.
 * @param height The body height.
 * @return A string containing the advice.
 */
public static String getBSAAdvice(int weight, int height) {
    double BSA = 0.20247 * Math.pow(height, 0.725) / Math.pow(weight, 0.425);
    String advice = doubleToString(BSA) + "\n";
    advice += HavUIUtils.getLocalizedString("BSA.squarem") + "\n";
    advice += HavUIUtils.getLocalizedString("BSA.info") + "\n";
    return advice;
}

/**
 * Gives Lean Body Weight (LBW) advice.
 * The LBW is calculated from the sex, weight and the height with the
 * following formulas:
 * $LBW (men) = (1.10 \times \text{weight (kg)}) - 128 ((\text{weight (kg)})^2) /
 * (100 \times \text{height (m)})^2)$
 * $LBW (women) = (1.07 \times \text{weight (kg)}) - 148 ((\text{weight (kg)})^2) /
 * (100 \times \text{height (m)})^2)$
 * @param gender The gender, <code>male</code> for male, <code>female</code> for female.
 * @param weight The body weight.
 * @param height The body height.
 * @return A string containing the advice.
 */
public static String getLBWAdvice(int weight, int height, String gender) {

}
public static String getBMIAdvice(int gender, int weight, int height) {
    double BMI = 0;
    String advice = ""
;
    if (gender == 0) {
        BMI = (1.0 * weight / (height * height));
    } else if (gender == 1) {
        BMI = (1.07 * weight / (height * height));
    } else {
        System.out.println("Calculator.java: Got invalid gender parameter!");
        BMI = 0;
    }

    advice = doubleToString(BMI) + ""
;
    return advice;
}

/**
 * Gives Ideal Body Weight (IBW) advice.
 * The IBW is calculated from the sex and the height with the
 * following formulas:
 *  minimum IBW(men) = height(m) * height(m) * 20.1
 *  maximum IBW(men) = height(m) * height(m) * 25
 *  minimum IBW(women) = height(m) * height(m) * 20.7
 *  maximum IBW(women) = height(m) * height(m) * 23.8
 * @param gender The gender. <code>0</code> for male, <code>1</code> for female.
 * @param weight The body weight.
 * @param height The body height.
 * @return A string containing the advice.
 * /
public static String getIBMAVice(int gender, int weight, int height) {
    double IBW;
    double height_in_meters = height / 100.0;
    String advice = ""
;
    if (gender == 0) {
        IBW = height_in_meters * height_in_meters * 25;
    } else if (gender == 1) {
        IBW = height_in_meters * height_in_meters * 23.8;
    } else {
        System.out.println("Calculator.java: Got invalid gender parameter!");
        IBW = 0;
    }

    advice = doubleToString(IBW) + ""
;
    return advice;
}

/**
 * Calculate overweight. Used for IBW advice
 * @param height The height.
 * @return The overweight.
 */
private static int checkWeight(int height) {
    int overweight = 0;
    if (height == 145) {
        overweight = 63;
    } else if (height > 145 && height <= 148) {
        overweight = 66;
    } else if (height > 148 && height <= 151) {
        overweight = 68;
    } else if (height > 151 && height <= 154) {
        overweight = 72;
    } else if (height > 154 && height <= 157) {
        overweight = 74;
    } else if (height > 157 && height <= 160) {
        overweight = 77;
    } else if (height > 160 && height <= 163) {
        overweight = 80;
    } else if (height > 163 && height <= 166) {
        overweight = 83;
    } else if (height > 166 && height <= 169) {
        overweight = 86;
    }
    return overweight;
}
C.7 com.notethefly.mhp.remotescreen

C.7.1 Client.java

/**
 * description: Client class for displaying stuff remotely on the TV screen. This class should run on a portable device with a limited screen. It connects to the server that runs on the STH.
 * Author: Tom Geelen
 * Version: $Id$
 * Copyright: NotTheFly Component Solutions BV
 */

generate from com.notethefly.mhp.remotescreen;

public class Client {
    /** Text to send for assigning to no variable */
    public static final String NONE = "void";

    /** Command to create a new screen. Should only be used once per session. */
    public static final int CREATE_SCREEN = 6;

    /** Command to create a new object */
    public static final int NEW = 1;

    /** Command to invoke a method of an object */
    public static final int METHOD = 2;

    /** Command to invoke a static method of a class */
    public static final int STATIC_METHOD = 3;

    /** Command to destroy the screen and terminate the connection */
    public static final int QUIT = 4;

    DataInputStream in;
    DataOutputStream out;

    /** Creates a new Client object. */
    public Client(InputStream in, OutputStream out) {
        this.in = new DataInputStream(in);
        this.out = new DataOutputStream(out);
    }

    /** Create a new remote screen with specified dimensions. This should only be called once per session. The resulting container is stored in a variable called "screen". */
    public RemoteScreen createScreen(int x, int y, int w, int h) throws RemoteScreenException, IOException {
        out.writeInt(CREATE_SCREEN);
        out.writeInt(x);
        out.writeInt(y);
        out.writeInt(w);
        out.writeInt(h);
        out.flush();
        return screen;
    }
}

Error! No text of specified style in document.
Error! No text of specified style in document.
Error! No text of specified style in document.

Page 80 of 128
/**
 * Creates a new object remotely. The no-argument constructor is used to
 * create the object.
 * @param var The name of the variable to reference the newly created object.
 * @param classname The fully qualified name of the class to instantiate.
 */
public void newObject(String var, String classname, Argument args[]) throws RemoteScreenException, IOException {
    out.writeInt(NEW);
    out.writeUTF(var);
    out.writeUTF(classname);
    out.writeInt(args.length);
    for (int i = 0; i < args.length; i++) {
        if (args[i].isVar) {
            out.writeUTF("var");
        } else {
            out.writeUTF(args[i].type);
            args[i].newValue();
        }
        out.flush();
        handleResponse();
    }
}

/**
 * Invokes a method of a remote object.
 * @param resultvar The name of the variable where the return value of the
 * call should be stored, or <code>NONE</code> to discard the returned value.
 * @param var The name of the variable for the object that contains the method.
 * @param methodName The name of the method to invoke.
 * @param args The arguments for the method.
 */
public void method(String resultvar, String var, String methodName, Argument args[]) throws RemoteScreenException, IOException {
    out.writeInt(METHOD);
    out.writeUTF(var);
    out.writeUTF(methodName);
    out.writeInt(args.length);
    for (int i = 0; i < args.length; i++) {
        if (args[i].isVar) {
            out.writeUTF("var");
        } else {
            out.writeUTF(args[i].type);
            args[i].newValue();
        }
    }
    out.writeUTF(resultvar);
    out.flush();
    handleResponse();
}

/**
 * Invokes a static method of a remote class.
 * @param resultvar The name of the variable where the return value of the
 * call should be stored, or <code>NONE</code> to discard the returned value.
 * @param classname The fully qualified name of the class that contains the method.
 * @param methodName The name of the method to invoke.
 * @param args The arguments for the method.
 */
public void staticMethod(String resultvar, String classname, String methodName, Argument args[]) throws RemoteScreenException, IOException {
    out.writeInt(STATIC_METHOD);
    out.writeUTF(classname);
    out.writeUTF(methodName);
    out.writeInt(args.length);
    for (int i = 0; i < args.length; i++) {
        if (args[i].isVar) {
            out.writeUTF("var");
        } else {
            out.writeUTF(args[i].type);
            args[i].newValue();
        }
    }
    out.writeUTF(resultvar);
    out.flush();
    handleResponse();
}

/**
 * Destroys the remote screen and closes the connection to the server.
 */
public void quit() throws RemoteScreenException, IOException {
    out.writeInt(EXIT);
    out.flush();
    handleResponse();
    try {
        in.close();
        out.close();
    } catch (IOException e) {
        in = null;
        out = null;
    }
}

/**
 * Helper method that handles the server responses. On errors it prints a
 * message on stdout.
 */
private void handleResponse() throws RemoteScreenException, IOException {
    try {
        String resp = in.readUTF();
        if (!resp.equals("ok")) return;
        String errmsg:
    }
}
if (resp.equals("error")) {
    errmsg = 'Error: ' + resp; 
} else {
    errmsg = 'Unknown response: ' + resp; 
}

// We got an error; clean up.
try {
    in.close();
    catch (IOException e) {
        out.close();
        in = null;
        out = null;
    }
    throw new RemoteScreenException(errmsg);
}
catch (IOException e) {
    throw new RemoteScreenException("IOException: " + e.getMessage());
}
*/

/**
 * This method sends example commands.
 * @throws RemoteScreenException When a communication error occurs or a
 * command fails.
 */
public void example() throws RemoteScreenException, IOException {
    createScreen(60, 60, 400, 200);
    newObj("comp", "java.awt.Canvas", new Argument[] { });
    method(NONE, "comp", "setBounds", new Argument[] { 
        new Argument(0), 
        new Argument(0), 
        new Argument(100), 
        new Argument(200) 
    });
    method(NONE, "screen", "add", new Argument[] { new Argument("java.awt.Component", "comp") });
    method(NONE, "gfx", "setGraphics", new Argument[] { 
        new Argument("java.awt.Color", "green") 
    });
    newObj("gfx", "fillRect", new Argument[] { 
        new Argument(30), 
        new Argument(30), 
        new Argument(60), 
        new Argument(80) 
    });
    newObj("white", "java.awt.Color", new Argument[] { new Argument(255), new Argument(255), new Argument(255) });
    method(NONE, "gfx", "drawString", new Argument[] { new Argument("Hello TV!") });
    method(NONE, "screen", "repaint", new Argument[] { });
}

/**
 * Represents an argument for a remote method call.
 */
public class Argument {
    public boolean isVar;
    public String type;
    String stringValue = "";
    int intValue = 0;
    boolean boolValue = false;

    /**
     * Creates a new variable argument.
     * @param type The type of the argument. This should be the type that
     * the method expects, not the actual variable type!
     * @param name The name of the variable.
     */
    public Argument(String type, String name) {
        this.type = type;
        this.stringValue = name;
    }

    /**
     * Creates a string constant argument.
     * @param str The string value.
     */
    public Argument(String str) {
        isVar = false;
        this.type = "java.lang.String";
        this.stringValue = str;
    }

    /**
     * Creates an integer constant argument.
     * @param i The int value.
     */
    public Argument(int i) {
        isVar = false;
        this.type = "int";
        this.intValue = i;
    }

    /**
     * Creates a boolean constant argument.
     * @param b The boolean value.
     */
    public Argument(boolean b) {
        isVar = false;
        this.type = "boolean";
        this.boolValue = b;
    }

    /**
     * Sends the value of this argument to the DataOutputStream.
     */
    public void writeValue() throws IOException {
        if (type.equals("int")) {
            // Write integer value
        } else if (type.equals("boolean")) {
            // Write boolean value
        } else if (type.equals("string")) {
            // Write string value
        }
    }
}
C.7.2 Server.java

```java
package com.notthefly.nobex.*;
import java.io.*;
import java.net.Socket;

public class Server implements NOBEXServerListener {

    NOBEXServer server;
    PipedOutputStream commandStream = null;
    PipedInputStream resultStream = null;

    /**
     * The object name we use in our NOBEX replies */
    String objectName = "(none)";

    /** Creates a new server object. This will start a new thread *
     * that starts listening for incoming connections. *
     * Throws IOException When binding to the port fails. */
    public Server() throws IOException {
        server = new NOBEXServer(new TrAServerSocket("OBEX"), 5913);
        commandStream = new PipedOutputStream(commandStream);
        resultStream = new PipedInputStream(resultStream);
    }

    public void serverException(Exception e) {
        System.out.println("NOBEX Server Exception: " + e.getMessage());
        e.printStackTrace();
    }

    public InputStream getStream(String name, Integer length) {
        return null;
    }

    /** Handle incoming data, i.e. a command for the RemoteScreen. *
     * Param name The name of the object that is transferred using the *
     * underlying NOBEX protocol. *
     * Return The stream that is read by the (link Screen) class. */
    public OutputStream putFile(String name) {
        // Use the same object name for our replies as we receive.
        objectName = name;
        System.out.println("PUT request: " + name);
        return commandStream;
    }

    /** Handle TRANSFER_DONE and DISCONNECT events. On TRANSFER_DONE (which is *
     * for receiving a command) we send back the response. On DISCONNECT we *
     * return null. *
     * Param type The type of event. *
     * Param data Event data. */
    public void serverNotify(int type, int data) {
        if (type == NOBEXServerListener.TRANSFER_DONE) {
            System.out.println("Server.java: End-of-transfer. send reply now...!");
        }
        try {
            //TrASocket sock = new TrASocket("ANY", "OBEX");
            TrASocket sock = new TrASocket("10.1.1.130", 5913);
            NOBEXClient client = new NOBEXClient(sock.getInputStream(), sock.getOutputStream());
        }
    }

    public static void main(String[] args) {
        System.out.println("Server.java: starting...");
        client.connect();
        System.out.println("Server.java: client.connect() done.");
    }
```
C.7.3 Screen.java

/**
 * description: Represents the TV display for remote apps.
 * It handles the commands for an incoming connection.
 * @author: Tom Geezen
 * @version: 1.0
 * copyright: NotTheFly Component Solutions BV
 */
package com.notthefly.mhp.remotescreen;

import java.net.Socket;
import java.io.*;
import java.awt.*;
import org.hnivi.util;
import java.util.Enumeration;
import java.lang.reflect.*;

public class Screen implements Runnable {
    /** Command to create a new screen. Should only be used once per session. */
    public static final int CREATE_SCREEN = 0;
    /** Command to create a new object */
    public static final int NEW = 1;
    /** Command to invoke a method of an object */
    public static final int METHOD = 2;
    /** Command to invoke a static method of a class */
    public static final int STATIC_METHOD = 3;
    /** Command to destroy the screen and terminate the connection */
    public static final int QUIT = 4;

    Socket sock;
    /** Reader for the input stream. */
    DataInputStream in;
    /** A pushback version of the raw inputstream, used to skip the length */
    bytes from Siemens packets.
    PushbackInputStream pbIn;
    DataOutputStream out;
    HashTable vars;
    Container screen = null;
    /** Creates a new Screen object. It starts a new thread to handle */
    incoming commands.
    * @param in the input stream for the communication.
    * @param out the output stream for the communication.
    */
    public Screen(InputStream in, OutputStream out) throws IOException {
        this.sock = null;
        pbIn = new PushbackInputStream(in);
        this.in = new DataInputStream(pbIn);
        this.out = new DataOutputStream(out);
        (new Thread(this)).start();
    }

    /** Main thread that handles incoming commands. */
}
public void run() {
    boolean done = false;
    int cmd;
    vars = new Hashtable();

    String classname;
    String name;
    System.out.println("Ready to receive commands...");
    try {
        while (!done) {
            skipIntsAndBytes();
            cmd = in.readInt();
            System.out.println("Got command: "+ cmd);
            switch (cmd) {
                case CREATE_SCREEN:
                    int x = in.readInt();
                    int y = in.readInt();
                    int w = in.readInt();
                    int h = in.readInt();
                    System.out.println("requesting hScreen");
                    //HSceneTemplate template = new HSceneTemplate();
                    //template.setPreference(HSceneTemplate.SCENE_PIXEL_LOCATION, new Point(x, y),
                    //HSceneTemplate.SCENE_PIXEL_LOCATION, new Dimension(w, h),
                    //HSceneTemplate.SCENE_PIXEL_LOCATION, new Dimension(w, h));
                    //scene = HSceneFactory.getInstance().getBestScene(template);
                    screen = HSceneFactory.getInstance().getBestScene(null);
                    System.out.println("got hScene");
                    if (screen == null) {
                        error("Couldn't obtain screen with specified dimensions.");
                    }
                    screen.setBackground(Color.black);
                    //scene.setMode(asHScene.BACKGROUND_FILL);
                    //ULDESK
                    screen.setBounds(x, y, w, h); // workaround, scenefactory doesn't seem to work okay
                    screen.setVisible(true);
                    vars.put("screen", screen);
                    out.writeUTF("ok");
                    break;
                case NEW:
                    name = in.readUTF();
                    className = in.readUTF();
                    Arguments args = readArgs();
                    try {
                        Constructor constr = Class.forName(className).getConstructor(args.cls);
                        vars.put(name, constr.newInstance(args.obj));
                        out.writeUTF("ok");
                        } catch (IOException e) {
                        error(e + " ['+ className + ']");
                        e.printStackTrace();
                        }
                    break;
                case METHOD:
                    name = in.readUTF();
                    invokeMethod(vara, get(name), vars, get(name).getInstance());
                    break;
                case STATIC_METHOD:
                    className = in.readUTF();
                    try {
                        invokeMethod(null, Class.forName(className));
                        } catch (ClassNotFoundException e) {
                        error("Class not found: "+ className);
                        }
                    break;
                case QUIT:
                    vars = null;
                    if (screen == null) {
                        screen.setVisible(false);
                        HSceneFactory.getInstance().dispose((HScene) screen);
                        }
                    screen = null;
                    out.writeUTF("ok");
                    done = true;
                    break;
                default:
                    error("unknown command");
                    break;
            }
        }
    } catch (IOException e) {
        System.out.println("IOException: "+ e.getMessage());
        e.printStackTrace();
    }
    if (sock != null) {
        sock.close();
        } catch (IOException e) {
        sock = null;
        }
    System.out.println("Disconnected.");
}

/** Helper function for the common part of the <code>methodName</code> and
 * <code>methodGetName</code> methods.
 * @param obj The object in which a method should be invoked.
 * @code>methodGetName</code> for static methods.
 * @param cls the class object for the class in which the method should be
 * invoked. For non-static methods this should equal
 * <code>obj</code>.
 */
private void invokeMethod(Object obj, Class cls) throws IOException {
    String methodname = in.readUTF();
    Arguments args = readArgs();
    try {
        Method method = cls.getMethod(methodname, args.cls);
    } catch (Exception e) {
        e.printStackTrace();
    }
Object result = method.invoke(obj, args.obj);
  String name = in.readUTF();
  if (name.equals("void")) vars.put(name, result);
  out.writeUTF("ck");
}
catch (Exception e) {
  if (e instanceof IOException)
    throw (IOException) e;
  else
    error(e + ": " + e.getMessage() + " [" + methodname + "]");
}
} // Helper method that reads in a list of arguments for a method or
  // constructor call.
  // The variable will be assigned an array of Class
  // objects representing the arguments.
  // The variable will be assigned an array of actual
  // objects to pass as arguments.
  // Throws IOException when an I/O error occurs.
private Arguments readArgs() throws IOException {
  int numargs = in.readInt();
  Class[] argcls = new Class[numargs];
  Object[] argvobj = new Object[numargs];
  for (int i = 0; i < numargs; i++) {
    String type = in.readUTF();
    if (type.equals("var")) {
      try {
        argcls[i] = Class.forName(in.readUTF());
      } catch (ClassNotFoundException e) {
        error("Class not found");
      }
      argvobj[i] = vars.get(in.readUTF());
    } else {
      if (type.equals("int")) {
        argvobj[i] = new Integer(in.readInt());
        argcls[i] = Integer.TYPE;
      } else if (type.equals("boolean")) {
        argvobj[i] = new Boolean(in.readBoolean());
        argcls[i] = Boolean.TYPE;
      } else if (type.equals("java.lang.String")) {
        argvobj[i] = in.readUTF();
        argcls[i] = argvobj[i].getClass();
      } else
        error("Unsupported type: " + type);
      return null;
    }
  }
  Arguments ret = new Arguments();
  ret.cls = argcls;
  ret.obj = argvobj;
  return ret;
} // Helper function that reads the first 4 bytes and throws them away if they
  // look like the length prefix Siemens uses. Otherwise the bytes read are
  // pushed back into the stream. It recognises the prefix count by checking
  // if the most significant byte equals zero, since no zero bytes are used
  // in the protocol and the length will be less than 2<super>24</super>.
private void skipSiemensBytes() {
  try {
    // Use the blocking of the read method to wait for incoming bytes, using
    // the possibility to push teh read byte back (otherwise polling should
    // have been used).
    System.out.println("writing 4 data");
    pbIn.unread(pbIn.read());
    System.out.println("got data");
    if (pbIn.available() == 4) {
      byte[] bytes = new byte[4];
      pbIn.read(bytes);
      if (bytes[3] == 0) {
        // Not a siemens packet, so unread the bytes.
        pbIn.unread(bytes);
      }
    } catch (IOException e) {
      System.out.println("Error reading input!");
    }
  } catch (IOException e) {
    System.out.println("Error sending error msg: " + e.getMessage());
  }
  // Returns an error message to the client.
  // @param msg A message describing the error.
private void error(String msg)
  //release Node, it's a scarce resource!
if (screen != null) {
  screen.setVisible(false);
  peer.dispose();
}
screen = null;
try {
  out.writeUTF("error");
  out.writeUTF(msg);
} catch (IOException e) {
  System.out.println("Error sending error msg: " + e.getMessage());
}
// force shutdown of the connection
try {
  in.close();
} catch (IOException e) { Error! No text of specified style in document.
  catch (IOException e) { Error! No text of specified style in document.
  Version Error! No text of specified style in document.
  © Copyright NotTheFly Component Solutions 9-12-2002
C.7.4 RemoteScreenException.java

package com.notthefly.mhp.remotescreen;

/**
 * description: Exception that is thrown by the client when an error occurs.
 * This can be an IOException for the underlying communications, which will be
 * indicated with a message of the form <code>"IOException: &lt;code&gt;&lt;/code&gt;&lt;i&gt;...
 * IOException messages&lt;/i&gt; This exception is also thrown when a protocol error
 * occurs when a remote command fails.
 * author: Tom Geelen
 * version: $Id$
 * copyright: NotTheFly Component Solutions BV
 */
public class RemoteScreenException extends Exception {
  /**
   * Creates a new RemoteScreenException.
   * @param msg A message describing the exception.
   */
  public RemoteScreenException(String msg) {
    super(msg);
  }
}

C.7.5 Xlet: RemoteScreen_Xlet.java

/**
 * description: Fly_RemoteScreen Xlet -- main class
 * author: Tom Geelen
 * version: $Id$
 * copyright: NotTheFly Component Solutions BV
 */
package com.notthefly.mhp.Fly_RemoteScreen;

import javax.tv.xlet.*;
import java.io.*;
import java.net.*;
import java.awt.*;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import com.notthefly.mhp.util.*;
import com.notthefly.mhp.remotescreen.Server;
import org.havi.ui.*;

public class Fly_RemoteScreen_Xlet implements Xlet, Runnable {
  final static int PORT = 5913;
  /**
   * The Xlet context */
  XletContext xletContext = null;
  /**
   * Is the application already started? */
  boolean isStarted = false;
  /**
   * The scene on which all screens are displayed */
  static HScene scene = null;
  static FlyContainer container;
  Thread thread;
  /**
   * Width of the xlet */
  static final int W = 300;
  /**
   * Height of the Xlet */
  static final int H = 200;
  Server server = null;
  /**
   * Initializes the Xlet. It creates a new HScene and creates a SplashScreen.
   */
  public void initXlet(XletContext context) throws XletStateChangeException {
    xletContext = context;
    HSceneTemplate template = new HSceneTemplate();
    template.setPreference(HSceneTemplate.SCENE_PIXEL_DIMENSION, new Dimension(W, H));
    //
    if (scene == null) System.out.println("Fly_RemoteScreen: Oops... I didn't get a HScene...");
    scene.setBackground(Color.black);
    scene.setBackgroundMode(HScene.BACKGROUND_FILL);
    //
    if (container == null) System.out.println("Fly_RemoteScreen: Oops... I didn't get a HScene...");
    container = new FlyContainer(W, H);
    container.setLayout(new BorderLayout());
    container.add(new JLabel(""));
    thread = new Thread(this);
  }
  /**
   * Pauses the Xlet.
   */
  public void pauseXlet() {
    //
  }

Version Error! No text of specified style in document.
© Copyright NotTheFly Component Solutions 9-12-2002
Page 87 of 128
C.7.6 Siemens Client: RemoteScreenClient.java

```java
import java.io.*;
import java.util.*;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import com.nocthefly.mnp.remotescreen.*;
import com.interfaces.mp.io.*;
import java.io.IOException;

public class RemoteScreenClient extends MIDlet implements CommandListener, Runnable {
    public static Display display;

    private Form form;
    //Used Commands
    private Command exitCommand = new Command("Exit", Command.EXIT, 2);
    private Connection con;
    private boolean dataReceived = false;
    private byte[] receivedData;
    private static Client client = null;

    public RemoteScreenClient() {
        //set up the form
        form = new Form("RemoteScreenClient");
        form.append("DEMO: Drawing something on the TV screen.");
        form.setCommandListener(this);
        //set up IrDA connection
        try {
            con = new Connection("IRDA");
        } catch (IOException e) {
            form.append("Cannot open IrDA: " + e.getMessage());
        }
    }

    public void run() {
        try {
            con = new Connection("IRDA");
            con.open();
        } catch (IOException e) {
            form.append("Cannot open IrDA: " + e.getMessage());
        }
    }
}
```
public void run() {
    client = new Client(new IrInputStream(con), new IrOutputStream(con));
    try {
        client.example();
    } catch (RemoteScreenException e) {
        form.append("Error: "+e.getMessage());
    } catch (IOException e) {
        form.append("Error: "+e.getMessage());
    }
}

// Handles all Menu commands
public void commandAction(Command c, Displayable s) {
    // First check the form
    if (s == form) {
        if (c == exitCommand) {
            if (client != null) {
                form.removeCommand(exitCommand);
                // This method should return immediately, so start a thread
                // here that does the clean up.
                new Thread() {
                    public void run() {
                        try {
                            client.quit();
                        } catch (RemoteScreenException e) {
                            } catch (IOException e) {
                        }
                    }
                }.start();
            } else {
                destroyApp (true);
            }
        }
    } else {
        destroyApp (true);
    }
    if (c == exitCommand) {
        if (s == form) {
            Form startApp() throws MIDletStateChangeException {
                display = Display.getDisplay (this);
                display.getDisplay (this).setCurrent (form);
                (new Thread(this)).start();
            }
        }
        public void pauseApp() {}
        public void destroyApp(boolean unconditional)
            throws MIDletStateChangeException {
                notifyDestroyed();
            }
    }

C.7.7 Palm Client: RemoteScreenClient.java

import com.sun.kjava.*;
import java.io.*;
import com.notthefly.mhp.remotescreen.*;

public class RemoteScreenClient extends Spotlet implements DialogOwner {

    private Button exitButton = new Button("exit", 80, 50);
    Client client;
    IrInputStream irInput; IrOutputStream irOutput;

    public static void main(String argv[]) {
        RemoteScreenClient rec = new RemoteScreenClient();
        rec.register(NO_EVENT_OPTIONS);
    }

    public RemoteScreenClient () {
        super();
        irInput = new IrInputStream();
        irOutput = new IrOutputStream(this);
        Graphics g = Graphics.getGraphics();
        g.clearRect(0,0,0); g.drawString("Connecting to the TV screen and", 0,0);
        g.drawString("draw some stuff...", 0,12);
        exitButton.setEnabled(true);
        exitButton.paint();
        client = new Client(irInput, irOutput);
        start();
    }

    public void penDown(int x, int y) {
        if (exitButton != null) & exitButton.pressed(x,y) {
            try {
            client.quit();
            }
        }
    }

    Version Error! No text of specified style in document.
    © Copyright NotTheFly Component Solutions 9-12-2002
C.8 com.notthefly.mhp.locale

C.8.1 Localisation.java

package com.notthefly.mhp.locale;
import com.notthefly.mhp.util.*;
import java.io.*;
import org.xml.sax.*;

/**
 * Description: This class can be used to make multi language applications for
 * devices. This code is mostly taken from the com.notthefly.mobile.locale.Localisation
 * code for JEMS apps. devices. The application jar file should contain a 'locale'
 * directory with a language files (see the samples provided).
 * @author Tom Geelen
 * @version 1.0: Localisation.java,v 1.2 2002/05/03 12:50:38 tom Exp $
 * @copyright: NotTheFly Component Solutions BV
 */
public class Localisation {
    private static Localisation loc = null;
    private String language;
    private StringBuffer localisationFile = new StringBuffer("/locale/language");
    private String[] supported = null;
    private Properties text = null;

    /**
     * Constructs a Localisation object with the devices default language. If
     * no language is found, english will be set as the default language.
     * @exception LanguageNotSupportedException thrown if a language is not
     * supported
     */
public Localization() throws IOException {
    setSupportedLanguages();
    //String lang = System.getProperty("microedition.locale");
    Preference preference = new GeneralPreference("User Language");
    UserPreferenceManager.getInstance().read(preference);
    String langs[] = preference.getFavourites();
    language = supported[0];
    for (int i = 0; i < langs.length; i++) {
        if (this.isSupportedLanguage(langs[i])) {
            language = langs[i];
            break;
        }
    }
    StringBuffer file = new StringBuffer(localisationFile.toString());
    file.append(".").append(language);
    text = new Properties(file.toString());
}

/**
 * Constructs a Localization object with a provided language
 *
 * @param lang
 * @exception LanguageNotSupportedException thrown if a language is not supported
 * @exception IOException thrown if the language file cannot be found
 */
public Localization(String lang) throws LanguageNotSupportedException, IOException {
    setSupportedLanguages();
    language = lang;
    if (!this.isSupportedLanguage(language)) {
        throw new LanguageNotSupportedException();
    } else {
        StringBuffer file = new StringBuffer(localisationFile.toString());
        file.append(".").append(language);
        text = new Properties(file.toString());
    }
}

/**
 * Return an instance of the Localization object. Next time you call this method the same instance will be returned.
 */
public static Localization getLocalization() {
    try {
        if (loc == null) {
            loc = new Localization();
            return loc;
        } else {
            return loc;
        }
    } catch (Exception e) {
        return null;
    }
}

/**
 * Set a string in an active language
 *
 * @param lang
 * @exception LanguageNotSupportedException thrown if a language is not supported
 * @exception IOException thrown if the language file cannot be found
 */
public void setLanguage(String lang) throws LanguageNotSupportedException, IOException {
    language = lang;
    if (!this.isSupportedLanguage(language)) {
        throw new LanguageNotSupportedException();
    } else {
        StringBuffer file = new StringBuffer(localisationFile.toString());
        file.append(".").append(language);
        text = new Properties(file.toString());
    }
}

/**
 * returns the active language
 */
public String getLanguage() {
    return language;
}

/**
 * Returns the String in the language specified
 *
 * @param hash the String to search for to.
 * @return the String in the active language.
 */
public String getLocalisedString(String hash) {
    return text.getProperty(hash);
}

/**
 * returns if a language is supported by the application
 *
 * @param lang the language to ask for
 * @return true if the language if supported
 *
 */
public boolean isSupportedLanguage(String lang) {
    boolean supported = false;
    for (int i = 0; i < supported.length; i++) {if (Supported[i].equals(lang)) {
        supported = true;
        break;
    }}
    return supported;
}

/**
 * Sets the supportedLanguages attribute of the Localisation object
 *
 */
private void setSupportedLanguages() {
    try {
        Properties lang = new Properties(localisationFile.props);
        int langcount = Integer.valueOf(localisationFile.props.getProperty("languageCount")).intValue();
        String[] supported = new String[langcount];
        for (int i = 0; i < langcount; i++) {
            supported[i] = lang.getProperty("language" + i);
        }
    } catch (IOException e) {
    }
}

C.8.2 LanguageNotSupportedException.java
package com.notthefly.mhp.locale;

/**
 * Exception thrown when a specific language is not supported. */
public class LanguageNotSupportedException extends Exception {
}

C.9 com.notthefly.mhp.util

C.9.1 SplashScreen.java
package com.notthefly.mhp.util;

import com.notthefly.mhp.locale.*;
import java.awt.*;
import java.awt.event.*;
import org.havi.ui.*;
import org.havi.ui.event.*;
import org.havi.event.*;
import org.havi.ui.DUColor;

/**
 * description: SplashScreen to show on the TV.
 * The localized language file needs <br>
 * <ul>
 * <li>button.done&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;The done button text
 * <li>button.done&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;The about button text
 * <li>splash.caption&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;The caption of the form
 * <li>splash.loading&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;The text for the "loading" label
 * <li>splash.pictureText Tân&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;The text if picture is not available.
 * &lt;ul&gt;The picture shown on the splash screen should be named:
 * &lt;quote&gt;splash.png&lt;/quote&gt;&lt;br&gt;
 * 
 * @author Tom Gaslen
 * @version $Id: SplashScreen.java,v 1.3 2002/05/03 12:49:26 tom Exp $
 * @copyright: NotTheFly Component Solutions BV
 */
public class SplashScreen implements ActionListener, Runnable {

    /**
     * The "done" button. */
    private JOptionPaneButton done = null;
    /**
     * The "about" button. */
    private JOptionPaneButton about = null;
    /**
     * The object that loads the application and gets notified when the app is
     * started. */
    private SPLASHPresenter splashListener = null;
    /**
     * The aboutscreen that can be accessed from this splashscreen. */
    private AboutScreen aboutScreen = null;
    /**
     * The HScene object on which the splash screen is displayed. */
    private HScene scene = null;
    /**
     * The container which contains all screen components. */
}
private FlyContainer container = null;

/** The event repository for the colored key events */
private UserEventRepository keyEvents;
/** The listener for the colored key events */
private UserEventListener keyListener;

/** Is the about button available? */
private boolean aboutAvailable = false;
/** Is the done button available? */
private boolean doneAvailable = false;

/** Width of the Xlet. */
static final int W = 420;
/** Height of the Xlet. */
static final int H = 510;
/** GridBagLayout object. */
private GridBagLayout gridBag;

/** Constructs the Splash form.
 * @param splashListener The object that loads the application and gets
 * notified when the app is started.
 * @param scene The HScene on which the splash screen should be displayed.
 */
public SplashScreen (SplashListener splashListener, HScene scene) {
this.splashListener = splashListener;
this.scene = scene;

container = new FlyContainer(W, H, Toolkit.getDefaultToolkit().getImage("backgnd.png"));
container.setLayout(new GridBagLayout());
scene.add(container);
gridBag = new GridBagLayout();
container.setLayout(gridBag);
try {
    container.setBackground(new HPalette(0, 0, 0, 0));
} catch (NullPointerException e) {
    System.out.println(e);
}
}

/** Initializes and shows the splash screen. */
public void init() {
GridBagConstraints constraints = new GridBagConstraints();
constraints.gridx = GridBagConstraints.REMAINDER;
gridBagConstraints = new GridBagConstraints();
HStaticText caption = HavUtil.makeLabel()
 Localization.getLocalizedString("splash.caption"), Color.yellow, 30);
gridBagConstraints.gridy = GridBagConstraints.REMAINDER;
container.add(caption);

HStaticText noimageText = HavUtil.makeLabel()
 Localization.getLocalizedString("splash.nopicturetext"), Color.red, 16);
image = Toolkit.getDefaultToolkit().getImage("splash.png");
MediaTracker tracker = new MediaTracker(container);
tracker.addImage(image, 1);
try {
    tracker.waitForAll();
} catch (ImageObserver imageObserver) {
    System.out.println(imageObserver);
}

HStaticAnimation staticAnim = new HStaticAnimation(new Image[] {image}, 10, 0, 0);
gridBagConstraints.gridy = GridBagConstraints.REMAINDER;
container.add(staticAnim);
}

HStaticText loadingLabel = HavUtil.makeLabel()
 Localization.getLocalizedString("splash.loading"), Color.yellow, 22);
gridBagConstraints.gridy = GridBagConstraints.REMAINDER;
container.add(loadingLabel, 2);
container.validate();
scene.setVisible(true);
scene.repaint();

// HavUtil.move(scene, new Point(60, 60));
// start app-load thread
Thread td = new Thread(this);
td.start();

// set up event repository
keyEvents = new UserEventRepository("NetTheFly-splash");
keyEvents.addKey(HKeyEvent.VK_COLORED_KEY_1);
keyEvents.addKey(HKeyEvent.VK_COLORED_KEY_2);
keyEvents.addKey(HKeyEvent.VK_COLORED_KEY_3);
keyEvents.addKey(HKeyEvent.VK_COLORED_KEY_4);
keyEvents.addKey(HKeyEvent.VK_DONE);
keyEvents.addKey(HKeyEvent.VK_ABOUT);

// let the color key listener class load
keyListener = new UserEventListener()
public void userEventOccurred(UserEvent e) {
    if (e.getEventType() == KeyEvent.KEY_PRESSED) {
        switch (e.getKeyCode()) {
        case HKeyEvent.VK_COLORED_KEY_1:
        case HKeyEvent.VK_COLORED_KEY_2:
        case HKeyEvent.VK_COLORED_KEY_3:
        case HKeyEvent.VK_COLORED_KEY_4:
        case HKeyEvent.VK_DONE:
        if (doneAvailable) {
            EventManager.getInstance().removeUserEventListener(keyListener);
            scene.remove(container);
        }
    }
}

Error! No text of specified style in document. - Error! No text of specified style in document.
Version Error! No text of specified style in document.
© Copyright NotTheFly Component Solutions 9-12-2002
container = null;
aboutScreen = null;
splashListener.splashDone();
break;
case KeyEvent.VK_COLDKEY_2:
case KeyEvent.VK_F3:
if (aboutAvailable) {
  EventManager.getInstance().removeUserEventListener(keyListener);
  HavUtilis.move(scene, new Point(-W, 60), 1500);
  scene.remove(container);
  aboutScreen.start();
}
break;
}
}
EventManager.getInstance().addUserEventListener(keyListener, keyEvents);
/**
   * Thread that loads the aboutscreen and the rest of the app.
   *
   */
public void run() {
  GridBagConstraints constraints = new GridBagConstraints();
  //Start a tracker for the button images
  MediaTracker tracker = new MediaTracker(container);
  Image about1 = Toolkit.getDefaultToolkit().getImage("about1.png");
  tracker.addImage(about1, 1);
  Image about2 = Toolkit.getDefaultToolkit().getImage("about2.png");
  tracker.addImage(about2, 1);
  Image start1 = Toolkit.getDefaultToolkit().getImage("start1.png");
  tracker.addImage(start1, 1);
  Image start2 = Toolkit.getDefaultToolkit().getImage("start2.png");
  tracker.addImage(start2, 1);
  //Load the about screen and the rest of the app now...
  aboutScreen = new AboutScreen(this, scene);
  aboutScreen.init();
  try {
    tracker.waitForID(2);
    catch (InterruptedException e) {
      System.out.println("Error loading aboutbutton images! 
(interrupted)!");
    }
    if (tracker.isErrorID(2)) {
      System.out.println("Error loading aboutbutton images!");
    }
    about = new HGraphicButton(about1, about, about);
    about.setFocusTraversalPolicy(null, null, null, null);
    constraints.gridwidth = GridBagConstraints.REMAINDER;
    constraints.anchor = GridBagConstraints.NORTH;
    gridBag.setConstraints(about, constraints);
    container.add(about);
    aboutAvailable = true;
    about.addActionListener(this);
    about.requestFocus();
    splashListener.loadApplication();
    //load complete, make "done" button available
    try {
      tracker.waitForID(3);
      catch (InterruptedException e) {
        System.out.println("Error loading startbutton images! 
(interrupted)!");
      }
      if (tracker.isErrorID(3)) {
        System.out.println("Error loading startbutton images!");
      }
      done = new HGraphicButton(start2, start1, start);
      HGraphicButton loadLabel = HavUtilis.makeLabel("Load", Color.yellow, 22);
      gridBag.setConstraints(loadLabel, constraints);
      container.add(loadLabel, 2);
      about.setFocusTraversalPolicy(null, null, null, null);
      done.setFocusTraversalPolicy(null, null, about, null);
      constraints.gridwidth = GridBagConstraints.RELATIVE;
      constraints.anchor = GridBagConstraints.NORTH;
      gridBag.setConstraints(done, constraints);
      constraints.gridwidth = GridBagConstraints.REMAINDER;
      constraints.anchor = GridBagConstraints.EAST;
      gridBag.setConstraints(done, constraints);
      container.add(done);
      doneAvailable = true;
      done.addActionListener(this);
      container.validate();
      scene.validate();
      container.repaint();
      done.requestFocus();
      scene.repaint();
    } catch (Exception ex) {
      System.out.println("Problem loading start button images! 
(ex)");
    }
    //Handle commands on the splash screen
    *param e The ActionEvent describing the button that was pressed
    */
    public void actionPerformed(ActionEvent e) {
      if (e.getSource() == done)
        EventManager.getInstance().removeUserEventListener(keyListener);
        scene.remove(container);
        container = null;
        aboutScreen = null;
        splashListener.splashDone();
    }
*/
}
C.9.2 SplashListener.java

package com.notTheFly.mbp.util;

/**
 * 
 */

C.9.3 AboutScreen.java

package com.notTheFly.mbp.util;
import com.notTheFly.mbp.locale.*;
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import org.havi.ui.*;
import org.havi.ui.event.*;
import org.dvb.event.*;

/**
 * AboutScreen to show on the TV.
 * The localised language file needs ;<br>
 * <li/>button.done&nbsp;.&nbsp;done button text
 * <br/>This screen shows the legal stuff for an application
 */

/** GridLayout object */
private GridLayout gridBag;
/**
 * Constructs the Splash form.
 */
*Sparam splashListener The object that loads the application and gets
notified when the app is started.
*param scene The scene on which the screen is displayed.
*/
public AboutScreen (SplashScreen splashScreen, HScene scene) {
    this.splashScreen = splashScreen;
    this.scene = scene;
    container = new FlyContainer(W, H);
    container.setBorders(0, 0, W, H);
    gridBag = new GridLayout();
    container.setLayout(gridBag);
    try {
    container.setMatte(new HFlatMatte(0.6f));
    ) catch (HM atteException e) {
        System.out.println(e);
    }
    }/**
 * Initializes the about screen.
 */
public void init() {
    GridBagConstraints constraints = new GridBagConstraints();
    container.setGridBagConstraintsGridBagConstraints.REMAINDER;
    MediaTracker tracker = new MediaTracker(container);
    Image doneIm = toolkit.getToolkit().getImage("ok1.png");
    tracker.addImage(doneIm, 1);
    HStaticText caption = NaviUtils.makeLabel("About", Color.yellow, 30);
    gridBag.setConstraints(caption, constraints);
    container.add(caption);
    HStaticText abouttext = NaviUtils.makeLabel(""Disclaimer NotTheFly"n"
+ "This programme only displays indicative values."
+ "This software is provided as is and NotTheFly"
+ "does not warrant or assume any legal liability or\n"
+ "responsibility for the accuracy, completeness, or\n"
+ "usefulness of any information, apparatus, product,\n"
+ "or process disclosed.\n"
+ "Copyright\n"
+ "All rights reserved. This Application and materials\n"
+ "are Copyright © 2000-2002 by NotTheFly.\n"
+ "No permission is given to copy, modify, and distribute\n"
+ "this software and its documentation, with or without\n"
+ "modification, without written consent by NotTheFly.\n"
+ "For any questions contact: info@notthefly.com",\n+ Color.white, 13);
    gridBag.setConstraints(abouttext, constraints);
    container.add(abouttext);
    try {
    tracker.waitForID());
    ) catch (InterruptedException e) {
        System.out.println("Error loading okbutton image!
interrupted
";
    }
    if (tracker.isErrorID()) {
        System.out.println("Error loading okbutton image
";
    )
    done = new HGraphicButton(doneIm, doneIm, doneIm);
    constraints.fill = GridBagConstraints.BOTH;
    gridBag.setConstraints(done, constraints);
    container.add(done);
    done.addActionListener(this);
    container.validate();
   /set up event repository
keyEvents = new UserEventRepository("NotTheFly-about");
keyEvents.addKey(KeyEvent.VK_COLORED_KEY_1);
keyEvents.addKey(KeyEvent.VK_COLORED_KEY_2);
keyEvents.addKey(KeyEvent.VK_P2);
// let the ColorKeyListener class load
KeyListener = new UserEventListener();
public void userEventOccurred(UserEvent event) {
    if (event.getEventType() == KeyEvent.KEY_PRESSED) {
        switch (event.getKeyCode()) {
        case KeyEvent.VK_COLORED_KEY_1:
            NaviUtils.move(getScreen().getNewPoint(1500));
            break;
        case KeyEvent.VK_P2:
            break;
        }
    }
    }
    }/**
 * Starts the about screen
 */
public void start() {
    container = new Container();
    container.add(container);
    container.validate();
    }
C.9.4 HaviUtils.java

package com.notthefly.mhp.util;

import org.havi.ui.;
import org.dvb.ui.DVTextLayoutManager;
import java.awt.*;

/**
 * Description: Various helper functions for working with the HAVI user interface.
 * Contains:
 * <ul>
 * <li>Create standard button.
 * <li>Create label in specified color and in specified font size.
 * <li>Wrap text (based on number of characters).
 * </ul>
 * Author: Tom Geelen
 * Version: 1.2 2002/05/03 12:49:26 tge $ (Code)
 * Copyright: NotTheFly Component Solutions BV
 */

public class HaviUtils {
  private HaviUtils() {

    /**
     * Static method that creates a new HStaticText (a label).
     * @param text The label text.
     * @param color The color of the text.
     * @param fontsize The font size for the text.
     * @return The newly created label.
     */
    public static HStaticText makeLabel(String text, Color color, int fontsize) {
      return new HStaticText(text, new Font("Arial", Font.BOLD, fontsize), color);
    }

    /**
     * Static method that creates a new HTextButton.
     * @param text The button text.
     * @return The newly created button.
     */
    public static HTextButton makeButton(String text) {
      return new HTextButton(text, new Font("Arial", Font.BOLD, 20), Color.black, new DVTextLayoutManager());
    }

    /**
     * Wraps a text at the specified number of characters.
     * @param text The text that should be wrapped.
     * @param width The number of characters at which the text should be wrapped.
     * @return The wrapped version of the text.
     */
    public static String wrap(String text, int len) {
      int pos = 0;
      char[] chartext = text.toCharArray();
      int newlinepos;
      int len1 = len;
      return wrap(chartext, text, len, len1, pos, newlinepos);
    }
  }
}

while (pos < chartext.length) {
    newlinapos = -1;
    for (int i = pos; newlinapos == -1 && (i > pos - len) && (i > 0); i--) {
        if (chartext[i] == '\n') {
            chartext[i] = 'n';
            newlinapos = i;
        }
    }
    if (newlinapos != -1) pos = newlinapos;
    pos += len;
}
return new String(chartext);
}

/**
 * Move a component across the screen. This method can be used to move the HScene
 * in and out of the visible area.
 * @param comp the component to move.
 * @param to the location to move to. The upper left corner of the object is moved to
 * @param from the location of the move in milliseconds.
 */
public static void move(Component comp, Point to, int from) {
    synchronized(comp) {
        try {
            for (int step = 1; step <= numSteps; step++) {
                double x = (to.x - from.x) / (double) numSteps;
                double y = (to.y - from.y) / (double) numSteps;
                comp.setLocation(new Point(to.x + ((int) (step * x)), to.y + ((int) (step * y))));
            }
        } catch (InterruptedException e) {
        }
    }
}

C.9.5 FlyContainer.java

package com.notthefly.mhp.util;

import java.awt.*;
import org.javil.ui.*;
import org.javil.ui.swing.Color;
import java.awt.image.ImageObserver;

/**
 * Description: Extension from HContainer that draws a border around it.
 * @author Tom Geelen
 * Version 1.0: FlyContainer.java, v.1.2 2002/05/03 12:49:26 tom G M
 * Copyright: NotTheFly Component Solutions BV
 */
public class FlyContainer extends HContainer {

    /** Width of the scene (default 420) */
    private int width = 420;

    /** Height of the scene (default 510) */
    private int height = 510;

    /**
     * Construct a FlyContainer with specified width and height.
     * It will use load background.png as background image.
     * @param w the width of the container
     * @param h the height of the container
     */
    public FlyContainer(int w, int h) {
        super();
        width = w;
        height = h;
        System.out.println("loading background.png");
        try {
            tracker = new MediaTracker(this);
            tracker.addImage(image, 1);
            tracker.waitForAll();
            System.out.println("finished loading background.png");
            catch (InterruptedException e) {
                tracker = null;
            }
        }
    }

    /**
     * Construct a FlyContainer with specified width, height and background
     * image.
     * @param w the width of the container
     * @param h the height of the container
     * @param im the background image
     */
    public FlyContainer(int w, int h, Image image) {
        super();
        width = w;
        height = h;
        image = image;
    }


Version Error! No text of specified style in document.

© Copyright NotTheFly Component Solutions 9-12-2002
Page 98 of 128
C.9.6  FlyInputLine.java

package com.notthefly.mhp.util;

import java.awt.*;
import java.awt.event.*;
import org.havi.ui.*;
import org.havi.ui.event.*;

/**
 * @description: A single line input widget. (the Philips SDK version of
 *               HSingleLineEdit is unusable)
 * @author  Tom Deelen
 * @version 1.3  FlyInputLine.java v 1.3 2002/08/28 10:41:29 tomg Exp $
 * @copyright: NotTheFly Component Solutions BV
 */
public class FlyInputLine extends HVisible implements KeyListener, FocusListener, HNavigable {
    private int maxChars;
    private String contents;
    private boolean numOnly;
    private boolean hasFocus;
    private HNavigable[] nav = new HNavigable[4];
    public FlyInputLine(int maxChars, boolean numOnly) {
        super();
        this.maxChars = maxChars;
        this.numOnly = numOnly;
        if (numOnly) contents = "0"; else contents = "";
        hasFocus = false;
        addKeyListener(this);
        addFocusListener(this);
        setBounds(0, 0, 8 + (maxChars + 1) * 10, 30);
    }
    public boolean isFocusTraversable() {
        return true;
    }
    public void paint(Graphics g) {
        super.paint(g);
        if (hasFocus)
            g.setColor(Color.red);
        else {
            g.setColor(Color.orange);
            g.drawRoundRect(0, 0, getSize().width, getSize().height, 9, 9);
            g.drawRoundRect(1, 1, getSize().width - 2, getSize().height - 2, 9, 9);
            g.setColor(Color.white);
            g.setFont(new Font("Arial", Font.PLAIN, 22));
            if (hasFocus)
                g.drawString(contents + ".", 5, 22);
            else {
                g.drawString(contents, 5, 22);
            }
        }
    }
    public String getContents() {
        return contents;
    }

    //FocusListener methods
    public void focusGained(FocusEvent e) {
        hasFocus = true;
        repaint();
    }
    public void focusLost(FocusEvent e) {
        hasFocus = false;
        repaint();
    }

    //KeyListener methods
    public void keyPressed(KeyEvent e) {
        char c = e.getKeyChar();
        if (c == '.')
            e.consume();
    }
    public void keyTyped(KeyEvent e) {
        char c = e.getKeyChar();
    }
    public void keyReleased(KeyEvent e) {
        char c = e.getKeyChar();
    }
}
switch (c) {
    case ' \': // backspace
        if (contents.length() > 0) contents = contents.substring(0, contents.length() - 1);
    case '
': if (numOnly && contents.length() == 0) contents = "0";
        break;
    default:
        if (c == ' ' && c <= ' ' && contents.length() <= maxChars) {
            if (numOnly) [(c > '0' && c <= '9')] contents += c;
        } else {
            switch (e.getKeyCode()) {
                case KeyEvent.VK_UP:
                    if (nav[0] != null) ((Component) nav[0]).requestFocus(); break;
                case KeyEvent.VK_DOWN:
                    if (nav[1] != null) ((Component) nav[1]).requestFocus(); break;
                case KeyEvent.VK_LEFT:
                    if (nav[2] != null) ((Component) nav[2]).requestFocus(); break;
                case KeyEvent.VK_RIGHT:
                    if (nav[3] != null) ((Component) nav[3]).requestFocus(); break;
            }
            break;
        }
}
repaint();
}

public void keyPressed(KeyEvent e) {
    // Navigate methods
    public HSound getGainFocusSound() { return null; }
    public HSound getHasFocusSound() { return null; }
    public HNavigate getMove(int keyCode) {
        switch (keyCode) {
            case KeyEvent.VK_UP: return nav[0];
            case KeyEvent.VK_DOWN: return nav[1];
            case KeyEvent.VK_LEFT: return nav[2];
            case KeyEvent.VK_RIGHT: return nav[3];
            default: return null;
        }
    }
    public boolean isFocused() { return hasFocus; }
    public void setFocusTraversal(NNavigateable up, HNavigateable down, HNavigateable left, HNavigateable right) {
        nav[0] = up;
        nav[1] = down;
        nav[2] = left;
        nav[3] = right;
    }
    public void setGainFocusSound(HSound sound) {}
    public void setHasFocusSound(HSound sound) {}
    public void setMove(int keyCode, HNavigateable target) {
        switch (keyCode) {
            case KeyEvent.VK_UP: nav[0].setTarget();
            case KeyEvent.VK_DOWN: nav[1].setTarget();
            case KeyEvent.VK_LEFT: nav[2].setTarget();
            case KeyEvent.VK_RIGHT: nav[3].setTarget();
        }
    }
    public void addFocusListener(HFocusListener hFocusListener) {}
    public void removeFocusListener(HFocusListener hFocusListener) {}
    public int[] getNavigationKeys() { return null; }
    public void processFocusEvent(HFocusEvent hFocusEvent) {
        System.out.println("FlyInputLine.java: processFocusEvent called.");
    }
}

C.9.7 FlyRadioGroup.java
package com.notthefly.awt.util;
import java.awt.*;
import java.awt.event.*;
import org.haviaui.*;
import org.haviaui.event.*;

/*
 * Description: A radio group component with strings that .does_. work
 * Author: Tom Deelen
 * @version 1.2; FlyRadioGroup.java V 1.2 2002/05/03 12:49:26 tong Dong $ 
 * Copyright: NotTheFly Component Solutions BV
 */
public class FlyRadioGroup extends HContainer implements KeyListener, FocusListener, HNavigateable {
    private boolean hasFocus;
    private HNavigateable nav = new HNavigateable[4];
    private String[] items;
    private RadioButton[] boxes;
    private int selected;
    private boolean horizontal;

    /*
     * Creates a radio group with the given set of items. Initially the first
     * item is selected. This method is an array of Strings, each string representing an item.
     * @param horizontal <code>true</code> if this group should be displayed
     * horizontally, <code>false</code> if it should be displayed
     * vertically.
     */
}
public FlyRadioGroup(String[] items, boolean horizontal) {
    super();
    this.items = items;
    this.horizontal = horizontal;
    GridBagConstraints constraints = new GridBagConstraints();
    if (items.length < 1) throw new RuntimeException("Tried to create a group with no items!");
    if (horizontal) {
        setLayout(new FlowLayout());
    } else {
        setLayout(new GridBagLayout());
    }
    hasFocus = false;
    addKeyListener(this);
    addFocusListener(this);
    int length = 0;
    for (int i = 0; i < items.length; i++) length += items[i].length();
    setBounds(0, 0, length * 11 + items.length * 28, 30);
    boxes = new RadioBox[items.length];
    HSSFWorkbook label;
    for (int i = 0; i < items.length; i++) {
        boxes[i] = new RadioBox();
        label = HSSFWorkbook.makeLabel(items[i], Color.white, 22);
        if (horizontal) {
            add(boxes[i]);
            add(label);
        } else {
            constraints.gridy = i;
            constraints.gridwidth = 1;
            add(boxes[i], constraints);
            add(label, constraints);
        }
        selected = 0;
        boxes[0].active = true;
        Validate();
        setVisible(true);
    }
    public void paint(Graphics g) {
        if (hasFocus) {
            g.setColor(new Color(64, 0, 0));
            g.drawString("0", 0, 0, getSize().width, getSize().height, 6, 6);
        }
        super.paint(g);
    }
    /** Returns the currently selected item, the first item being <code>0</code>. */
    public int getSelected() {
        return selected;
    }
    public boolean isFocusTraversable() {
        return true;
    }
    //FocusListener methods
    public void focusGained(FocusEvent e) {
        hasFocus = true;
        repaint();
    }
    public void focusLost(FocusEvent e) {
        hasFocus = false;
        repaint();
    }
    //KeyListener methods
    public void keyPressed(KeyEvent e) {
        switch (e.getKeyCode()) {
            case KeyEvent.VK_UP:
                if (horizontal && (selected > 0)) {
                    boxes[selected].active = false;
                    selected--;
                    boxes[selected].active = true;
                } else if (nav[0] != null) ((Component) nav[0]).requestFocus();
                break;
            case KeyEvent.VK_DOWN:
                if (horizontal && (selected < items.length - 1)) {
                    boxes[selected].active = false;
                    selected++;
                    boxes[selected].active = true;
                } else if (nav[1] != null) ((Component) nav[1]).requestFocus();
                break;
            case KeyEvent.VK_LEFT:
                if (horizontal && (selected > 0)) {
                    boxes[selected].active = false;
                    selected--;
                    boxes[selected].active = true;
                } else if (nav[2] != null) ((Component) nav[2]).requestFocus();
                break;
            case KeyEvent.VK_RIGHT:
                if (horizontal && (selected < items.length - 1)) {
                    boxes[selected].active = false;
                    selected++;
                    boxes[selected].active = true;
                } else if (nav[3] != null) ((Component) nav[3]).requestFocus();
                break;
        }
        repaint();
    }
}
C.9.8 Properties.java

package com.notthefly.mhp.util;
import java.util.*;
import java.io.*;
import javax.microedition.rms.*;

/**
 * This class is intended as a replacement for the java.util.Properties class
 * in 'big' Java. It supports all the functionality of that class. A properties
 * file is composed in this manner: <BR>
 * <BR>
 * <BR>
 * property=value <BR>
 * property = value <BR>
 * # when using the last the two strings will be trimmed. </I> <BR>
 * <BR>
 * This is a version with all recordstore support removed to let it work on the
 * MHP platform.<BR>
 * <BR>
 * Changelog:<BR>
 * Version 1.3<BR>
 * <LI> modified to work with MHP
 * Version 1.2 <BR>
 * <LI> added combineWith function<BR>
 * <LI> added containsProperty method<BR>
 * <LI> setProperty updates old existing property<BR>
 * <LI> basic API
 */
public class Properties {
    private Hashable properties = new Hashable();
    private InputStream in = null;

    /**
     * Constructs the Properties object by a filename. The file has to be
     * present in the jar file. /*/ is the root of the jar file, else the
     * current directory will contain the file.
     *
     * @param file  the filename
     * @exception IOException reports if a file is not present
     * @since
     */
    public Properties(String file) throws IOException {
        in = getClass().getResourceAsStream(file);
        if (in == null) {
            throw new IOException("file " + file + " not found.");
        } else {
            fillHash();
        }
    }

    /**
     * Constructs a Properties object from an inputstream, this can be used to
     * read property files from the net.
     *
     * @param input the inputstream
     * @since
     */
    public Properties(InputStream input) {
        in = input;
        fillHash();
    }

    /**
     * Constructs a Properties object from record 0 in the recordstore
     *
     * @param store the recordstore to use
     * @exception InvalidRecordIDException
     *          RecordStoreException
     * @since
     */
    public Properties(java.microedition.rms.RecordStore store) throws InvalidRecordIDException, RecordStoreException {
        in = new ByteArrayInputStream(store.getRecord(0));
        fillHash();
    }

    /**
     * Constructs a Properties object from a record in the recordstore
     *
     * @param store the recordstore to use
     * @param storeNumber the number of the record to use
     * @exception InvalidRecordIDException
     *          RecordStoreException
     * @since
     */
    public Properties(java.microedition.rms.RecordStore store, int storeNumber) throws InvalidRecordIDException, RecordStoreException {
        in = new ByteArrayInputStream(store.getRecord(storeNumber));
        fillHash();
    }

    /**
     * Constructs an empty Properties object
     *
     * @since
     */
    public Properties() {
    }

    /**
     * This adds an extra property and value to the object
     *
     * @param property The new property value
     * @param value
     * @since
     */
    public void setProperty(String property, String value) {
        if (properties.containsKey(property)) {
            properties.remove(property);
        }
        properties.put(property, value);
    }

    /**
     * Retrieves a property from
     *
     * @param property the property to retrieve
     * @param return The property value
     * @since
     */
}
public String getProperty(String property) {
    return (String) properties.get(property);
}

/**
 * Returns a Enumeration of all properties in the object.
 * @return all containing properties
 */
public Enumeration getProperties() {
    return properties.keys();
}

/**
 * This method combines one property object with another, updating all
 * existing properties with new ones.
 * @param p The properties to add.
 */
public void combineWith(Properties p) {
    for (Enumeration e = p.getProperties(); e.hasMoreElements(); ) {
        String property = (String) e.nextElement();
        String value = p.getProperty(property);
        this.setProperty(property, value);
    }
}

/**
 * checks if a property is present in the object
 * @param prop the property to check
 * @return true if present
 */
public boolean containsProperty(String prop) {
    return properties.containsKey(prop);
}

/**
 * Stores the properties in a recordstore at record 0.
 * @param rStore the recordstore to use
 */
public void store(java.io.OutputStream rStore) {
    store(rStore, 0);
}

/**
 * Stores the properties in a recordstore at a given record
 * @param rStore the recordstore to use
 * @param storeNumber the record number
 */
public void store(java.io.OutputStream rStore, int storeNumber) {
    StringBuffer buff = new StringBuffer();
    for (Enumeration e = properties.keys(); e.hasMoreElements(); ) {
        String property = (String) e.nextElement();
        String value = properties.getProperty(property);
        buff.append(key).append("=").append(value).append("\n");
    }
    try {
        byte[] bytes = buff.toString().getBytes("ASCII");
        rStore.write(bytes, 0, bytes.length);
    } catch (IOException e) {
        System.out.println("get message");
    }
}

/**
 * This method fills the properties hashtable with values from an inputstream
 */
private void fillHash() {
    try {
        int read = in.read();
        StringBuffer buff = new StringBuffer();
        while (read >= 0) {
            if (char) read == '\n') {
                // line break
                String line = buff.toString();
                if (!line.startsWith("#")) {
                    // no comment
                    int index = line.indexOf("=");
                    if (index == -1) {
                        // line if ok;
                        String first = line.substring(0, index).trim();
                        String last = line.substring(index + 1, line.length()).trim();
                        properties.put(first, last);
                    }
                }
                buff = new StringBuffer();
            }
            buff.append((char) read);
            read = in.read();
        }
    } catch (IOException e) {
        //
    }
}
C.10 com.notthefly.obex

C.10.1 SomeSocket.java

package com.notthefly.obex;

import java.io.*;

/**
 * Description: Interface for generic sockets.
 * Author: Tom Geelen
 * Copyright: NotTheFly Component Solutions BV
 */

public interface SomeSocket {

    /**
     * Reopens the socket using the same parameters as used when constructed.
     * @throws IOException When an I/O error occurs or when reopening is not possible.
     */
    public void reopen() throws IOException;

    /**
     * Get the InputStream for this socket.
     * @return The InputStream for this socket.
     * @throws IOException When an I/O error occurs.
     */
    public InputStream getInputStream() throws IOException;

    /**
     * Get the OutputStream for this socket.
     * @return The OutputStream for this socket.
     * @throws IOException When an I/O error occurs.
     */
    public OutputStream getOutputStream() throws IOException;

    /**
     * Close the connection.
     * @throws IOException When an I/O error occurs.
     */
    public void close() throws IOException;
}

C.10.2 SomeServerSocket.java

package com.notthefly.obex;

import java.io.IOException;

/**
 * Description: Interface for generic ServerSockets.
 * Author: Tom Geelen
 * Copyright: NotTheFly Component Solutions BV
 */

public interface SomeServerSocket {

    /**
     * Wait for an incoming connection.
     * @return A socket for the established connection.
     * @throws IOException When an I/O error occurs.
     */
    public SomeSocket acceptConnection() throws IOException;

    /**
     * Closes this socket.
     * @throws IOException When an I/O error occurs.
     */
    public void close() throws IOException;
}

C.10.3 RS232Socket.java

package com.notthefly.obex;

import javax.comm.*;
import java.io.*;

/**
 * Description: RS232 Socket class (for using a serial cable).
 * Author: Tom Geelen
 * Copyright: NotTheFly Component Solutions BV
 */

public class RS232Socket implements SomeSocket {
    private String portname;
    /**
     * The communication speed for the port.
     * @param int baud;
     */
    private int baud;

    /**
     * Creates a new RS232Socket. It will use RL1 settings, a speed of 9600 baud
     * and RTS/CTS flow control.
     * @param portname The name of the port to use.
     */
    public RS232Socket(String portname) {
        this.portname = portname;
        this.baud = 9600;
    }

    /**
     * Open the serial port to the specified portname.
     * @param String portname
     * @throws IOException
     */
    public void open(String portname) throws IOException {
        // Open port
    }

    /**
     * Closes serial port connection.
     */
    public void close() {
        // Close port
    }

    /**
     * Sends a string to the serial port.
     * @param String data
     */
    public void send(String data) {
        // Send data
    }

    /**
     * Reads a string from the serial port.
     * @return String data
     */
    public String receive() {
        // Receive data
    }
}

Version Error! No text of specified style in document.
© Copyright NotTheFly Component Solutions 9-12-2002
public RS232Socket(String portName) throws NoSuchPortException, PortInUseException {
    this(portName, 9600);
}

/**
 * Creates a new RS232Socket. It will use 8N1 settings, and RTS/CTS flow
 * control.
 * @param portName The name of the port to use.
 * @param baud The communication speed for the port.
 */
public RS232Socket(String portName, int baud) throws NoSuchPortException, PortInUseException {
    this.portName = portName;
    this.baud = baud;
    port = CommPortIdentifier.getPortIdentifier(portName).open("OBEUX", 1000);
    try {
        (SerialPort) port).setSerialPortParams(
            baud,
            SerialPort.DATABITS_8,
            SerialPort.STOPBITS_1,
            SerialPort.PARITY_NONE
        );
        (SerialPort) port).setFlowControlMode(
            SerialPort.FLOWCONTROL_RTSCTS_OUT |
            SerialPort.FLOWCONTROL_RTSCTS_IN
        );
        //port.enableReceiveTimeout(5000);
        //port.setInpulBufferSize(MAX_PACKET_SIZE);
        //((SerialPort) port).setDTR(true); //voor ACT-IR1000
        catch (UnsupportedCommOperationException e) {
            System.out.println("RS232Socket: Could not set parameters for " + port + ",");
        }
    }
    catch (IOException e) {
        throw new IOException("Port already open.");
    }
    try {
        port = CommPortIdentifier.getPortIdentifier(portName).open("OBEUX", 1000);
    } catch (NoSuchPortException e) {
        throw new IOException("NoSuchPort: " + e.getMessage());
    } catch (PortInUseException e) {
        throw new IOException("PortInUse: " + e.getMessage());
    }
    }
}

/**
 * Reopens the socket using the same parameters as used when constructed.
 * @throws IOException When an I/O error occurs or when reopening is not possible.
 */
public void reopen() throws IOException {
    if (port != null) {
        try {
            port = CommPortIdentifier.getPortIdentifier(portName).open("OBEUX", 1000);
        } catch (NoSuchPortException e) {
            throw new IOException("NoSuchPort: " + e.getMessage());
        } catch (PortInUseException e) {
            throw new IOException("PortInUse: " + e.getMessage());
        }
    }
}

/**
 * Get the InputStream for this socket.
 * @return The InputStream for this socket.
 * @throws IOException When an I/O error occurs.
 */
public InputStream getInputStream() throws IOException {
    return port.getInputStream();
}

/**
 * Get the OutputStream for this socket.
 * @return The OutputStream for this socket.
 * @throws IOException When an I/O error occurs.
 */
public OutputStream getOutputStream() throws IOException {
    return port.getOutputStream();
}

/**
 * Close the connection.
 * @throws IOException When an I/O error occurs.
 */
public void close() throws IOException {
    port.close();
    port = null;
}

C.10.4 RS232ServerSocket.java

package com.notthefly.oobe;

import java.com.*;
import java.io.IOException;

/**
 * description: RS232 ServerSocket class (for using a serial cable).
 * @author Tom deelen
 * @version 1.1.1 2002/07/15 08:51:18 tong Exp $1
 * @copyright: NotTheFly Component Solutions BV
 */
public class RS232ServerSocket implements SomeServerSocket {
private RS232Socket sock;

/**
 * Creates a new RS232ServerSocket.
 * @param sock The RS232Socket to run the server on.
 */
public RS232ServerSocket(RS232Socket sock) {
    this.sock = sock;
}

/**
 * Wait for an incoming connection.
 * @return A socket for the established connection.
 * @throws IOException When an I/O error occurs.
 */
public RS232Socket accept() throws IOException {
    return sock.accept();
}


Version Error! No text of specified style in document.

© Copyright NotTheFly Component Solutions 9-12-2002 Page 106 of 128
public SomeSocket acceptConnection() throws IOException {    
    return sock;
}

/**
 * Closes this socket.
 * @throws IOException when an I/O error occurs.
 */
public void close() throws IOException {
    sock.close();
}

C.10.5 PhilipsSerialSocket.java

package com.notthefly.obex;
import com.philips.mhp.x.serial.SerialConnection;
import java.io.*;

/**
 * Description: Philips STB serial port Socket class.
 * Author: Tom Geelen
 * Copyright: NotTheFly Component Solutions BV
 */
public class PhilipsSerialSocket implements SomeSocket {
    SerialConnection con = null;
    RealInputStream ris = null;

    /**
     * Creates a new RS232Socket. It will use B96 settings, a speed of 9600 baud
     * and RTS/CTS flow control.
     * @param portName The name of the port to use.
     */
    public PhilipsSerialSocket() throws IOException {
        con = new SerialConnection("RS232");

    }

    /**
     * Reopens the socket using the same parameters as used when constructed.
     * @throws IOException When an I/O error occurs or when reopening is not
     * @param possible.
     */
    public void reopen() throws IOException {
        if (con == null) {
            throw new IOException("Port already open.");
        }
        con = new SerialConnection("RS232");
        ris = null;
    }

    /**
     * Get the InputStream for this socket.
     * @return The InputStream for this socket.
     * @throws IOException When an I/O error occurs.
     */
    public InputStream getInputStream() throws IOException {
        if (ris == null) {
            ris = new RealInputStream(con.getInputStream());
        }
        return ris;
    }

    /**
     * Get the OutputStream for this socket.
     * @return The OutputStream for this socket.
     * @throws IOException When an I/O error occurs.
     */
    public OutputStream getOutputStream() throws IOException {
        return con.getOutputStream();
    }

    /**
     * Close the connection.
     * @throws IOException When an I/O error occurs.
     */
    public void close() throws IOException {
        con.close();
        con = null;
        ris = null;
    }
}

C.10.6 RealInputStream.java

package com.notthefly.obex;
import java.io.;

/**
 * Description: Philips Serial InputStream bugs workaround (Philips doesn't do blocking)
 * Author: Tom Geelen
 * Version: $Id: RealInputStream.java,v 1.2 2002/08/28 10:30:00 tong Exp $
 * Copyright: NotTheFly Component Solutions BV
 */
public class RealInputStream extends InputStream {
    static int DELAY = 500;
    InputStream badStream;
    boolean closed = true;

    public RealInputStream(InputStream is) {
        badStream = is;
        closed = false;
    }
public int read(byte[] buffer) throws IOException {
  int result;
  while (((result = badStream.read(buffer, 0, 1)) == 0) {
    try {
      Thread.sleep(Delay);
    } catch (InterruptedException e) {
      if (closed) {
        throw new IOException("Stream is closed");
      }
    }
    if (result == -1) {
      return -1;
    }
  }
  return result;
}

public int read(byte[] buffer, int start, int len) throws IOException {
  int result;
  while (((result = badStream.read(buffer, start, len)) == 0) {
    try {
      Thread.sleep(Delay);
    } catch (InterruptedException e) {
      if (closed) {
        throw new IOException("Stream is closed");
      }
    }
  }
  return result;
}

public int available() throws IOException {
  return badStream.available();
}

public void close() throws IOException {
  closed = true;
  badStream.close();
}

C.10.7 TCPSocket.java
package com.notthefly.obex;
import java.net.*;
import java.io.*;
/**
 * Description: TCP Socket class (encapsulation of java.net.Socket).
 * Author: Tom Geelen
 * Version: $Id: TCPSocket.java,v 1.3 2002/07/15 08:50:30 tom Exp$
 * Copyright: NotTheFly Component Solutions BV
 */
public class TCPSocket implements SomeSocket {
  Socket sock;
  /**
   * Creates a new TCPSocket from an Socket object.
   * @param sock The socket for which to create an TCPSocket object.
   */
  public TCPSocket(Socket sock) {
    this.sock = sock;
  }
  /**
   * Creates a new TCPSocket that connects to the specified host and port.
   * @param host The IP-host to connect to (IP name or number).
   * @param port The port to connect to.
   * @throws IOException When an I/O error occurs.
   */
  public TCPSocket(String host, int port) throws IOException {
    sock = new Socket(host, port);
  }
  /**
   * Creates a new TCPSocket that connects to the specified host and port.
   * @param port The port to connect to.
   * @throws IOException When an I/O error occurs.
   */
  public TCPSocket(InetAddress addr, int port) throws IOException {
    sock = new Socket(addr, port);
  }
  /**
   * Reopens the socket using the same parameters as used when constructed.
   * @throws IOException When an I/O error occurs or when reopening is not
   * possible.
   */
  public void reopen() throws IOException {
    throw new IOException("Reopening not implemented.");
  }
  /**
   * Get the InputStream for this socket.
   * @return The InputStream for this socket.
   * @throws IOException When an I/O error occurs.
   */
  public InputStream getInputStream() throws IOException {
    return sock.getInputStream();
  }
}
C.10.8 TCPServerSocket.java

```java
package com.notthefly.obex;
import java.net.Socket;
import java.net.ServerSocket;
import java.io.IOException;

/**
 * Description: TCP ServerSocket class (encapsulation of
 * java.net.ServerSocket).
 * Author: Tom Geelen
 * Version: $Id: TCPServerSocket.java,v 1.2 2002/05/06 11:47:03 tong Exp $
 * Copyright: NotTheFly Component Solutions BV
 */
public class TCPServerSocket extends ServerSocket implements SomeServerSocket {

/**
 * Creates a new TCPServerSocket.
 * @param port The TCP port to listen to.
 * @throws IOException When an I/O error occurs.
 */
public TCPServerSocket(int port) throws IOException {
  super(port);
}

/**
 * Wait for an incoming connection.
 * @return A socket for the established connection.
 * @throws IOException When an I/O error occurs.
 */
public SomeSocket acceptConnection() throws IOException {
  return new TCPSocket(accept());
}

C.10.9 IrDASocket.java

```
public void reopen() throws IOException {
    throw new IOException("Reopening not implemented.");
}

/**
 * Get the InputStream for this socket.
 * @return The InputStream for this socket.
 * @throws IOException When an I/O error occurs.
 */
public InputStream getInputStream() throws IOException {
    return sock.getInputStream();
}

/**
 * Get the OutputStream for this socket.
 * @return The OutputStream for this socket.
 * @throws IOException When an I/O error occurs.
 */
public OutputStream getOutputStream() throws IOException {
    return sock.getOutputStream();
}

/**
 * Close the connection.
 * @throws IOException When an I/O error occurs.
 */
public void close() throws IOException {
    sock.close();
}

C.10.10  IrDAServerSocket.java
package com.notthefly.obex;
import hp.infrared.IrSocket;
import hp.infrared.IrServerSocket;
import java.io.IOException;

/**
 * Description: IrDAServerSocket class (encapsulation of
 * hp.infrared.IrServerSocket).
 * @author: Tom Geelen
 * @version: 0.2.2 2002/09/11 11:47:03 tomgeo $Ex
 * @copyright: NotTheFly Component Solutions BV
 */
public class IrDAServerSocket extends IrServerSocket implements SomeServerSocket {
    /**
     * Creates a new IrDAServerSocket.
     * @param service The name of the service on which to register. This
     * ServerSocket will listen for connections to this service.
     * @throws IOException When an I/O error occurs.
     */
    public IrDAServerSocket(String service) throws IOException {
        super(service);
    }

    /**
     * Wait for an incoming connection.
     * @return A socket for the established connection.
     * @throws IOException When an I/O error occurs.
     */
    public SomeSocket acceptConnection() throws IOException {
        return new IrDASocket(accept());
    }
}

C.10.11  ModemServerSocket.java
package com.notthefly.obex;
import java.net.Socket;
import java.net.ServerSocket;
import java.io.*;

/**
 * Description: ModemServerSocket class for using a modem. It will wait for
 * incoming calls (expects RING, then sends ATA, then expects CONNECT). Uses a
 * SomeSocket class for the underlying connection.
 * @author: Tom Geelen
 * @version: 0.2.2 2002/07/15 08:52:25 tomgeo $Ex
 * @copyright: NotTheFly Component Solutions BV
 */
public class ModemServerSocket implements SomeServerSocket {
    private SomeSocket sock;

    /**
     * Creates a new ModemServerSocket.
     * @param port The name of the port to open.
     * @throws IOException When an I/O error occurs.
     */
    public ModemServerSocket(SomeSocket sock) throws IOException {
        this.sock = sock;
    }

    /**
     * Wait for an incoming connection. It does so by waiting for "RING" to
     * appear on the underlying socket. It then replies with "ATA" (Answer modem
     * call) and finally waits for "CONNECT".
     * @return A socket for the established connection.
     * @throws IOException When an I/O error occurs.
     */
    public SomeSocket acceptConnection() throws IOException {
try {
    sock.reopen();
} catch (IOException e) {
    if (e.getMessage().equals("Port already open.")) {
        throw e;
    }
}

InputStream in = sock.getInputStream();
OutputStream out = sock.getOutputStream();
waitFor(in, "XHN\r\n\n");
out.write(65); //A
out.write(69); //E
out.write(69); //E
out.write(13); //\n
waitFor(in, new CONNECTStr()); //waitFor reads an extra char (the \n')
return sock;

/**
 * Reads an InputStream until a given string is read. NOTE: The character
 * following the string is also read from the stream.
 * @param in The InputStream to read
 * @param str The string to wait for.
 * @return true if the string is found.
 * @throws IOException when an I/O error occurs.
 */
private void waitFor(InputStream in, String str) throws IOException {
    char expect[] = str.toCharArray();
    int i;

    do {
        i = in.read();
        if (i >= 0) {
            if (i == expect.length) {
                for (int j = 0; j < expect.length; j++) {
                    if (i == expect[j]) {
                        i = i.read();
                    }
                }
                return true;
            } else {
                i = in.read();
            }
        }
    } while (i != expect.length);
}

/**
 * Closes this socket.
 * @throws IOException when an I/O error occurs.
 */
public void close() throws IOException {
    sock.close();
}

C.10.12 OBEXClient.java

package com.notthefly.obex;

import java.io.*;
import java.net Vector;
import java.util.*;
import java.util.Listed; //import rp.intrared.IrSocket;

/**
 * High level class for client side operations.
 * @author Tom Geelen
 * @version $Id: OBEXClient.java,v 1.4 2002/07/17 10:33:47 tomg Exp$
 * @copyright NotTheFly Component Solutions BV
 */
public class OBEXClient {
    private boolean isConnected; // Constructs a OBEX object.
    private int packetSize;
    private InputStream in;
    private OutputStream out;
    private OBEXHeader connectionId = null;

    private final int MAX_PACKET_SIZE = 0x1000; // 4 Kb

    /**
     * @return Whether the connection was succesfull.
     * @throws IOException when an I/O error occurs during sending/receiving.
     * @throws MalformedURLException when the received response is corrupt.
     */
    public boolean connect() throws IOException, MalformedURLException {
        return connect();
    }

    /**
     * Connect to a specific target.
     * @param target The target to connect to (see OBEX spec).
     * @return Whether the connection was succesfull.
     * @throws IOException when an I/O error occurs during sending/receiving.
     * @throws MalformedURLException when the received response is corrupt.
     */
    public boolean connect(String target) throws IOException, MalformedURLException {
        Vector headers = new Vector();
        if (target.length() > 0) {
headers.addElement(new OBEXByteSequenceHeader(OBEXHeader.TARGET, target.getBytes()));
} else {
    // Use OBEXBasicPacket
    OBEXCommand cmd = new OBEXCommand(OBEXCommand.CONNECT, true, headers, MAX_PACKET_SIZE);
    out.write(cmd.getData());
    out.flush();
    OBEXResponse resp = new OBEXResponse(readPacket(), true);
    if (resp.getStatusCode() == OBEXResponse.OK) {
        isConnected = true;
    } else {
        packetSize = resp.getMaxPacketSize();
        headers = resp.getHeaders();
        connectionId = null;
        for (Enumeration e = headers.elements(); e.hasMoreElements(); ) {
            OBEXHeader header = (OBEXHeader) e.nextElement();
            if (header.getType() == OBEXHeader.CONNECTION_ID) {
                connectionId = header;
            }
        }
        isConnected = true;
    }
    return isConnected;
}
/*
 * Disconnect.
 * Throws IOException when an I/O error occurs during sending/receiving.
 * Throws MalformedOBEXPacketException when the received response is
 * corrupt.
 */
public void disconnect() throws IOException, MalformedOBEXPacketException {
    if (!isConnected) return;
    Vector headers = new Vector();
    if (connectionId != null) {
        headers.addElement(connectionId);
    }
    OBEXCommand cmd = new OBEXCommand(OBEXCommand.DISCONNECT, true, headers, MAX_PACKET_SIZE);
    out.write(cmd.getData());
    out.flush();
    try {
        new OBEXResponse(readPacket());
        catch (IOException e) {
            // Disconnect in IetFMP layer is legal now, so don't throw an exception when
            // the connection was closed by the other side.
            if (!e.getMessage().equals("Unexpected end of input stream.") )
                throw e;
        }
        isConnected = false;
    }
    /*
     * Sends an object. This is often a file, but can be any stream.
     * @param objectStream An input stream which delivers the data for the
     * object.
     * @param name The name of the object (i.e. the filename when a file is
     * sent).
     * @param length The length of the object in bytes.
     * @param listener An OBEXClientListener that listens for progress. May be
     * null.
     */
    public boolean sendObject(InputStream objectStream, String name, int length, OBEXClientListener listener)
        throws IOException, MalformedOBEXPacketException {
        return sendObject(objectStream, name, length, 0, listener);
    }
    /*
     * Sends an object. This is often a file, but can be any stream.
     * @param objectStream An input stream which delivers the data for the
     * object.
     * @param name The name of the object (i.e. the filename when a file is
     * sent).
     * @param length The length of the object in bytes.
     */
    public boolean sendObject(InputStream objectStream, String name, int length, int objectClass, OBEXClientListener listener)
        throws IOException, MalformedOBEXPacketException {
    if (!isConnected) return false;
    OBEXCommand cmd;
    OBEXResponse resp;
    Vector headers;
    headers = new Vector();
    if (connectionId != null) {
        headers.addElement(connectionId);
    }
    headers.addElement(new OBEXUnicodeHeader(OBEXHeader.NAME, name));
    headers.addElement(new OBEXIntHeader(OBEXHeader.LENGTH, length));
    headers.addElement(new OBEXUnicodeHeader(OBEXHeader.OBJECT_CLASS, objectClass));
    return sendObject(objectStream, name, length, 0, listener);
}
public int receiveObject(OutputStream objectStream, String name, OBEXClientListener listener) throws IOException, MalformedURLException {  
    if (!isConnected) return -1;
    OBEXCommand cmd;
    OBEXResponse resp;
    Vector headers;

    headers = new Vector();
    if (connectionId != null) {    
      headers.addElement(connectionId);
    }
    headers.addElement(new OBEXUnencodedHeader(OBEXHeader.NAME, name));
    cmd = new OBEXCommand(OBEXCommand.GET, true, headers);
    int respCode;
    do {  
      out.write(cmd.getData());
      out.flush();
      resp = new OBEXResponse(readPacket());
      respCode = resp.getCode();
      if (respCode != OBEXResponse.OK && respCode != OBEXResponse.CONTINUE) return respCode;
      int bytesRead = 0;
      for (Enumeration enum = resp.getHeaders().elements(); enum.hasMoreElements(); ) {    
        OBEXHeader header = (OBEXHeader) enum.nextElement();
        if (header.getType() == OBEXHeader.SYSCON || header.getType() == OBEXHeader.END_OF_BODY) {  
          bytes += (OBEXBytesSequenceHeader) header.getBytesSequence();
          out.write(bytes);
          bytesRead += bytes.length;
        }
      }
      if (listener != null) {    
        listener.clientProgress(bytesReceived);
      }
    } while (respCode == OBEXResponse.CONTINUE);
    return OBEXResponse.OK;
  }

  // Converting a byte into an unsigned int.
  // * @param b The byte to convert.
  // * @return The byte as an unsigned int.
  private static int bytetoint(byte b) {    
    int i = b;
    if (i < 0) i = 0x100;
    return i;
  }
C.10.13 OBEXClientListener.java

package com.notthefly.obex;

/**
 * Description: Interface class for getting progress reports
 * @author Tom Geelen
 * @version V1.0 2002/04/12 08:16:50
 */
public interface OBEXClientListener {
    /**
     * This callback method is called for every data packet sent or
     * received. This method should return very quickly.
     * @param bytes The number of bytes sent or received so far.
     */
    void clientProgress(int bytes);
}

C.10.14 OBEXServer.java

package com.notthefly.obex;

import java.io.*;
import java.util.*;

/**
 * Description:
 * @author Tom Geelen
 * @version V1.0 2002/04/17 11:12:28
 */
public class OBEXServer implements Runnable {
    private SomeServerSocket srvSocket;
    private InputStream in; // OBEX data stream
    private OutputStream out; // OBEX data stream
    private OBEXServerListener listener;
    // Connection state variables
    private int packetSize;
    private InputStream input; // input stream for PUT
    private OutputStream output; // output stream for GET
    private boolean isClosing;
    private boolean isPutting;
    private boolean connected;
    private boolean closedStreams;
    private int length;
    private int bytesSent;
    private boolean isReceived;
    private boolean isReceived;

    private final int MAX_PACKET_SIZE = 0x1000; // 4 Kb
    // private final int MAX_PACKET_SIZE = 128;

    /**
     * Constructs an OBEXServer object. Streams for GET/PUT commands will be closed
     * at the end of the transfer.
     * @param listener Object that supplies the streams for reading and
     * writing OBEX objects and gets notified of any exceptions.
     */
    public OBEXServer(SomeServerSocket srvSocket, OBEXServerListener listener) {
        this(srvSocket, listener, true);
    }

    /**
     * Constructs an OBEXServer object.
     * @param listener Object that supplies the streams for reading and
     * writing OBEX objects and gets notified of any exceptions.
     * @param closeStreams When <code>false</code>, the streams that are supplied
     * by the OBEXServerListener are not closed by the server, otherwise they
     * are closed at the end of the GET/PUT request.
     */
    public OBEXServer(SomeServerSocket srvSocket, OBEXServerListener listener, boolean closeStreams) {
this.listener = listener;
this.arvSocket = arvSocket;
this.closeStreams = closeStreams;
Thread td = new Thread(this);
   td.start();
}

/**
 * Main server thread, waiting for connections.
 */
public void run() {
   SomeSocket sock = null;

   while (true) {
      try {
         sock = arvSocket.acceptConnection();
         in = sock.getInputStream();
         out = sock.getOutputStream();
         catch (IOException e) {
            if (killed) return;
            listener.serverException(e);
            sock = null;
         }
         listener.serverNotify(0, OBEXServerListener.CONNECT, 0);
         processConnection();
         listener.serverNotify(0, OBEXServerListener.DISCONNECT, 0);
      }
      if (sock != null) {
         try {
            sock.close();
         }
         catch (IOException e) {
            listener.serverException(e);
         }
      }
   }
}

/**
 * Kills the server thread.
 */
public void kill() {
   connected = false;
   killed = true;
   try {
      arvSocket.close();
   }
   catch (IOException e) {
      System.out.println("Error closing server socket!");
   }
}

/**
 * Process a connection.
 */
private void processConnection() {
   OBEXCommand cmd;
   OBEXResponse resp;
   input = null;
   output = null;
   isSetting = false;
   isPutting = false;
   connected = true;
   length = null;
   pytesSent = 0;
   bytesReceived = 0;
   packetSize = 118;  // Safe fallback value in case we get no CONNECT cmd.

   while (connected) {
      try {
         cmd = new OBEXCommand(readPacket());
         resp = handleCommand(cmd);
         out.write(resp.getData());
         out.flush();
      }
      catch (Exception e) {
         if (killed || e.getMessage().equals("Unexpected end of input stream.")) && !isSetting &&
            !isPutting) {
            // IrLMF disconnect. Since OBEX DISCONNECT is not required, this
            // is a valid end of the session.
            if (closeStreams) {
               try {
                  input.close();
               }
               catch (Exception ex) {}
               try {
                  output.close();
               }
               catch (Exception ex) {}
            }
            input = null;
            output = null;
            connected = false;
         } else {
            // Pass exceptions (IOException, MalformedURLException) to
            // listener
            listener.serverException(e);
         }
      }
   }
}

/**
 * Handle a single command.
 * Serves cmd The OBEXCommand to handle.
 * Return the OBEXResponse for the supplied command.
 */
private OBEXResponse handleCommand(OBEXCommand cmd) throws IOException {
   OBEXResponse resp = null;
   Vector headers;
   switch (cmd.getCommand()) {
   case [ CONNECT ]:---------------------------

case OBEXCommand.CONNECT:
    packetSize = cmd.getMaxPacketSize() - 6;
    resp = new OBEXResponse(MAX_PACKET_SIZE);
    break;

    //== GET ]================================================================
    case OBEXCommand.GET:
        if (isGetting) {
            String name = "*";
            for (Enumeration enum = cmd.getHeaders().elements(); enum.hasMoreElements();)
            {
                OBEXHeader header = (OBEXHeader) enum.nextElement();
                if (header.getType() == OBEXHeader.NAME) {
                    name = ((OBEXTimestampHeader) header).getName();
                }
            }
            if (name.equals("*")) {
                // no or empty name supplied
                resp = new OBEXResponse(OBEXResponse.BAD_REQUEST);
            } else {
                length = new Integer(0);
                input = listener.getFile(name, length);
                byteLeft = length.intValue();
                if (input == null) {
                    // file not found
                    resp = new OBEXResponse(OBEXResponse.NOT_FOUND);
                } else {
                    // request ok, first packet
                    isGetting = true;
                    headers = new Vector();
                    headers.addElement(new OBEXIntHeader(OBEXHeader.END_OF_BODY, length.intValue()));
                    resp = new OBEXResponse(OBEXResponse.OK, headers);
                    listener.serverNotify(OBEXServerListener.TRANSFER_DONE, 0);
                }
            }
        }
        headers = new Vector();
        if (byteLeft <= packetSize) {
            byte[] buffer = new byte[byteLeft];
            for (int i = 0; i < byteLeft; i++) {
                i += input.read(buffer, i, byteLeft - i);
            }
            if (closeStream) {
                input.close();
            }
            headers.addElement(new OBEXTimestampHeader(OBEXHeader.END_OF_BODY, buffer));
            resp = new OBEXResponse(OBEXResponse.OK, headers);
            isGetting = false;
            listener.serverNotify(OBEXServerListener.TRANSFER_DONE, 0);
        } else {
            // in-between packet
            byte[] buffer = new byte[packetSize];
            for (int i = 0; i < packetSize; i++) {
                i += input.read(buffer, i, packetSize - i);
            }
            headers.addElement(new OBEXTimestampHeader(OBEXHeader.END_OF_BODY, buffer));
            resp = new OBEXResponse(OBEXResponse.CONTINUE, headers);
            byteLeft -= packetSize;
            listener.serverNotify(OBEXServerListener.PROGRESS, length.intValue() - byteLeft);
        }
        break;

    //== [ PUT ]================================================================
    case OBEXCommand.PUT:
        if (isPutting) {
            String name = "*";
            for (Enumeration enum = cmd.getHeaders().elements(); enum.hasMoreElements();)
            {
                OBEXHeader header = (OBEXHeader) enum.nextElement();
                if (header.getType() == OBEXHeader.NAME) {
                    name = ((OBEXTimestampHeader) header).getName();
                }
            }
            if (name.equals("*")) {
                // no or empty name supplied
                resp = new OBEXResponse(OBEXResponse.BAD_REQUEST);
            } else {
                output = listener.writeFile(name);
                if (output == null) {
                    // file couldn't be opened for some reason
                    resp = new OBEXResponse(OBEXResponse.INTERNAL_SERVER_ERROR);
                } else {
                    isPutting = true;
                    byteReceived = 0;
                }
            }
        }
        if (isPutting) {
            for (Enumeration enum = cmd.getHeaders().elements(); enum.hasMoreElements();)
            {
                OBEXHeader header = (OBEXHeader) enum.nextElement();
                if (header.getType() == OBEXHeader.BODY) {
                    byte[] data = (OBEXByteSequenceHeader) header).getBytes();
                    output.write(data);
                    byteReceived += data.length;
                }
            }
            if (cmd.isFinal) {
                isPutting = false;
                if (closeStream) {
                    output.close();
                }
                output = null;
                resp = new OBEXResponse(OBEXResponse.OK);
                listener.serverNotify(OBEXServerListener.TRANSFER_DONE, 0);
                // else {
                resp = new OBEXResponse(OBEXResponse.CONTINUE);
            } else {
C.10.15 OBEXServerListener.java

package com.notthefly.obex;
import java.io.InputStream;
import java.io.OutputStream;
/**
 * Description: Interface class for interacting with the OBEXServer.
 * Base OBEXServer
 * @author Tom Geelen
 * @version $Id: OBEXServerListener.java,v 1.2 2002/04/12 08:16:54 tom Exp$
 * (Base OBEXServer), copyright: NotTheFly Component Solutions BV
 */
public interface OBEXServerListener {
    /**
     * Connect event */
    public final static int CONNECT = 0;
    /**
     * Progress event */
    public final static int PROGRESS = 1;
    /**
     * Transfer done event */
    public final static int TRANSFER_DONE = 2;
    /**
     * Disconnect event */
    public final static int DISCONNECT = 3;
    /**
     * All exceptions that occur in the server thread are passed back via
     * this method.<p>
     * Exceptions that can occur are: <UL>
     * <LI>$<ulink harmless=0>OBEXPacketException</ulink>: Thrown when there is a parse
     * error in a received packet.<p>
     * </LI></UL>
     */
}
C.10.16 OBEXCommand.java

package com.notthefly.obex;
import java.util.Vector;
import java.util.Enumeration;
/**
 * description: Class representing an OBEX command packet. More info on
 * the OBEX protocol can be found on http://www.irta.org/
 * author: Tom Gweilen
 * version: $Id: OBEXCommand.java,v 1.2 2002/05/21 10:35:46 tong Exp$
 * see also OBEXResponse
 * copyright: NotTheFly Component Solutions BV
 * 
 * public class OBEXCommand {
 *  /** CONNECT opcode */
 *  public static final byte CONNECT = (byte) 0x00;
 *  /** DISCONNECT opcode */
 *  public static final byte DISCONNECT = (byte) 0x01;
 *  /** PUT opcode */
 *  public static final byte PUT = (byte) 0x02;
 *  /** GET opcode */
 *  public static final byte GET = (byte) 0x03;
 *  /** SKYPATH opcode */
 *  public static final byte SKYPATH = (byte) 0x05;
 *  /** ABORT opcode */
 *  public static final byte ABORT = (byte) 0x07;
 *  /** The raw bytes of this packet */
 *  private byte[] rawBytes;
 *  /** The command opcode */
 *  private byte command;
 *  /** The final bit */
 *  private boolean finalBit;
 *  /** The packet length */
 *  private int length;
 *  /** The OBEX headers in this packet */
 *  private Vector headers = new Vector();
 *  /** The OBEX protocol version */
 *  private byte version = (byte) 0x10;
 *  /** Flags for CONNECT and SKYPATH (mostly unused) */
 *  private byte flags = (byte) 0x00;
 *  /** Constants for SKYPATH (unused) */
 *  private byte constants = (byte) 0x00;
 *  /** The maximum packet size */
 *  private int maxPacketSize = 0;
 *  /** Converts a byte to an unsigned int. */
 *  private int byteToUint(byte b) {
 *    int i = b;
 *    if (i < 0) i += 0x100;
 *    return i;
 *  }
 *  /** Creates an OBEXCommand from the specified parameters. */
 *  @param command The opcode for the OBEX command.
 *  @param finalBit <code>true</code> if this is the last packet for the
 *  command, <code>false</code> otherwise.
 *  @param headers A vector containing the (link OBEXHeader) for this
 *  command.
 *  @param maxPacketSize The maximum packet size this "device" can handle
 *  */
 */
public OBEXCommand(byte command, boolean finalBit, Vector headers, int maxPacketSize) {
    this.command = command;
    this.finalBit = finalBit;
    this.headers = headers;
    this.maxPacketSize = maxPacketSize;

    switch (command) {
    case CONNECT: length = 7; break;
    case SETUPPATH: length = 5; break;
    default: length = 3; break;
    }

    for (Enumeration enum = headers.elements(); enum.hasMoreElements(); ) {
        length += ((OBEXHeader) enum.nextElement()).getData().length;
    }

    int index;
    rawData = new byte[length];
    if (finalBit) {
        rawData[0] = (byte) (byteToInt(command) | 0x80);
    } else {
        rawData[0] = command;
    }

    rawData[1] = (byte) (length >> 8);
    rawData[2] = (byte) (length & 0xFF);

    switch (command) {
    case CONNECT:
        rawData[3] = version;
        rawData[4] = flags;
        rawData[5] = (byte) (maxPacketSize >> 8);
        rawData[6] = (byte) (maxPacketSize & 0xFF);
        index = 7;
        break;
    case SETUPPATH:
        rawData[3] = flags;
        rawData[4] = constants;
        index = 5;
        break;
    default:
        index = 3;
        break;
    }

    OBEXHeader header;
    byte[] headerData;
    for (Enumeration enum = headers.elements(); enum.hasMoreElements(); ) {
        header = (OBEXHeader) enum.nextElement();
        headerData = header.getData();
        for (int i = 0; i < headerData.length; i++) {
            rawData[index] = headerData[i];
            index++;
        }
    }
}

/**
 * Creates an OBEXCommand from the specified parameters. For the maximum
 * packet size (only used for CONNECT commands) the default value of 128 is
 * used.
 * @param command The opcode for the OBEX command.
 * @param finalBit <code>true</code> if this is the last packet for the
 * command, <code>false</code> otherwise.
 * @param headers A vector containing the {Glink OBEXHeader}s for this
 * command.
 * @return
 */
public OBEXCommand(byte[] data) throws MalformedObexPacketException {
    int headerstart;

    if (data.length < 3) throw new MalformedObexPacketException("Packet too short");
    rawdata = data;
    command = (byte) (data[0] & 0x0F);
    finalBit = (data[0] & 0x80) == 0x80;
    length = (data[1] & 0xFF) + byteToInt(data[2]);
    if (length > data.length) throw new MalformedObexPacketException("Incomplete packet");

    switch (command) {
    case CONNECT:
        if (length < 5) throw new MalformedObexPacketException("Incorrect length field");
        version = data[3];
        flags = data[4];
        maxPacketSize = byteToInt(data[5]) * 0x100 + byteToInt(data[6]);
        headerstart = 7;
        break;
    case SETUPPATH:
        if (length < 5) throw new MalformedObexPacketException("Incorrect length field");
        flags = data[3];
        constants = data[4];
        headerstart = 5;
        break;
    case DISCONNECT:
    case PUT:
    case GET:
    case ABORT:
        default:
            headerstart = 3;
            break;
    }
}
headers = OBEXHeader.decodeHeaders(data, headerstart, length);

/**
 * Gives a string representation of the opcode of this command.
 * @return A string representation of the the opcode of this command.
 */
public String commandAsString()
{
    switch (command) {
    case 0x00: return "CONNECT" + (version & 0x0F) + " ", + Integer.toHexString((int) command) + " ");
    case 0x01: return "DISCONNECT";
    case 0x02: return "PUT";
    case 0x03: return "GET";
    case 0x05: return "SETUP";
    case 0x0F: return "ABORT";
    default: return "UNKNOWN_CMD ", + Integer.toHexString((int) command) + " ");
    }
}

/**
 * Gets the raw packet data for this command.
 * @return The raw packet data for this command.
 */
public byte[] getData()
{
    return rawdata;
}

/**
 * Gets the maximum packet size for this command.
 * @return The maximum packet size if this is a CONNECT packet,
 * <code>true</code> otherwise.
 */
public int getMaxPacketSize()
{
    return maxPacketSize;
}

/**
 * Gets the opcode for this command.
 * @return The opcode for this command.
 */
public byte getCommand()
{
    return command;
}

/**
 * Check whether this command has the final bit set.
 * @return <code>true</code> if the final bit is set; <code>false</code> if it is not set. If the final bit is set, this
 * command is split over multiple packets.
 * @return <code>true</code> when the final bit is set on this command,
 * <code>false</code> otherwise.
 */
public boolean isFinal()
{
    return finalBit;
}

/**
 * Gets the OBEX headers for this command.
 * @return A vector containing the OBEX headers for this command.
 */
public Vector getHeaders()
{
    return headers;
}

C.10.17 OBEXResponse.java

package com.notthefly.obex;
import java.util.Vector;
import java.util.Enumeration;
/**
 * Description: Class representing an OBEX response packet. More info on
 * the OBEX protocol can be found on http://www.irda.org/
 * Author: Tom Geelen
 * Version: 5.0; OBEXResponse.java, v 1.3 2002/07/16 10:21:02 tomg Exp $
 * See OBEXCommand
 * copyright: NotTheFly Component Solutions BV
 */
public class OBEXResponse
{
    /** "100 - Continue" response */
    public static final byte CONTINUE = (byte) 0x90;
    /** "200 - OK" response */
    public static final byte OK = (byte) 0xA0;
    /** "400 - Bad Request" response */
    public static final byte BAD_REQUEST = (byte) 0xC0;
    /** "403 - Forbidden" response */
    public static final byte FORBIDDEN = (byte) 0xC3;
    /** "404 - Not Found" response */
    public static final byte NOT_FOUND = (byte) 0xC4;
    /** "500 - Internal server error" response */
    public static final byte INTERNAL_SERVER_ERROR = (byte) 0xD0;
    /** "501 - Not implemented" response */
    public static final byte NOT_IMPLEMENTED = (byte) 0xD1;
    /** The raw bytes of this packet */
    protected byte[] rawdata;
    /** The response code */
    protected byte code;
    /** The OBEX headers of this response */
    protected Vector headers;
    /** The maximum packet size (only for connection responses) */
    protected int maxPacketSize;
    /** Converts a byte to an unsigned int. */
}
private static int bytetoInt(byte b) {
    int i = b;
    if (i < 0) i += 0x100;
    return i;
}

/**
 * Constructs a response from the specified raw packet data.
 * @param connect Indicates whether this is a connection response or a
 * normal response. <code>true</code> for a connection response.
 * @param data The raw packet data.
 */
public OBEXResponse(byte[] data, boolean connect) throws MalformedOBEXPacketException {
    int length; int headerstart;
    code = data[0];
    length = (byteToInt(data[1]) << 8) | byteToInt(data[2]);
    if (connect) {
        maxPacketSize = byteToInt(data[5]) * 0x100 + byteToInt(data[6]);
        headerstart = 7;
    } else {
        maxPacketSize = 0;
        headerstart = 3;
    }
    headers = OBEXHeader.decodeHeaders(data, headerstart, length);

    /**
     * Constructs a response from the specified raw packet data. The data should
     * not be a connection response.
     * @param data The raw packet data.
     */
    public OBEXResponse(byte[] data) throws MalformedOBEXPacketException {
this(data, false);
}

/**
 * Constructs a response with specified response code and no headers.
 * @param code The response code.
 */
public OBEXResponse(byte code) {
this(code, new Vector());
}

/**
 * Constructs a response with specified response code.
 * @param code The response code.
 * @param headers A vector with OBEX headers.
 */
public OBEXResponse(byte code, Vector headers) {
this.code = code;
this.headers = headers;
int length = 3;
for (Enumeration enum = headers.elements(); enum.hasMoreElements(); ) {
    length += ((OBEXHeader) enum.nextElement()).length;
}
rawdata = new byte[length];
rawdata[0] = code;
rawdata[1] = (byte) (length >> 8);
rawdata[2] = (byte) (length & 0xFF);

// Copy headers into raw data
int index = 3;
OBEXHeader header;
byte[] headerdatal; for (Enumeration enum = headers.elements(); enum.hasMoreElements(); ) {
    header = (OBEXHeader) enum.nextElement();
    headerdatal = header.getData(); for (int i = 0; i < headerdatal.length; i++) {
    rawdata[index] = headerdatal[i];
    index++;
    }
}

/**
 * Constructs a CONNECT response packet with specified max. packet size
 * @param maxPacketSize The maximum packet size.
 */
public OBEXResponse(int maxPacketSize) {
this.code = OK;
rawdata = new byte[7];
rawdata[0] = OK; // -- code OK
rawdata[1] = (byte) 0x00; // __packetsize = 0x0007
rawdata[2] = (byte) 0x07; // /!
rawdata[3] = (byte) 0x10; // -- OBEX version 1.0
rawdata[4] = (byte) 0x00; // -- flags, always 0x00
rawdata[5] = (byte) (maxPacketSize >> 8); // __ max packetsize
rawdata[6] = (byte) (maxPacketSize & 0xFF); // /!
headers = new Vector();
}

/**
 * Gets the raw packet data for this response.
 * @return The raw packet data for this response.
 */
public byte[] getData() {
return rawdata;
}

/**
 * Gets the headers from this response.
 * @return The headers from this response.
 */
public Vector getHeaders() {
return headers;
}
public abstract class OBEXHeader {
    /**
     * COUNT header type. */
    public final static byte COUNT = 0x00;
    /**
     * NAME header type. */
    public final static byte NAME = 0x01;
    /**
     * TYPE header type. */
    public final static byte TYPE = 0x02;
    /**
     * LDMTH header type. */
    public final static byte LENGTH = 0x03;
    /**
     * TIME header type. */
    public final static byte TIME = 0x04;
    /**
     * DESCRIPTION header type. */
    public final static byte DESCRIPTION = 0x05;
    /**
     * TARGET header type. */
    public final static byte TARGET = 0x06;
    /**
     * HTTP header type. */
    public final static byte HTTP = 0x07;
    /**
     * BODI header type. */
    public final static byte BODI = 0x08;
    /**
     * END OF BODI header type. */
    public final static byte ENDOFBODI = 0x09;
    /**
     * WHO header type. */
    public final static byte WHO = 0x0A;
    /**
     * CONNECTION ID header type. */
    public final static byte CONNECTION_ID = 0x0B;
    /**
     * APPLICATION PARAMETERS header type. */
    public final static byte APP_PARAMS = 0x0C;
    /**
     * AUTHENTICATION CHALLENGE header type. */
    public final static byte AUTH_CHALLENGE = 0x0D;
    /**
     * AUTHENTICATION RESPONSE header type. */
    public final static byte AUTH_RESPONSE = 0x0E;
    /**
     * OBJECT CLASS header type. */
    public final static byte OBJECT_CLASS = 0x0F;

    /**
     * The header type */
    protected byte headertype;

    /**
     * The raw data for this header. */
    protected byte[] rawdata;

    /**
     * Gives a string representation of the value of the header.
     * @return A string representation of the value of the header.
     */
    public abstract String valueAsString();

    /**
     * Gets the type of this header.
     * @return the type of this header.
     */
    public byte getType() {
        return headertype;
    }
}

C.10.18 OBEXHeader.java
public byte[] getData() {  
    return rawdata;
}

/**
 * Sets the type of this header
 * @param type The new type for this header
 */
public void setTyp(byte type) {
    // code
}

/**
 * Gives a string representation of the type of this header
 * @return A string representation of the type of this header
 */
public String typeAsString() {
    switch (type) {
    case 0x00: return "Count";
    case 0x01: return "Name";
    case 0x02: return "Type";
    case 0x03: return "Length";
    case 0x04: return "Time";
    case 0x05: return "Description";
    case 0x06: return "Target";
    case 0x07: return "Http";
    case 0x08: return "Body";
    case 0x09: return "End-Of-Body";
    case 0x0A: return "Who";
    case 0x0B: return "Connection-Id";
    case 0x0C: return "Application-Parameters";
    case 0x0D: return "Authentication-Challenge";
    case 0x0E: return "Authentication-Response";
    case 0x0F: return "Object-ID";
    default: return "Unknown-Header(" + Integer.toHexString((byte)0x00) + ")";
    }
}

/**
 * Gives a string representation of this header
 * @return A string representation of this header
 */
public String toString() {
    return typeAsString() + ": "+ valueAsString();
}

private static int byteToT(int byte) {
    int i = 0;
    if (i < 0) i += 0x100;
    return i;
}

/**
 * Static method to decode headers from raw packet data
 * @param data The raw packet data.
 * @param startIndex The index in the data where the headers start.
 * @param stopIndex The index in the data where the headers end
 * @param code The code of the header data. A common value is <code>data.length</code>.
 * @param headers A vector containing the HEADER object representing
 * the headers in the data.
 * @return The decoded headers.
 */
public static Vector decodeHeaders(byte[] data, int startIndex, int stopIndex) throws MalformedURLException {

    Vector result = new Vector();

    byte headerType = (byte) (data[startIndex] & 0x3F);
    switch (((int) data[startIndex] & 0x0F) {
    case 0x00:// & not Unicode
        hLength = byteToT(data[startIndex] + 1) * 0x100 + byteToT(data[startIndex] + 2) - 3;
        startIndex += 3;
        if (hLength < 0) throw new MalformedURLException("Illegal header data.");
        char[] cvalue = new char[hLength / 2];
        for (int i = 0; i < hLength; i++) {  
            cvalue[i / 2] = (char) {  
                byteToT(data[startIndex] + i) * 0x100 + byteToT(data[startIndex] + i + 1);
            }
        }
        header = new OHXUnicodeHeader(headerType, new String(cvalue));
        startIndex += hLength;
        break;
    case 0x4B://byte
        hLength = byteToT(data[startIndex] + 1) * 0x100 + byteToT(data[startIndex] + 2) - 3;
        startIndex += 3;
        if (hLength < 0) throw new MalformedURLException("Illegal header data.");
        byte[] bvalue = new byte[hLength];
        for (int i = 0; i < hLength; i++) {  
            bvalue[i] = data[startIndex + i];
        }
        header = new OHXByteSequenceHeader(headerType, bvalue);
        startIndex += hLength;
        break;
    case 0x5A:///byte
        header = new OHXByteHeader(headerType, data[startIndex] + 1);
        startIndex += 1;
        break;
    case 0x5E:///byte
        break;
    case 0x60:///byte
        break;
    case 0x7E:///byte
        break;
    case 0x80:///byte
        break;
    case 0x81:///byte
        break;
    case 0x82:///byte
        break;
    case 0x83:///byte
        break;
    case 0x84:///byte
        break;
    case 0x85:///byte
        break;
    case 0x86:///byte
        break;
    case 0x87:///byte
        break;
    case 0x88:///byte
        break;
    case 0x89:///byte
        break;
    case 0x8A:///byte
        break;
    case 0x8B:///byte
        break;
    case 0x8C:///byte
        break;
    case 0x8D:///byte
        break;
    case 0x8E:///byte
        break;
    case 0x8F:///byte
        break;
    case 0x90:///byte
        break;
    case 0x91:///byte
        break;
    case 0x92:///byte
        break;
    case 0x93:///byte
        break;
    case 0x94:///byte
        break;
    case 0x95:///byte
        break;
    case 0x96:///byte
        break;
    case 0x97:///byte
        break;
    case 0x98:///byte
        break;
    case 0x99:///byte
        break;
    case 0x9A:///byte
        break;
    case 0x9B:///byte
        break;
    case 0x9C:///byte
        break;
    case 0x9D:///byte
        break;
    case 0x9E:///byte
        break;
    case 0x9F:///byte
        break;
    case 0xA0:///byte
        break;
    case 0xA1:///byte
        break;
    case 0xA2:///byte
        break;
    case 0xA3:///byte
        break;
    case 0xA4:///byte
        break;
    case 0xA5:///byte
        break;
    case 0xA6:///byte
        break;
    case 0xA7:///byte
        break;
    case 0xA8:///byte
        break;
    case 0xA9:///byte
        break;
    case 0xAA:///byte
        break;
    case 0xAB:///byte
        break;
    case 0xAC:///byte
        break;
    case 0xAD:///byte
        break;
    case 0xAE:///byte
        break;
    case 0xAF:///byte
        break;
    case 0xB0:///byte
        break;
    case 0xB1:///byte
        break;
    case 0xB2:///byte
        break;
    case 0xB3:///byte
        break;
    case 0xB4:///byte
        break;
    case 0xB5:///byte
        break;
    case 0xB6:///byte
        break;
    case 0xB7:///byte
        break;
    case 0xB8:///byte
        break;
    case 0xB9:///byte
        break;
    case 0xBA:///byte
        break;
    case 0xBB:///byte
        break;
    case 0xBC:///byte
        break;
    case 0xBD:///byte
        break;
    case 0xBE:///byte
        break;
    case 0xBF:///byte
        break;
    case 0xC0:///byte
        break;
    case 0xC1:///byte
        break;
    case 0xC2:///byte
        break;
    case 0xC3:///byte
        break;
    case 0xC4:///byte
        break;
    case 0xC5:///byte
        break;
    case 0xC6:///byte
        break;
    case 0xC7:///byte
        break;
    case 0xC8:///byte
        break;
    case 0xC9:///byte
        break;
    case 0xCA:///byte
        break;
    case 0xCB:///byte
        break;
    case 0xCC:///byte
        break;
    case 0xCD:///byte
        break;
    case 0xCE:///byte
        break;
    case 0xCF:///byte
        break;
    case 0xD0:///byte
        break;
    case 0xD1:///byte
        break;
    case 0xD2:///byte
        break;
    case 0xD3:///byte
        break;
    case 0xD4:///byte
        break;
    case 0xD5:///byte
        break;
    case 0xD6:///byte
        break;
    case 0xD7:///byte
        break;
    case 0xD8:///byte
        break;
    case 0xD9:///byte
        break;
    case 0xDA:///byte
        break;
    case 0xDB:///byte
        break;
    case 0xDC:///byte
        break;
    case 0xDD:///byte
        break;
    case 0xDE:///byte
        break;
    case 0xDF:///byte
        break;
    case 0xE0:///byte
        break;
    case 0xE1:///byte
        break;
    case 0xE2:///byte
        break;
    case 0xE3:///byte
        break;
    case 0xE4:///byte
        break;
    case 0xE5:///byte
        break;
    case 0xE6:///byte
        break;
    case 0xE7:///byte
        break;
    case 0xE8:///byte
        break;
    case 0xE9:///byte
        break;
    case 0xEA:///byte
        break;
    case 0xEB:///byte
        break;
    case 0xEC:///byte
        break;
    case 0xED:///byte
        break;
    case 0xEE:///byte
        break;
    case 0xEF:///byte
        break;
    case 0xF0:///byte
        break;
    case 0xF1:///byte
        break;
    case 0xF2:///byte
        break;
    case 0xF3:///byte
        break;
    case 0xF4:///byte
        break;
    case 0xF5:///byte
        break;
    case 0xF6:///byte
        break;
    case 0xF7:///byte
        break;
    case 0xF8:///byte
        break;
    case 0xF9:///byte
        break;
    case 0xFA:///byte
        break;
    case 0xFB:///byte
        break;
    case 0xFC:///byte
        break;
    case 0xFD:///byte
        break;
    case 0xFE:///byte
        break;
    case 0xFF:///byte
        break;
    }
    return result;
}
C.10.19 OBEXByteHeader.java

package com.notthefly.obex;

/**
 * Description: Class representing an OBEX single byte header.
 * Author: Tom Geelen
 * Revision: $Id: OBEXByteHeader.java,v 1.2 2002/05/17 12:01:50 tong Exp$
 * Copyright: NotTheFly Component Solutions BV
 */
public class OBEXByteHeader extends OBEXHeader {
  /** The value of the header */
  private byte value;

  /**
   * Creates an OBEXByteHeader
   * @param type The type of the header
   * @param val The value for the header
   */
  public OBEXByteHeader(byte type, byte val) {
    headerType = type;
    value = val;
    rawdata = new byte[2];
    rawdata[0] = (byte) (type | 0x80);
    rawdata[1] = val;
  }

  /**
   * Sets the value for this header
   * @param b The new value for the header
   */
  public void setByte(byte b) {
    value = b;
  }

  /**
   * Gets the value for this header
   * @return The value for the header
   */
  public byte getByte() {
    return value;
  }

  /**
   * Gives a string representation of the value of the header
   * @return A string representation of the value of the header
   */
  public String valuesAsString() {
    return "0x" + Integer.toHexString((new Byte(value)).intValue());
  }
}

C.10.20 OBEXIntHeader.java

package com.notthefly.obex;

/**
 * Description: Class representing an OBEX four-byte quantity header.
 * Author: Tom Geelen
 * Revision: $Id: OBEXIntHeader.java,v 1.2 2002/05/17 12:01:50 tong Exp$
 * Copyright: NotTheFly Component Solutions BV
 */
public class OBEXIntHeader extends OBEXHeader {
  /** The value of the header */
  private int value;

  /**
   * Creates an OBEXIntHeader
   * @param type The type of the header
   * @param val The value for the header
   */
  public OBEXIntHeader(byte type, int val) {
    headerType = type;
    value = val;
    rawdata = new byte[4];
    rawdata[0] = (byte) (type | 0xC0);
    rawdata[1] = (byte) (val & 0xFF000000) >> 24;
    rawdata[2] = (byte) (val & 0xFFFF0000) >> 16;
    rawdata[3] = (byte) (val & 0xFF0000) >> 8;
    rawdata[4] = (byte) (val & 0xFF);
  }

  /**
   * Sets the value for this header
   * @param i The new value for the header
   */
  public void setInt(int i) {
    value = i;
  }

C.10.21  OBEXByteSequenceHeader.java

package com.notthefly.obex;

/**
 * Gets the value for this header
 * @return The value for the header
 */
public int getInt() {
    return value;
}

/**
 * Gives a string representation of the value of the header
 * @return A string representation of the value of the header
 */
public String valueAsString() {
    return new Integer(value).toString();
}

C.10.22  OBEXUnicodeHeader.java

package com.notthefly.obex;


Version Error! No text of specified style in document.

© Copyright NotTheFly Component Solutions 9-12-2002 Page 125 of 128
public class OBEXUnicodeHeader extends OBEXHeader {
  /** The value of the header */
  private String val;

  public OBEXUnicodeHeader(byte type, String val) {
    headerType = type;
    value = val;
    int length = (val.length() * 2) + 5;
    rawdata = new byte[length];
    rawdata[0] = type; //type bits are 00
    rawdata[1] = (byte) ((length & 0xFF00) >> 8);
    rawdata[2] = (byte) (length & 0xFF);
    chars = val.toCharArray();
    for (int i = 0; i < chars.length; i++) {
      rawdata[2 * i + 3] = (byte) (chars[i] >> 8);
      rawdata[2 * i + 4] = (byte) (chars[i] & 0xFF);
    }
    rawdata[length - 2] = 0;
    rawdata[length - 1] = 0;
  }

  public void setUnicode(String str) {
    value = str;
  }

  public String getUnicode() {
    return value;
  }

  public String valueAsString() {
    return value;
  }
}

C.10.23  OBEXApp.java

package com.notthefly.obeex;
import java.io.*;
import java.util.HashMap;
import java.util.List;

/**
 * description: Program that talks the OBEEX protocol.
 * author: Tom Greens
 * copyright: NotTheFly Component Solutions BV
 */
public class OBEXApp {

  /** Main method. Starts in receive mode when the application is started
   * without arguments and starts in sending mode when an argument is given.
   */
  public static void main(String[] args) {
    System.out.println("Usage: java OBEXApp [server | client]");
    try {
      if (args.length == 0) {
        System.out.println("Usage: java OBEXApp [server | client]");
        System.out.println("-m [mode] -t [timeout] -h [host] -p [port]");
        System.out.println("-u [username] -p [password] -f [file]");
        System.out.println("-d [dump file]");
        System.out.println("-f [file]");
        System.out.println("-d [dump file]");
      }
      else {
        if (args[0].equals("server")) {
          System.out.println("** Starting OBEEXServer **");
          mod = new OBEEXServer();
          mod.start();
        }
        else {
          if (args[0].equals("client") || args[0].equals("client2")) {
            System.out.println("** Starting OBEEXClient **");
            client = new OBEEXClient();
            client.start();
          }
          else {
            System.out.println("** Starting OBEEXServer **");
            mod = new OBEEXServer();
            mod.start();
            System.out.println("** OBEEXServer started **");
          }
        }
      }
    }
  }
}

public class OBEEXServer {
  public void serverException(Exception e) {
    System.out.println("Exception!");
    System.exit(1);
  }

  public InputStream getFile(String name, Integer length) {
    File file = new File(name);
    if (file.isDirectory()) {
      System.out.println("Directory not supported");
      System.exit(1);
    }
    else {
      try {
        FileInputStream fis = new FileInputStream(file);
        byte[] buffer = new byte[length.intValue()];
        fis.read(buffer);
        return new ByteArrayInputStream(buffer);
      }
      catch (IOException e) {
        System.out.println("Error copying file");
        System.exit(1);
      }
    }
  }

  public static void main(String[] args) {
    if (args.length == 0) {
      System.out.println("Usage: java OBEXApp [server | client]");
      System.out.println("-m [mode] -t [timeout] -h [host] -p [port]");
      System.exit(1);
    }
    else {
      if (args[0].equals("server")) {
        System.out.println("** Starting OBEEXServer **");
        mod = new OBEEXServer();
        mod.start();
      }
      else {
        if (args[0].equals("client") || args[0].equals("client2")) {
          System.out.println("** Starting OBEEXClient **");
          client = new OBEEXClient();
          client.start();
        }
        else {
          System.out.println("** Starting OBEEXServer **");
          mod = new OBEEXServer();
          mod.start();
          System.out.println("** OBEEXServer started **");
        }
      }
    }
  }
}

MalformedObexPacketException.java

package com.notthefly.obex;

/**
 * Exception thrown when an error occurs parsing an OBEX packet.
 * @author Tom Geelen
 * @version $Id: MalformedObexPacketException.java,v 1.1 2002/04/05 08:38:44 tong Exp $
 * copyright NotTheFly Component Solutions BV
 */
public class MalformedObexPacketException extends Exception {
    /**
     * Constructs a MalformedObexPacketException with the specified message.
     * @param msg A message describing the parse error.
     */
    public MalformedObexPacketException(String msg) {
        super(msg);
    }
}