J2ME Bluetooth Programming

Master's Thesis

André N. Klingsheim
Department of Informatics
University of Bergen

30th June 2004

The NoWires Research Group

http://www.nowires.org

http://wireless.klings.org

Preface

This Master's thesis gives insights into the technologies needed to develop Java Bluetooth applications for mobile devices. Bluetooth, Java 2 Micro Edition (J2ME), and Java APIs for Bluetooth Wireless Technology (JABWT) are discussed. The necessary infrastructure for developing Java Bluetooth applications are also described. Descriptions of how different Bluetooth actions like *inquiry* and *service discovery* are done with the Java API are provided. Code samples are included as well, highlighting the functionality available in JABWT.

Books on JABWT programming became available during the writing of this thesis. Simple functionality is often explained in a far too complex way in these books, making it hard for developers to get started with Java Bluetooth programming. This thesis aims to give a clean and basic introduction to the simple parts of JABWT before the more complex functionality is explained. Also, a broad view of the technology is given, enabling developers to see where JABWT has its place among other technologies. Different software platforms, development tools, and Java Bluetooth enabled devices are discussed.

Acknowledgments

I would first of all like to thank Professor Kjell Jørgen Hole for giving me the opportunity to work with the Java and Bluetooth technologies. He has been highly available and highly supportive throughout the whole process of writing this thesis. I also thank my common-law spouse Eli for her patience, understanding and encouragement. I would like to thank my parents, Tor and Barbro, for invaluable support through the years. Even though they do not know much about Java or Bluetooth, they still have shown great interest in my work. My fellow students must not be forgotten. It has been a privilege to make their acquaintance, I will surely miss the monthly gatherings at our favorite uncle Lauritz.

Table of Contents

Chapter 1 Introduction..........1
1.1 Structure of thesis..........2

Chapter 2 Bluetooth..........5
2.1 Bluetooth architecture..........6
2.2 Piconet and scatternet..........8
2.3 Bluetooth links..........11
2.4 Device discovery (inquiry) and service discovery..........12
2.5 Bluetooth services..........13
2.6 Bluetooth profiles..........15
2.7 Bluetooth qualification..........16
2.8 Bluetooth security..........17
2.8.1 Security modes..........17
2.8.2 Pairing and bonding (authentication)..........18
2.8.3 Encryption..........18
2.8.4 Authorization..........19
2.8.5 Security manager..........19
2.8.6 Security mode 2..........20

Chapter 3 Java 2 Micro Edition (J2ME)..........21
3.1 Configurations and profiles..........22
3.2 Connected Limited Device Configuration (CLDC)..........22
3.2.1 Generic Connection Framework (GCF)..........23
3.2.2 CLDC versions and requirements..........23
3.2.3 CLDC security..........24
3.3 Mobile Information Device Profile (MIDP)..........25
3.3.1 MIDP version 1..........26
3.3.2 MIDP version 2..........27
3.4 MIDlets..........29
3.4.1 OEM-specific applications..........30
3.4.2 MIDlet suites..........30
3.4.3 MIDlet deployment..........31
3.5 Java APIs for Bluetooth Wireless Technology (JABWT)..........32
3.5.1 Security..........33

Chapter 4 Infrastructure..........35
4.1 Linux workstation..........35
4.2 Sun wireless toolkits..........35
4.3 Rococo Impronto simulator..........36
4.4 Smartphones..........37
4.4.1 Nokia 6600..........37
4.4.2 Sony Ericsson P900..........37
4.5 Web server, www.klings.org/nowires/..........38
4.6 IDEs..........39

Chapter 5 J2ME and JABWT programming..........41
5.1 Structure of Bluetooth MIDlet..........41
5.2 Device discovery (Inquiry)..........43

5.3 Service search....48
5.4 RFCOMM links with JABWT....52
5.4.1 RFCOMM server....53
5.4.2 RFCOMM client....54
5.4.3 RFCOMM connection parameters....55
5.5 Service records and JABWT....57
5.5.1 Retrieving information from service records with JABWT....57
5.5.2 Manipulating service records with JABWT....59
5.6 Pitfalls....62
5.6.1 RFCOMM flow control....62
5.6.2 RFCOMM EOF....63
5.6.3 Removal of service records....63
5.6.4 Populating service records....63
5.6.5 Inquiry with P900....64
5.6.6 ServiceRecordHandle format....64

Chapter 6 Sample applications....65
6.1 Bluetooth browser....65
6.2 Bluetooth benchmark....68
6.2.1 Benchmark server....69
6.2.2 Benchmark client....72

Chapter 7 Summary and conclusions....75
7.1 Summary....75
7.2 Conclusions....76
7.3 Further work....76

Appendix A BTBrowser....79
BTBrowserMIDlet.java....79

Appendix B BTBenchmark....93
Server.java....93
Client.java....103
StatusCanvas.java....116

Appendix C KlingsLib....121
BTServiceAttributeId.java....121
BTProtocol.java....127
BTServiceClass.java....132
BTUUIDTool.java....142
BluetoothInfoCanvas.java....144
BluetoothServiceRecordCanvas.java....150
CanvasHelper.java....162

List of Tables

Table 2.1 Descriptions of Bluetooth protocol layers........7
Table 2.2 Service record attributes........14
Table 2.3 Bluetooth foundation profiles........16
Table 3.1 CLDC packages........24
Table 3.2 MIDP 1.0 requirements........26
Table 3.3 MIDP 1.0 packages........26
Table 3.4 MIDP 2.0 requirements........28
Table 3.5 MIDP 2.0 packages........28
Table 3.6 JABWT packages........32
Table 5.1 RFCOMM connection parameters........56
Table 6.1 Benchmark results........69

List of Figures

Figure 2.1 The Bluetooth protocol stack....6
Figure 2.2 A typical piconet....8
Figure 2.3 Scatternet....9
Figure 2.4 Piconet with two nodes....10
Figure 2.5 Scatternet with 3 nodes....10
Figure 2.6 Piconet with 3 nodes....11
Figure 2.7 The Service Discovery DataBase (SDDB)....13
Figure 2.8 Data element construct....14
Figure 3.1 High level view of J2ME....21
Figure 3.2 CLDC position in J2ME architecture....23
Figure 3.3 MIDP position in J2ME architecture....25
Figure 3.4 MIDlet architecture overview....29
Figure 3.5 MIDlet deployment....31
Figure 3.6 JABWT position in J2ME architecture....32
Figure 4.1 Screenshot of Impronto Simulator....36
Figure 4.2 The Nokia 6600....37
Figure 4.3 The Sony Ericsson P900....38
Figure 6.1 Cached/known devices....65
Figure 6.2 Initiating device discovery....66
Figure 6.3 Device discovery in progress....66
Figure 6.4 Device discovery completed....67
Figure 6.5 Service search on P900....67
Figure 6.6 Service attributes....68
Figure 6.7 Service attributes....68
Figure 6.8 Benchmark server main menu....70
Figure 6.9 Benchmark server settings....70
Figure 6.10 Benchmark server started....71
Figure 6.11 Client connected to server....71
Figure 6.12 Benchmark server, transfer results....72
Figure 6.13 Benchmark client, device discovery....72
Figure 6.14 Benchmark client, selection of data amount....73
Figure 6.15 Benchmark client, transfer status screen....73
Figure 6.16 Benchmark client, transfer finished 74

Important acronyms

Connected Limited Device Configuration (CLDC)..22
Generic Access Profile (GAP)..15
Generic Connection Framework (GCF)...23
Java 2 Micro Edition (J2ME)..2
Java APIs for Bluetooth Wireless Technology (JABWT)..2
Logical Link Control and Adaption Protocol (L2CAP)...11
MIDlets...29
Mobile Information Device (MID)..23
Mobile Information Device Profile (MIDP)...22
RFCOMM...12
Serial Port Profile (SPP)...15
Service Discovery DataBase (SDDB)...13
Service Discovery Protocol (SDP)...7
Universally Unique IDentifier (UUID)..14
Wireless ToolKits (WTKs)...35

1 Introduction

Wireless technologies are becoming more and more popular around the world. Consumers appreciate the wireless lifestyle, relieving them of the well known “cable chaos” that tends to grow under their desk. Nowadays, the world would virtually stop if wireless communications suddenly became unavailable. Both our way of life and the global economy are highly dependent on the flow of information through wireless mediums like television and radio. Cellphones have become highly available during the last decade. Now virtually everyone owns a cellphone, making people available almost wherever they are. Many companies are highly dependent on their employees having cellphones, some companies have even decided not to employ stationary phone systems but instead use cellphones exclusively throughout the organization.

New wireless technologies are introduced at an increasing rate. During the last few years the IEEE 802.11 [1] technologies have started to spread rapidly, enabling consumers to set up their own wireless networks. This constitutes an important change in how wireless communications are made available to consumers. Wireless networks are no longer provided by big corporations alone, they can just as well be implemented by individuals. Our society is becoming more and more dependent on wireless communications as new areas of use are introduced.

The Bluetooth wireless technology is also spreading rapidly. The number of Bluetooth chipsets shipped per year has doubled from 2002 to a total of 69 million chipsets in 2003 [2]. The majority of these Bluetooth chipsets are used in mobile phones. An interesting aspect is that consumers are highly dependent on having a cellphone, and the Bluetooth technology is included in the majority of new cellphones. The Bluetooth technology will therefore spread because of the general need for cellphones. As an increasing number of useful Bluetooth applications become available, many consumers will already have Bluetooth devices and be ready to start using Bluetooth PANs (Personal Area Networks) where all their Bluetooth devices communicate with one another [3].

The number of Java enabled mobile phones worldwide is over 250 million according to a press release from Sun (dated February 19, 2004) titled: “Java technology is everywhere, surpasses 1.5 billion devices worldwide” [4]. The number of Java enabled mobile phones will continue to increase. Nokia states that they have already shipped tens of millions Java enabled handsets and that most of their new handset models announced will support Java [5].

Java enabled mobile phones have already been on the market for some years. Due to the very resource constrained mobile phones available a few years ago, Java applications were not very sophisticated and did not hit the mass-market the way many had hoped. As seen in the rest of the software and hardware industry, games play an important role in driving the development of both hardware and software forward. It is therefore interesting to see that a large market has emerged lately for

Java games targeting mobile devices. Processing power, available memory, screen size, and screen resolution are increasing as new Java enabled mobile devices enter the market. Newly released Java applications are accordingly sophisticated, and will help to spread the Java technology usage even further.

The Java APIs for Bluetooth Wireless Technology (JABWT) ties the Java technology and the Bluetooth technology together. JABWT is made available in some of the latest smartphones and will probably be available also in low-end cellphones in the future. One can easily imagine different scenarios where JABWT would be useful, e.g. the functionality of existing Java games is extended to support multi-player games using Bluetooth connectivity. Other interesting scenarios emerge as well, such as a consumer using a Java Bluetooth enabled mobile phone to pay for a soda by connecting to a Bluetooth enabled soda vending-machine. A good prediction is that JABWT will first find its use in multi-player Java games, making the Java and Bluetooth technologies well-known to consumers. Thereafter we will probably see other types of Java Bluetooth applications, such as small-amount payment applications.

At the time of writing there are only two books discussing JABWT [6], [7]. These books provide quite complex code samples for Java Bluetooth programming, making it hard to grasp how JABWT is used. There are a few discussion boards with high activity on the Internet where developers help each other [8], [9], [10]. Although discussion boards may help developers with a specific problem, they usually do not give a sufficient overview of the technology. This thesis gives a broad overview of Java and Bluetooth technologies before discussing JABWT and its details. Code samples are provided, showing how JABWT specific code is used in a J2ME application. The amount of J2ME specific code is kept at a minimum in order to draw attention to what is really important, namely the JABWT specific code.

The intended audience for this thesis are students working with Java and Bluetooth technologies, and Java 2 Micro Edition (J2ME) application developers seeking knowledge about the Bluetooth technology and JABWT. It is assumed that the reader is familiar with the J2ME technology. Individuals whom are unfamiliar with any of the technologies discussed should not expect to understand all the details in this thesis. However, they will get a broad overview and may use this thesis as a starting point for their studies on the involved technologies. References to in-depth information are included throughout the thesis, enabling the interested reader to quickly find relevant background information.

1.1 Structure of thesis

This thesis will give an introduction to the J2ME technology, the Bluetooth technology, and JABWT available in mobile devices. The infrastructure used when developing JABWT applications will be described. Programming with J2ME and

JABWT will be discussed thoroughly, highlighting functionality and irregularities in JABWT. The sample applications developed during the work with this thesis will be explained and demonstrated.

Chapter 2 gives an overview of the Bluetooth technology. Different aspects of the technology are discussed, starting with a general overview of the Bluetooth architecture. Important concepts such as Bluetooth networks, Bluetooth services, Bluetooth profiles, device discovery, and service discovery are explained. Chapter 2 also looks into the Bluetooth security model.

Chapter 3 presents a brief introduction to the J2ME technology. It is assumed that the reader is familiar with J2ME. Readers unfamiliar with J2ME will get an understanding of what it is and may use the references to collect extensive background information.

Chapter 4 provides an overview of the infrastructure needed to develop Java and Bluetooth applications. Available development tools are discussed, in addition to the Java Bluetooth enabled smartphones that were used to test Java Bluetooth applications.

Chapter 5 contains code samples and explanations on how JABWT is used when developing applications. Basic operations such as device discovery and service discovery are described first, before more complex functionality is explained.

Chapter 6 describes the demo applications supplied in Appendix A and Appendix B, the *Bluetooth browser* and *Bluetooth benchmark* applications.

Chapter 7 contains a summary of this thesis and some important conclusions.

2 Bluetooth

This chapter gives a brief introduction to the Bluetooth technology. The Bluetooth architecture will be explained, in addition to basic Bluetooth actions like *device discovery* and *service discovery*. Bluetooth services and some important details about *service records* will be explained as well. After reading this chapter, developers should have sufficient knowledge about the Bluetooth technology to start application development with JABWT. For the interested reader, references to in-depth information about the Bluetooth technology are included throughout the chapter. This chapter is based on my supervisor's Bluetooth lectures available on his website [11], the Bluetooth book by Bray and Sturman [12], and the Bluetooth Specification version 1.1 [13] available for download on the Bluetooth Special Interest Group (SIG) website [14].

Bluetooth is a low cost, low power, short-range radio technology intended to replace cable connections between cellphones, PDAs and other portable devices. It can clean up your desk considerably, making wires between your workstation, mouse, laptop computer etc. obsolete. Ericsson Mobile Communications started developing the Bluetooth system in 1994, looking for a replacement to the cables connecting cellphones and their accessories. The Bluetooth system is named after a tenth-century Danish Viking king, Harald Blåtand, who united and controlled Norway and Denmark. The first Bluetooth devices hit the market around 1999.

The Bluetooth SIG is responsible for further development of the Bluetooth standard. Sony Ericsson, Intel, IBM, Toshiba, Nokia, Microsoft, 3COM, and Motorola are some of the companies involved in the SIG. The composition of the Bluetooth SIG is one of the major strengths of the Bluetooth technology. The mixture of both noticeable software and hardware suppliers participating in the further development of the Bluetooth technology ensures that Bluetooth products are made available to end-users. Microsoft supports Bluetooth in their Microsoft Windows Operating System (OS), hence, Bluetooth software is made available to the vast majority of the desktop software market. At the time of writing, Intel is including Bluetooth technology in several new mainboard chipsets, especially for laptop computers. Both Nokia and Sony Ericsson include Bluetooth technology in their latest cellphones. This all adds up to a wide availability of the Bluetooth technology for end-users. Information of more commercial nature about the Bluetooth technology is available on the Bluetooth technology website [15].

This thesis describes the Bluetooth Specification version 1.1, the Bluetooth version implemented in most mobile devices at the moment [13]. However, the Bluetooth 1.2 specification is already completed and the Bluetooth 2.0 specification is in the works. At the time of writing, Enhanced Data Rate (EDR) Bluetooth has just been introduced by the Bluetooth SIG, raising the gross air data rate from 1 Mbps to 2 Mbps or 3

Mbps. Devices conforming to these new specifications will probably show up shortly after the completion of this Master thesis.

2.1 Bluetooth architecture

The Bluetooth specification aims to allow Bluetooth devices from different manufacturers to work with one another, so it is not sufficient to specify just a radio system. Because of this, the Bluetooth specification does not only outline a radio system but a complete protocol stack to ensure that Bluetooth devices can discover each other, explore each other's services, and make use of these services.

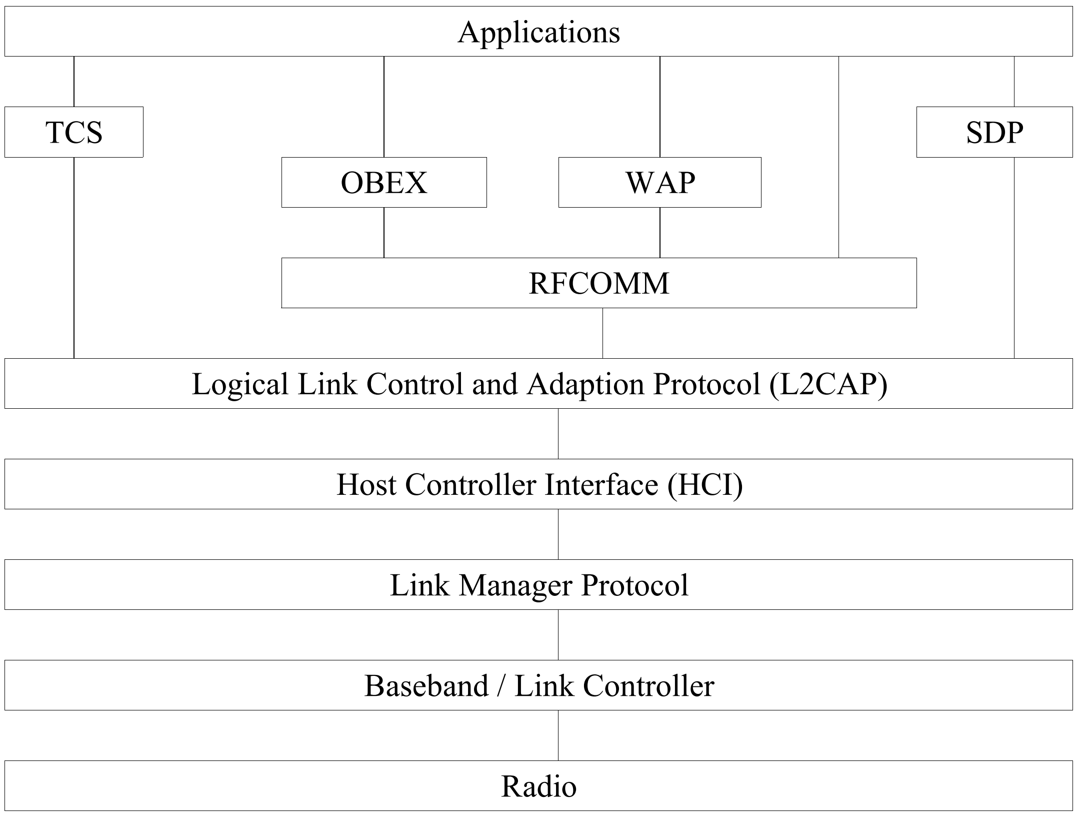

Applications
TCS
SDP
OBEX
WAP
RFCOMM
Logical Link Control and Adaption Protocol (L2CAP)
Host Controller Interface (HCI)
Link Manager Protocol
Baseband / Link Controller
Radio

Figure 2.1 The Bluetooth protocol stack

The Bluetooth stack is made up of many layers, as shown in Figure 2.1. The HCI is usually the layer separating hardware from software and is implemented partially in software and hardware/firmware. The layers below the HCI are usually implemented in hardware and the layers above the HCI are usually implemented in software. Note that resource constrained devices such as Bluetooth headsets may have all functionality implemented in hardware/firmware. Table 2.1 gives a short description of each layer shown in Figure 2.1.

Layer	Description
Applications	Bluetooth profiles guide developers on how applications should use the protocol stack
Telephony Control System (TCS)	Provides telephony services
Service Discovery Protocol (SDP)	Used for service discovery on remote Bluetooth devices
WAP and OBEX	Provide interfaces to higher layer parts of other communications protocols
RFCOMM	Provides an RS-232 like serial interface
L2CAP	Multiplexes data from higher layers and converts between different packet sizes
HCI	Handles communication between the host and the Bluetooth module
Link manager Protocol	Controls and configures links to other devices
Baseband and Link Controller	Controls physical links, frequency hopping and assembling packets
Radio	Modulates and demodulates data for transmission and reception on air

Table 2.1 Descriptions of Bluetooth protocol layers

The interested reader will find further information about the layers of the Bluetooth stack in the Bluetooth book by Bray and Sturman [12] and in the Bluetooth specification [13].

Application developers do not need to know all the details about the layers in the Bluetooth stack. However, an understanding of how the Bluetooth radio works is of importance. The Bluetooth radio is the lowest layer of Bluetooth communication. The Industrial, Scientific and Medical (ISM) band at 2.4 GHz is used for radio communication. Note that several other technologies use this band as well. Wi-Fi technologies like IEEE 802.11b/g and kitchen technologies like microwave ovens may cause interference in this band [1].

The Bluetooth radio utilizes a signaling technique called Frequency Hopping Spread Spectrum (FHSS). The radio band is divided into 79 sub-channels. The Bluetooth radio uses one of these frequency channels at a given time. The radio jumps from channel to channel spending 625 microseconds on each channel. Hence, there are 1600 frequency hops per second. Frequency hopping is used to reduce interference caused by nearby Bluetooth devices and other devices using the same frequency band. Adaptive Frequency Hopping (AFH) is introduced in the Bluetooth 1.2 specification and is useful if your device communicates through both Bluetooth and Wi-Fi simultaneously (e.g. a laptop computer with both Bluetooth and Wi-Fi equipment). The frequency hopping algorithm can then avoid using Bluetooth channels overlapping the Wi-Fi channel in use, hence avoiding interference between your own radio communications.

Every Bluetooth device is assigned a unique Bluetooth address, being a 48-bit hardware address equivalent to hardware addresses assigned to regular Network Interface Cards (NICs). The Bluetooth address is used not only for identification, but also for synchronizing the frequency hopping between devices and generation of keys in the Bluetooth security procedures.

2.2 Piconet and scatternet

A *piconet* is the usual form of a Bluetooth network and is made up of one *master* and one or more *slaves*. The device initiating a Bluetooth connection automatically becomes the master. A piconet can consist of one master and up to seven active slaves. The master device is literally the master of the piconet. Slaves may only transmit data when transmission-time is granted by the master device, also slaves may not communicate directly with each other, all communication must be directed through the master. Slaves synchronize their frequency hopping with the master using the master's clock and Bluetooth address.

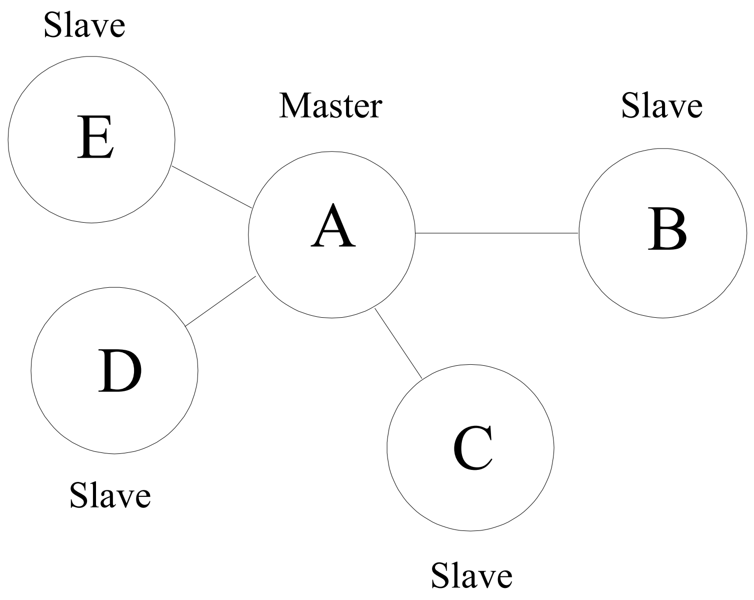


Figure 2.2 A typical piconet

Piconets take the form of a star network, with the master as the center node, shown in Figure 2.2. Two piconets may exist within radio range of each other. Frequency hopping is not synchronized between piconets, hence different piconets will randomly collide on the same frequency.

When connecting two piconets the result will be a *scatternet.* Figure 2.3 shows an example, with one intermediate node connecting the piconets. The intermediate node must time-share, meaning it must follow the frequency hopping in one piconet at the time. This reduces the amount of time slots available for data transfer between the intermediate node and a master, it will at least cut the transfer rate in half. It is also important to note that neither version 1.1 nor version 1.2 of the Bluetooth specification define how packets should be routed between piconets. Hence, communication between piconets cannot be expected to be reliable.

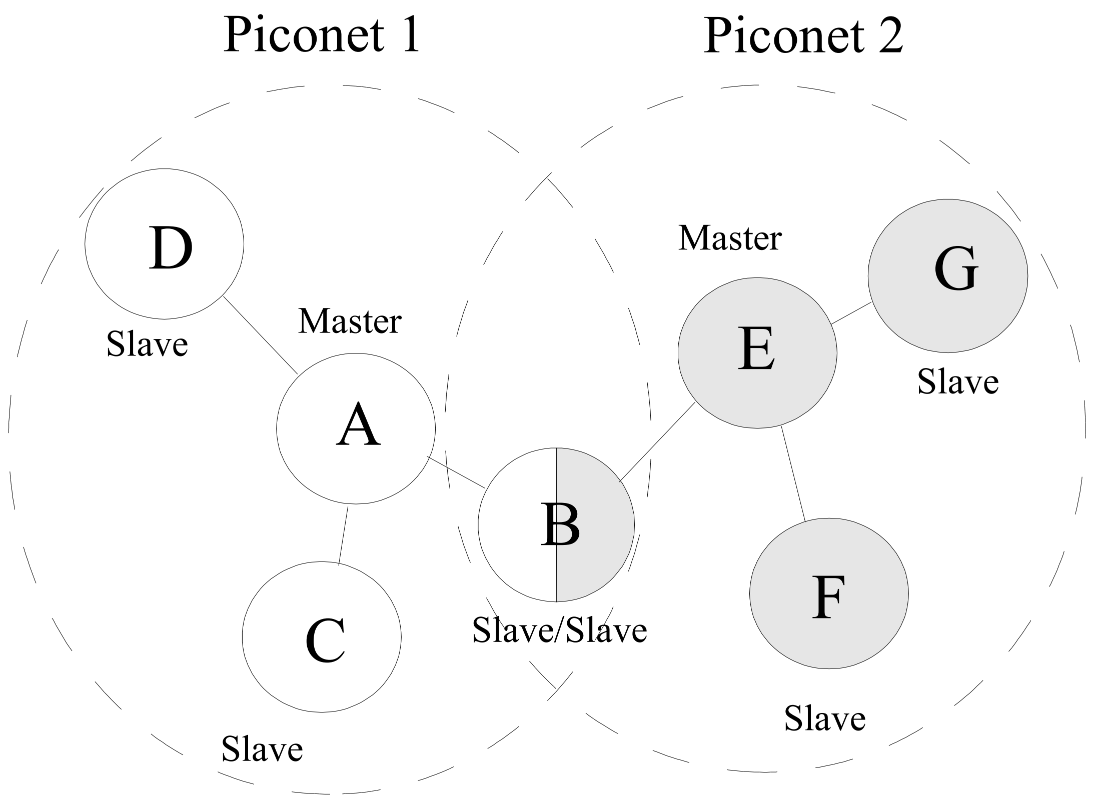


Figure 2.3 Scatternet

Role-switching enables two devices to switch roles in a piconet. Consider the following example: You have two devices A and B. Device A connects to device B, hence, device A becomes the master of the piconet consisting of devices A and B as shown in Figure 2.4.

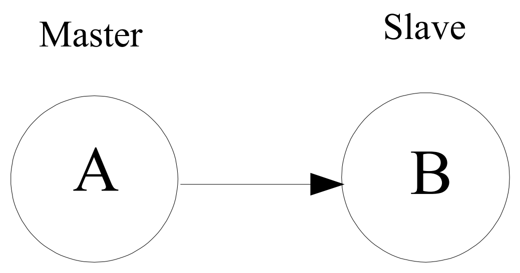


Figure 2.4 Piconet with two nodes

Then a device C wants to join the piconet. Device C connects to the master device, A. Since device C initiated the connection it will automatically become the master of the connection between device C and device A. We now have two masters, hence, we have two piconets. Device A is the intermediate node between these piconets, being the master for device B and the slave for device C, as seen in Figure 2.5.

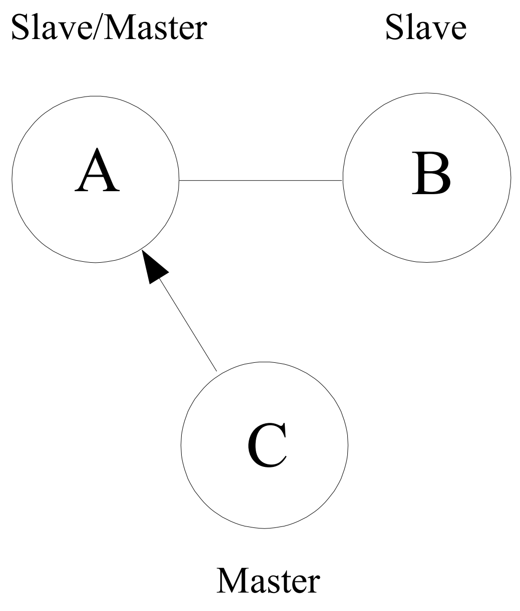


Figure 2.5 Scatternet with 3 nodes

Figure 2.6 shows that a role-switch between device A and device C will give us one piconet where A is the master and both B and C are slaves. We see that when a new device wants to be part of a piconet we actually need a role-switch to make this happen, else we get a scatternet.

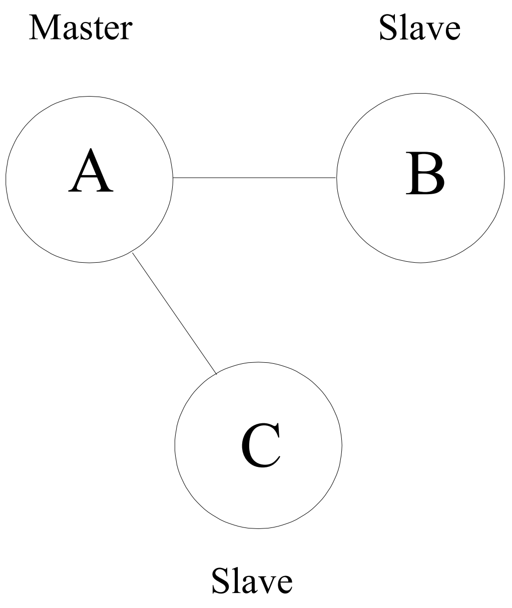


Figure 2.6 Piconet with 3 nodes

2.3 Bluetooth links

Two types of physical links are defined in version 1.1 of the Bluetooth specification, Synchronous Connection Oriented (SCO) links and Asynchronous ConnectionLess (ACL) links. The SCO and ACL links are part of the baseband specification.

SCO links are intended for audio transmission. When setting up a SCO link time slots are reserved for transmission of data, thus providing a Quality of Service (QoS) guarantee. Lost or erroneous packages are not re-transmitted which makes sense for voice transmissions. All SCO links operate at 64 kbps. A master device can have up to three simultaneous SCO links at a time, all to the same slave or to different slaves. Slave devices can have up to three SCO links to the Master device.

ACL links are intended for data communication. An ACL link provides error-free transmission of data which means that lost or erronous packets are re-transmitted. No QoS guarantee is provided. The maximum data rate at the application level is around 650 kbps for an ACL link. A master device can have a number of ACL links to a number of different devices, but only one ACL link can exist between two devices. User data is usually transferred to and from the Logical Link Control and Adaption Protocol (L2CAP) layer of the Bluetooth stack. Application developers usually refer to L2CAP and RFCOMM links when talking about Bluetooth links. To be precise, L2CAP and RFCOMM are separate layers in the Bluetooth stack which rely on an ACL physical link for data transmission.

L2CAP provides multiplexing between different higher layer protocols over a single physical ACL link, enabling several logical data links to be set up between two Bluetooth devices. L2CAP also provides segmentation and reassembly of packets from higher layers. Different protocols use different packet sizes, some of these may need to be segmented in order to be sent over an ACL link due to package size constraints. An ACL packet can have a maximum of 339 bytes of payload data, while an L2CAP packet can have a maximum of 65,535 bytes of payload data.

The RFCOMM layer emulates RS-232 serial ports and serial data streams. RFCOMM relies on L2CAP for multiplexing multiple concurrent data streams and handling connections to multiple devices. The majority of Bluetooth profiles make use of the RFCOMM protocol because of its ease of use compared to direct interaction with the L2CAP layer.

2.4 Device discovery (inquiry) and service discovery

Due to the ad-hoc nature of Bluetooth networks, remote Bluetooth devices will move in and out of range frequently. Bluetooth devices must therefore have the ability to discover nearby Bluetooth devices. When a new Bluetooth device is discovered, a service discovery may be initiated in order to determine which services the device is offering.

The Bluetooth Specification refers to the *device discovery* operation as *inquiry*. During the inquiry process the inquiring Bluetooth device will receive the Bluetooth address and clock from nearby discoverable devices. The inquiring device then has identified the other devices by their Bluetooth address and is also able to synchronize the frequency hopping with discovered devices, using their Bluetooth address and clock.

Devices make themselves discoverable by entering the *inquiry scan* mode. In this mode frequency hopping will be slower than usual, meaning the device will spend a longer period of time on each channel. This increases the possibility of detecting inquiring devices. Also, discoverable devices make use of an Inquiry Access Code (IAC). Two IACs exist, the General Inquiry Access Code (GIAC) and the Limited Inquiry Access Code (LIAC). The GIAC is used when a device is general discoverable, meaning it will be discoverable for a undefined period of time. The LIAC is used when a device will be discoverable for only a limited period of time.

Different Bluetooth devices offer different sets of services. Hence, a Bluetooth device needs to do a *service discovery* on a remote device in order to obtain information about available services. Service searches can be of a general nature by polling a device for all available services, but can also be narrowed down to find just a single service. The service discovery process uses the Service Discovery Protocol (SDP). A

SDP client must issue SDP requests to a SDP server to retrieve information from the server's service records.

2.5 Bluetooth services

Bluetooth devices keep information about their Bluetooth services in a Service Discovery DataBase (SDDB) as shown in Figure 2.7. The SDDB contains *service record* entries, [13, p. 340], where each service record contains attributes describing a particular service. Each service has its own entry in the SDDB.

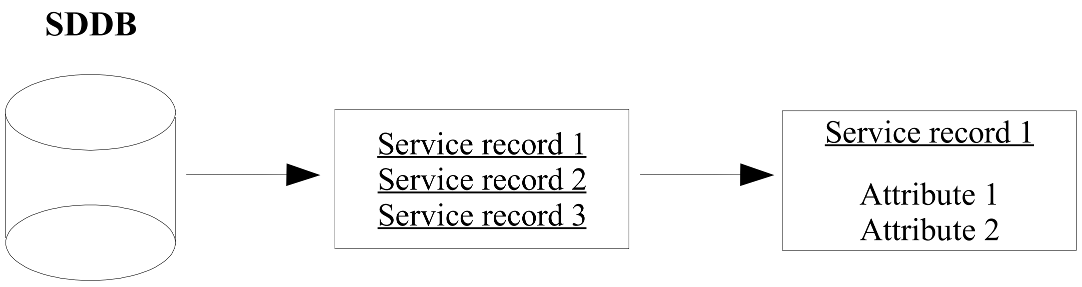


Figure 2.7 The Service Discovery DataBase (SDDB)

Remote devices can retrieve service records during service discovery and will then possess all information required to use the services described. We see from Figure 2.7 that a service record has several attributes. Each attribute is assigned an *attribute ID*, being a hexadecimal identifier. Table 2.2 shows the most common attributes' names, IDs and data types. Note that only two attributes are required to exist in a service record, the ServiceRecordHandle (attribute ID 0x0000) and the ServiceClassIDList (attribute ID 0x0001) attributes. Usually there exist several additional attributes in service records describing common Bluetooth services.

Attribute Name	Attribute ID	Attribute Value Type
ServiceRecordHandle	0x0000	32-bit unsigned integer
ServiceClassIDList	0x0001	Data Element Sequence (of UUIDs)
ServiceRecordState	0x0002	32-bit unsigned integer
ServiceID	0x0003	UUID
ProtocolDescriptorList	0x0004	Data Element Sequence (of UUIDs and protocol-specific parameters) or Data Element Alternative

Attribute Name	Attribute ID	Attribute Value Type
BrowseGroupList	0x0005	Data Element Sequence (of UUIDs)
LanguageBaseAttributeIDList	0x0006	Data Element Sequence (of language parameters for supported languages)
ServiceInfoTimeToLive	0x0007	32-bit unsigned integer
ServiceAvailability	0x0008	8-bit unsigned integer
BluetoothProfileDescriptorList	0x0009	Data Element Sequence (of UUIDs)
DocumentationURL	0x000A	URL
ClientExecutableURL	0x000B	URL
IconURL	0x000C	URL

Table 2.2 Service record attributes

Different attributes contain values of various types and sizes. To cope with this, *data elements* are used for storing values. A data element consists of a data element type descriptor and a data field as seen in Figure 2.8. The data element type descriptor contains information about the type and size of the data and the data field contains the actual data. A remote device will then know what kind of data and how much data it is receiving when retrieving a service attribute.

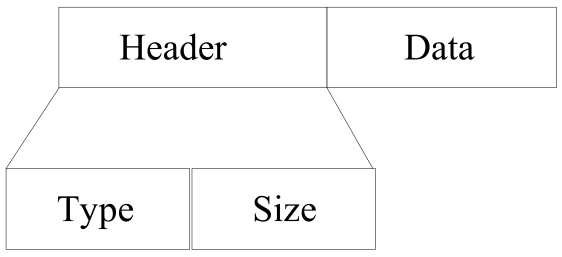


Figure 2.8 Data element construct

The Universally Unique IDentifier (UUID), [13, p. 345], is the data type used for identifying services, protocols and profiles etc. A UUID is a 128-bit identifier that is guaranteed to be unique across all time and space. The Bluetooth technology uses different variants of UUIDs, short UUIDs and long UUIDs, to reduce the burden of storing and transferring 128-bit UUID values. A range of short UUID values has been pre-allocated for often-used services, protocols and profiles, and is listed in the *Bluetooth Assigned Numbers* document on the Bluetooth Membership website [14].

More details about attributes can be found in the Bluetooth Specification 1.1 [13, Part E], and in the book by Bray and Sturman [12, Ch. 11].

2.6 Bluetooth profiles

Bluetooth profiles provide a well defined set of higher layer procedures and uniform ways of using the lower layers of Bluetooth. The profiles guide developers on how to implement a given end-user functionality using the Bluetooth system. This section is based on [3].

The profiles released with the Bluetooth specification version 1.1 are called foundation profiles. Table 2.3 gives an overview and a short description of these profiles.

Profile	Description
Generic Access Profile (GAP)	The basis for all profiles in the Bluetooth system. The GAP defines basic Bluetooth functionality like setting up L2CAP links, handling security modes and discoverable modes
Serial Port Profile (SPP)	Provides serial port (RS-232) emulation based on the RFCOMM part of the Bluetooth stack
Dial Up Networking Profile (DUNP)	Defines functionality for using a Bluetooth device as a Dial Up Networking gateway
FAX Profile	Defines functionality for using a Bluetooth device as a FAX gateway
Headset Profile	Defines the functionality required to do audio transfer with e.g. a wireless Bluetooth headset
LAN Access Point Profile	Defines functionality for using a Bluetooth device as a LAN access point
Generic Object Exchange Profile (GOEP)	Provides support for the OBjext EXchange (OBEX) protocol over Bluetooth links
Object Push Profile	Defines functionality for exchanging vCard and vCalendar objects, based on the GOEP
File Transfer Profile	Defines functionality for navigating through folders and copying/deleting/creating a file or folder on a Bluetooth device, based on the GOEP
Synchronization Profile	Defines functionality for synchronizing Object Stores containing IrMC objects (vCard, vCalendar, vMessaging and vNotes objects) between Bluetooth devices, based on the GOEP

Profile	Description
Intercom Profile	Enables Bluetooth devices to establish a direct communication link similar to intercom communcation
The Cordless Telephony Profile	Enables Bluetooth devices to act as regular cordless phones communicating with e.g. an ISDN gateway

Table 2.3 Bluetooth foundation profiles

Using profiles ensure interoperability between different devices from different Original Equipment Manufacturers (OEMs). Consumers should be able to buy a cellphone from one vendor and a headset from another and have them working nicely together assuming that both devices implement the headset profile. New profiles are defined continuously by Bluetooth SIG Working groups.

2.7 Bluetooth qualification

This section is based on [12, Ch. 23-24]. New Bluetooth products cannot use the Bluetooth brand for marketing purposes before the products have passed the Bluetooth qualification program. This is to ensure interoperability between Bluetooth devices. When a product has passed this qualification program consumers can be sure that the product will work with other qualified Bluetooth products. The Bluetooth Qualification website [16] contains information for companies who wish to get their Bluetooth devices qualified. The requirements for qualification is split into four categories:

- Bluetooth radio link requirements
- Bluetooth protocol requirements
- Bluetooth profile requirements
- Bluetooth information requirements

Qualification tests are carried out on samples of a Bluetooth product. Three levels of Bluetooth qualification are used to ensure that a Bluetooth product meets the qualification requirements:

- Qualification testing to ensure conformance with the Bluetooth core specification
- Interoperability testing to ensure that devices work with one another at the profile level
- Checking documentation to ensure it conforms to the Bluetooth brand book

In addition to the qualification of sample products, all Bluetooth products have a *test mode* which is used to test that the radio performance of the real products conform with the samples used for regulatory and qualification testing.

2.8 Bluetooth security

Security is important when communicating without wires. If your device is discoverable, anyone in the vicinity can do a device discovery and find your Bluetooth device. They may determine which services your device is offering and try to connect to them. Another problem is eavesdropping, which can be done very easily when communicating without wires. In order to handle these threats, the Bluetooth specification defines a security model based on three components: authentication, encryption and authorization. In addition, three *security modes* are defined, enforcing different levels of security. A *security manager* is used to handle the security transactions in the Bluetooth system.

2.8.1 Security modes

Security modes are part of the GAP profile. All qualified Bluetooth devices must have an implementation of the GAP profile, hence all Bluetooth devices will have implemented a security mode. The OEM must decide which security mode to support when implementing the GAP profile on a Bluetooth device. On more powerful devices such as a laptop computer, the user may have the option to select the desired security mode. The ability to select security modes is available in e.g. the Bluetooth software accompanying 3COM USB Bluetooth devices. The GAP defines three security modes:

1. No security
2. Service level enforced security
3. Link level enforced security

In security mode 1, devices will never initiate any security procedure. Support for authentication is optional. This security mode is not seen in many devices at the time of writing, it was probably used in early Bluetooth devices.

Security mode 2 is the security mode used for the majority of Bluetooth devices. Security is enforced at the service level, hence the service decides whether security is required or not. Note that in service mode 2 security procedures are initiated by the higher Bluetooth layers after the Bluetooth link is created by the lower layers. This enables developers to create services and decide if a service should require security. Security mode 2 will be discussed further in Section 2.8.6.

In security mode 3, security procedures are initiated during the setup of a Bluetooth link. If security measures fail, the link setup will fail. Observe that security procedures are initiated by the lower layers of the Bluetooth stack in security mode 3. Application developers have no influence on the security settings when setting up a Bluetooth link. Security mode 3 is useful for Bluetooth devices which have factory preset settings and is not configurable by the user, e.g. Bluetooth headsets.

2.8.2 Pairing and bonding (authentication)

Bonding is the procedure of a Bluetooth device authenticating another Bluetooth device, and is dependent on a shared authentication key. If the devices do not share an authentication key, a new key must be created before the bonding process can complete. Generation of the authentication key is called *pairing*. The pairing process involves generation of an initialization key and an authentication key, followed by mutual authentication. The initialization key is based on user input, a random number and the Bluetooth address of one of the devices. The user input is referred to as a Personal Identification Number (PIN) or *passkey* and may be up to 128-bits long. The passkey is the shared secret between the two devices. The authentication key is based on random numbers and Bluetooth addresses from both devices. The initialization key is used for encryption when exchanging data to create the authentication key, and is thereafter discarded. When the pairing process is completed, the devices have authenticated each other. Both devices share the same authentication key, often called a *combination key* since both devices have contributed to the creation of the key.

When two devices have completed the pairing process they may store the authentication key for future use. The devices are then *paired* and may authenticate each other through the bonding process without the use of a passkey. Devices will stay paired until one device requests a new pairing process, or the authentication key is deleted on either of the devices. Storing the authentication key is useful for devices frequently connecting to each other, such as a laptop computer frequently connecting to the dial-up networking service on a cellphone. The bonding procedure can then complete without user input and the user is relieved of figuring out a new passkey every time he or she wants to connect to the Internet.

2.8.3 Encryption

When two devices have authenticated each other encryption may be requested for the Bluetooth link by either of the devices. Before encryption can begin, the devices must negotiate encryption mode and key-size for the encryption key. There are three encryption modes:

- no encryption
- encrypt both point-to-point and broadcast packets
- only encrypt point-to-point packets

When only two devices are connected, the *point-to-point packets encryption* mode is a natural choice. The *no encryption* mode will only be selected if either of the devices do not support encryption. When encryption has been requested and both devices support encryption, the size of the encryption key is negotiated. The master device will then suggest its largest supported key-length. The slave device may then accept or reject this key-length. If the slave accepts, all is well and encryption may be started. If the slave rejects, the master can suggest a shorter key-length or decide to terminate the connection. This procedure is repeated until the devices agree on a key-length or the master decides to terminate the link. Key-lengths from 8-128 bits are supported for encryption keys. This is due to export restrictions from the U.S. to some countries.

2.8.4 Authorization

Authorization is the process of giving a remote Bluetooth device permission to access a particular service. In order to be authorized the remote device must first be authenticated through the bonding process. Access may then be granted on a temporary or a permanent basis. The *trust* attribute is related to authorization, linking authorization permissions to a particular device. A trusted device may connect to a Bluetooth service, and the authorization process will complete successfully without user interaction. This means that the previously mentioned user with the laptop computer and cellphone may completely avoid user interaction with the cellphone when connecting to the Internet. By marking the laptop computer as a trusted device on the cellphone, the laptop computer may be authorized automatically when connecting to the dial-up networking service on the cellphone.

2.8.5 Security manager

In order to keep track of which devices are trusted and the different levels of authorization for different services, security information needs to be stored in security databases. Two databases are used, one for devices and one for services. Several layers need access to these security databases. The security manager allows uniform access to the security databases for all layers and is responsible for entering and extracting information from the security databases. Hence, all exchange of information from the different layers and the security databases goes through the security manager. Applications and protocols must register with the security manager in order to use security features.

Other important tasks handled by the security manager are to query the user for a passkey during the pairing process and query the user for an authorization response when a remote device tries to connect to a service that requires authorization. The security manager must also provide an user interface to configure security settings on the device.

2.8.6 Security mode 2

The Bluetooth security white paper [17] defines a security architecture which may be used to implement security mode 2 service level enforced security. Device security levels are defined, splitting devices into three security categories:

- *Trusted devices* are bonded devices marked as trusted in the device database, and can be given unrestricted access to all services.
- *Known untrusted devices* are bonded devices not marked as trusted in the device database. Access to services may be restricted.
- *Unknown devices* are not paired. These devices are untrusted and access to services may be restricted.

The security white paper also defines service security levels by splitting services into three security categories:

- *Open services* with no security requirements. Any device can access these.
- *Authentication-only services* accessible to any bonded device.
- *Authentication and authorization servic*es accessible to trusted devices only.

When working with recent smartphones these categories are recognized in the menus where devices can be bonded, and trust can be granted. This indicates that OEMs use the Bluetooth security white paper when implementing security mode 2. Note that with this implementation the user may not have as fine-grained control as he or she may wish. It could be of interest to mark a device as trusted, but give it access to only a subset of services. This is not possible, a trusted device will have access to all services. The interested reader can download the Bluetooth security white paper from the Bluetooth SIG website [14].

3 Java 2 Micro Edition (J2ME)

This chapter gives an overview of the J2ME technology. The J2ME architecture is described in general before the components in the J2ME technology are introduced. J2ME applications are also discussed in general, and it is explained how they are made available to end users. Finally, JABWT is discussed, showing where it has its place in the J2ME architecture.

J2ME is a highly optimized Java runtime environment. J2ME is aimed at the consumer and embedded devices market. This includes devices such as cellular telephones, Personal Digital Assistants (PDAs) and other small devices.

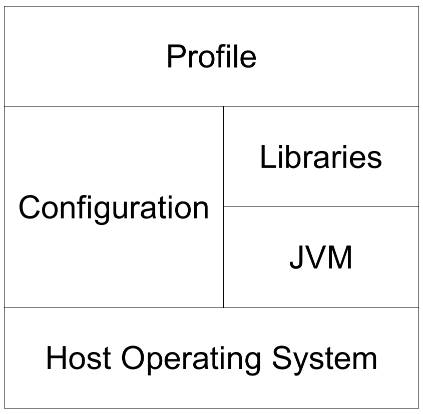


Figure 3.1 High level view of J2ME

Figure 3.1 shows the J2ME architecture. Java 2 Standard Edition (J2SE) developers should be familiar with Java Virtual Machines (JVMs) and at least one host Operating System (OS). *Profiles* and c*onfigurations* are introduced in J2ME and will be outlined in Section 3.1.

The OS will vary on different mobile devices. Some devices run the Symbian OS [18], others run some other OS developed by the manufacturer. It is therefore up to the manufacturers to implement a JVM for their specific platform compliant with the JVM Specification and Java Language Specification.

3.1 Configurations and profiles

Mobile devices come with different form, features and functionality, but often use similar processors and have similar amounts of memory. Therefore *configurations* were created, defining groups of products based on the available processor power and memory of each device. A configuration outlines the following:

- The Java programming language features supported
- The JVM features supported
- The basic Java libraries and Application Programming Interfaces (APIs) supported

There are two standard configurations for the J2ME at this time, Connected Device Configuration (CDC) and Connected Limited Device Configuration (CLDC). The CDC is targeted toward powerful devices like Internet TVs and car navigation systems. The CLDC is targeted toward less powerful devices like mobile phones and PDAs. The vast majority of Java enabled mobile devices available to consumers today use CLDC. The CDC will therefore not be discussed in this thesis. The interested reader can find more information about CDC on Sun Microsystems' CDC product website [19].

A *profile* defines a set of APIs which reside on top of a configuration and offers access to device specific capabilities. The Mobile Information Device Profile (MIDP) is a profile to be used with the CLDC and provides a set of APIs for use by mobile devices. These APIs include classes for user interface, persistent storage and networking. The MIDP is outlined in Section 3.3. Specifications, APIs and other J2ME related information can be found on Sun Microsystems' J2ME website [20].

3.2 Connected Limited Device Configuration (CLDC)

The CLDC is the result of a Java Community Process [21] expert group JSR 30 [22] consisting of a number of industrial partners.

The main goal of the CLDC Specification is to standardize a highly portable minimum-footprint Java application development platform for resource-constrained, connected devices.

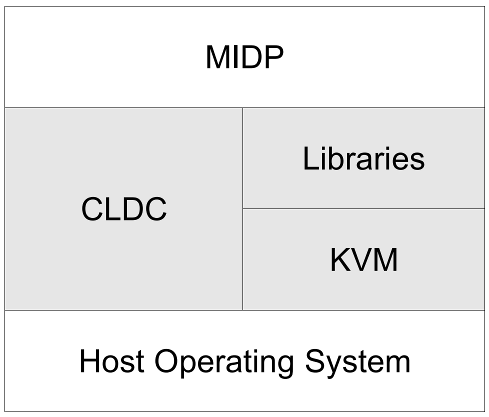


Figure 3.2 CLDC position in J2ME architecture

Figure 3.2 shows that CLDC is core technology designed to be the basis for one or more profiles. CLDC defines a minimal subset of functionality from the J2SE platform. Hence, the CLDC does not define device-specific functionality in any way, but instead defines the basic Java libraries and functionality available from the Kilo Virtual Machine (KVM). The KVM got its name because it includes such a small subset of the J2SE JVM that its size can be measured in kilobytes.

It is important to note that the CLDC does not define any optional features. Hence, developers are sure their applications will work on any device with a compliant CLDC implementation.

3.2.1 Generic Connection Framework (GCF)

During development of the CLDC the familiar J2SE `java.io` and `java.net` APIs were considered to large to fit in memory of a resource constrained Mobile Information Device (MID), so the GCF was created as a replacement. As the name implies, the GCF provides a generic approach to connectivity. The GCF is used to create connections such as datagram or stream connections. JABWT makes use of the GCF when creating Bluetooth links. This way, the Java code used to create a Bluetooth link is equivalent to the Java-code used to create other types of communication links. The GCF is defined in the `javax.microedition.io` API.

3.2.2 CLDC versions and requirements

Two versions of the CLDC have been defined, version 1.0 and version 1.1. CLDC 1.1 adds a few new features over CLDC 1.0. Floating point support is the most important feature added. Several minor bugfixes have also been added. CLDC 1.1 is intended to be backwards compatible with version 1.0. Developers should note that the minimum memory requirement has been raised from 160 KB in version 1.0 to 192 KB in version 1.1 due to the added floating point support.

Package	**Provides**
`java.io`	Provides classes for input and output through data streams
`java.lang`	Provides classes that are fundamental to the Java programming language
`java.lang.ref`	Provides support for weak references
`java.util`	Contains the collection classes, and the date and time facilities
`javax.microedition.io`	Classes for the GCF

Table 3.1 CLDC packages

CLDC consists of the Java packages shown in Table 3.1. Observe that GUI-libraries and the `java.net` library are unavailable. All packages are subsets of the corresponding packages from J2SE, except the `javax.microedition.io` package which is introduced in the CDLC. `Java.io` provides basic input and output streams, but not file streams or other libraries for persistent storage. The streams from `java.io` are used with stream connections from the `javax.microedition.io` package. Note that user interface, networking support and persistent storage are addressed by the MIDP.

One does not usually develop programs based solely on the packages provided by the CLDC since they only provide the most basic functionality from J2SE. Downloading a java program based solely on the CLDC to a cellular phone would actually be impossible. These devices only support applications based on MIDP, which implies the use of CLDC. Hence, we use the CLDC and MIDP in combination.

Specifications, APIs and other CLDC-related information are available at Sun Microsystems' CLDC product website [23].

3.2.3 CLDC security

The security model of the CLDC is defined at three different levels, low-level security, application-level security and, end-to-end security [24]. Low-level security ensures that the application follows the semantics of the Java programming language. It also ensures that an ill-formed or maliciously encoded class file does not crash or in any other way harm the target device. In a standard Java virtual machine implementation this is guaranteed by a class file verifier, which ensures that the bytecodes and other items stored in class files cannot contain illegal instructions, cannot be executed in an illegal order, and cannot contain references to invalid memory locations or memory areas outside the Java object memory. However, the conventional J2SE class verifier takes a minimum of 50 kB binary code space and typically at least 30-100 kB of dynamic Random Access Memory (RAM) at runtime. This is not ideal for small, resource constrained devices. Because of this, a different approach is used for class file verification in CLDC. Class files are preverified off-device, usually on the workstation used by the developer to compile the applications. The preverification process will add some information to the classes, making runtime verification much easier. The result is that the implementation of the class verifier in Sun's KVM requires about 10 kB of Intel x86 binary code and less than 100 bytes of dynamic RAM at runtime for typical class files.

Application-level security means that the application will run in the CLDC sandbox-model. The application should only have access the resources and libraries permitted by the Java application environment. This means that the application programmer must not be able to modify or bypass the standard class loading mechanisms of the virtual machine. The CLDC sandbox model also requires that a closed, predefined set of Java APIs is available to the application programmer, defined by the CLDC, profiles (e.g. MIDP) and manufacturer-specific classes. The application programmer must not be able to override, modify, or add any classes to the protected `java.*`, `javax.microedition.*`, profile-specific or manufacturer-specific packages.

End-to-end security usually requires a number of advanced security solutions (e.g. encryption and authentication). The CLDC expert group decided not to mandate a single end-to-end security mechanism. Therefore, all end-to-end security solutions are assumed to be implementation dependent and outside the scope of the CLDC specification.

3.3 Mobile Information Device Profile (MIDP)

The MIDP is a set of APIs that resides on top of the CLDC as shown in Figure 3.3, providing features such as user interface, networking support and persistent storage. Two version of the MIDP exist at the time of writing, MIDP 1.0 and MIDP 2.0. They will both be outlined in this chapter.

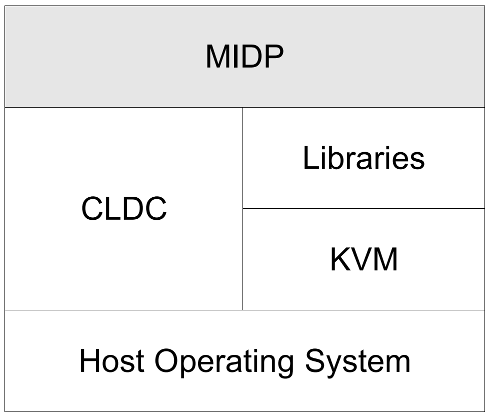


Figure 3.3 MIDP position in J2ME architecture

Specifications, APIs and other MIDP-related information are available at Sun Microsystems' MIDP website [25].

3.3.1 MIDP version 1

The MIDP version 1.0a is the result of the work carried out by a Java Community Process expert group, JSR 37 [26], consisting of a number of industrial partners. The MIDP 1.0a specification defines the architecture and the associated APIs needed for application development for mobile information devices. The MIDP targets MIDs. To be classified as a MID, a device should have the minimum characteristics listed in Table 3.2:

Display:	Pixels: 96x54
Display depth:	1-bit
Pixel shape (aspect ratio):	approximately 1:1
Input:	One- or two-handed keyboard or touch screen
Memory:	128 KB of non-volatile memory for the MIDP components 8 KB of non-volatile memory for application-created persistent data 32 KB of volatile memory for the Java runtime environment
Networking:	Two-way, wireless, possibly intermittent, with limited bandwidth

Table 3.2 MIDP 1.0 requirements

The MIDP adds a few packages on top of the CLDC, shown in Table 3.3:

Package	**Provides**
`javax.microediton.lcdui`	Provides classes for user interface
`javax.microedition.midlet`	Defines MIDP applications and the interactions between the application and the environment in which the application runs
`javax.microedition.rms`	Provides persistent storage (Record Management System)

Table 3.3 MIDP 1.0 packages

Since these packages are added on top of the CLDC, the MIDP API will also include all CLDC packages. It is worth noting that the MIDP adds a few extra interfaces and classes to existing packages in the CLDC. One of these is the `HttpConnection` interface which gives a framework for HTTP connections, by extending the functionality in the GCF. The `TimerTask` class in `java.util` is another example. Developers should therefore use the MIDP API when programming for MIDs, thus having access to all classes and interfaces provided by both the CLDC and the MIDP.

Application management in terms of fetching, installing, selecting, running and removing MIDlets is not specified by the MIDP 1.0a. These issues are handled by the Application Manager, which is implemented in a device specific way by the OEM. Hence, application management is handled in a device specific way. Note that application management is specified in the new MIDP 2.0 specification.

The MIDP 1.0a relies on the security model of the CLDC and specifies no additional security features except the semantics implied by the MIDP application model. The CLDC security model takes care of sufficient low-level and application-level security. Hence, neither the CLDC or the MIDP 1.0a addresses end-to-end security. This was first introduced in MIDP 2.0.

3.3.2 MIDP version 2

The MIDP version 2.0 is the result of the Java Community Process expert group JSR-118 [27]. The MIDP 2.0 specification defines an enhanced architecture and the associated APIs needed for application development for mobile information devices. The specification is based on the MIDP 1.0 specification, providing backwards compatibility so that MIDlets written for MIDP 1.0 can execute in MIDP 2.0 environments.

Display:	Pixels: 96x54
Display depth:	1-bit
Pixel shape (aspect ratio):	approximately 1:1
Input:	One- or two-handed keyboard or touch screen
Memory:	256 KB of non-volatile memory for the MIDP components 8 KB of non-volatile memory for application-created persistent data 128 KB of volatile memory for the Java runtime environment

Networking:	Two-way, wireless, possibly intermittent, with limited bandwidth
Sound:	The ability to play tones, either via dedicated hardware or via software algorithm

Table 3.4 MIDP 2.0 requirements

Table 3.4 shows that requirements for display, input and networking are the same as for MIDP 1.0. Memory requirements have been raised in the MIDP 2.0 specification. There must be 256 KB of non-volatile memory for the MIDP implementation, beyond what is required for the CLDC and 128 KB of volatile memory for the Java runtime. Requirements for sound have been added. The ability to play tones is now made a requirement.

MIDP 2.0 is backwards compatible with MIDP 1.0, hence it provides all functionality defined in the MIDP 1.0 specification. In addition it provides Over-The-Air (OTA) provisioning. This feature was left to OEMs to provide in the MIDP 1.0 specification. An enhanced user interface has been defined, making applications more interactive and easier to use. Table 3.5 shows the packages provided by MIDP 2.0.

Package	**Provides**
`javax.microediton.lcdui`	Provides classes for user interface
`javax.microedition.midlet`	Defines MIDP applications and the interactions between the application and the environment in which the application runs
`javax.microedition.rms`	Provides persistent storage (Record Management System)
`javax.microedition.lcdui.game`	Provides functionality useful for game development
`javax.microedition.media`	Provides the Audio Building Block (ABB)
`javax.microedition.pki`	Provides functionality for handling certificates

Table 3.5 MIDP 2.0 packages

Media support has been added through the ABB, giving developers the ability to add tones, tone sequences and WAV files even if the Mobile Media API (MMAPI) optional package is not available.

Game developers now have access to a Game API providing a standard foundation for building games. This API takes advantage of native device graphic capabilities.

MIDP 2.0 adds support for HTTPS, datagram, sockets, server sockets and serial port communication.

Push architecture is introduced in MIDP 2.0. This makes it possible to activate a MIDlet when the device receives information from a server. Hence, developers may develop event driven applications utilizing carrier networks. An example of this could be a SMS MIDlet, which would be activated when a new incoming SMS arrived at the device.

End-to-end security is provided by HTTPS and SSL/TLS protocol access over the IP (Internet Protocol) network. The ability to set up secure connections is a leap forward for MIDP programming. A wide range of application models require encryption of data and may now utilize the security model of MIDP 2.0 based on open standards.

3.4 MIDlets

MIDP applications are called MIDlets. Even though Figure 3.4 defines MIDlets as applications built using the MIDP and CLDC only, one usually also refer to OEM-specific applications as MIDlets. MIDlets are usually distributed in MIDlet suites, available on the Internet through WAP.

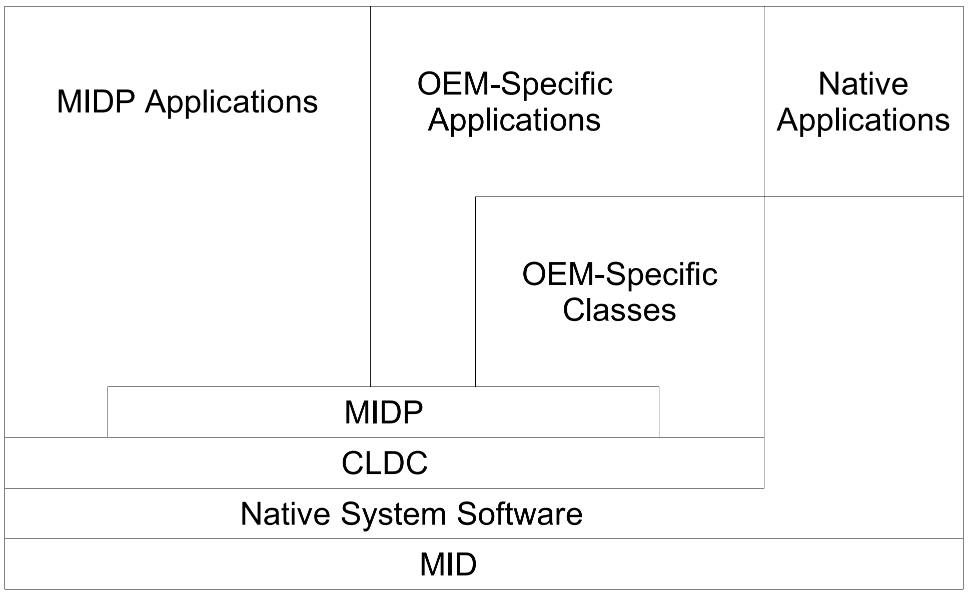


Figure 3.4 MIDlet architecture overview

The MIDP defines an application model to allow the limited resources of the device to be shared by multiple MIDlets. The application model defines the following:

- What a MIDlet is
- How it is packaged
- Runtime environment available to the MIDlet
- The MIDlet's behavior so that the device can manage its resources
- Packaging of MIDlets forming a MIDlet-suite

3.4.1 OEM-specific applications

OEM-specific applications rely on OEM-specific classes. Both Nokia and Sony Ericsson have their own classes for user interface and device specific functionality. E.g. vibration is provided through these specific classes. One of the main advantages of MIDlets is their portability. If you use OEM-specific classes you sacrifice this portability. Using Nokia classes will for certain give you an error if you run your MIDlet on a Sony Ericsson device.

3.4.2 MIDlet suites

MIDlets are usually available through MIDlet suites. A MIDlet suite consists of two files, a .jar and a .jad file. The Java ARchive (JAR) file contains compiled classes in a compressed and preverified format. Several MIDlets may be included in a MIDlet suite. Hence, the JAR file will contain all these MIDlet classes. This enables multiple MIDlets to share resources, like common libraries included in the MIDlet suite or data stored on the device. Because of security constraints, a MIDlet may only access the resources associated with its own MIDlet suite. This applies to all resources, such as libraries it may depend on or data stored on the MID.

The Java Application Descriptor (JAD) file is a plain text file containing information about a MIDlet suite. All MIDlets must be named in this file, the size of the JAR file must be included (and be correct!) and the URL to the JAR file must be present. In addition, the MIDlet suite version number is included here. This is essential information for a MID. The MID will always download the JAD file first and inspect its contents. If the MIDlet suite is already installed, it will know if a newer version is available. The size of the JAR file is important information, the MID can determine if there is enough memory available to install the MIDlet suite. If all is well the MID can go to the supplied URL and download the JAR file. Other attributes may be included as well. Midlet vendor and other information may be included on a nice-to-know basis. The MIDP specifications list available attributes and can be downloaded from the respective JSR web pages [26], [27].

3.4.3 MIDlet deployment

MIDlet suites can be deployed to a webserver and made available for download. Deploying to a webserver is the most common method for making MIDlet suites widely available, but MIDlet suites may also be transferred to a MID using e.g. a Bluetooth connection or a cable connection. The Java Bluetooth applications described later in this thesis are deployed to the author's webserver and may be downloaded through a HTTP or WAP enabled browser.

Figure 3.5 shows that a mobile device may connect to the Internet by using a GSM or GPRS connection. A MIDlet can then be downloaded from a webserver on the Internet, simply by entering the URL to the desired MIDlet's .jad file in a HTTP or WAP browser.

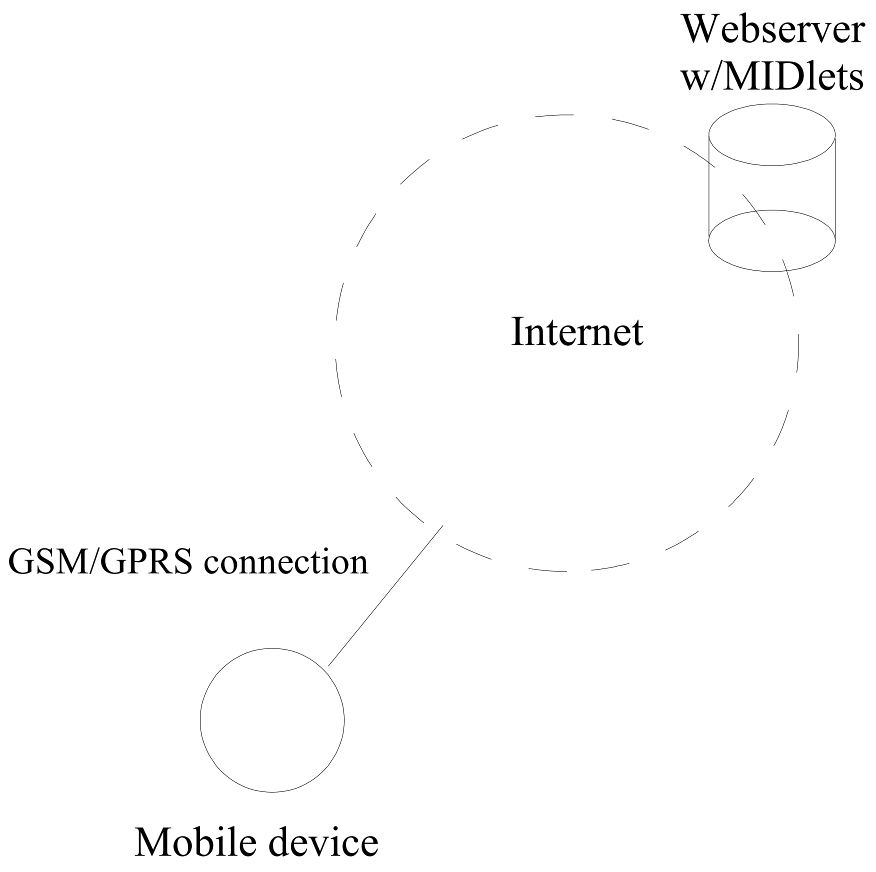


Figure 3.5 MIDlet deployment

3.5 Java APIs for Bluetooth Wireless Technology (JABWT)

JABWT was defined by a Java Community Process expert group JSR-82 [28]. The JABWT specification defines an optional J2ME package for Bluetooth wireless technology.

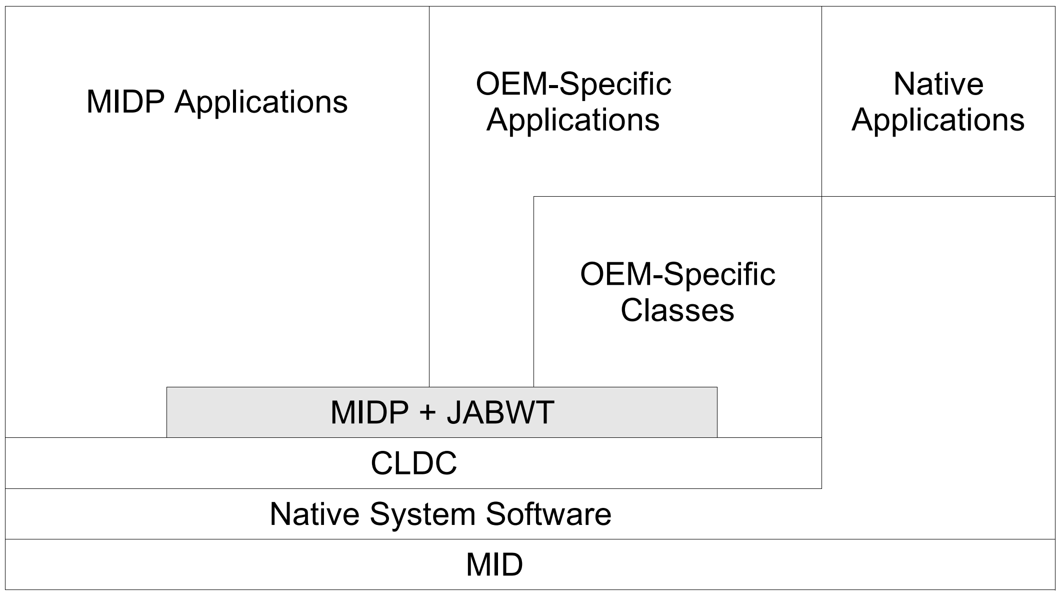


Figure 3.6 JABWT position in J2ME architecture

Figure 3.6 shows that JABWT operates on top of the CLDC and is intended to extend the capabilities of profiles like the MIDP. JABWT use the GCF, defined in the CLDC specification, for Bluetooth communication. JABWT consists of two packages, listed in Table 3.6.

Package	Provides
`javax.bluetooth`	The core Bluetooth API.
`javax.obex`	The Object Exchange (OBEX) API.

Table 3.6 JABWT packages

These packages are separate optional packages so the CLDC implementation may include either of the packages or both of them.

The `javax.bluetooth` package provides an API for device discovery and service discovery (see Section 2.4). In addition it provides functionality for setting up services of your own and customization of local service records. Setting up L2CAP and RFCOMM connections is available through an extension to the GCF from the CLDC.

The `javax.obex` package provides an API for the OBject EXchange (OBEX) protocol. This package is not implemented on the Nokia 6600 smartphone. It is not tied to the Bluetooth API alone but is intended to be of more general use. At the time of writing the `javax.obex` package is not implemented on any available smartphones, hence it will not be discussed in this thesis.

3.5.1 Security

This section is based on [7, Sec. 3.2]. The JABWT specification itself does not define any security models. It depends on the security models available through the Bluetooth stack. However, it does define how JABWT should interact with the lower layers in the Bluetooth stack responsible for security features. The device must have a Bluetooth Control Center (BCC) to which JABWT applications can direct their security requests. The BCC is the central authority for local Bluetooth device settings. It controls security settings and provides lists of devices both known and trusted by the local device. The BCC is responsible for pairing devices and providing authorization for connection request. All of these functions must be included in the BCC. It is not clear what the relationship between the BCC and the Bluetooth security manager is. A natural assumption is that the BCC relies on the Bluetooth security manager to carry out security related actions. The security manager is discussed in Section 2.8.5.

The BCC may have other capabilities like setting the Bluetooth friendly-name of the device, setting timeouts for the baseband layer, determining how connectable and discoverable modes are set, resetting the local device or enumerating services on the local device. This BCC functionality is implementation dependent and may vary between OEMs an their devices. Some implementations may provide a GUI to the BCC while others provide hard-coded defaults in the BCC. For example, a headset will provide only defaults in the BCC since it does not have an input device or screen. When sending a request to the BCC one should always check if the request was fulfilled by the BCC. One is not guaranteed that the BCC can fulfill the request at the given time.

4 Infrastructure

Software developers have a choice of platform and development tools. This chapter describes the infrastructure needed to develop Java Bluetooth applications. In the initial face of writing this thesis, available Java Bluetooth application development tools were explored. It was not clear how JABWT would be made available in consumer devices, since there actually were no such devices available at the time. Several vendors, such as Atinav and Rococo amongst others, offered JABWT implementations and Bluetooth stacks implemented in Java. A decision had to be made on which stack to use. During the fall of 2003 the Nokia 6600 smartphone was released as the first device implementing JABWT. Shortly after, the Sony Ericsson P900 smartphone was released being the second device on the market providing JABWT.

4.1 Linux workstation

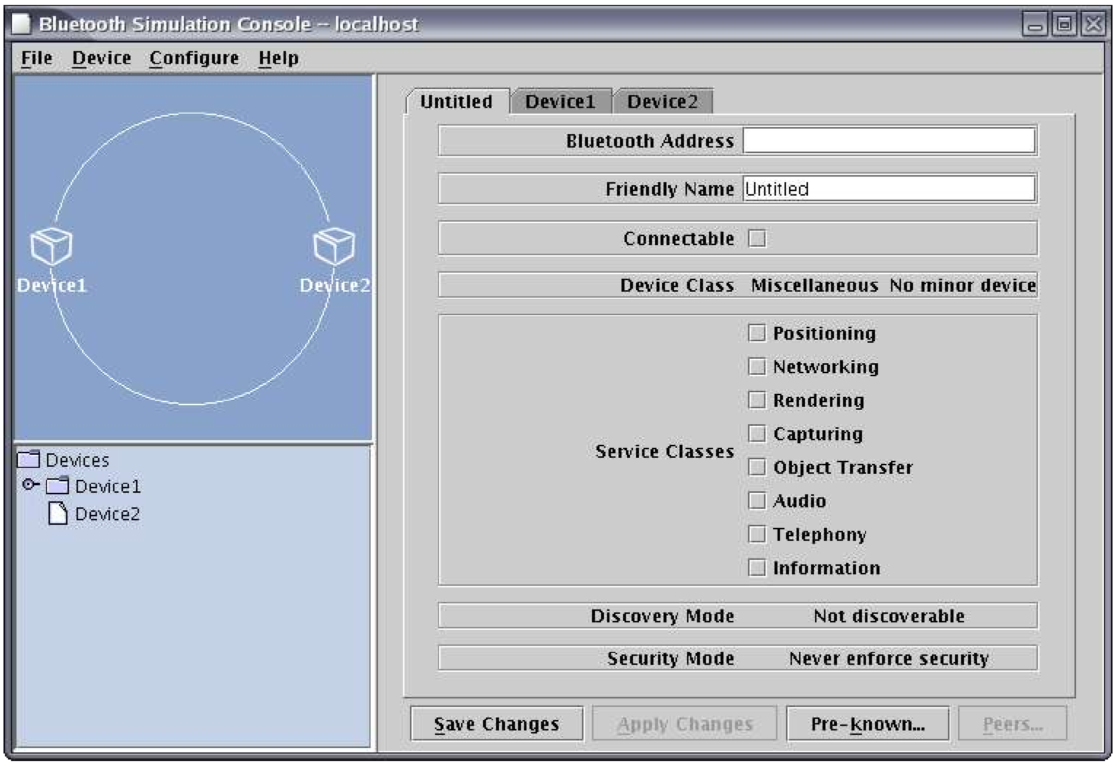
Linux is the OS of choice as Java development platform for the NoWires research group [29]. There are several Linux distributions to choose from and most of them are available for free. During the process of selecting development tools for Java Bluetooth programming it came clear that all necessary development tools for J2ME application development were available for the Linux platform, as well as the Microsoft Windows platform. RedHat Linux 9 has been the platform for all the work with this thesis and have produced a series of how-to documents explaining how to get a Bluetooth USB dongle working, installing development tools etc. The interested reader should consult the website related to this thesis [30].

4.2 Sun wireless toolkits

Sun provides Wireless ToolKits (WTKs) which enables development, preverification and packaging of MIDlet suites. There are toolkits available for both MIDP 1.0 and MIDP 2.0 application development. These toolkits include emulators, enabling developers to test their applications before they are deployed to a J2ME enabled device. In addition, simple graphical tools are available for debugging and building MIDlet suites. Several Integrated Developer Environments (IDEs) support the WTKs, such as NetBeans, JBuilder and Eclipse. If the WTKs are used from an IDE, MIDlet suites may be built, preverified and packaged automatically. This is highly recommended in the long run. The WTKs are available for download at Sun Microsystems' wireless toolkit web pages [31].

4.3 Rococo Impronto simulator

The Impronto simulator is developed by Rococo Software Ltd. and can be purchased through their website [32]. The Impronto simulator provides a simulation of a complete Bluetooth environment and enables development of Java Bluetooth applications without the need for Bluetooth hardware. The simulator supports both J2ME and J2SE applications, enabling developers to have J2ME and J2SE Java applications communicating with each other. J2ME applications run in the emulator from the Sun wireless toolkit 1.04 integrate with the simulator without problems. Bluetooth MIDlets may therefore be fully developed and extensively tested before on-device testing commences. It is important to note that no emulator or simulator can replace on-device testing, but they can save developers a lot of time during the initial development and testing phases. Simulated Bluetooth devices are visualized in the simulator, as shown in Figure 4.1.



Bluetooth Address	
Friendly Name	Untitled
Connectable	☐
Device Class	Miscellaneous No minor device
Service Classes	☐ Positioning ☐ Networking ☐ Rendering ☐ Capturing ☐ Object Transfer ☐ Audio ☐ Telephony ☐ Information
Discovery Mode	Not discoverable
Security Mode	Never enforce security

Figure 4.1 Screenshot of Impronto Simulator

Simulated devices are easily configurable, and device configurations may be saved for future use. This way, developers may create simulated devices with a variety of capabilities and settings for use in application testing.

4.4 Smartphones

During the work with this thesis two smartphones were used to test JABWT applications, the Nokia 6600 and the Sony Ericsson P900. These were the first two mobile phones to support JABWT.

4.4.1 Nokia 6600

The Nokia 6600 was the first smartphone to be released with support for JABWT. It was also one of the first mobile phones to support MIDP 2.0 Java applications. The Nokia 6600 has a 104 MHz ARM processor, 6 MB internal memory and a 32 MB Multimedia Memory Card (MMC). These specifications show what consumers can expect from future smartphones.

Figure 4.2 The Nokia 6600

During the work with this thesis JABWT applications were deployed to the 6600 using Bluetooth links. However, the 6600 does provide HTTP connectivity by both GSM and GPRS, and includes both a WAP browser and the Opera web-browser. Java applications can therefore also be downloaded from the Internet.

4.4.2 Sony Ericsson P900

The Sony Ericsson P900 smartphone was the second smartphone with JABWT support available on the market. It features a 156 MHz ARM processor, 16 MB internal memory and a 32 MB Memory Stick Duo memory card. The P900 also provides support for MIDP 2.0 applications. As with the Nokia 6600, Java applications may be deployed to the P900 using Bluetooth links, or via the Internet using GSM or GPRS connections. The P900 includes a WAP browser and a XHTML browser.

Figure 4.3 The Sony Ericsson P900

JABWT was initially not available in the P900. Through Sony Ericsson's software update service, the P900 software can be upgraded. In the latest releases of the P900 software, JABWT is available.

4.5 Web server, www.klings.org/nowires/

In order to make MIDlet suites available on the Internet, a web server is needed. Fortunately, the author was already running a Linux powered web server hosting his personal domain when starting the work on this thesis. With some minor configuration changes the web server could be used to publish MIDlet suites. The web-server is running the Fedora Core 1 Linux distribution, with Apache as the web server software. The web pages describing this thesis [30] are also served from this server. The interested reader can download the sample applications discussed in Chapter 6 from these web pages.

The web pages have become quite popular, especially the collection of How-Tos describing how to install and use various software needed to develop Java Bluetooth applications. At the time of writing the average number of unique visits per month is nearly 2000. In April 2004 the Bluetooth browser application was downloaded 58 times. There are postings from several developers on multiple developer forums referring to the web pages, generating increased traffic to the website. This confirms that there is an audience for this thesis.

4.6 IDEs

During the work with this thesis several IDEs have been explored. Many IDEs exist for J2SE development, but it was not clear how many IDEs were usable for J2ME development. Three IDEs were explored, NetBeans, Jbuilder and Eclipse.

The NetBeans IDE [33] is easy to use and provides all the basic functionality expected by an IDE. Code completion, organizing projects and CVS (Concurrent Versions System) [34] integration are provided. One of the really helpful features of NetBeans is its ability to show JavaDoc for suggested methods during code completion. There were no problems using the Impronto simulator with NetBeans. The update feature was also a nice feature, enabling easy updates and easy installation of new modules to the IDE. Integrating the WTK1.x with NetBeans was very easy thanks to the automatic installation procedure. Unfortunately the WTK2.x was not available for automatic download and installation, and was hard to integrate manually. Some instability was experienced with the IDE, it would sometimes hang. NetBeans was not the most responsive IDE either, so after a while other IDEs were explored.

Borland JBuilder [35] is great software. The JBuilder IDE was tested for a 30-day trial period. With the JBuilder Mobile Edition add-on package a developer has everything he or she needs for developing J2ME applications. There were no problems using the Impronto simulator with the JBuilder IDE. The only downside with Borland JBuilder Mobile edition is that it is quite expensive. Due to a limited research budget a switch to the Eclipse IDE was made.

The Eclipse IDE [36] is also great software. It is open-source and freely available. Using native GUI libraries, it gives the developer a much better look-and-feel than the other IDEs. Eclipse is a framework for Java development, meaning that its functionality can be extended by the use of plugins. One suitable plugin was found for J2ME application development during the work with this thesis, the EclipseME plugin [37]. With the EclipseME plugin installed, MIDlet suites can be created easily with the Eclipse IDE. Unfortunately there is a bug in the EclipseME plugin at the time of writing, which will make Eclipse start the J2ME emulator without external libraries supplied as an argument to the emulator. This will cause a problem, since the emulator needs an external Bluetooth library to communicate with the Impronto simulator. To work around this problem, the emulator can be started from command-line with correct arguments. The author has filed a bug report to the EclipseME developer so this issue will hopefully be resolved in the near future. MIDlets which do not rely on external libraries are not affected by this issue.

5 J2ME and JABWT programming

This chapter will give a thorough description on how to program with JABWT and J2ME. Code samples are provided throughout the chapter, aiming to show the reader how a complete Bluetooth MIDlet is built. This chapter is partly based on [7]. The code samples in this chapter are simplified as much as possible, highlighting the Bluetooth specific Java code instead of describing both J2ME and JABWT issues simultaneously. The event-driven nature of J2ME applications tend to raise the complexity of the source code, making it difficult to understand the structure of the application. The simplified code samples provided in this chapter should make it easier for the reader to understand JABWT programming. It is assumed that the reader is familiar with J2ME programming. Readers who lack general knowledge about J2ME programming should consult [38].

The first code samples provided in this chapter show the structure of a Bluetooth MIDlet. Functionality will be added to these code samples as new functionality is explained. Note that these code samples are not fully functional MIDlets. However, they can be used as a starting point for a complete application. Method declarations have font typeface bold to increase the readability of the code.

5.1 Structure of Bluetooth MIDlet

This section shows the structure in a Java/Bluetooth MIDlet. Several MIDlet examples are available on Sun Microsystems' Mobility website [39].

Usually an event-driven MIDlet with no Bluetooth support looks like this:

```
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Displayable;
import javax.microedition.midlet.MIDlet;

public class YourMidlet extends MIDlet implements CommandListener {

        public void startApp() {

        }

        public void pauseApp() {
        }

        public void destroyApp(boolean unconditional) {

        }

        public void commandAction(Command c, Displayable d) {

        }
}
```

The first three methods, `startApp()`, `pauseApp()` and `destroyApp()` are needed for any MIDlet. They come from extending the MIDlet class. The next method, `commandAction()` comes from the `CommandListener` interface. This is needed to catch command events. The MIDlet is extended to support Bluetooth communication in the next code sample.

During device discovery and service discovery, events will be delivered to a `DiscoveryListener` object when devices or services are found or the device discovery or service discovery is completed. An object implementing the `DiscoveryListener` interface is used to catch these events. The MIDlet will then look like this:

```
import javax.bluetooth.DiscoveryListener;
import javax.bluetooth.DeviceClass;
import javax.bluetooth.ServiceRecord;
import javax.bluetooth.RemoteDevice;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Displayable;
import javax.microedition.midlet.MIDlet;

public class YourMidlet extends MIDlet implements CommandListener,
                                              DiscoveryListener {

    public void startApp() {

    }

    public void pauseApp() {

    }

    public void destroyApp(boolean unconditional) {

    }

    public void commandAction(Command c, Displayable d) {

    }
    public void deviceDiscovered(RemoteDevice remoteDevice,
                                 DeviceClass deviceClass) {

    }

    public void inquiryCompleted(int param) {

    }

    public void serviceSearchCompleted(int transID, int respCode) {

    }

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

    }
}
```

The last four methods, `deviceDiscovered()`, `inquiryCompleted()`, `serviceSearchCompleted()` and `servicesDiscovered()` are used to catch events during device discovery and service discovery. Device discovery and service discovery will be outlined in the two next sections.

5.2 Device discovery (Inquiry)

Device discovery, introduced in Section 2.4, is the first step required when browsing nearby Bluetooth devices. When we have discovered nearby devices we can find out which services they offer. Note that no UI specific code is included in the following examples, only Bluetooth specific code.

To use any Bluetooth related methods you need to obtain a reference to the `LocalDevice` object by calling the `LocalDevice.getLocalDevice()` method. The obtained `LocalDevice` object gives access to Bluetooth properties for the device, such as the Bluetooth address, friendly name and discovery mode. We will use the `LocalDevice` object to obtain a `DiscoveryAgent` object. The `DiscoveryAgent` object is used for device discovery and service discovery.

The MIDlet now looks like this:

```
import java.util.Vector;
import javax.bluetooth.BluetoothStateException;
import javax.bluetooth.DiscoveryAgent;
import javax.bluetooth.DiscoveryListener;
import javax.bluetooth.LocalDevice;
import javax.bluetooth.DeviceClass;
import javax.bluetooth.ServiceRecord;
import javax.bluetooth.RemoteDevice;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Displayable;
import javax.microedition.midlet.MIDlet;

public class yourMIDlet extends MIDlet implements CommandListener,
                                                  DiscoveryListener {


    private LocalDevice local = null;
    private DiscoveryAgent agent = null;

    private Vector devicesFound = null;

    public void startApp() {

        /*  Add your MIDlet specific code here.
         *  You probably want to show the user
         *  a welcome screen.
         *  The call to doDeviceDiscovery() is
         *  here for the example's sake. You
         *  should call doDeviceDiscovery() when
         *  the user actively asks for it.
         */


        doDeviceDiscovery();
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }

    public void commandAction(Command c, Displayable d) {

    }

    public void deviceDiscovered(RemoteDevice remoteDevice,
                                 DeviceClass deviceClass) {

    }

    public void inquiryCompleted(int param) {

    }

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

    public void serviceSearchCompleted(int transID, int respCode) {
    }
    private void doDeviceDiscovery() {

        try {
            local = LocalDevice.getLocalDevice();
        }catch (BluetoothStateException bse) {

            // Error handling code here
        }

        agent = local.getDiscoveryAgent();

        devicesFound = new Vector();

        try {

            if(!agent.startInquiry(DiscoveryAgent.GIAC,this)) {

                // Inquiry not started, error handling code here
            }
        }catch(BluetoothStateException bse) {

            // Error handling code here
        }
    }

}
```

Device discovery is started using the `LocalDevice` and `DiscoveryAgent` objects. Observe that the `doDeviceDiscovery()` method is called in the `startApp()` method. Searching with the GIAC parameter will find devices which are general discoverable (see Section 2.4). The `DiscoveryListener` parameter is `this`, meaning our MIDlet. When devices are discovered or the search is complete, events will be delivered to our MIDlet. Note that the `startInquiry()` method returns immediately, returning `true` if the device discovery was initiated or `false` if the device discovery process was not started. An event will be delivered to the MIDlet when the device discovery is completed. This is important to take into account when designing the flow of execution in the MIDlet.

The `deviceDiscovered()` and `inquiryCompleted()` methods are used to catch events related to the device discovery process. When a device is discovered the `deviceDiscovered()` method of the object `this` will be called. The parameter `remoteDevice` will be the discovered device, it is up to us to decide what to do with it. Note that we do not know how many devices will be discovered. A `Vector` will therefore be an appropriate data structure to save discovered devices.

The `inquiryCompleted()` method is called when the inquiry ends. The status code supplied in the parameter `param` should always be checked. The complete code for device discovery follows:

```
import java.util.Vector;
import javax.bluetooth.BluetoothStateException;
import javax.bluetooth.DiscoveryAgent;
import javax.bluetooth.DiscoveryListener;
import javax.bluetooth.LocalDevice;
import javax.bluetooth.DeviceClass;
import javax.bluetooth.ServiceRecord;
import javax.bluetooth.RemoteDevice;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Displayable;
import javax.microedition.midlet.MIDlet;

public class yourMIDlet extends MIDlet implements CommandListener,
                                                  DiscoveryListener {

    private LocalDevice local = null;
    private DiscoveryAgent agent = null;

    Vector devicesFound = null;

    public void startApp() {

        /* Add your MIDlet specific code here.
         * You probably want to show the user
         * a welcome screen.
         * The call to doDeviceDiscovery() is
         * here for the example's sake. You
         * should call doDeviceDiscovery when
         * the user actively asks for it.
         */


        doDeviceDiscovery();
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }

    public void commandAction(Command c, Displayable d) {

    }

    public void deviceDiscovered(RemoteDevice remoteDevice,
                                 DeviceClass deviceClass) {

        devicesFound.addElement(remoteDevice);
    }

    public void inquiryCompleted(int param) {

        /* We should give the user an alert based on the
         * inquiry status code
         */
        switch (param) {

            case DiscoveryListener.INQUIRY_COMPLETED:

                /* Inquiry completed normally, add appropriate code
                 * here
                 */

                break;

            case DiscoveryListener.INQUIRY_ERROR:

                // Error during inquiry, add appropriate code here.

                break;

            case DiscoveryListener.INQUIRY_TERMINATED:

                /* Inquiry terminated by agent.cancelInquiry()
                 * Add appropriate code here.
                 */

                break;
        }
    }

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

    public void serviceSearchCompleted(int transID, int respCode) {
    }

    private void doDeviceDiscovery() {

        try {
            local = LocalDevice.getLocalDevice();
        }catch (BluetoothStateException bse) {

            // Error handling code here
        }

        agent = local.getDiscoveryAgent();

        devicesFound = new Vector();

        try {

            if(!agent.startInquiry(DiscoveryAgent.GIAC,this)) {

                // Inquiry not started, error handling code here
            }
        }catch(BluetoothStateException bse) {

            // Error handling code here
        }
    }
}
```

Discovered devices are kept in the `DevicesFound` vector by adding them as they are discovered. When our search ends, we check if everything went as expected and can alert the user by adding appropriate code in our switch-statement.

5.3 Service search

After device discovery is completed it is time to find out which services are offered by the discovered devices. This is accomplished by doing a service discovery on the device of interest.

The `servicesDiscovered()` and `serviceSearchCompleted()` methods must be implemented. They will handle the events occuring when services are found or the service discovery completes. In addition, the `doServiceSearch()` method has been added to show how a service discovery is initiated. This method will start a service discovery on the `RemoteDevice` supplied as a parameter, and is called in the `inquiryCompleted()` method.

The complete example Bluetooth MIDlet looks like this:

```
import java.util.Vector;
import javax.bluetooth.UUID;
import javax.bluetooth.BluetoothStateException;
import javax.bluetooth.DiscoveryAgent;
import javax.bluetooth.DiscoveryListener;
import javax.bluetooth.LocalDevice;
import javax.bluetooth.DeviceClass;
import javax.bluetooth.ServiceRecord;
import javax.bluetooth.RemoteDevice;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Displayable;
import javax.microedition.midlet.MIDlet;


public class YourMIDlet extends MIDlet implements CommandListener,
                                                  DiscoveryListener {

    private LocalDevice local = null;
    private DiscoveryAgent agent = null;

    private Vector devicesFound = null;
    private ServiceRecord[] servicesFound = null;

    public void startApp() {
        /*  Add your MIDlet specific code here.
         *  You probably want to show the user
         *  a welcome screen.
         *  The call to doDeviceDiscovery() is
         *  here for the example's sake. You
         *  should call doDeviceDiscovery() when
         *  the user actively asks for it.
         */

        doDeviceDiscovery();
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }

    public void commandAction(Command c, Displayable d) {

    }

    public void deviceDiscovered(RemoteDevice remoteDevice,
                                 DeviceClass deviceClass) {

        devicesFound.addElement(remoteDevice);
    }

    public void inquiryCompleted(int param) {

        /* We should give the user an alert based on the
         * inquiry status code
         */

        switch (param) {

            case DiscoveryListener.INQUIRY_COMPLETED:

                /* Inquiry completed normally, so we
                 * initiate a service discovery on the first
                 * device found. That is, if we actually
                 * found any devices.
                 */

                 if (devicesFound.size() > 0){

                       doServiceSearch( (RemoteDevice)
                                    devicesFound.firstElement());
                 }
                break;

            case DiscoveryListener.INQUIRY_ERROR:

                //Error during inquiry, add appropriate code here

                break;
            case DiscoveryListener.INQUIRY_TERMINATED:

                /*
                 * Inquiry terminated by agent.cancelInquiry()
                 * Add appropriate code here
                 */

                break;
        }
    }

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

        /* Services discovered, keep reference to the ServiceRecord
         * array
         */

        servicesFound = serviceRecord;
    }

    public void serviceSearchCompleted(int transID, int respCode) {

            switch(respCode) {

            case DiscoveryListener.SERVICE_SEARCH_COMPLETED:

                /*
                 * Service search completed successfully
                 * Add appropriate code here
                 */

                break;

            case
               DiscoveryListener.SERVICE_SEARCH_DEVICE_NOT_REACHABLE:

                // device not reachable, add appropriate code here

                break;

            case DiscoveryListener.SERVICE_SEARCH_ERROR:

                // Service search error, add appropriate code here

                break;

            case DiscoveryListener.SERVICE_SEARCH_NO_RECORDS:

                // No records found, add appropriate code here

                break;

            case DiscoveryListener.SERVICE_SEARCH_TERMINATED:
                /*
                 * Search terminated by agent.cancelServiceSearch()
                 * Add appropriate code here
                 */

                break;
        }

    }



    private void doDeviceDiscovery() {

        try {
            local = LocalDevice.getLocalDevice();
        }catch (BluetoothStateException bse) {

            // Error handling code here
        }

        agent = local.getDiscoveryAgent();

        devicesFound = new Vector();

        try {

            if(!agent.startInquiry(DiscoveryAgent.GIAC,this)) {

                // Inquiry not started, error handling code here
            }
        }catch(BluetoothStateException bse) {

            // Error handling code here
        }
    }

    private void doServiceSearch(RemoteDevice device) {

        /*
         * Service search will always give the default attributes:
         * ServiceRecordHandle (0x0000), ServiceClassIDList (0x0001),
         * ServiceRecordState (0x0002), ServiceID (0x0003) and
         * ProtocolDescriptorList (0x004).
         *
         * We want additional attributes, ServiceName (0x100),
         * ServiceDescription (0x101) and ProviderName (0x102).
         *
         * These hex-values must be supplied through an int array
         */

        int[] attributes = {0x100,0x101,0x102};

        /*
         * Supplying UUIDs in an UUID array enables searching for
         * specific services. PublicBrowseRoot (0x1002) is used in
         * this example. This will return any services that are
         * public browseable. When searching for a specific service,
         * the service's UUID should be supplied here.
         */

        UUID[] uuids = new UUID[1];
        uuids[0] = new UUID(0x1002);

        try {
            agent.searchServices(attributes,uuids,device,this);
        } catch (BluetoothStateException e) {

            // Error handling code here
        }
    }
}
```

The `searchServices()` method will return immediately, returning a transaction ID for the service discovery. The transaction ID is used to identify the particular service discovery. When the service discovery completes, an event will be delivered to the MIDlet.

This example Bluetooth MIDlet will hopefully be of help to J2ME application developers getting started with Bluetooth programming. Study the Bluetooth Browser source code in Appendix A to see what a fully functional Bluetooth MIDlet looks like.

5.4 RFCOMM links with JABWT

After device discovery and service discovery are completed a client has all the information needed to set up a communication link to a service on the server. L2CAP links are the basis for all communication through JABWT. An L2CAP link can be set up by supplying a "btl2cap://hostname:[PSM | UUID];parameters" type of URL as parameter to the `Connector.open()` method. The PSM (Protocol/Service Mulitplexer) value is similar in function to the port number of a traditional service on an IP (Internet Protocol) network and is used by a client connecting to a server. The UUID value is used when setting up a service on a server. Care must be taken by the developer when using L2CAP links. The developer must keep track of the receive MTU (Maximum Transfer Unit) and transmit MTU and divide data into chunks so that data may be sent in packages with appropriate payload size. These concerns increase the complexity of Bluetooth application development.

Fortunately, the Bluetooth Serial Port Profile (SPP) is implemented and available through JABWT. The SPP is based on the RFCOMM protocol, providing RS-232 serial port emulation for Bluetooth links. The RFCOMM protocol is again based on L2CAP, so it is actually an L2CAP link used for data transfer. Setting up an RFCOMM connection is just as easy as setting up an L2CAP connection. The URL supplied as parameter to the `Connector.open()` is of the form:

"btspp://hostname:[CN | UUID];parameters." The CN (Channel Number) value is similar to the port number of a service on an IP network, and is used by a client connecting to a server. The UUID value is used when setting up a service on a server.

RFCOMM takes care of all the important details, like receive MTU and transmit MTU and dividing data into chunks so that the MTUs are not violated. RFCOMM should therefore be the preferred link type for most developers.

5.4.1 RFCOMM server

When creating a custom Bluetooth service, it should be assigned a unique UUID. This will ensure that clients do not confuse your service with other services. When creating a custom Bluetooth service a 128-bit UUID must be created. This can be done very easily on a Linux system. The `uuidgen -t` command will generate a 128-bit UUID based on the current system time and the 48-bit hardware address of the system's NIC. The 48-bit hardware address portion of the UUID will ensure that no one else will generate the same UUID, assuming they are generating the UUID in a similar way. The 80-bit portion of the UUID which is based on the system time will ensure that two identical UUIDs are not generated on the same system, assuming that the system time is not reset. Note that the `uuidgen -t` command will generate UUIDs with the following format: f3d68400-b4c7-11d8-80a2-000bdb544cb1. JABWT does not support UUIDs with hyphens, so if the sample UUID is to be used it must have the following format: f3d68400b4c711d880a2000bdb544cb1. The UUID will be added to the ServiceClassIDList attribute (attribute ID 0x0001) by the JABWT implementation.

Setting up an RFCOMM server is fairly easy, as shown by the following code:

```
LocalDevice local = null;
StreamConnectionNotifier server = null;
StreamConnection conn = null;

String connectionURL =
      "btspp://localhost:393a84ee7cd111d89527000bdb544cb1;"
    + "authenticate=false;encrypt=false;name=RFCOMM Server";

/*
 * Make sure the device is discoverable, else clients cannot
 * find our service
 */

try {
        local = LocalDevice.getLocalDevice();
        local.setDiscoverable(DiscoveryAgent.GIAC);
} catch (BluetoothStateException e) {

        // Error handling code here
}

/*
 * First get a StreamConnectionNotifier. It is used to get
 * the service record created for us, so we can manipulate
 * the service record.
 */

try {
    server = (StreamConnectionNotifier)
              Connector.open(connectionURL);
} catch (IOException e1) {

        // Error handling code here

}

/*
 * acceptAndOpen() will register the service record in the
 * Bluetooth SDDB and block until a client connects.
 */

try {
        conn = server.acceptAndOpen();
} catch (IOException e2) {

        // Error handling code here

}
```

This is actually all the code needed to create a service record with UUID: 393a84ee7cd111d89527000bdb544cb1 and support for RFCOMM links. Additional parameters have been specified, neither authentication or encryption are enforced. Connection parameters are outlined in Section 5.4.3. The service will have service name "RFCOMM Server," this will be added in attribute ID 0x100 in the service record. All other attributes are set automatic by the implementation. When a client connects to the service, the server can obtain input and output streams from the `StreamConnection` object `conn` and communication can begin. The Bluetooth benchmark application in Appendix B shows what a complete JABWT server MIDlet may look like.

5.4.2 RFCOMM client

Clients will typically do an inquiry and service discovery before connecting to a server. The example service from Section 5.4.1 can be found by specifying "393a84ee7cd111d89527000bdb544cb1" as the UUID to search for during service discovery.

When the desired service is found, a connection can be created with very little effort:

```
/*
 * The service we want to connect to has been found earlier
 * (by service discovery) and the service record is
 * referencedthrough the object named: service (of type
 * ServiceRecord)
 */

StreamConnection conn = null;

/*
 * The connection URL is extracted from the service record.
 * Authentication and encryption is disabled. The client does
 * not require to be the master of the connection (the false
 * parameter)
 */

String connectionURL = service.getConnectionURL(
                      ServiceRecord.NOAUTHENTICATE_NOENCRYPT,
                      false);

/*
 * A StreamConnection is obtained from the connection URL
 */

try {
     conn = (StreamConnection) Connector.open(connectionURL);
} catch (IOException e) {
        // Error handling code here

}

/*
 * The conn object is now a working StreamConnection, from
 * which input/output streams can be obtained, enabling
 * communication.
 */
```

The sample code shows just how little code is needed to set up a client connection. To see how this fits into a complete MIDlet you should have a look at the source code for the benchmark MIDlets found in Appendix B.

5.4.3 RFCOMM connection parameters

Several parameters may be included in the connection URL when connecting to a Bluetooth service or creating a Bluetooth service. Security settings are configured with these parameters. Setting the ServiceName attribute and controlling master/slave switches are also done with these parameters. The parameters are listed in Table 5.1 along with their values.

Name	**Valid values**	**Client or server**
authenticate	true, false	Both
encrypt	true, false	Both
authorize	true, false	Server
Name	Any valid string	Server
master	true, false	Both

Table 5.1 RFCOMM connection parameters

The authenticate, encrypt, and authorize parameters are related to security, see Section 2.8. If these parameters are not included in the connection URL, they are implicit set to false. Setting the authenticate parameter to true will force authentication of the remote device before the connection is made. Encryption can only be initiated if both devices have authenticated each other. Therefore, the authentication parameter does not have to be set int the connection URL if the encrypt parameter is set to `true`. The same applies to the authorize parameter, since authorization also requires authentication. In the following server connection URL the service name will be BTService, the remote device will be authenticated and authorized, but the other parameters will be set to false:

"btspp://localhost:393a84ee7cd111d89527000bdb544cb1;authorize=true;name=
BTService"

If the sample service also required encryption and to be the master of the connection, the connection URL would be:

"btspp://localhost:393a84ee7cd111d89527000bdb544cb1;encrypt=true;authorize=true
;name= BTService;master=true"

Clients can only use the authenticate, encrypt and master parameters. Usually clients build their connection URLs with the `getConnectionURL()` method, which has two parameters. One is the security settings and the other is the master setting.

Note that there are combinations of parameters that are invalid. Setting authenticate to false and encrypt to true will give an error.

5.5 Service records and JABWT

This section describes how developers handle service records with the JABWT and is closely related to Section 2.5 and partly based on [7, Sec. 7.3]. Service records are fetched during service discovery in order to obtain detailed information about particular services. When creating a Bluetooth service and making it available to the world, developers may want to manipulate the service attributes manually in order to give clients sufficient information about the service. Handling service records is not trivial for the unexperienced developer. The examples provided in books on Java Bluetooth programming and on the Internet are usually quite complex. This section aims to explain service record usage in-depth and also provide code samples for advanced service record usage. Only JABWT specific code is shown in the code samples, not complete multi-threaded MIDlet structures. Hopefully, this will make it easier for developers to understand the essentials of service record usage with JABWT.

5.5.1 Retrieving information from service records with JABWT

Every service available on a Bluetooth device is described by a service record. A Bluetooth device maintains service records for all its services in the SDDB. When a client searches for services, the relevant service records from the SDDB are returned to the client. The client may then explore the different attributes of the returned records to find out more about each service. Note that in-depth knowledge about service records is not necessary to start developing JABWT applications since the JABWT implementation automatically creates basic service records for services. However, creating customized service records requires knowledge about how data is stored internally in service records. The *Bluetooth Assigned Numbers* document available at the Bluetooth SIG website [14] is very useful when working with service records. The BluetoothServiceRecordCanvas.java source file in Appendix C shows how the most common service attributes are retrieved from service records.

Data types for service record attributes

Attributes contain different data types such as integers or string values. The data is represented using a data element construct, as outlined in Section 2.5. JABWT provides a `DataElement` class, which has a similar structure to the data element construct described in the Bluetooth Specification. When fetching the value of a specific attribute, a `DataElement` object is returned by JABWT. Detailed information about service record attributes can be found in the Bluetooth specification [13, p. 366]. See Table 2.2 for a list of common service attributes.

Default attributes

Some attributes are retrieved by default by JABWT when fetching a service record. These default attributes are: the ServiceRecordHandle, ServiceClassIDList, ServiceRecordState, ServiceID, and ProtocolDescriptorList.

The ServiceRecordHandle (attribute ID 0x0000) uniquely identifies the service record in the SDDB. It is similar to a primary key in an ordinary database. This makes it possible to run several instances of the same service.

The ServiceClassIDList (attribute ID 0x0001) is a list of the serviceclasses the service belongs to.

The ServiceRecordState (attribute ID 0x0002) is a counter which is updated every time the service record changes. This enables clients to confirm that a cached service record is still valid.

The ServiceID (attribute ID 0x0003) is a 128-bit ID identifying a particular instance of the service. This attribute is useful if the same service is described in more than one SDP server.

The ProtocolDescriptorList (attribute ID 0x0004) is a list of protocols supported by your service. If your service supports e.g. RFCOMM this attribute will contain both L2CAP (0x0100) and RFCOMM (0x0003).

With these attributes in place a client has enough information to connect to a service. The `getConnectionURL()` method available in the `ServiceRecord` class will use the mandatory attributes to build a valid connection URL.

Optional attributes

In addition to the default attributes, several optional attributes may exist in the retrieved service record. Three interesting attributes are the ServiceName (attribute ID 0x100), ServiceDescription (attribute ID 0x101) and ProviderName (attribute ID 0x102), which are represented as strings in `DataElement` objects. If they are present in the service record they may be retrieved quite easily. The following example shows how to retrieve the ServiceName attribute.

```
/*
 * ServiceRecord sr is obtained during service discovery,
 * now we just extract a value from it.
 */

String name = null;

DataElement elm = sr.getAttributeValue(0x100);

/*
 * Always check for null here. If the attribute is not
 * set, null will be returned. Also check the datatype
 * set in the header of the DataElement to make sure that
 * the DataElement really contains a string.
 */

if (elm != null && elm.getDataType() == DataElement.STRING) {
        name = (String) elm.getValue();
}
```

To get the ServiceDescription or ProviderName attributes, just switch the hex value attribute ID used in this example. This was a small and simple example. Look at the source code in BluetoothServiceRecordCanvas.java in Appendix C to see how to handle more complex attributes.

5.5.2 Manipulating service records with JABWT

When creating Bluetooth services the JABWT implementation will handle the creation of basic service records automatically. However, developers may require more control of the data contained in a service record. This section gives an introduction to service record manipulation.

A server creates a Bluetooth service by calling the `Connector.open()` method with a Bluetooth specific connection URL as parameter. The `Connector.open()` method returns a `StreamConnectionNotifier` object, which in turn is used to actually start the service. A service record is created automatically for the service and is editable when the service record is created, but before the service is actually started. The JABWT implementation will initialize the mandatory attributes of the service record automatically. Only one optional attribute, the ServiceName attribute (attribute ID 0x100), can be supplied in the connection URL. All other optional attributes must be set manually. The Server.java source code in Appendix B shows how to set several of the optional attributes.

Setting optional attributes

Service search can be done using the PublicBrowseRoot value (0x1002) and will then reveal all services which are browseable. This way, searching for a specific service is not required. Clients may instead find several available services during service discovery. The PublicBrowseRoot value must be manually added to the BrowseGroupList (attribute ID 0x1005) attribute in the service record created by the JABWT implementation. The following code sample will show how to do this.

```
StreamConnectionNotifier server = null;
ServiceRecord sr = null;

String conURL =
    "btspp://localhost:1b34b730983d11d8898d000bdb544cb1;"
        + "authenticate=false;encrypt=false;name=BTDemoApp";
```

```
/*
 * When creating a StreamConnectionNotifier a service record
 * will automatically be created for us, describing the new
 * service. The service will have the Serial Port (0x1101)
 * value in the ServiceClassIDList (id 0x0001) attribute. The
 * service record will also have both the L2CAP (0x0100) and
 * RFCOMM (0x0003) values in the ProtocolDescriptorList (id
 * 0x0004) attribute. Other mandatory attributes will be set
 * automatically by the JABWT implementation. The optional
 * ServiceName (id 0x100) attribute will be set to the name
 * parameter, "BTDemoApp" in this case.
 */

try {

        server = (StreamConnectionNotifier)
                     Connector.open(conURL);

} catch (IOException e1) {

        // Error handling code here

}

/*
 * The automatically created service record can be obtained
 * from the LocalDevice object, using the reference to the
 * StreamConnectionNotifier.
 */

try {
        sr = local.getRecord(server);
}
catch (IllegalArgumentException iae){

        // Error handling code here

}

/*
 * We create a new DataElement and set its content correctly.
 */

DataElement elm = null;

/*
 * Setting public browse root in the browsegrouplist
 * attribute. The BrowseGroupList (id 0x0005) attribute
 * contains a DataElement sequence, which in turn contains
 * DataElements with UUIDs. The DataElement sequence must be
 * created before we can add DataElements to it.
 */

elm = new DataElement(DataElement.DATSEQ);
elm.addElement(new DataElement(
        DataElement.UUID,new UUID(0x1002)));

/*
 * The DataElement is now prepared. It must be added to the
 * appropriate attribute ID, in this case the
 * BrowseGroupList.
 */

sr.setAttributeValue(0x0005,elm);

/*
 * Finally, the service record must be updated in our
 * LocalDevice.
 */

try {
        local.updateRecord(sr);
} catch (ServiceRegistrationException e3) {

        // Error handling code here

}

/*
 * The call to acceptAndOpen() will enter the service record
 * in the device's Service Discovery DataBase (SDBB) and
 * start the service. The service will then be browseable to
 * clients.
 */

StreamConnection conn = null;

try {
        conn = server.acceptAndOpen();
}
catch (ServiceRegistrationException sre){

        // Error handling code here

}
catch (IOException e2) {

        // Error handling code here
}

/*
 * At this point a client is connected. input and output
 * streams can be obtained from the conn object,
 * communication can begin.
 */
```

The sample code shows all the code needed to create a Bluetooth service and make it public browseable. Note that there are some concerns when doing communication with MIDlets. Communication should be done in a separate thread. This issue is not discussed further in this section. The interested reader can find some valuable links about threads on the website describing this thesis [30].

Other optional but useful attributes are the ServiceDescription (attribute ID 0x101) and ServiceProviderName (attribute ID 0x102) attributes. How to set these attributes is shown by the following code sample:

```
DataElement elm = null;

//Setting ServiceDescription
elm = new DataElement(DataElement.STRING,
                      "Bluetooth demo service");

record.setAttributeValue(0x101,elm);

//Setting ServiceProviderName
elm = null;
elm = new DataElement(DataElement.STRING,
                      "Demo provider name");

record.setAttributeValue(0x102,elm);
```

The samples provided here give an introduction to manipulation of service records. Developers who wish to do more advanced modifications to service records should consult the JABWT API available for download at the JSR-82 website [28].

5.6 Pitfalls

Care must be taken when developing Bluetooth MIDlets. A Java/Bluetooth implementation is cutting edge technology and has its flaws. During development of the sample Bluetooth MIDlets the experience was that not everything works as expected. At the time of writing the following issues apply to the Nokia 6600 and Sony Ericsson P900 smartphones.

5.6.1 RFCOMM flow control

When a RFCOMM link is set up and ready for use, it is expected that one should be able to send some data from one device to another and shut down the link without problems. It turns out that the link is shut down, possibly before the data is transfered. Hence, one way communication is not reliable over RFCOMM links. Nasty exceptions are thrown, like the "Symbian error = - 36" exception. This was discovered during development of the benchmark program, when setting up a link, transferring some data from device A to device B and then shutting down the link.

The solution to this problem is to transfer some amount of data from device A to device B and then send a byte back from device B to device A before shutting down the link. Note that the connection must flushed in order to send data.

Care must also be taken concerning the amount of data sent at once. It seems one of the buffers in use in the 6600 and P900 smartphones is 512 bytes. When trying to transfer more than 512 bytes at once, the receiver gets 512 bytes and then the connection will freeze. Sending 512 bytes, flushing, sending 1 byte back, and flushing works fine. The connection is then kept alive.

5.6.2 RFCOMM EOF

All streams have an end, they say. This is also the case for Bluetooth links. When using an `InputStream` obtained from the `StreamConnection` object you should get -1 when there is no more data in the stream. In the real world, this is not the case. You get an exception, once again the "Symbian error = - 36" exception.

One way to work around this is by first of all send the number of bytes to transfer, and then send the bytes. The other device will then know how much data it will receive and can stop reading when all data is received, thus avoiding the exception. Doing this is fairly easy, since a `DataInputStream` and `DataOutputStream` can be obtained from the `StreamConnection`. Simply send an `int` with the number of bytes you intend to transfer.

5.6.3 Removal of service records

Service records are added to the Bluetooth SDDB when you call the `acceptAndOpen()` method on the `StreamConnectionNotifier` object. This is a blocking operation and should therefore be done in a seperate thread. To remove the Service record from the Bluetooth SDDB one must call the `close()` method on the `StreamConnectionNotifier`. This is **very** important. If the MIDlet exits without doing this, the service record will stay in the Bluetooth SDDB, despite the fact that the MIDlet is dead. This may or may not have security implications, it has not been investigated just how much damage one can do with a dangling service record.

5.6.4 Populating service records

The `ServiceRecord` class provides a `populateRecord()` method. This method should make the device go on the air to retrieve additional attributes from the remote device's service record. When using this method it returned `false` every time, indicating that it could not retreive any additional attributes. This did not make sense since the device was able to retrieve these additional attributes during the initial service discovery. The conclusion is: retrieve the attributes you want in the initial service discovery. The `populateRecord()` method does not work as expected.

5.6.5 Inquiry with P900

The P900 discovers all nearby devices but the `inquiryCompleted()` method is always called with `INQUIRY_ERROR` as parameter when the inquiry completes normally. A possible explanation is that Sony Ericsson has swapped the INQUIRY_ERROR and INQUIRY_COMPLETED values. This could not be tested properly since it is not known how to force an error during inquiry.

5.6.6 ServiceRecordHandle format

An interesting issue was discovered during the development of the Bluetooth browser application. The ServiceRecordHandle is 32-bit ID which uniquely identifies each service record within an SDP server. Doing service discovery with Bluetooth tools available for Linux, this handle would look like e.g. 0x10003. The JABWT implementation reports the value 0x3000100 for the same service. This seems to be a big-endian/low-endian problem. Both Nokia and Sony Ericsson have been asked to comment on this issue. No response has been received at the time of writing.

6 Sample applications

During the work with this thesis two sample Bluetooth MIDlets were developed. It was a logical approach to first investigate the device discovery and service discovery related parts of JABWT. The result of this investigation was the *Bluetooth browser* application. Next, Bluetooth links had to be explored so a *Bluetooth benchmark* MIDlet was developed. This way, it could be investigated how Bluetooth links were set up and what transfer rates they could provide. Both the Bluetooth browser and the Bluetooth benchmark programs are available for download from the klings.org WAP site [40]. Both sample applications depend on features from the *KlingsLib* library in Appendix C. The KlingsLib library provides tools to format textual information so it can be displayed correctly on screen, print the attributes of a service record, print Bluetooth system properties, work with short and long UUIDs, and map short UUIDs to the corresponding protocol, service class or profile name. In addition, constants for all numbers in the Bluetooth Assigned Numbers document are provided. Interested developers can download the source code for both applications and the KlingsLib library from the author's website [30].

6.1 Bluetooth browser

The Bluetooth browser application can search for discoverable Bluetooth devices, search for services on discovered devices and show the content of the most common attributes used in service records. The screenshot in Figure 6.1 shows the screen users will see when starting up the Bluetooth browser.

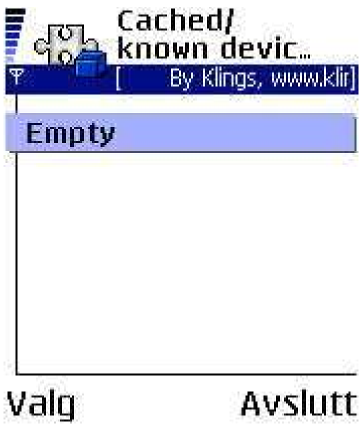


Figure 6.1 Cached/known devices

Cached and known devices is a list of devices stored by the BCC. The Bluetooth browser has not yet been on the air, so this list will only contain devices seen in the past. In this case there are no known devices. A device discovery process may be initiated through the menu, as seen in Figure 6.2.

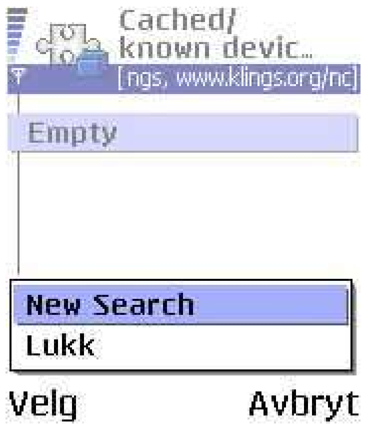
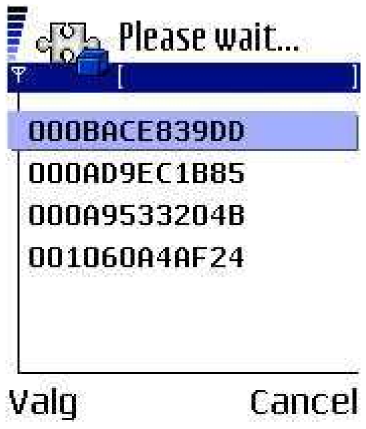


Figure 6.2 Initiating device discovery

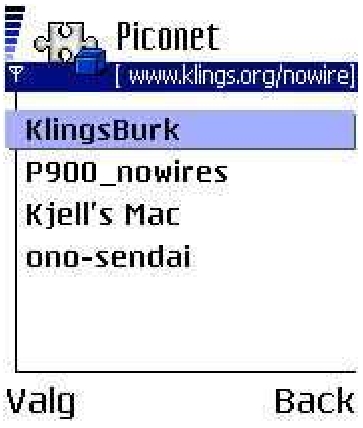
A device discovery will be started immediately when *New Search* is selected. The Bluetooth browser will then show a list of discovered Bluetooth devices, as seen in Figure 6.3. Initially this list is empty, but is populated as devices are discovered.



```
000BACE839DD
000AD9EC1B85
000A9533204B
001060A4AF24
```

Figure 6.3 Device discovery in progress

Only Bluetooth addresses will be shown in the list of discovered devices during the device discovery process. The Bluetooth friendly name must be retrieved after a device is discovered. The Bluetooth browser will retrieve friendly names for all devices after device discovery is completed. The user will then be presented a screen equivalent to the screenshot in Figure 6.4.

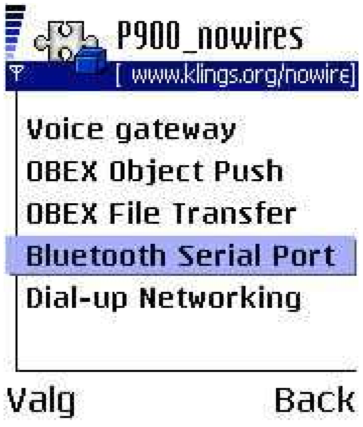


Piconet
[www.klings.org/nowire]

KlingsBurk
P900_nowires
Kjell's Mac
ono-sendai

Figure 6.4 Device discovery completed

When device discovery is completed and Bluetooth friendly names are retrieved for all available devices, the user may initiate a service discovery on a device by selecting it from the list. When the service discovery completes, the user will be shown a list of discovered services. Figure 6.5 shows the result from a service discovery on the Sony Ericsson P900 smartphone.



P900_nowires
[www.klings.org/nowire]

Voice gateway
OBEX Object Push
OBEX File Transfer
Bluetooth Serial Port
Dial-up Networking

Figure 6.5 Service search on P900

The user may now select any of the services from the list in order to investigate that service's attributes. Selecting the *Bluetooth Serial Port* service will reveal the attributes shown in Figure 6.6.

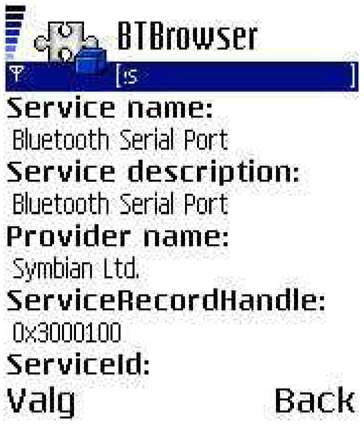

BTBrowser
[:S]
Service name:
Bluetooth Serial Port
Service description:
Bluetooth Serial Port
Provider name:
Symbian Ltd.
ServiceRecordHandle:
0x3000100
ServiceId:
Valg Back

Figure 6.6 Service attributes

Due to the limited screen size on the Nokia 6600, the user must scroll down to see the rest of the attributes, shown in Figure 6.7.

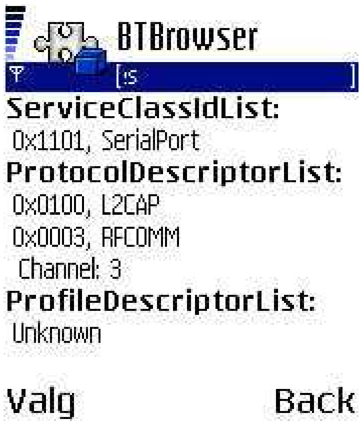

BTBrowser
[:S]
ServiceClassIdList:
0x1101, SerialPort
ProtocolDescriptorList:
0x0100, L2CAP
0x0003, RFCOMM
Channel: 3
ProfileDescriptorList:
Unknown
Valg Back

Figure 6.7 Service attributes

The Bluetooth browser enables users to search for nearby discoverable Bluetooth devices. In addition, a service discovery may be done on the discovered devices, revealing all public browseable services. Each service may be examined in detail, showing the most common service attributes.

6.2 Bluetooth benchmark

The Bluetooth benchmark applications enable users to test the throughput of RFCOMM links between Java Bluetooth enabled devices using various amounts of data. Security settings are demonstrated, users may request security features for a

Bluetooth link and verify if the request was fulfilled or not when the connection is made.

The Bluetooth benchmark MIDlet suite contains two MIDlets, a *server* MIDlet and a *client* MIDlet. The server MIDlets must be run on a discoverable Bluetooth device. The client MIDlet will first do a device discovery, showing the user a list of nearby devices. The user should then select a device from the list, initiating a service discovery on that device. If the benchmark service is found, the client will automatically connect to the service. The benchmark MIDlets use a RFCOMM link for communication, due to the fact that most developers will prefer RFCOMM over L2CAP. The benchmark MIDlets have a simple user interface. The reason for this is to keep MIDlet activity to a minimum, so that as much resources as possible are available for the communication. The highest possible transfer rate should then be achieved.

Table 6.1 shows that the transfer rate achieved varies, depending on which devices are involved in the transfer. These numbers are not affected when enabling or disabling encryption on the Bluetooth link.

Devices	Result
Nokia 6600 to Nokia 6600	10-11 KB/s
Nokia 6600 to Sony Ericsson P900	10-11 KB/s
PC to Nokia 6600	25 KB/s

Table 6.1 Benchmark results

In the PC-to-Nokia 6600 test, the benchmark MIDlet was run through the ME4SE emulator [41], using the Rococo developer kit for Linux [32] to interact with the Linux Bluez Bluetooth stack [42]. It is quite interesting that PC-to-smartphone communication gives a higher transfer rate than smartphone-to-smartphone communication. This means that the smartphones are not able to send data at the same rate as the PC. Where the bottleneck is located is difficult to say, since the details of how JABWT is implemented is not known.

6.2.1 Benchmark server

The benchmark server will create a Bluetooth service and make it available to nearby Bluetooth devices. RFCOMM is used as the communication protocol. The user can change security settings through the *Settings* menu and start the server. The screenshot in Figure 6.8 shows the main menu in the benchmark server application.

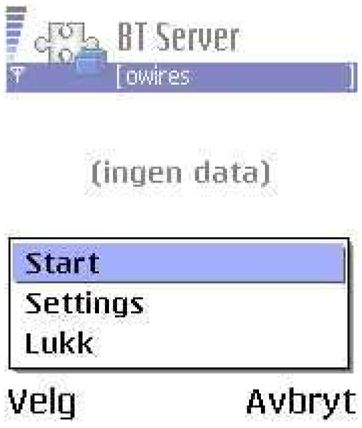


Figure 6.8 Benchmark server main menu

The user may change the settings for the connection through the *Settings* menu, shown in Figure 6.9.

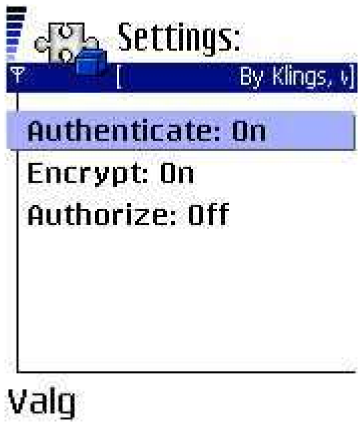
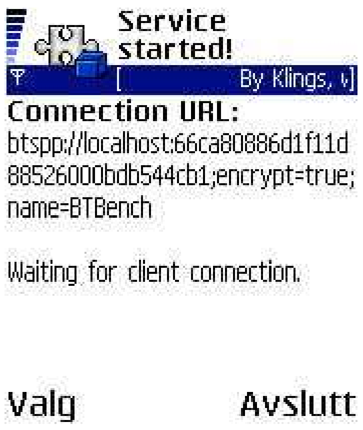

Settings:
[By Klings, v]
Authenticate: On
Encrypt: On
Authorize: Off
Valg

Figure 6.9 Benchmark server settings

The user has no option for the master parameter because the Nokia 6600 does not support master/slave switches. However, security related parameters may be configured by the user. When the settings are configured the user may start the server. A service record will be created by JABWT and entered into the SDDB. Benchmark clients may then find the service through the service discovery process and connect to the service. The user is presented the screen seen in Figure 6.10 when the server is started and is waiting for client connections.



Service started!

[By Klings, v]

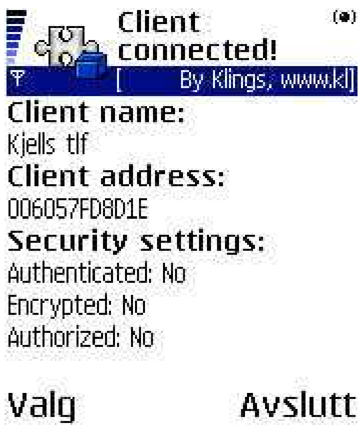
Connection URL:

btspp://localhost:66ca80886d1f11d88526000bdb544cb1;encrypt=true;name=BTBench

Waiting for client connection.

Figure 6.10 Benchmark server started

The connection URL used to create the service is printed so the user can verify that the settings are correct. The screen indicates that the server is waiting for client connections. When a client connects to the service a paring process may be initiated by the BCC if authentication is required. The BCC will query the user for a passkey if necessary, or complete the authentication process without user interaction if the devices are already paired. When the client is connected and data transfer commences a screen equivalent to Figure 6.11 is shown.



Client connected!

[By Klings, www.kl]

Client name:

Kjells tlf

Client address:

006057FD8D1E

Security settings:

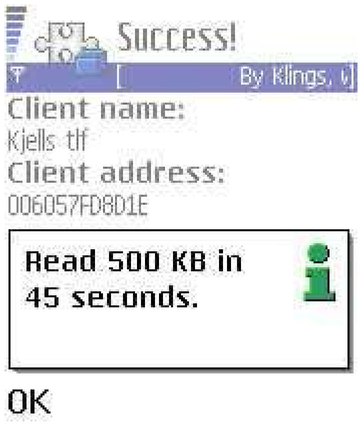
Authenticated: No

Encrypted: No

Authorized: No

Figure 6.11 Client connected to server

The user will see details about the connected client and security settings for the Bluetooth link, as reported by the BCC. When the transfer completes the user will get a summary of how much data was sent and the elapsed time, as seen in Figure 6.12.


Success!

[By Klings, w]

Client name:

Kjells tlf

Client address:

006057FD8D1E

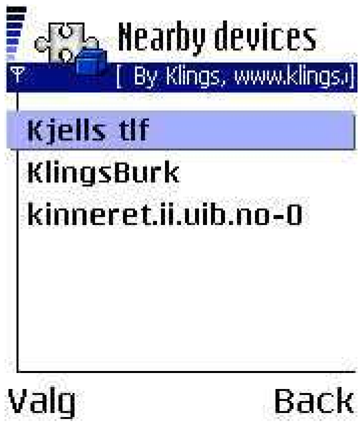
Read 500 KB in 45 seconds.

Figure 6.12 Benchmark server, transfer results

The service can be restarted if the user wants to do another benchmark.

6.2.2 Benchmark client

The benchmark client is similar to the benchmark server. Device discovery and service discovery must be done by the client before it can connect to the server. When nearby devices have been discovered, the user may select a device from a list in order to search for the benchmark service, as seen in Figure 6.13.


Nearby devices

[By Klings, www.klings.]

Kjells tlf

KlingsBurk

kinneret.ii.uib.no-0

Figure 6.13 Benchmark client, device discovery

If the selected device provides the benchmark service, the user will be notified and may configure the settings for the connection through a *Settings* screen similar to the one in the server application. A connection will be made and the data transfer will commence when the user selects the amount of data to transfer from the screen shown in Figure 6.14.

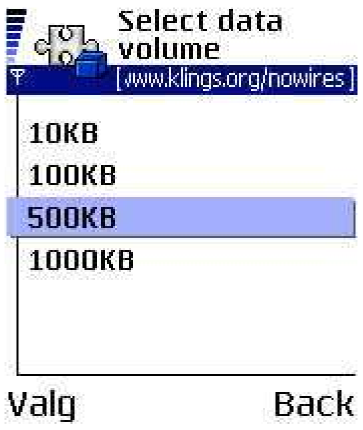

Select data volume
[www.klings.org/nowires]
10KB
100KB
500KB
1000KB
Valg
Back

Figure 6.14 Benchmark client, selection of data amount

To notify the user that the transfer is in progress, a status screen will be displayed. Figure 6.15 shows that the user is presented details about the server and the security settings for the connection.

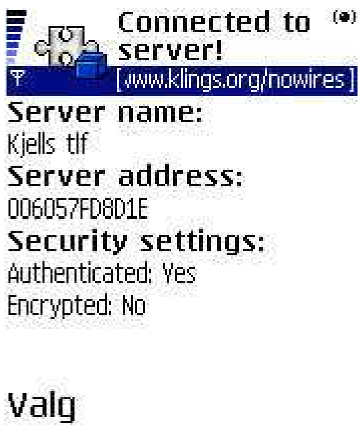

Connected to server!
[www.klings.org/nowires]
Server name:
Kjells tlf
Server address:
006057FD8D1E
Security settings:
Authenticated: Yes
Encrypted: No
Valg

Figure 6.15 Benchmark client, transfer status screen

When the transfer is finished, the user is told how much data was transferred, as shown in Figure 6.16. The server will display how much time elapsed during the data transfer.

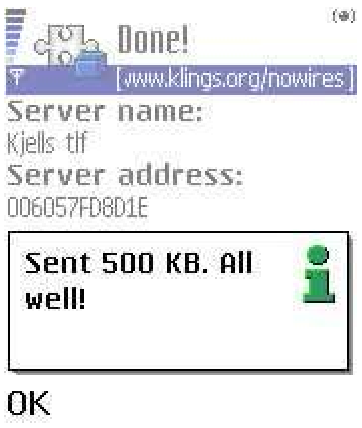

Done!
[www.klings.org/nowires]
Server name:
Kjells tlf
Server address:
006057FD8D1E
Sent 500 KB. All well!
OK

Figure 6.16 Benchmark client, transfer finished

7 Summary and conclusions

7.1 Summary

This thesis has described and evaluated Bluetooth and J2ME technologies—helping the reader understand the foundation for the JABWT technology. The Bluetooth technology is advanced, and the large and complex specification makes it hard to get a sufficient overview of the technology. The Bluetooth architecture was therefore described in general, before important Bluetooth concepts such as device discovery, service discovery, creation of services, and service record usage were discussed. The Bluetooth security model was also explained since JABWT relies on security features available in Bluetooth.

The reader was expected to be familiar with the J2ME technology. Still, an overview of the J2ME technology was given so that readers with little or no knowledge of J2ME could understand the architecture and investigate the technology further on their own. References were included to J2ME resources, suitable as starting points for further study of J2ME.

JABWT was described thoroughly. It was explained where JABWT has its place in the J2ME architecture and how all important Bluetooth operations are done through the API. Simple code samples were provided to show the structure of a Bluetooth application and make it easier for the reader to understand how the JABWT actually works. All key functionality: device discovery, service discovery, creating services, connecting to services, customizing service records, and setting security parameters for Bluetooth links were explained and documented through code samples. Additional source code is included in the appendices, showing complete Bluetooth enabled J2ME applications. Some noticeable bugs in JABWT implementations on current smartphones were explained and workarounds were suggested. Developers aware of these bugs can save a lot of time spent on debugging.

Infrastructure needed to develop JABWT applications was described. Available development tools were discussed, giving the reader an overview of which tools are available for Java Bluetooth development. It can be a time consuming task to both find and test development tools. The reader can use the tools described in this thesis to quickly start Java Bluetooth development.

Source code for sample applications using the JABWT functionality discussed in this thesis is included in the appendices and is also available from the author's website [30]. A separate library, KlingsLib, with J2ME and Java Bluetooth related tools was developed during the work with this thesis. KlingsLib, its source code and JavaDoc documentation are also available on the website. From the author's WAP pages [40], the applications can be downloaded and installed to Java Bluetooth enabled cellphones.

7.2 Conclusions

JABWT is cutting edge technology. After completing the work with this thesis it is clear that JABWT implementations have some flaws and do not always work as expected. Several of these flaws were investigated and described, and methods to avoid these problems were given. At the time of writing, extra care must be taken by Java Bluetooth developers when using JABWT due to the implementation flaws on Java Bluetooth devices. It is expected that future implementations of JABWT will be of better quality, JABWT will then provide a nice set of tools for developers working with Java Bluetooth applications.

Much of the information in this thesis is also available on the author's website [30]. The traffic to this website has increased with the amount of Java Bluetooth information published there. Now there are nearly 2000 unique visits each month to these pages and several developer forums are linking to them. J2ME developers and students around the world who are starting their work on the Java Bluetooth technology contact the author for help. This confirms that the interest for the Java Bluetooth technology is increasing, and that this thesis has a growing audience.

7.3 Further work

During the work with this thesis it was investigated how JABWT could and should be used. Some of the JABWT functionality can be a bit complex for Java developers with little knowledge of Bluetooth. Creating a new API that simplifies the use of JABWT could be of interest. There was not enough time available to look at L2CAP links, so investigating L2CAP links is also of interest.

Security is an important aspect of Java Bluetooth programming. It is of great interest to study how the BCC relates to the Bluetooth security manager. Some irregularities were found when using the security functions on the smartphones, studying these issues further would be very valuable. Looking into how the Bluetooth security manager is implemented on Bluetooth devices is also an exciting subject.

Investigating Bluetooth profiles, especially the Sim Access profile, would be interesting since it can be used for small-amount payment solutions.

Appendix A BTBrowser

BTBrowserMIDlet.java

```
/*
 * BTBrowserMIDlet.java
 *
 * Version 1.0
 *
 * 22. June 2004
 *
 * Copyright (c) 2004, Andre N. Klingsheim
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * Redistributions of source code must retain the above copyright notice, this
 * list of conditions and the following disclaimer.
 *
 * Redistributions in binary form must reproduce the above copyright notice,
 * this list of conditions and the following disclaimer in the documentation
 * and/or other materials provided with the distribution.
 *
 * Neither the name of the NoWires research group nor the names of its
 * contributors may be used to endorse or promote products derived from this
 * software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

package org.klings.j2me.BTBrowser;

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

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


import javax.bluetooth.*;
import org.klings.wireless.j2me.*;
import org.klings.wireless.BluetoothNumbers.*;


public class BTBrowserMIDlet extends MIDlet implements CommandListener,

DiscoveryListener{

      /* Reference to the Display object to do anything useful with UI*/
    private Display display = null;

    /* Reference to LocalDevice to do anything concerning Bluetooth */
    private LocalDevice local=null;

    /* Reference to a DiscoveryAgent to do inquiry or service searches */
    private DiscoveryAgent agent=null;

    /* Globally available Vectors for devices and services found */
    private Vector deviceVector = null;
    private Vector serviceVector = null;

    /* Global Ticker used for most screens in this application */
    private Ticker tic = null;

    /*
     * Global UI list accessible to deviceDiscovered(...) so we can
     * show devices and services as they are discovered, giving the user a
     * feeling of progress in the program
     */

    private List deviceList = null;
    private List serviceList = null;

    private String clientExecutableURL = null;
    private String documentationURL = null;

    private BluetoothServiceRecordCanvas rCanvas = null;

    /*
     * Global list of interesting service attributes used when populating and
     * showing service records
     */


    int[] attrs = {
            BTServiceAttributeId.SDP_SERVICENAME,
            BTServiceAttributeId.SDP_SERVICEDESCRIPTION,
            BTServiceAttributeId.SDP_PROVIDERNAME,
            BTServiceAttributeId.SDP_SERVICEINFOTIMETOLIVE,
            BTServiceAttributeId.SDP_SERVICEAVAILABILITY,
            BTServiceAttributeId.SDP_BLUETOOTHPROFILEDESCRIPTORLIST,
            BTServiceAttributeId.SDP_DOCUMENTATIONURL,
            BTServiceAttributeId.SDP_CLIENTEEXECUTABLEURL,
            BTServiceAttributeId.SDP_ICONURL,
                  };

    /* Globally available commands used for most menus */

    Command exitCommand = new Command("Exit",Command.EXIT,2);
    Command searchCommand = new Command("New Search",Command.SCREEN, 1);
    Command backCommand = new Command("Back", Command.BACK, 1);
    Command cancelCommand = new Command("Cancel",Command.CANCEL,1);
    Command openURL = new Command("Open URL",Command.CANCEL,2);

    /* Integer to keep track of which menu is active */
    int currentMenu = 0;

    /* Boolean stating if an inquriy is in progress or not */
    boolean inquiring = false;

    /*
     * Integer to keep track of service searches. Needed to cancel an
     * ongoing service search
     */
    int serviceSearch = 0;

    public void startApp() {

        display = Display.getDisplay(this);

        Alert a = null;
        try {
            local = LocalDevice.getLocalDevice();

        }catch(BluetoothStateException bse) {
            a = new Alert("Bluetooth error",
                        "Either Bluetooth must be turned on, or your "+
                              "device does not support JABWT",
                              null,AlertType.ERROR);

            a.setTimeout(Alert.FOREVER);
            a.addCommand(exitCommand);
            a.setCommandListener(this);
        }


        tic = new Ticker("By Klings, www.klings.org/nowires");
        mainMenu(a);
    }

    public void pauseApp() {

        display = null;
        local=null;
        //attrs = null;
        tic = null;

    }

    public void destroyApp(boolean unconditional) {

        notifyDestroyed();
    }

    /*
     * mainMenu() will present the user a list of cached and preknown devices,
     * and provide a menu to initiate device discovery (inquiry) in order to
     * find nearby Bluetooth devices
     */
    private void mainMenu(Alert a) {

        List knownDevices = new List("Cached/known devices",List.IMPLICIT);

        if (deviceVector == null) deviceVector = new Vector();
        if (agent == null) agent = local.getDiscoveryAgent();

        /* Retrieve PREKNOWN devices and add them to our Vector */

        RemoteDevice[] devices = agent.retrieveDevices(
                  DiscoveryAgent.PREKNOWN);

        String name = null;

        if (devices != null) {

            /*
             * Synchronize on vector to obtain object lock before loop.
             * Else, object lock will be obtained every iteration.
             */
            synchronized(deviceVector) {

                for (int i = devices.length-1;i >=0;i--) {
                    deviceVector.addElement(devices[i]);

                    try {
                        name = devices[i].getFriendlyName(false);

                    }catch (IOException ioe) {
                        name = devices[i].getBluetoothAddress();
                    }
                    if (name.equals(""))
                       name = devices[i].getBluetoothAddress();

                    knownDevices.insert(0,name,null);
                }
            } //End synchronized
        }

        /* Retrieve cached devices and add them to our Vector */

        devices = null; devices = agent.retrieveDevices(
                  DiscoveryAgent.CACHED);

        if (devices !=null) {

            synchronized(deviceVector) {
                for (int i = devices.length-1;i >=0;i--) {
                    deviceVector.addElement(devices[i]);

                    try {
                        name = devices[i].getFriendlyName(false);
                    }catch (IOException ioe) {
                        name = devices[i].getBluetoothAddress();
                    }
                    if (name.equals(""))
                       name = devices[i].getBluetoothAddress();

                    knownDevices.insert(0,name,null);
                }
            }
        }

        /* Indicate to user if the Vector is empty */

        if (deviceVector.isEmpty()) {
            knownDevices.append("Empty",null);
        }

        knownDevices.setTicker(tic);
        knownDevices.addCommand(exitCommand);
        knownDevices.addCommand(searchCommand);
        knownDevices.setCommandListener(this);

        /* If we have an Alert, show it... */

        if (a ==null ) display.setCurrent(knownDevices);
        else display.setCurrent(a,knownDevices);

        currentMenu = 1;

    }

    /*
     * deviceScreen() will show the user a list of devices found
     * during device discovery (inquiry)
     */
    private void deviceScreen(Alert a) {


        /*
         * if currentmenu < 3 we are in screen with known/cached devices or
         * screen with discovered devices and have issued a device search
         * deviceList is then reinitialized by startInquiry(), hence we
         * must add these commands again
         */

        if (currentMenu < 3) {

            deviceList.setTicker(tic);
            if(inquiring ) {
                deviceList.setTitle("Please wait...");
                deviceList.addCommand(cancelCommand);

            }else {
                deviceList.setTitle("Devices:");
                deviceList.removeCommand(cancelCommand);
                deviceList.addCommand(exitCommand);
                deviceList.addCommand(searchCommand);
                deviceList.addCommand(backCommand);

            }

            deviceList.setCommandListener(this);
        }

        /* Display Alert if any... */
        if (a == null) display.setCurrent(deviceList);
        else display.setCurrent(a,deviceList);
        currentMenu = 2;
    }

    /*
     * startInquiry() will reinitialize the list of devices, and initiate
     * a device discovery (inquiry) repopulating the list of devices
     */
    private void startInquiry() {

        Alert a = new Alert("Inquiry status",null,null,AlertType.INFO);

        if (agent ==null) agent = local.getDiscoveryAgent();

        /* Get rid of old search results in vector and deviceList */

        deviceVector.removeAllElements();
        deviceList = null; deviceList = new List("Devices",List.IMPLICIT);

        /* Start the actual inquiry */

        try {
            inquiring = agent.startInquiry(DiscoveryAgent.GIAC, this);
        }catch(BluetoothStateException bse) {
            a.setType(AlertType.ERROR);
            a.setString("Bluetooth error while starting inquiry");
            mainMenu(a);
            return;
        }

        if (inquiring) {
            a.setString("Inquiry started");
            deviceScreen(a);
        }
        else {
            a.setType(AlertType.ERROR);
            a.setString("Error starting inquiry");

            //With no Inquiry we have no need for this any more.
            deviceList = null;
            mainMenu(a);
        }
    }

    /*
     * getFriendlyNames() will contact all devices in the deviceVector and
     * retrieve their friendly names, if available
     */
    private void getFriendlyNames() {
        String name = null;
        for (int i = deviceVector.size() -1; i>= 0;i--) {

            try {
                name = ((RemoteDevice)
                        deviceVector.elementAt(i)).getFriendlyName(false);
            }catch (IOException ioe) {

                  /*
```

```
                 * An IOException may occur if the remote device can not be
                 * contacted or the remote device could not provide its name.
                 * In that case we leave the Bluetooth address in the list,
                 * and move on to the next device found.
                 */
                continue;
            }

            if (!name.equals("")) {
                deviceList.set(i, name,null);
            }
        }

    }

    /*
     * startServiceSearch() will initiate a service search on the supplied
     * remote device.
     */
    private void startServiceSearch(RemoteDevice rDevice) {

        Alert a = null;

        //Prepare serviceVector
        if (serviceVector == null) serviceVector = new Vector();
        else serviceVector.removeAllElements();

        serviceList = null; serviceList = new List("",List.IMPLICIT);
        try {
            serviceList.setTitle( rDevice.getFriendlyName(false));
        }catch (IOException ioe) {
            serviceList.setTitle(rDevice.getBluetoothAddress());
        }

        /*
         * Search for services containing the PublicBrowseRoot UUID (0x1002)
         * Should give us all public browseable services
         */

        UUID[] uuids = new UUID[1];
        uuids[0] = new UUID(0x1002);

        /*
         * Start the actual service search, using the attrs array initialized
         * earlier, the UUID array and the remote device we want to do a
         * service search on
         */
        try {
            serviceSearch = agent.searchServices(attrs, uuids, rDevice, this);
        }catch (BluetoothStateException bse) {
            a = new Alert("Bluetooth error", "Error starting service search",
                          null, AlertType.ERROR);
            deviceScreen(a);
            return;
        }

        a = new Alert("Status","Service search started. Please wait!",
                    null, AlertType.INFO);
        deviceScreen(a);
    }

    /*
     * serviceScreen() will present a list of discovered services to the user
     */
    private void serviceScreen(Alert a) {

        //Make sure we only add these commands once
        if (currentMenu <= 2) {

            serviceList.setTicker(tic);

            if(serviceSearch == 0) {
                serviceList.addCommand(exitCommand);
                serviceList.addCommand(backCommand);
                serviceList.removeCommand(cancelCommand);
            }else{
                serviceList.addCommand(cancelCommand);
            }

            serviceList.setCommandListener(this);
        }

        if (a == null) display.setCurrent(serviceList);
        else display.setCurrent(a,serviceList);
        currentMenu = 3;
    }

    /*
     * showService() will create a recordCanvas showing service attributes
     * to the user
     */
    private void showService(int index) {

      ServiceRecord s = (ServiceRecord) serviceVector.elementAt(index);
      if (rCanvas == null){
            rCanvas = new BluetoothServiceRecordCanvas(s);

            rCanvas.addCommand(exitCommand);
            rCanvas.addCommand(backCommand);
            rCanvas.setCommandListener(this);
      }else{


        rCanvas.setServiceRecord(s);
        rCanvas.removeCommand(openURL);

      }

      clientExecutableURL = rCanvas.getClientExecutableURL();
        documentationURL = rCanvas.getDocumentationURL();

        if (clientExecutableURL !=null || documentationURL != null)
            rCanvas.addCommand(openURL);

        //rCanvas.setTicker(tic);
        display.setCurrent(rCanvas);
        currentMenu = 4;
    }

    private void openURLScreen(){

      List list = new List("Open URL",List.IMPLICIT);

      list.append("Documentation",null);
      list.append("Client Executable",null);
      list.addCommand(backCommand);
      list.setCommandListener(this);
      currentMenu = 5;
    }

    private void openURL(String url){

      boolean mustExit = false;
      Alert a = new Alert("Error","",null,AlertType.ERROR);
      try {
                mustExit = platformRequest(url);
            } catch (ConnectionNotFoundException e) {
                a.setString("This device cannot open URLs requested by Java "+
                            "programs. See http://www.klings.org/nowires "+
                            "for information.");
                display.setCurrent(a,rCanvas);
                currentMenu = 4;
                return;
            }

            if (mustExit){
                a.setType(AlertType.INFO);
                a.setTitle("Notification");
                a.setString("The URL MAY be opened when this application exits."+
                            " Choose exit to open URL now.");
                display.setCurrent(a,rCanvas);
                currentMenu = 4;
            }
    }

    /*
     * deviceDiscovered() is called by the JABWT implementation when a device
     * is discovered during device discovery (inquiry)
     */
    public void deviceDiscovered(javax.bluetooth.RemoteDevice remoteDevice,
            javax.bluetooth.DeviceClass deviceClass) {

        //Add device to vector in case of further use
        deviceVector.addElement(remoteDevice);

        /* Add vector to active list, making devices show up as they are added
         * Add only BT address, since getting the name requires going on air
         * Will get friendly name later, device is probably quite busy now
         */
        deviceList.append(remoteDevice.getBluetoothAddress(),null);

    }

    /*
     * inquiryCompleted() is called by the JABWT implementation when device
     * discovery (inquiry) is completed
     */
    public void inquiryCompleted(int param) {
        inquiring = false;
        Alert a = new Alert("Inquiry status",null,null,AlertType.INFO);
        switch(param) {

            /*
             * If inquiry completed normally, give the user an alert stating
             * that no devices were found, or the number of devices discovered.
             * Also, retrive friendly names, if devices were discoverd.
             */
            case DiscoveryListener.INQUIRY_COMPLETED:

                if (deviceVector.size() == 0) {
                    a.setString("No devices found!");
                    deviceList.append("Empty",null);
                }else {
                    getFriendlyNames();
                    a.setString(deviceVector.size() + " devices found!");
                }
                deviceScreen(a);
                break;

            /*
             * Alert the user if an error occured during device discovery
             * (inquiry). Show list of devices found before error occured.
             */
            case DiscoveryListener.INQUIRY_ERROR:
                a.setType(AlertType.ERROR);


                if(deviceVector.size() > 0) {
                    a.setString("Error occured, but " + deviceVector.size()+
                            " devices found anyway!");
                    getFriendlyNames();
                    deviceScreen(a);
                }
                else{
                    a.setString("Error occured during inquiry.");
                    mainMenu(a);
                }
                break;

            /*
             * If the user requests termination of the device discovery
             * (inquiry), alert the user that the process is actually
             * terminated.
             */
            case DiscoveryListener.INQUIRY_TERMINATED:

                a.setString("Search terminated");

                if(deviceVector.size() > 0) {
                getFriendlyNames();
                deviceScreen(a);
                }else{
                mainMenu(a);
                }
            break;
        }

    }

    /*
     * servicesDiscovered() is called by the JABWT when services are discovered
     * during service search. The transID parameter identifies the particular
     * service search that returned results. The serviceRecord array is the
     * services found during the search.
     */
    public void servicesDiscovered(int transID,
            ServiceRecord[] serviceRecord) {

        DataElement nameElement = null;

        synchronized(serviceVector) {

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

                nameElement = (DataElement)
                                    serviceRecord[i].getAttributeValue(0x100);

                if (nameElement != null
                        && nameElement.getDataType() == DataElement.STRING) {
                    serviceList.append((String) nameElement.getValue(),null);
                    serviceVector.addElement(serviceRecord[i]);
                }
            }
        }
    }


    /*
     * serviceSearchCompleted() is called by the JABWT implementation when
     * service search completes. the transID parameter identifies a particular
     * service search, the responseCode indicates why the service search is
     * ended.
     */
    public void serviceSearchCompleted(int transID, int responseCode) {

      /*
       * serviceSearch is a handle to the active service search. Set this to
       * 0 since the search is ended.
       */
      serviceSearch = 0;

        Alert a = new Alert("Search status",null,null,AlertType.INFO);

        switch(responseCode) {

            /*
             * Service search completed normally. Show the user the results.
             */
            case DiscoveryListener.SERVICE_SEARCH_COMPLETED:
                a.setString("Service search complete");
                serviceScreen(a);

                break;
            /*
             * The remote device was not reachable, making it really hard to
             * search for services.
             */
            case DiscoveryListener.SERVICE_SEARCH_DEVICE_NOT_REACHABLE:

                a.setString("Device not reachable");
                deviceScreen(a);
                break;

            /*
             * Some error occured during service search, alert the user.
             */
            case DiscoveryListener.SERVICE_SEARCH_ERROR:
                a.setType(AlertType.ERROR);
                a.setString("Error during service search");
                deviceScreen(a);
                break;

            /*
             * No service were returned by the remote device, alert the user.
             */
            case DiscoveryListener.SERVICE_SEARCH_NO_RECORDS:
                a.setString("No services found");
                deviceScreen(a);
                break;

            /*
             * Service search termination requested by user, alert the user
             * that the search is indeed terminated.
             */
            case DiscoveryListener.SERVICE_SEARCH_TERMINATED:
                a.setString("Search terminated");
                deviceScreen(a);
                break;
        }


    }


    /*
     * The commandAction() is the central nerve of a command-driven MIDlet.
     * We must check wich command is selected and also keep track of which
     * menu that was active when the command was selected. We make use of the
     * currentMenu Integer to achieve this.
     */
    public void commandAction(Command c,Displayable d) {

        if (c == exitCommand) destroyApp(true);

        else if(c == searchCommand) {
            startInquiry();
        }
        else if(c == backCommand) {

            /*
             * User wants to go back, check where we are and fulfill
             * the request.
             */
            switch(currentMenu) {

                case 2:
                    mainMenu(null);
                    break;

                case 3:
                    deviceScreen(null);
                    break;

                case 4:
                    serviceScreen(null);
                    break;

                case 5:
                    display.setCurrent(rCanvas);
                    currentMenu = 4;
                    break;
            }
        }else if(c == cancelCommand){

            /*
             * The user wants to cancel either a device discovery or a service
             * search. Check where we are, and fulfill the request.
             */
            switch(currentMenu) {

                case 2:
                    if (inquiring) {
                    inquiring = !agent.cancelInquiry(this);
                    if (inquiring) {
                        deviceScreen(new Alert("Error",
                                   "Could not stop inquiry or inquiry not started",
                                             null, AlertType.ERROR));
                        return;
                    }
                    }else if (serviceSearch > 0
                             && agent.cancelServiceSearch(serviceSearch)) {
                        deviceScreen(new Alert("Status",
                                   "Service search terminated",null,
                                             AlertType.INFO));
                    }

                    serviceSearch = 0;
                    break;

                case 3:

                    if(!agent.cancelServiceSearch(serviceSearch)) {
                        serviceScreen(new Alert("Error",
                                   "No active service search",null,
                                             AlertType.ERROR));
                    }

                    break;
            }
        }else if (c == openURL){

            openURLScreen();

    }else if (c == List.SELECT_COMMAND) {

            /*
             * The user has selected something from a list. Find out where
             * we are, get the List currently displayed and act according to
             * this.
             */
            List list = (List) display.getCurrent();
            int index = list.getSelectedIndex();

            switch (currentMenu) {

                case 1: // Main list of known devices


                case 2: //List of newly found devices
                    if (serviceSearch == 0 && ! deviceVector.isEmpty())
                        startServiceSearch(
                                    (RemoteDevice)deviceVector.elementAt(index));
                    break;

                case 3: //Browse service
                    if(! serviceVector.isEmpty()) showService(index);

                    break;

                    case 5: //The user wants to open an URL

                        switch(index){
                              case 0:
                                    openURL(documentationURL);
                                    break;

                              case 1:

                                    openURL(clientExecutableURL);
                                    break;

                        }

            }//End switch for list-index
        }// End if for List-command

    }//End commandaction

}//End all
```

Appendix B   BTBenchmark

Server.java

```
/*
 * Server.java
 *
 * Version 1.0
 *
 * 22. June 2004
 *
 * Copyright (c) 2004, Andre N. Klingsheim
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * Redistributions of source code must retain the above copyright notice, this
 * list of conditions and the following disclaimer.
 *
 * Redistributions in binary form must reproduce the above copyright notice,
 * this list of conditions and the following disclaimer in the documentation
 * and/or other materials provided with the distribution.
 *
 * Neither the name of the NoWires research group nor the names of its
 * contributors may be used to endorse or promote products derived from this
 * software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

package org.klings.wireless.j2me.BTBenchmark;

import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;
import javax.bluetooth.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;
import java.io.*;
import org.klings.wireless.j2me.*;

public class Server extends MIDlet implements CommandListener, Runnable{
      /* Display and Bluetooth LocalDevice */
      private Display display = null;
      private LocalDevice local = null;

      private List list = null;
      private Ticker tic = null;

      /* Commands */
      private Command exitCommand = new Command("Exit",Command.EXIT,2);
      private Command startCommand = new Command("Start",Command.SCREEN,2);
      private Command settingsCommand = new Command("Settings",Command.SCREEN,2);
      private Command backCommand = new Command("Back", Command.BACK, 1);
      private Command btInfoCommand = new Command("BTInfo",Command.SCREEN,2);

      /* Notifier for client connections */
      private StreamConnectionNotifier server = null;

      /* Connection for communication */
      private StreamConnection conn = null;
      private InputStream in = null;
      private OutputStream out = null;

      /* Connection parameters */
      private boolean authenticate = false;
      private boolean encrypt = false;
      private boolean authorize = false;
      private boolean master = false;

      /* Canvas to show status to user */
      private StatusCanvas s=null;

      /* Initialized in createService(), defined globally so it is accessible
       * by the communicating thread.
       */
      private String connectionURL;

      /* Thread for blocking functionality */

      private Thread t = null;

      /* Default constructor */
      public Server() {
            super();

      }

      protected void startApp() throws MIDletStateChangeException {

            display = Display.getDisplay(this);

            tic = new Ticker("By Klings, www.klings.org/nowires");

            mainMenu(null);
      }

      protected void pauseApp() {
            tic = null;
      }
```

```
    protected void destroyApp(boolean arg0) throws MIDletStateChangeException {

          /*
           * Ensure that cleanUp() is called before MIDlet exits. cleanUp()
           * will close the notifier, which will remove the service record from
           * device Bluetooth SDDB. Very important!
           */

          cleanup();

          /*
           * If the communication Thread is waiting for clients, an exception
           * occurs when we close the notifier. The thread will terminate,
           * wait for it to terminate.
           */
          if(t!=null){
                try {
                      t.join();
                } catch (InterruptedException e) {
                      // TODO Auto-generated catch block
                      e.printStackTrace();
                }
          }

          notifyDestroyed();
    }
    public void commandAction(Command c, Displayable arg1) {

          if (c == exitCommand) {

                try {
                      destroyApp(true);
                } catch (MIDletStateChangeException e) {

                }
          }
          else if(c == startCommand){

                /*
                 *  Start a server thread. createService() returns true
                 * if the service was created successfully.
                 */

                if (createService()){

                      /* Show status screen to user */
                      s = new StatusCanvas();
                      s.waitForClient(connectionURL);
                      s.addCommand(exitCommand);
                      s.setCommandListener(this);
                      s.setTicker(tic);
                      display.setCurrent(s);

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

                }else{
                      mainMenu(new Alert("Error","Error creating service",null,
                                  AlertType.ERROR));
```

```
            }

        }else if(c == settingsCommand){

            settings();
        }else if (c == backCommand){

            mainMenu(null);
    }else if (c == btInfoCommand){

            BluetoothInfoCanvas btic = new BluetoothInfoCanvas();
            btic.addCommand(backCommand);
            btic.setCommandListener(this);
            display.setCurrent(btic);


    }else if (c == List.SELECT_COMMAND) {

            /* There is only one list. Check what the user selected */
            List list = (List) display.getCurrent();

            int index = list.getSelectedIndex();

            switch (index){

                case 0:

                    authenticate = !authenticate;
                    if(!authenticate) {
                        encrypt = false;
                        authorize = false;
                    }
                    break;

                case 1:
                    encrypt = !encrypt;
                    if(encrypt) authenticate = true;
                    break;

                case 2:

                    authorize = !authorize;
                    if (authorize) authenticate = true;
                    break;

                case 3:

                    master = !master;
                    break;
            }

            /* Update settings screen so it reflects the user's choice */
        settings();
        }

    }

    /* Display list of connection settings */
    private void settings() {
        List sett = new List("Settings:",List.IMPLICIT);

        sett.append("Authenticate: " + (authenticate?"On":"Off"),null);
        sett.append("Encrypt: " + (encrypt?"On":"Off"),null);
        sett.append("Authorize: " + (authorize?"On":"Off"),null);

        if ("true".equals(LocalDevice.getProperty("bluetooth.master.switch")))
        sett.append("Master: " + (master?"Yes":"No"),null);

        sett.addCommand(startCommand);
        sett.setCommandListener(this);

        display.setCurrent(sett);
    }

    /* Display main menu */
    private void mainMenu(Alert a){

        list = new List("Benchmark Server",List.IMPLICIT);
        list.setTicker(tic);
        list.addCommand(exitCommand);
        list.addCommand(startCommand);
        list.addCommand(settingsCommand);
        list.addCommand(btInfoCommand);
        list.setCommandListener(this);

        if(a == null){
        display.setCurrent(list);
        }else display.setCurrent(a,list);
    }

    /* Create ServiceRecord, set desired attributes */
    private boolean createService(){

        ServiceRecord record = null;

        Alert a = new Alert("Error","",null,AlertType.ERROR);
        a.setTimeout(Alert.FOREVER);

        /* Get the Bluetooth LocalDevice */
        try {
            local = LocalDevice.getLocalDevice();
            local.setDiscoverable(DiscoveryAgent.GIAC);
        } catch (BluetoothStateException e) {
            a.setString("Error getting localdevice or setting discovery mode");
            display.setCurrent(a,list);
            e.printStackTrace();
            return false;
        }

        String param ="";

        /* Add parameters to connection URL */
        if(encrypt) param = "encrypt=true;";
        else if(authorize) param="authorize=true;";
        else if(authenticate) param="authenticate=true;";
```

```
        param += "name=BTBench";

        if(master) param +=";master=true";

        connectionURL = "btspp://localhost:66ca80886d1f11d88526000bdb544cb1;"
              + param;


        /* Get the notifier, will also generate a ServiceRecord */
        try {
              server = (StreamConnectionNotifier) Connector.open(connectionURL);
        } catch (IOException e1) {

              return false;
        }

        /* Get the ServiceRecord associated with the notifier */
        try {
              record = local.getRecord(server);
        }
        catch (IllegalArgumentException iae){
              return false;

        }

        /*
         * Manipulate the ServiceRecord to meet our needs. Some code is
         * disabled, but is included to show how attributes are set.
         */
        DataElement elm = null;

        /*
         * Set public browse root in browsegrouplist, making service
         * public browseable
         */
        elm = new DataElement(DataElement.DATSEQ);
        elm.addElement(new DataElement(DataElement.UUID,new UUID(0x1002)));
        record.setAttributeValue(0x0005,elm);


        /* Set service description */
        elm = new DataElement(DataElement.STRING,"BT Benchmark service");
        record.setAttributeValue(0x101,elm);

        /* Set service provider name */
        elm = new DataElement(
                    DataElement.STRING,"Klings, NoWires Research Group");
        record.setAttributeValue(0x102,elm);

        /* Set serviceInfoTimeToLive */
        /*elm = new DataElement(DataElement.U_INT_4,10000);
        record.setAttributeValue(0x0007,elm);*/

        /* Set serviceAvailability */
        /*elm = new DataElement(DataElement.U_INT_1,255);
        record.setAttributeValue(0x0008,elm);*/

        /* Set documentationURL */
        elm = new DataElement(
                    DataElement.URL,"http://wap.klings.org/btbenchmark.wml");
        record.setAttributeValue(0x000A,elm);

        /* Set clientExecutableURL */
        elm = new DataElement(

        DataElement.URL,"http://wap.klings.org/java/btbenchmark.jad");
        record.setAttributeValue(0x000B,elm);

        /* Set iconURL */
        /*elm = new DataElement(
         *          DataElement.URL,"http://klings.org/java/BTBenchmark.ico");
        record.setAttributeValue(0x000C,elm);*/

        /* Update the record, else changes are lost */
        try {
              local.updateRecord(record);
        } catch (ServiceRegistrationException e3) {

              return false;

        }
        return true;
  }

  /* run() method executed by communication thread */
  public void run() {

        /*
         * If the notifier is not available, a service has not been
         * created.
         */
        if(server == null) return;

        Alert a = new Alert("Error","",null,AlertType.ERROR);
        a.setTimeout(Alert.FOREVER);


        /*
         * Open the notifier. This is a blocking operation. Now the
         * ServiceRecord will be entered in the Bluetooth SDDB and
         * the server is ready for client connections.
         */
        try {
                    conn = server.acceptAndOpen();
              }
              catch (ServiceRegistrationException sre){
                    a.setString("Error creating service record");
                    display.setCurrent(a,list);

                    cleanup();
                    return;
              }
              catch (IOException e2) {
                    if (t != null){
                    a.setString("Error starting server. " + e2.getMessage());
                    display.setCurrent(a,list);
                    }
```

```
            cleanup();
            return();
      }

      /* A client has connected! Retrieve information about the
       * remote device and display it to the user.
       */
      RemoteDevice dev = null;
      String name,address;
      boolean authorized = false;

      try {
            dev = RemoteDevice.getRemoteDevice(conn);
            name = dev.getFriendlyName(false);
            address = dev.getBluetoothAddress();
            authorized = dev.isAuthorized(conn);
      } catch (IOException e) {
            name = "Unknown";
            address = "Unknown";
      }


      s.connectedToClient(name,address,dev.isAuthenticated(),
                  dev.isEncrypted(),authorized);
      display.setCurrent(s);

      /*
       * The user now knows who connected. Get streams and start
       * communication.
       */
      try {
            in = conn.openInputStream();
            out = conn.openOutputStream();
      } catch (IOException e4) {
            a.setString("Error opening input/output streams:" +
                        e4.getMessage());
            display.setCurrent(a,list);

            cleanup();
            t = null;
            return;
      }

      DataInputStream dais = new DataInputStream(in);

      byte[] data = new byte[512];

      //int data;
      long timer = 0;
      boolean keepOn = true;
      int bytesRead = 0;
      int iterations = 0;

      /* read the number of iterations */
      try {
            iterations = dais.readInt();
      }catch (EOFException eof ){

            a.setString("EOF on first read.");
```

```
                        display.setCurrent(a,list);

                        cleanup();
                        list.addCommand(startCommand);
                        return;
                } catch (IOException e5) {
                        a.setString("Error on first read.");
                        display.setCurrent(a,list);


                        cleanup();
                        list.addCommand(startCommand);
                        return;
                }

                /* Get the time and do actual communication */
                try{
                        timer = System.currentTimeMillis();

                        for (int i = iterations; i>0;i--){

                                bytesRead = dais.read(data,0,data.length);
                                out.write(0);
                                out.flush();
                        }
                        timer = System.currentTimeMillis() - timer;
                }catch (IOException ioe) {

                        a.setString(bytesRead +
                                " bytes read before communication error occured. " +
                                        ioe.getMessage());
                        display.setCurrent(a,list);

                        cleanup();
                        list.addCommand(startCommand);
                        return;
                }

                /* Communication complete. Cleanup connections */
                cleanup();
                long sec = timer / 1000;
                long transferred = iterations/2;

                /* Display statistics to user */
                a.setType(AlertType.INFO);
                a.setTitle("Success!");
                a.setString("Read " + transferred +" KB in " + sec + " seconds.\n"
                                +transferred/sec + " KBps");


                display.setCurrent(a,list);
                list.addCommand(startCommand);
                list.addCommand(settingsCommand);
                list.setTitle("Run complete.");



        }
```

```
      /* Close streams and notifier */
      private void cleanup() {

            try {

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

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

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

                  if (server != null){

                        server.close();
                  }
            }catch (IOException ioe){

                  // Error occurred.
            }
      }



}
```

Client.java

```
/*
 * Client.java
 *
 * Version 1.0
 *
 * 22. June 2004
 *
 * Copyright (c) 2004, Andre N. Klingsheim
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * Redistributions of source code must retain the above copyright notice, this
 * list of conditions and the following disclaimer.
 *
 * Redistributions in binary form must reproduce the above copyright notice,
 * this list of conditions and the following disclaimer in the documentation
 * and/or other materials provided with the distribution.
 *
 * Neither the name of the NoWires research group nor the names of its
 * contributors may be used to endorse or promote products derived from this
 * software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */
package org.klings.wireless.j2me.BTBenchmark;

import javax.microedition.midlet.MIDlet;
import javax.microedition.lcdui.*;
import java.util.*;
import java.io.*;

import javax.microedition.io.*;


import javax.bluetooth.*;
import org.klings.wireless.j2me.*;

public class Client extends MIDlet implements CommandListener,
                                              DiscoveryListener,
                                              Runnable{

    private Display display = null;
    private LocalDevice local=null;
    private DiscoveryAgent agent=null;
    /* Vectors to store devices an services in */
    private Vector deviceVector = null;
    private Vector serviceVector = null;

    private Ticker tic = null;

    /* Global list accessible to deviceDiscovered(...) */
    private List deviceList = null;
    private List serviceList = null;

    /* lobal serviceRecord accessible to several threads */
    private ServiceRecord globalRecord = null;

    private StreamConnection conn = null;
    private DataOutputStream out = null;
    private InputStream in = null;

    /* Connection parameters */
    boolean authenticate,encrypt,master=false;

    private int iterations = 0;


    private Command exitCommand = new Command("Exit",Command.EXIT,2);
    private Command searchCommand = new Command("New Search",Command.SCREEN, 1);
    private Command backCommand = new Command("Back", Command.BACK, 1);
    private Command cancelCommand = new Command("Cancel",Command.CANCEL,1);
    private Command settingsCommand = new Command("Settings",Command.SCREEN,1);
    private Command btInfoCommand = new Command("BTInfo",Command.SCREEN,2);

    int currentMenu = 0;

    /* Keep track of ongoing device discoveries */
    boolean inquiring = false;

    /* Keep track of ongoing service discoveries */
    int serviceSearch = 0;

    public void startApp() {

        display = Display.getDisplay(this);

        try {
            local = LocalDevice.getLocalDevice();

        }catch(BluetoothStateException bse) {
            System.err.println("LocalDevice error");
        }

        tic = new Ticker("By Klings, www.klings.org/nowires");
        mainMenu(null);
    }

    public void pauseApp() {
        display = null;
        local = null;
        tic = null;
    }
    public void destroyApp(boolean unconditional) {

        cleanUp();

        notifyDestroyed();
    }

    /* Main menu, list of cached/known Bluetooth devices */
    private void mainMenu(Alert a) {

        List knownDevices = new List("Cached/known devices",List.IMPLICIT);

        if (deviceVector == null) deviceVector = new Vector();
        if (agent == null) agent = local.getDiscoveryAgent();

        RemoteDevice[] devices = agent.retrieveDevices(
                  DiscoveryAgent.PREKNOWN);

        String name = null;

        if (devices != null) {

            /*
             * Synchronize on vector before running through loop, since
             * Vector methods are thread safe. We obtain the object lock
             * on the Vector once, instead of every iteration. Add devices
             * to the deviceVector.
             */
            synchronized(deviceVector) {

                for (int i = devices.length-1;i >=0;i--) {
                    deviceVector.addElement(devices[i]);

                    try {
                        name = devices[i].getFriendlyName(false);

                    }catch (IOException ioe) {
                        name = devices[i].getBluetoothAddress();
                    }
                    if (name.equals(""))
                        name = devices[i].getBluetoothAddress();

                    knownDevices.append(name,null);
                }
            } //End synchronized
        }
        devices = agent.retrieveDevices(DiscoveryAgent.CACHED);

        if (devices !=null) {

            synchronized(deviceVector) {
                for (int i = devices.length-1;i >=0;i--) {
                    deviceVector.addElement(devices[i]);

                    try {
                        name = devices[i].getFriendlyName(false);
                    }catch (IOException ioe) {
                        name = devices[i].getBluetoothAddress();
                    }
                    if (name.equals(""))
                        name = devices[i].getBluetoothAddress();

                    knownDevices.append(name,null);
                }
            }
        }

        /* No cached/known devices, notify the user */
        if (deviceVector.isEmpty()) {
            knownDevices.append("Empty",null);
        }

        knownDevices.setTicker(tic);
        knownDevices.addCommand(exitCommand);
        knownDevices.addCommand(searchCommand);
        knownDevices.addCommand(btInfoCommand);
        knownDevices.setCommandListener(this);

        /* Show Alert, if available */
        if (a ==null ) display.setCurrent(knownDevices);
        else display.setCurrent(a,knownDevices);

        currentMenu = 1;

    }

    /* Show list of discovered devices */
    private void deviceScreen(Alert a) {


        /* if currentmenu < 3 we are in screen with known/cached devices or
         * screen with discovered devices and have issued a device search
         * deviceList is then reInitialized by startInquiry(), hence we
         *  must add these commands again
         */
        if (currentMenu < 3) {

            deviceList.setTicker(tic);
            if(inquiring) {
                deviceList.addCommand(cancelCommand);
            }else{
                deviceList.removeCommand(cancelCommand);
                deviceList.addCommand(exitCommand);
                deviceList.addCommand(searchCommand);
                deviceList.addCommand(backCommand);

            }

            deviceList.setCommandListener(this);
        }

        if (a == null) display.setCurrent(deviceList);
        else display.setCurrent(a,deviceList);
        currentMenu = 2;
    }

    /* Start device discovery */
```

```
    private void startInquiry() {

        Alert a = new Alert("Inquiry status",null,null,AlertType.INFO);


        if (agent ==null) agent = local.getDiscoveryAgent();

        /* Remove old search results in vector and deviceList */
        deviceVector.removeAllElements();
        deviceList = new List("Nearby devices",List.IMPLICIT);

        /* Start the actual device discovery */
        try {
            inquiring = agent.startInquiry(DiscoveryAgent.GIAC, this);
        }catch(BluetoothStateException bse) {
            a.setType(AlertType.ERROR);
            a.setString("Bluetooth error while starting inquiry");
            mainMenu(a);
            return;
        }

        /* Notify the user if inquiry was started or not */
        if (inquiring) {
            a.setString("Inquiry started");
            deviceScreen(a);
        }
        else {
            a.setType(AlertType.ERROR);
            a.setString("Error starting inquiry");

            /* With no Inquiry we have no need for this any more. */
            deviceList = null;
            mainMenu(a);
        }
    }

    /* Retrieve friendly names for all devices in deviceVector */
    private void getFriendlyNames() {
      String name = null;

      synchronized(deviceVector){
            for (int i = deviceVector.size() -1; i>= 0;i--) {

                  try {
                        name = ( (RemoteDevice)
                              deviceVector.elementAt(i)).getFriendlyName(false);
                  }catch (IOException ioe) {
                        continue;
                  }
                  if (!name.equals("")) {
                        deviceList.set(i, name,null);
                  }
            }// End for
      }// End synchronized

    }

    /* Start service discovery on remote device */
    private void startServiceDiscovery(RemoteDevice rDevice) {
        Alert a = null;

        /* Prepare serviceVector and service list*/
        if (serviceVector == null) serviceVector = new Vector();
        else serviceVector.removeAllElements();

        serviceList = new List("",List.IMPLICIT);
        try {
            serviceList.setTitle( rDevice.getFriendlyName(false));
        }catch (IOException ioe) {
            serviceList.setTitle(rDevice.getBluetoothAddress());
        }

        UUID[] uuids = new UUID[1];

        /* Add the BTBenchmark UUID to an array */
        uuids[0] = new UUID("66ca80886d1f11d88526000bdb544cb1",false);

        /*
         * Retrieve default attributes, services with BTBenchmark UUID on
         * the remote device.
         */
        try {
            int transid = agent.searchServices(null, uuids, rDevice, this);
        }catch (BluetoothStateException bse) {
            a = new Alert("Bluetooth error", "Error starting service search",
                        null, AlertType.ERROR);
            deviceScreen(a);
            return;
        }

    }

    /* Select amount of data to transfer */
    private void selectDataScreen(Alert a) {

      List tList = new List("Select amount of data ",List.IMPLICIT);
        tList.append("10KB",null);
        tList.append("100KB",null);
        tList.append("500KB",null);
        tList.append("1000KB",null);

        tList.addCommand(settingsCommand);
        tList.addCommand(backCommand);
        tList.setCommandListener(this);

        if (a == null) display.setCurrent(tList);
        else display.setCurrent(a,tList);
        currentMenu = 3;
    }

    /*
     * Set the number of iterations to use by the thread when communicating,
     * then start the communication Thread.
     */
    private void doTransfer(int index) {

      switch(index){
            case 0: iterations = 20; break;
            case 1: iterations = 200; break;
            case 2: iterations = 1000; break;
            case 3: iterations = 2000; break;
      }

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

    }

    /* Display settings menu */
    private void settingsScreen(){

      List settings = new List("Settings",List.IMPLICIT);
      settings.append("Authenticate: " + (authenticate? "Yes":"No"),null);
      settings.append("Encrypt: " + (encrypt? "Yes":"No"),null);

      if ("true".equals(LocalDevice.getProperty("bluetooth.master.switch")))
      settings.append("Master: " + (master? "Yes":"No"),null);

      settings.addCommand(backCommand);
      settings.setCommandListener(this);
      display.setCurrent(settings);
      currentMenu = 31;

    }

    /* A device is discovered. Add it to the deviceVector */
    public void deviceDiscovered(javax.bluetooth.RemoteDevice remoteDevice,
            javax.bluetooth.DeviceClass deviceClass) {

        /* Add device to vector in case of further use */
        deviceVector.addElement(remoteDevice);

        /*
         * Add device to active list, making devices show up as they are
         * discoverd. Add only BT address, getting the name requires the
         * device to go on air and the device is probably quite busy now.
         */

        deviceList.append(remoteDevice.getBluetoothAddress(),null);

    }

    /* Device discovery completed. Get friendly names. */
    public void inquiryCompleted(int status) {
        inquiring = false;
        Alert a = new Alert("Inquiry status",null,null,AlertType.INFO);

        /* Check status */
        switch(status) {

            case DiscoveryListener.INQUIRY_COMPLETED:

                if (deviceVector.size() == 0) {
                    a.setString("No devices found!");
                    deviceList.append("Empty",null);
                }else {
                    getFriendlyNames();
                    a.setString(deviceVector.size() + " devices found!");
                }
                deviceScreen(a);
                break;

            case DiscoveryListener.INQUIRY_ERROR:
                a.setType(AlertType.ERROR);
                a.setString("Error occured.");
                mainMenu(a);
                break;

            case DiscoveryListener.INQUIRY_TERMINATED:

                a.setString("Search terminated");

                if(deviceVector.size() > 0) {
                getFriendlyNames();
                deviceScreen(a);
                }else{
                mainMenu(a);
                }
                break;
        }

    }

    /* ServiceSearch completed. */
    public void serviceSearchCompleted(int transID, int respCode) {
        serviceSearch = 0;
        Alert a = new Alert("Search status",null,null,AlertType.INFO);

        /* Check resonse Code */
        switch(respCode) {

            case DiscoveryListener.SERVICE_SEARCH_COMPLETED:
                a.setString("Service found!");
                selectDataScreen(a);

                break;

            case DiscoveryListener.SERVICE_SEARCH_DEVICE_NOT_REACHABLE:

                a.setString("Device not reachable");
                deviceScreen(a);
                break;

            case DiscoveryListener.SERVICE_SEARCH_ERROR:
                a.setType(AlertType.ERROR);
                a.setString("Error during service search");
                deviceScreen(a);
                break;

            case DiscoveryListener.SERVICE_SEARCH_NO_RECORDS:
                a.setString("No services found");
                deviceScreen(a);
                break;
            case DiscoveryListener.SERVICE_SEARCH_TERMINATED:
                a.setString("Search terminated");
                deviceScreen(a);
                break;
        }


    }

    /* Services were discovered. */
    public void servicesDiscovered(int transID,
            ServiceRecord[] serviceRecord) {

        DataElement nameElement = null;
        String name = null;
        RemoteDevice dev = serviceRecord[0].getHostDevice();

        /*
         * Keep the discovered service record in globalRecord so it is
         * available to the communication thread.
         */
        globalRecord=serviceRecord[0];

    }

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

        if (c == exitCommand) destroyApp(true);

        else if(c == searchCommand) {
            startInquiry();
        }
        else if(c == backCommand) {
            switch(currentMenu) {

                  case 11:
                case 2:
                    mainMenu(null);
                    break;

                case 3:
                    deviceScreen(null);
                    break;

                case 31:
                case 4:
                    selectDataScreen(null);
                    break;

            }
        }else if(c == cancelCommand){

            switch(currentMenu) {

                  case 2: // Device discovery in progress
                        if (agent.cancelInquiry(this)) {

                              inquiring = false;
                        deviceScreen(null);

                    }else{
                        deviceScreen(new Alert("Error",
                            "Could not stop inquiry or inquiry not started",
                                    null, AlertType.ERROR));
                        inquiring = false;
                    }

                    break;

                case 3: //Service discovery in progress

                    if(!agent.cancelServiceSearch(serviceSearch)) {
                        selectDataScreen(new Alert("Error",
                                "No active service search",null,
                                    AlertType.ERROR));
                    }

                    break;


            }
        }else if(c == settingsCommand){

            settingsScreen();

        }else if (c == btInfoCommand){
            BluetoothInfoCanvas canv = new BluetoothInfoCanvas();
            canv.addCommand(backCommand);
            canv.setCommandListener(this);
            display.setCurrent(canv);
            currentMenu = 11;
        }
        else if (c == List.SELECT_COMMAND) {

            List list = (List) display.getCurrent();
            int index = list.getSelectedIndex();

            switch (currentMenu) {

                case 1: // Main list of known devices


                case 2: //List of newly found devices
                    if (! deviceVector.isEmpty())
                        startServiceDiscovery( (RemoteDevice)
                                deviceVector.elementAt(index));
                    break;

                case 3: //Browse service

                    doTransfer(index);

                    break;

                    case 31: //Settings changed

                        switch(index){
                              case 0:

                                    authenticate = !authenticate;
                                    if(!authenticate) encrypt = false;
                                    break;

                              case 1:
                                    encrypt = !encrypt;
                                    if(encrypt) authenticate=true;
                                    break;

                              case 2:
                                    master = !master;
                                    break;
                        }

                        settingsScreen();
                    break;
            }//End switch for list-index
        }// End if for List-command

    }//End commandaction

    /* Code executed by the communication thread */
      public void run() {

            /* Generate connection URL based on user selections */
            String conURL = null;

            if(encrypt){
                  conURL = globalRecord.getConnectionURL(
                              ServiceRecord.AUTHENTICATE_ENCRYPT,master);
            }else if (authenticate){

                  conURL = globalRecord.getConnectionURL(
                              ServiceRecord.AUTHENTICATE_NOENCRYPT,master);
            }else{
                  conURL = globalRecord.getConnectionURL(
                              ServiceRecord.NOAUTHENTICATE_NOENCRYPT,master);
            }

            StreamConnection conn = null;

            Alert a = new Alert("Error",conURL,null,AlertType.ERROR);
            a.setTimeout(Alert.FOREVER);

            /* Connect to the Benchmark server */
            try {
                   conn = (StreamConnection) Connector.open(conURL);
            } catch (IOException e) {
                  a.setString("Error creating connection\n URL used:\n"+conURL);
                  deviceScreen(a);
                  cleanUp();
                  return;
            }

            /* Display information about the server */
            StatusCanvas s = new StatusCanvas();
            RemoteDevice dev = globalRecord.getHostDevice();
            String name = "Unknown";
            try {
                  name = dev.getFriendlyName(false);
            } catch (IOException e4) {
                  name = "Unretrievable";
            }

            s.connectedToServer(name,dev.getBluetoothAddress(),
                        dev.isAuthenticated(),dev.isEncrypted());
            display.setCurrent(s);

            /* Open streams */
            try {
                  out = conn.openDataOutputStream();
                  in = conn.openInputStream();
            } catch (IOException e1) {
                  a.setString("Error opening streams");
                  display.setCurrent(a,serviceList);
                  cleanUp();
                  return;
            }


            /* Write the number of iterations */
            byte[] bytes = new byte[512];
            try {
                  out.writeInt(iterations);
                  out.flush();
            } catch (IOException e2) {
                  a.setString("Error doing first write");
                  display.setCurrent(a,serviceList);
                  cleanUp();
                  return;
            }

            /* Send the actual data */
            try {

                  for (int i = iterations; i > 0; i--){

                        out.write(bytes);
                        out.flush();
                        in.read();
                  }
            } catch (IOException e3) {
                  a.setString("Error writing main load");
                  display.setCurrent(a,serviceList);
                  cleanUp();
                  return;
            }

            /* Clean up streams and connection */
            cleanUp();

            /* Display the number of KB sent */
            a.setString("Sent " + iterations/2 + " KB. All well!");
            a.setType(AlertType.INFO);
            a.setTitle("Done!");
            deviceScreen(a);
      }

      /* Close streams and streamconnection */
      private void cleanUp(){

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

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

                  if(conn != null){
                        conn.close();
                  }
            } catch (IOException e) {
                  // Error occurred
            }
      }
}
```

StatusCanvas.java

```
/*
 * StatusCanvas.java
 *
 * Version 1.0
 *
 * 22. June 2004
 *
 * Copyright (c) 2004, Andre N. Klingsheim
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * Redistributions of source code must retain the above copyright notice, this
 * list of conditions and the following disclaimer.
 *
 * Redistributions in binary form must reproduce the above copyright notice,
 * this list of conditions and the following disclaimer in the documentation
 * and/or other materials provided with the distribution.
 *
 * Neither the name of the NoWires research group nor the names of its
 * contributors may be used to endorse or promote products derived from this
 * software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */
package org.klings.wireless.j2me.BTBenchmark;

import javax.microedition.lcdui.*;
import org.klings.wireless.j2me.*;


public class StatusCanvas extends Canvas{

      private final int WAITING_FOR_CLIENT = 1;
      private final int CONNECTED_TO_CLIENT = 2;
      private final int CONNECTED_TO_SERVER = 3;

      /* Anchor for text */
      private final int ANCHOR = Graphics.LEFT|Graphics.TOP;

      /* indent */
      private final int X = 2;

      private int mode = 0;
      private boolean authentication,encryption,authorization,isServer;
      /* Different fonts for different types of text */
      private Font plain,bold;

    /*Size of canvas */
      private int canvasHeight, canvasWidth;

    /* Height of fonts */
      private int plainHeight,boldHeight;

    /* Keep track of where we are in the canvas */
      private int y = 0;

    /* Custom String */
      private String remoteName,remoteAddress,connectionURL;

      public StatusCanvas() {

            super();

            /*Get the canvas size */
        canvasHeight = this.getHeight();
        canvasWidth = this.getWidth();

        /* Get height of fonts */
        plain = Font.getFont(Font.FACE_SYSTEM, Font.STYLE_PLAIN,
                  Font.SIZE_MEDIUM);
        bold = Font.getFont(Font.FACE_SYSTEM, Font.STYLE_BOLD,
                  Font.SIZE_MEDIUM);

        /* heights to compute where to draw. */
        plainHeight = plain.getHeight();
        boldHeight = bold.getHeight();

        /*Set Custom to null */
        remoteName = null;
        remoteAddress = null;
        connectionURL = null;


      }

      protected void paint(Graphics g) {

        /* Initialize the canvas */
        g.setColor(0xffffff);
        g.fillRect(0,0, getWidth(),getHeight());

        /* We want black text */
        g.setColor(0x000000);
        g.setFont(plain);

        if (isServer) serverStatus(g);
        else clientStatus(g);

      }



      public void waitForClient(String connectionURL){
            /* this must be a server */
            isServer = true;
            mode = WAITING_FOR_CLIENT;
            this.connectionURL = connectionURL;
            this.repaint();
      }

      public void connectedToClient(String deviceName, String deviceAddress,
              boolean authentication,boolean encryption,boolean authorization){

            /* this must be a server */
            isServer = true;
            remoteName = deviceName;
            remoteAddress = deviceAddress;
            this.authentication = authentication;
            this.encryption = encryption;
            this.authorization = authorization;
            mode = CONNECTED_TO_CLIENT;
            this.repaint();
      }
      public void connectedToServer(String deviceName, String deviceAddress,
              boolean authentication,boolean encryption){
            /* this must be a client */
            isServer = false;
            remoteName = deviceName;
            remoteAddress = deviceAddress;
            this.authentication = authentication;
            this.encryption = encryption;


            mode = CONNECTED_TO_SERVER;
            this.repaint();
      }

      private void serverStatus(Graphics g){

            switch (mode){

              case WAITING_FOR_CLIENT:
                  this.setTitle("Service started!");
                  y = 2;

                  y+= CanvasHelper.printString("Connection URL:",X,y,ANCHOR,
                          bold,canvasWidth-X,g);

                  y+= CanvasHelper.printString(connectionURL,X,y,ANCHOR,
                          plain,canvasWidth-X,g);

                  y+=plainHeight;

                  y+=CanvasHelper.printString("Waiting for client connection.",
                          X,y,ANCHOR,plain,canvasWidth-X,g);

                  break;

              case CONNECTED_TO_CLIENT:
                  this.setTitle("Client connected!");
                  y = 2;
```

```
                        y+= CanvasHelper.printString("Client name:",X,y,ANCHOR,
                                    bold,canvasWidth-X,g);
                        y+= CanvasHelper.printString(remoteName,X,y,ANCHOR,
                                    plain,canvasWidth-X,g);

                        y+= CanvasHelper.printString("Client address:",X,y,ANCHOR,
                                    bold,canvasWidth-X,g);
                        y+= CanvasHelper.printString(remoteAddress,X,y,ANCHOR,
                                    plain,canvasWidth-X,g);

                        y+= CanvasHelper.printString("Security settings:",X,y,ANCHOR,
                                    bold,canvasWidth-X,g);
                        y+= CanvasHelper.printString("Authenticated: " +
                                    (authentication ? "Yes" : "No"),X,y,ANCHOR,
                                    plain,canvasWidth-X,g);
                        y+= CanvasHelper.printString("Encrypted: "+
                                    (encryption ? "Yes" : "No"),X,y,ANCHOR,
                                    plain,canvasWidth-X,g);
                        y+= CanvasHelper.printString("Authorized: " +
                                    (authorization ? "Yes" : "No"),X,y,ANCHOR,
                                    plain,canvasWidth-X,g);

                        break;

            }

      }

      private void clientStatus(Graphics g){

            if (mode == CONNECTED_TO_SERVER){

                  this.setTitle("Connected to server!");
                  y = 2;

                  y+= CanvasHelper.printString("Server name:",X,y,ANCHOR,
                              bold,canvasWidth-X,g);
                  y+= CanvasHelper.printString(remoteName,X,y,ANCHOR,
                              plain,canvasWidth-X,g);

                  y+= CanvasHelper.printString("Server address:",X,y,ANCHOR,
                              bold,canvasWidth-X,g);
                  y+= CanvasHelper.printString(remoteAddress,X,y,ANCHOR,
                              plain,canvasWidth-X,g);

                  y+= CanvasHelper.printString("Security settings:",X,y,ANCHOR,
                              bold,canvasWidth-X,g);
                  y+= CanvasHelper.printString("Authenticated: " +
                              (authentication ? "Yes" : "No"),X,y,ANCHOR,
                              plain,canvasWidth-X,g);
                  y+= CanvasHelper.printString("Encrypted: "+
                              (encryption ? "Yes" : "No"),X,y,ANCHOR,
                              plain,canvasWidth-X,g);

            }

      }

}
```

Appendix C KlingsLib

BTServiceAttributeId.java

```
/*
 * BTServiceAttributeId.java
 *
 * Version 1.0
 *
 * 9. June 2004
 *
 * Copyright (c) 2004, Andre N. Klingsheim
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * Redistributions of source code must retain the above copyright notice, this
 * list of conditions and the following disclaimer.
 *
 * Redistributions in binary form must reproduce the above copyright notice,
 * this list of conditions and the following disclaimer in the documentation
 * and/or other materials provided with the distribution.
 *
 * Neither the name of the NoWires research group nor the names of its
 * contributors may be used to endorse or promote products derived from this
 * software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

package org.klings.wireless.BluetoothNumbers;

/**
 * Defines constants for all Bluetooth service attribute IDs found in Section
 * 4.5 &quot;Attribute identifier codes Numeric IDS&quot; in the Bluetooth
 * Assigned Numbers document. The names are prefixed with the acronym for the
 * profile they are related to. Hence, attributes related to the Service
 * Discovery Protocol (SDP) are prefixed SDP_. Consult the Bluetooth Assigned
 * Numbers document for more information.
 * <p>Last updated 09. june 2004.
 *
 * @author  Andr&eacute; N. Klingsheim
 * @version 1.0
 * @since 1.0
 * @see <a  href=
 * "https://www.bluetooth.org/foundry/assignnumb/document/service_discovery">
 * Bluetooth Assigned Numbers</a>
 */
public final class BTServiceAttributeId {

    /**
     * Defines the <code>ServiceRecordHandle</code> short UUID value.<p>
     * <code>SDP_SERVICERECORDHANDLE</code> is set to the constant value 0x0000.
     */
    public static final int SDP_SERVICERECORDHANDLE = 0x0000;

    /**
     * Defines the <code>ServiceClassIDList</code> short UUID value.<p>
     * <code>SDP_SERVICECLASSIDLIST</code> is set to the constant value 0x0001.
     */
    public static final int SDP_SERVICECLASSIDLIST = 0x0001;

    /**
     * Defines the <code>ServiceRecordState</code> short UUID value.<p>
     * <code>SDP_SERVICERECORDSTATE</code> is set to the constant value 0x0002.
     */
    public static final int SDP_SERVICERECORDSTATE = 0x0002;

    /**
     * Defines the <code>ServiceID</code> short UUID value.<p>
     * <code>SDP_SERVICEID</code> is set to the constant value 0x0003.
     */
    public static final int SDP_SERVICEID = 0x0003;

    /**
     * Defines the <code>ProtocolDescriptorList</code> short UUID value.<p>
     * <code>SDP_PROTOCOLDESCRIPTORLIST</code> is set to the constant value 0x0004.
     */
    public static final int SDP_PROTOCOLDESCRIPTORLIST = 0x0004;

    /**
     * Defines the <code>BrowseGroupList</code> short UUID value.<p>
     * <code>SDP_BROWSEGROUPLIST</code> is set to the constant value 0x0005.
     */
    public static final int SDP_BROWSEGROUPLIST = 0x0005;

    /**
     * Defines the <code>LanguageBaseAttributeIDList</code> short UUID value.<p>
     * <code>SDP_LANGUAGEBASEATTRIBUTEIDLIST</code> is set to the constant value
     * 0x0006.
     */
    public static final int SDP_LANGUAGEBASEATTRIBUTEIDLIST = 0x0006;

    /**
     * Defines the <code>ServiceInfoTimeToLive</code> short UUID value.<p>
     * <code>SDP_SERVICEINFOTIMETOLIVE</code> is set to the constant value 0x0007.
     */
    public static final int SDP_SERVICEINFOTIMETOLIVE = 0x0007;

    /**
     * Defines the <code>ServiceAvailability</code> short UUID value.<p>
     * <code>SDP_SERVICEAVAILABILITY</code> is set to the constant value 0x0008.
     */
    public static final int SDP_SERVICEAVAILABILITY = 0x0008;

    /**
     * Defines the <code>BluetoothProfileDescriptorList</code> short UUID value.<p>
     * <code>SDP_BLUETOOTHPROFILEDESCRIPTORLIST</code> is set to the constant value
     * 0x0009.
     */
    public static final int SDP_BLUETOOTHPROFILEDESCRIPTORLIST = 0x0009;

    /**
     * Defines the <code>DocumentationURL</code> short UUID value.<p>
     * <code>SDP_DOCUMENTATIONURL</code> is set to the constant value 0x000A.
     */
    public static final int SDP_DOCUMENTATIONURL = 0x000A;

    /**
     * Defines the <code>ClientExecutableURL</code> short UUID value.<p>
     * <code>SDP_CLIENTEEXECUTABLEURL</code> is set to the constant value 0x000B.
     */
    public static final int SDP_CLIENTEEXECUTABLEURL = 0x000B;

    /**
     * Defines the <code>IconURL</code> short UUID value.<p>
     * <code>SDP_ICONURL = 0x000C</code> is set to the constant value 0x000C.
     */
    public static final int SDP_ICONURL = 0x000C;

    /**
     * Defines the <code>AdditionalProtocolDescriptorLists</code> short UUID
     * value.<p>
     * <code>SDP_ADDITIONALPROTOCOLDESCRIPTORLISTS</code> is set to the constant
     * value 0x000D.
     */
    public static final int SDP_ADDITIONALPROTOCOLDESCRIPTORLISTS = 0x000D;

    /**
     * Defines the <code>GroupID</code> short UUID value.<p>
     * <code>SDP_GROUPID = 0x0200</code> is set to the constant value 0x0200.
     */
    public static final int SDP_GROUPID = 0x0200;

    /**
     * Defines the <code>IpSubnet</code> short UUID value.<p>
     * <code>PAN_IPSUBNET</code> is set to the constant value 0x0200.
     */
    public static final int PAN_IPSUBNET = 0x0200;

    /**
     * Defines the <code>VersionNumberList</code> short UUID value.<p>
     * <code>SDP_VERSIONNUMBERLIST</code> is set to the constant value 0x0200.
     */
    public static final int SDP_VERSIONNUMBERLIST = 0x0200;

    /**
     * Defines the <code>ServiceDatabaseState</code> short UUID value.<p>
     * <code>SDP_SERVICEDATABASESTATE</code> is set to the constant value 0x0201.
     */
    public static final int SDP_SERVICEDATABASESTATE = 0x0201;

    /**
     * Defines the <code>ServiceVersion</code> short UUID value.<p>
     * <code>SERVICEVERSION</code> is set to the constant value 0x0300.
     */
    public static final int SERVICEVERSION = 0x0300;

    /**
     * Defines the <code>Externalnetwork</code> short UUID value.<p>
     * <code>CTP_EXTERNALNETWORK</code> is set to the constant value 0x0301.
     */
    public static final int CTP_EXTERNALNETWORK = 0x0301;

    /**
     * Defines the <code>Network</code> short UUID value.<p>
     * <code>HFP_NETWORK</code> is set to the constant value 0x0301.
     */
    public static final int HFP_NETWORK = 0x0301;

    /**
     * Defines the <code>SupportedDataStoresList</code> short UUID value.<p>
     * <code>SYNCP_SUPPORTEDDATASTORESLIST</code> is set to the constant value
     * 0x0301.
     */
    public static final int SYNCP_SUPPORTEDDATASTORESLIST = 0x0301;

    /**
     * Defines the <code>FaxClass1Support</code> attribute ID value.<p>
     * <code>FAXP_FAXCLASS1SUPPORT</code> is set to the constant value 0x0302.
     */
    public static final int FAXP_FAXCLASS1SUPPORT = 0x0302;

    /**
     * Defines the <code>Remoteaudiovolumecontrol</code> attribute ID value.<p>
     * <code>GAP_REMOTEAUDIOVOLUMECONTROL</code> is set to the constant value
     * 0x0302.
     */
    public static final int GAP_REMOTEAUDIOVOLUMECONTROL = 0x0302;

    /**
     * Defines the <code>FaxClass2_0Support</code> attribute ID value.<p>
     * <code>FAXCLASS20SUPPORT</code> is set to the constant value 0x0303.
     */
    public static final int FAXCLASS20SUPPORT = 0x0303;

    /**
     * Defines the <code>SupportedFormatsList</code> attribute ID value.<p>
     * <code>OPP_SUPPORTEDFORMATSLIST</code> is set to the constant value 0x0303.
     */
    public static final int OPP_SUPPORTEDFORMATSLIST = 0x0303;

    /**
     * Defines the <code>FaxClass2Support</code> attribute ID value.<p>
     * <code>FAXCLASS2SUPPORT</code> is set to the constant value 0x0304.
     */
    public static final int FAXCLASS2SUPPORT = 0x0304;

    /**
```

```
     * Defines the <code>AudioFeedbackSupport</code> attribute ID value.<p>
     * <code>AUDIOFEEDBACKSUPPORT</code> is set to the constant value 0x0305.
     */
    public static final int AUDIOFEEDBACKSUPPORT = 0x0305;

    /**
     * Defines the <code>NetworkAddress</code> attribute ID value.<p>
     * <code>WAP_NETWORKADDRESS</code> is set to the constant value 0x0306.
     */
    public static final int WAP_NETWORKADDRESS = 0x0306;

    /**
     * Defines the <code>WAPGateWay</code> attribute ID value.<p>
     * <code>WAP_WAPGATEWAY</code> is set to the constant value 0x0307.
     */
    public static final int WAP_WAPGATEWAY = 0x0307;

    /**
     * Defines the <code>HomePageURL</code> attribute ID value.<p>
     * <code>WAP_HOMEPAGEURL</code> is set to the constant value 0x0308.
     */
    public static final int WAP_HOMEPAGEURL = 0x0308;

    /**
     * Defines the <code>WAPStackType</code> attribute ID value.<p>
     * <code>WAP_WAPSTACKTYPE</code> is set to the constant value 0x0309.
     */
    public static final int WAP_WAPSTACKTYPE = 0x0309;

    /**
     * Defines the <code>SecurityDescription</code> attribute ID value.<p>
     * <code>PAN_SECURITYDESCRIPTION</code> is set to the constant value 0x030A.
     */
    public static final int PAN_SECURITYDESCRIPTION = 0x030A;

    /**
     * Defines the <code>NetAccessType</code> attribute ID value.<p>
     * <code>PAN_NETACCESSTYPE</code> is set to the constant value0x030B.
     */
    public static final int PAN_NETACCESSTYPE = 0x030B;

    /**
     * Defines the <code>MaxNetAccessrate</code> attribute ID value.<p>
     * <code>PAN_MAXNETACCESSRATE</code> is set to the constant value 0x030C.
     */
    public static final int PAN_MAXNETACCESSRATE = 0x030C;

    /**
     * Defines the <code>IPv4Subnet</code> attribute ID value.<p>
     * <code>PAN_IPV4SUBNET</code> is set to the constant value 0x030D.
     */
    public static final int PAN_IPV4SUBNET = 0x030D;

    /**
     * Defines the <code>IPv6Subnet</code> attribute ID value.<p>
     * <code>PAN_IPV6SUBNET</code> is set to the constant value 0x030E.
     */
    public static final int PAN_IPV6SUBNET = 0x030E;
    /**
     * Defines the <code>SupportedCapabalities</code> attribute ID value.<p>
     * <code>IMAGING_SUPPORTEDCAPABILITIES</code> is set to the constant value
     * 0x0310.
     */
    public static final int IMAGING_SUPPORTEDCAPABILITIES = 0x0310;

    /**
     * Defines the <code>SupportedFeatures</code> attribute ID value.<p>
     * <code>SupportedFeatures</code> is set to the constant value 0x0311.
     */
    public static final int IMAGING_SUPPORTEDFEATURES = 0x0311;

    /**
     * Defines the <code>SupportedFeatures</code> attribute ID value.<p>
     * <code>HFP_SUPPORTEDFEATURES</code> is set to the constant value 0x0311.
     */
    public static final int HFP_SUPPORTEDFEATURES = 0x0311;

    /**
     * Defines the <code>SupportedFunctions</code> attribute ID value.<p>
     * <code>IMAGING_SUPPORTEDFUNCTIONS</code> is set to the constant value 0x0312.
     */
    public static final int IMAGING_SUPPORTEDFUNCTIONS = 0x0312;

    /**
     * Defines the <code>TotalImagingDataCapacity</code> attribute ID value.<p>
     * <code>IMAGING_TOTALIMAGINGDATACAPACITY</code> is set to the constant value
     * 0x0313.
     */
    public static final int IMAGING_TOTALIMAGINGDATACAPACITY = 0x0313;

    /**
     * Defines the <code>ServiceName</code> attribute ID value.<p>
     * <code>SDP_SERVICENAME</code> is set to the constant value 0x100.
     */
    public static final int SDP_SERVICENAME = 0x100;

    /**
     * Defines the <code>ServiceDescription</code> attribute ID value.<p>
     * <code>SDP_SERVICEDESCRIPTION</code> is set to the constant value 0x101.
     */
    public static final int SDP_SERVICEDESCRIPTION = 0x101;

    /**
     * Defines the <code>ProviderName</code> attribute ID value.<p>
     * <code>SDP_PROVIDERNAME</code> is set to the constant value 0x102.
     */
    public static final int SDP_PROVIDERNAME = 0x102;


    /**
     * Useless default constructor.
     * @deprecated
     */
    public BTServiceAttributeId(){

    }
}
```

BTProtocol.java

```
/*
 * BTProtocol.java
 *
 * Version 1.0
 *
 * 09. June 2004
 *
 * Copyright (c) 2004, Andre N. Klingsheim
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * Redistributions of source code must retain the above copyright notice, this
 * list of conditions and the following disclaimer.
 *
 * Redistributions in binary form must reproduce the above copyright notice,
 * this list of conditions and the following disclaimer in the documentation
 * and/or other materials provided with the distribution.
 *
 * Neither the name of the NoWires research group nor the names of its
 * contributors may be used to endorse or promote products derived from this
 * software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

package org.klings.wireless.BluetoothNumbers;

/**
 * Defines constants for all Bluetooth protocol numbers found in Section 4.3
 * &quot;Protocols&quot; in the Bluetooth Assigned Numbers document.
 * Maps short Bluetooth protocol UUIDs to corresponding protocol name.
 * <p>Last updated 09. june 2004.
 *
 * @author  Andr&eacute; N. Klingsheim
 * @version 1.0
 * @since 1.0
 * @see <a href=
 *"https://www.bluetooth.org/foundry/assignnumb/document/service_discovery">
 * Bluetooth Assigned Numbers</a>
 */
public class BTProtocol{

      /**
       * Defines the <code>SDP</code> short UUID value.<p>
       * <code>SDP</code> is set to the constant value 0x0001.
       */
    public static final int SDP = 0x0001;

    /**
     * Defines the <code>UDP</code> short UUID value.<p>
     * <code>UDP</code> is set to the constant value 0x0002
     */
    public static final int UDP = 0x0002;

    /**
     * Defines the <code>RFCOMM</code> short UUID value.<p>
     * <code>RFCOMM</code> is set to the constant value 0x0003
     */
    public static final int RFCOMM = 0x0003;
    /**
     * Defines the <code>TCP</code> short UUID value.<p>
     * <code>TCP</code> is set to the constant value 0x0004
     */
    public static final int TCP = 0x0004;
    /**
     * Defines the <code>TCS_BIN</code> short UUID value.<p>
     * <code>TCS_BIN</code> is set to the constant value 0x0005
     */
    public static final int TCS_BIN = 0x0005;

    /**
     * Defines the <code>TCS_AT</code> short UUID value.<p>
     * <code>TCS_AT</code> is set to the constant value 0x0006
     */
    public static final int TCS_AT = 0x0006;

    /**
     * Defines the <code>OBEX</code> short UUID value.<p>
     * <code>OBEX</code> is set to the constant value 0x0008
     */
    public static final int OBEX = 0x0008;

    /**
     * Defines the <code>IP</code> short UUID value.<p>
     * <code>IP</code> is set to the constant value 0x0009
     */
    public static final int IP = 0x0009;

    /**
     * Defines the <code>FTP</code> short UUID value.<p>
     * <code>FTP</code> is set to the constant value 0x000A
     */
    public static final int FTP = 0x000A;

    /**
     * Defines the <code>HTTP</code> short UUID value.<p>
     * <code>HTTP</code> is set to the constant value 0x000C
     */
    public static final int HTTP = 0x000C;

    /**
     * Defines the <code>WSP</code> short UUID value.<p>
     * <code>WSP</code> is set to the constant value 0x000E
     */
    public static final int WSP = 0x000E;

    /**
     * Defines the <code>BNEP</code> short UUID value.<p>
     * <code>BNEP</code> is set to the constant value 0x000F
     */
    public static final int BNEP = 0x000F;

    /**
     * Defines the <code>UPNP</code> short UUID value.<p>
     * <code>UPNP</code> is set to the constant value 0x0010
     */
    public static final int UPNP = 0x0010;

    /**
     * Defines the <code>HIDP</code> short UUID value.<p>
     * <code>HIDP</code> is set to the constant value 0x0011
     */
    public static final int HIDP = 0x0011;

    /**
     * Defines the <code>HardcopyControlChannel</code> short UUID value.<p>
     * <code>HARDCOPYCONTROLCHAN</code> is set to the constant value 0x0012
     */
    public static final int HARDCOPYCONTROLCHANNEL = 0x0012;

    /**
     * Defines the <code>HardcopyDataChannel</code> short UUID value.<p>
     * <code>HARDCOPYDATACHANNEL</code> is set to the constant value 0x0014
     */
    public static final int HARDCOPYDATACHANNEL = 0x0014;

    /**
     * Defines the <code>HardcopyNotification</code> short UUID value.<p>
     * <code>HARDCOPYNOTIFICATION</code> is set to the constant value 0x0016
     */
    public static final int HARDCOPYNOTIFICATION = 0x0016;

    /**
     * Defines the <code>AVCTP</code> short UUID value.<p>
     * <code>AVCTP</code> is set to the constant value 0x0017
     */
    public static final int AVCTP = 0x0017;

    /**
     * Defines the <code>AVDTP</code> short UUID value.<p>
     * <code>AVDTP</code> is set to the constant value 0x0019
     */
    public static final int AVDTP = 0x0019;

    /**
     * Defines the <code>CMTP</code> short UUID value.<p>
     * <code>CMTP</code> is set to the constant value 0x001B
     */
    public static final int CMTP = 0x001B;

    /**
     * Defines the <code>UDI_C_Plane</code> short UUID value.<p>
     * <code>UDI_C_Plane</code> is set to the constant value 0x001D
     */
    public static final int UDI_C_Plane = 0x001D;

    /**
     * Defines the <code>L2CAP</code> short UUID value.<p>
     * <code>L2CAP</code> is set to the constant value 0x0100
     */
    public static final int L2CAP = 0x0100;

    private static int[] BTProtos = {
            SDP,
                  UDP,
                  TCP,
                  TCS_BIN,
                  TCS_AT,
                  IP,
                  FTP,
                  HTTP,
                  WSP,
                  BNEP,
                  UPNP,
                  HIDP,
                  HARDCOPYCONTROLCHANNEL,
                  HARDCOPYDATACHANNEL,
                  HARDCOPYNOTIFICATION,
                  AVCTP,
                  AVDTP,
                  CMTP,
                  UDI_C_Plane,
                  OBEX,
                  RFCOMM,
                  L2CAP};

    private static String[] BTProtoStrings = {
            "SDP",
                  "UDP",
                  "TCP",
                  "TCS_BIN",
                  "TCS_AT",
                  "IP",
                  "FTP",
                  "HTTP",
                  "WSP",
                  "BNEP",
                  "UPNP",
                  "HIDP",
                  "HardcopyControlChannel",
                  "HardcopyDataChannel",
                  "HardcopyNotification",
                  "AVCTP",
                  "AVDTP",
                  "CMTP",
                  "UDI_C_Plane",
                  "OBEX",
                  "RFCOMM",
            "L2CAP"};

    /**
```

```
     * Useless default constructor.
     * @deprecated
     */
    public BTProtocol(){

    }

    /**
     * Returns the protocolname corresponding to the short UUID, according to the
Bluetooth Assigned Numbers document
     *
     * @param shortUUID The short UUID to look up.
     * @return Protocol name as <code>String</code>. <code>null</code> if the
shortUUID is
     * not specified in the Bluetooth Assigned Numbers document.
     */

    public static String protocolName(int shortUUID) {

        //let's search for our UUID.
        for (int i = BTProtos.length -1 ; i >= 0; i--) {

            if(shortUUID == BTProtos[i]) {

                  return BTProtoStrings[i];
            }
        }

        return null;
    }
}
```

BTServiceClass.java

```
/*
 * BTServiceClass.java
 *
 * Version 1.0
 *
 * 9. June 2004
 *
 * Copyright (c) 2004, Andre N. Klingsheim
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * Redistributions of source code must retain the above copyright notice, this
 * list of conditions and the following disclaimer.
 *
 * Redistributions in binary form must reproduce the above copyright notice,
 * this list of conditions and the following disclaimer in the documentation
 * and/or other materials provided with the distribution.
 *
 * Neither the name of the NoWires research group nor the names of its
 * contributors may be used to endorse or promote products derived from this
 * software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

package org.klings.wireless.BluetoothNumbers;

/**
 * Defines constants for all Bluetooth service class identifiers found in Section
 * 4.4 &quot;Service Class Identifiers and Names&quot; in the Bluetooth
 * Assigned Numbers document.
 * Maps short Bluetooth service class UUIDs to corresponding service class name.
 * <p>Last updated 09. june 2004.
 *
 * @author  Andr&eacute; N. Klingsheim
 * @version 1.0
 * @since 1.0
 * @see <a href=
 * "https://www.bluetooth.org/foundry/assignnumb/document/service_discovery">
 * Bluetooth Assigned Numbers</a>
 */

public final class BTServiceClass {
    /**
     * Defines the <code>ServiceDiscoveryServerServiceClassID</code> short UUID
     * value.<p>
     * <code>SERVICEDISCOVERYSERVERSERVICECLASSID</code> is set to the constant
     * value 0x1000.
     */
    public static final int SERVICEDISCOVERYSERVERSERVICECLASSID = 0x1000;

    /**
     * Defines the <code>BrowseGroupDescriptorServiceClassID</code> short UUID
     * value.<p>
     * <code>BROWSEGROUPDESCRIPTORSERVICECLASSID</code> is set to the constant value
     * 0x1001.
     */
    public static final int BROWSEGROUPDESCRIPTORSERVICECLASSID = 0x1001;

    /**
     * Defines the <code>PublicBrowseGroup</code> short UUID value.<p>
     * <code>PUBLICBROWSEGROUP</code> is set to the constant value 0x1002.
     */
    public static final int PUBLICBROWSEGROUP = 0x1002;

    /**
     * Defines the <code>SerialPort</code> short UUID value.<p>
     * <code>SERIALPORT</code> is set to the constant value 0x1101.
     */
    public static final int SERIALPORT = 0x1101;

    /**
     * Defines the <code>LANAccessUsingPPP</code> short UUID value.<p>
     * <code>LANACCESSUSINGPPP</code> is set to the constant value 0x1102.
     */
    public static final int LANACCESSUSINGPPP = 0x1102;

    /**
     * Defines the <code>DialupNetworking</code> short UUID value.<p>
     * <code>DIALUPNETWORKING</code> is set to the constant value 0x1103.
     */
    public static final int DIALUPNETWORKING = 0x1103;

    /**
     * Defines the <code>IrMCSync</code> short UUID value.<p>
     * <code>IRMCSYNC</code> is set to the constant value 0x1104.
     */
    public static final int IRMCSYNC = 0x1104;

    /**
     * Defines the <code>OBEXObjectPush</code> short UUID value.<p>
     * <code>OBEXOBJECTPUSH</code> is set to the constant value 0x1105.
     */
    public static final int OBEXOBJECTPUSH = 0x1105;

    /**
     * Defines the <code>OBEXFileTransfer</code> short UUID value.<p>
     * <code>OBEXFILETRANSFER</code> is set to the constant value 0x1106.
     */
    public static final int OBEXFILETRANSFER = 0x1106;

    /**
```

```
     * Defines the <code>IrMCSyncCommand</code> short UUID value.<p>
     * <code>IRMCSYNCCOMMAND</code> is set to the constant value 0x1107.
     */
    public static final int IRMCSYNCCOMMAND = 0x1107;

    /**
     * Defines the <code>Headset</code> short UUID value.<p>
     * <code>HEADSET</code> is set to the constant value 0x1108.
     */
    public static final int HEADSET = 0x1108;

    /**
     * Defines the <code>CordlessTelephony</code> short UUID value.<p>
     * <code>CORDLESSTELEPHONY</code> is set to the constant value 0x1109.
     */
    public static final int CORDLESSTELEPHONY = 0x1109;

    /**
     * Defines the <code>AudioSource</code> short UUID value.<p>
     * <code>AUDIOSOURCE</code> is set to the constant value 0x110A.
     */
    public static final int AUDIOSOURCE = 0x110A;

    /**
     * Defines the <code>AudioSink</code> short UUID value.<p>
     * <code>AUDIOSINK</code> is set to the constant value 0x110B.
     */
    public static final int AUDIOSINK = 0x110B;

    /**
     * Defines the <code>A_V_RemoteControlTarget</code> short UUID value.<p>
     * <code>A_V_REMOTECONTROLTARGET</code> is set to the constant value 0x110C.
     */
    public static final int A_V_REMOTECONTROLTARGET = 0x110C;

    /**
     * Defines the <code>AdvancedAudioDistribution</code> short UUID value.<p>
     * <code>ADVANCEDAUDIODISTRIBUTION</code> is set to the constant value 0x110D.
     */
    public static final int ADVANCEDAUDIODISTRIBUTION = 0x110D;

    /**
     * Defines the <code>A_V_RemoteControl</code> short UUID value.<p>
     * <code>A_V_REMOTECONTROL</code> is set to the constant value 0x110E.
     */
    public static final int A_V_REMOTECONTROL = 0x110E;

    /**
     * Defines the <code>VideoConferencing</code> short UUID value.<p>
     * <code>VIDEOCONFERENCING</code> is set to the constant value 0x110F.
     */
    public static final int VIDEOCONFERENCING = 0x110F;

    /**
     * Defines the <code>Intercom</code> short UUID value.<p>
     * <code>INTERCOM</code> is set to the constant value 0x1110.
     */
    public static final int INTERCOM = 0x1110;
    /**
     * Defines the <code>Fax</code> short UUID value.<p>
     * <code>FAX </code> is set to the constant value 0x1111.
     */
    public static final int FAX = 0x1111;

    /**
     * Defines the <code>HeadsetAudioGateway</code> short UUID value.<p>
     * <code>HEADSETAUDIOGATEWAY</code> is set to the constant value 0x1112.
     */
    public static final int HEADSETAUDIOGATEWAY = 0x1112;

    /**
     * Defines the <code>WAP</code> short UUID value.<p>
     * <code>WAP</code> is set to the constant value 0x1113.
     */
    public static final int WAP = 0x1113;

    /**
     * Defines the <code>WAP_CLIENT</code> short UUID value.<p>
     * <code>WAP_CLIENT</code> is set to the constant value 0x1114.
     */
    public static final int WAP_CLIENT = 0x1114;

    /**
     * Defines the <code>PANU</code> short UUID value.<p>
     * <code>PANU</code> is set to the constant value 0x1115.
     */
    public static final int PANU = 0x1115;

    /**
     * Defines the <code>NAP</code> short UUID value.<p>
     * <code>NAP</code> is set to the constant value 0x1116.
     */
    public static final int NAP = 0x1116;

    /**
     * Defines the <code>GN</code> short UUID value.<p>
     * <code>GN</code> is set to the constant value 0x1117.
     */
    public static final int GN = 0x1117;

    /**
     * Defines the <code>DirectPrinting</code> short UUID value.<p>
     * <code>DIRECTPRINTING</code> is set to the constant value 0x1118.
     */
    public static final int DIRECTPRINTING = 0x1118;

    /**
     * Defines the <code>ReferencePrinting</code> short UUID value.<p>
     * <code>REFERENCEPRINTING</code> is set to the constant value 0x1119.
     */
    public static final int REFERENCEPRINTING = 0x1119;

    /**
     * Defines the <code>Imaging</code> short UUID value.<p>
     * <code>IMAGING</code> is set to the constant value 0x111A.
     */
    public static final int IMAGING = 0x111A;
    /**
     * Defines the <code>ImagingResponder</code> short UUID value.<p>
     * <code>IMAGINGRESPONDER</code> is set to the constant value 0x111B.
     */
    public static final int IMAGINGRESPONDER = 0x111B;

    /**
     * Defines the <code>ImagingAutomaticArchive</code> short UUID value.<p>
     * <code>IMAGINGAUTOMATICARCHIVE</code> is set to the constant value 0x111C.
     */
    public static final int IMAGINGAUTOMATICARCHIVE = 0x111C;

    /**
     * Defines the <code>ImagingReferencedObjects</code> short UUID value.<p>
     * <code>IMAGINGREFERENCEDOBJECTS</code> is set to the constant value 0x111D.
     */
    public static final int IMAGINGREFERENCEDOBJECTS = 0x111D;

    /**
     * Defines the <code>Handsfree</code> short UUID value.<p>
     * <code>HANDSFREE</code> is set to the constant value 0x111E.
     */
    public static final int HANDSFREE = 0x111E;

    /**
     * Defines the <code>HandsfreeAudioGateway</code> short UUID value.<p>
     * <code>HANDSFREEAUDIOGATEWAY</code> is set to the constant value 0x111F.
     */
    public static final int HANDSFREEAUDIOGATEWAY = 0x111F;

    /**
     * Defines the <code>DirectPrintingReferenceObjectsService</code> short UUID
     * value.<p>
     * <code>DIRECTPRINTINGREFERENCEOBJECTSSERVICE</code> is set to the constant
     * value 0x1120.
     */
    public static final int DIRECTPRINTINGREFERENCEOBJECTSSERVICE = 0x1120;

    /**
     * Defines the <code>ReflectedUI</code> short UUID value.<p>
     * <code>REFLECTEDUI</code> is set to the constant value 0x1121.
     */
    public static final int REFLECTEDUI = 0x1121;

    /**
     * Defines the <code>BasicPrinting</code> short UUID value.<p>
     * <code>BASICPRINTING</code> is set to the constant value 0x1122.
     */
    public static final int BASICPRINTING = 0x1122;

    /**
     * Defines the <code>PrintingStatus</code> short UUID value.<p>
     * <code>PRINTINGSTATUS</code> is set to the constant value 0x1123.
     */
    public static final int PRINTINGSTATUS = 0x1123;

    /**
     * Defines the <code>HumanInterfaceDeviceService</code> short UUID value.<p>
     * <code>HUMANINTERFACEDEVICESERVICE</code> is set to the constant value 0x1124.
     */
    public static final int HUMANINTERFACEDEVICESERVICE = 0x1124;

    /**
     * Defines the <code>HardcopyCableReplacement</code> short UUID value.<p>
     * <code>HARDCOPYCABLEREPLACEMENT</code> is set to the constant value 0x1125.
     */
    public static final int HARDCOPYCABLEREPLACEMENT = 0x1125;

    /**
     * Defines the <code>HCR_Print</code> short UUID value.<p>
     * <code>HCR_PRINT</code> is set to the constant value 0x1126.
     */
    public static final int HCR_PRINT = 0x1126;

    /**
     * Defines the <code>HCR_Scan</code> short UUID value.<p>
     * <code>HCR_SCAN</code> is set to the constant value 0x1127.
     */
    public static final int HCR_SCAN = 0x1127;

    /**
     * Defines the <code>Common_ISDN_Access</code> short UUID value.<p>
     * <code>COMMON_ISDN_ACCESS</code> is set to the constant value 0x1128.
     */
    public static final int COMMON_ISDN_ACCESS = 0x1128;

    /**
     * Defines the <code>VideoConferencingGW</code> short UUID value.<p>
     * <code>VIDEOCONFERENCINGGW</code> is set to the constant value 0x1129.
     */
    public static final int VIDEOCONFERENCINGGW = 0x1129;

    /**
     * Defines the <code>UDI_MT</code> short UUID value.<p>
     * <code>UDI_MT</code> is set to the constant value 0x112A.
     */
    public static final int UDI_MT = 0x112A;

    /**
     * Defines the <code>UDI_TA</code> short UUID value.<p>
     * <code>UDI_TA</code> is set to the constant value 0x112B.
     */
    public static final int UDI_TA = 0x112B;

    /**
     * Defines the <code>Audio_Video</code> short UUID value.<p>
     * <code>AUDIO_VIDEO</code> is set to the constant value 0x112C.
     */
    public static final int AUDIO_VIDEO = 0x112C;

    /**
     * Defines the <code>SIM_Access</code> short UUID value.<p>
     * <code>SIM_ACCESS</code> is set to the constant value 0x112D.
     */
    public static final int SIM_ACCESS = 0x112D;

    /**
```

```
     * Defines the <code>PnPInformation</code> short UUID value.<p>
     * <code>PNPINFORMATION</code> is set to the constant value 0x1200.
     */
    public static final int PNPINFORMATION = 0x1200;

    /**
     * Defines the <code>GenericNetworking</code> short UUID value.<p>
     * <code>GENERICNETWORKING</code> is set to the constant value 0x1201.
     */
    public static final int GENERICNETWORKING = 0x1201;

    /**
     * Defines the <code>GenericFileTransfer</code> short UUID value.<p>
     * <code>GENERICFILETRANSFER</code> is set to the constant value 0x1202.
     */
    public static final int GENERICFILETRANSFER = 0x1202;

    /**
     * Defines the <code>GenericAudio</code> short UUID value.<p>
     * <code>GENERICAUDIO</code> is set to the constant value 0x1203.
     */
    public static final int GENERICAUDIO = 0x1203;

    /**
     * Defines the <code>GenericTelephony</code> short UUID value.<p>
     * <code>GENERICTELEPHONY</code> is set to the constant value 0x1204.
     */
    public static final int GENERICTELEPHONY = 0x1204;

    /**
     * Defines the <code>UPNP_Service</code> short UUID value.<p>
     * <code>UPNP_SERVICE</code> is set to the constant value 0x1205.
     */
    public static final int UPNP_SERVICE = 0x1205;

    /**
     * Defines the <code>UPNP_IP_Service</code> short UUID value.<p>
     * <code>UPNP_IP_SERVICE</code> is set to the constant value 0x1206.
     */
    public static final int UPNP_IP_SERVICE = 0x1206;

    /**
     * Defines the <code>ESDP_UPNP_IP_PAN</code> short UUID value.<p>
     * <code>ESDP_UPNP_IP_PAN</code> is set to the constant value 0x1300.
     */
    public static final int ESDP_UPNP_IP_PAN = 0x1300;

    /**
     * Defines the <code>ESDP_UPNP_IP_LAP</code> short UUID value.<p>
     * <code>ESDP_UPNP_IP_LAP</code> is set to the constant value 0x1301.
     */
    public static final int ESDP_UPNP_IP_LAP = 0x1301;

    /**
     * Defines the <code>ESDP_UPNP_L2CAP</code> short UUID value.<p>
     * <code>ESDP_UPNP_L2CAP</code> is set to the constant value 0x1302.
     */
    public static final int ESDP_UPNP_L2CAP = 0x1302;
    /**
     * Defines the <code>VideoSource</code> short UUID value.<p>
     * <code>VIDEOSOURCE</code> is set to the constant value 0x1303.
     */
    public static final int VIDEOSOURCE = 0x1303;

    /**
     * Defines the <code>VideoSink</code> short UUID value.<p>
     * <code>VIDEOSINK</code> is set to the constant value 0x1304.
     */
    public static final int VIDEOSINK = 0x1304;

      private static int[] ServiceClassIds = {
                  SERVICEDISCOVERYSERVERSERVICECLASSID,
                BROWSEGROUPDESCRIPTORSERVICECLASSID,
                PUBLICBROWSEGROUP,
                SERIALPORT,
                LANACCESSUSINGPPP,
                DIALUPNETWORKING,
                IRMCSYNC,
                OBEXOBJECTPUSH,
                OBEXFILETRANSFER,
                IRMCSYNCCOMMAND,
                HEADSET,
                CORDLESSTELEPHONY,
                AUDIOSOURCE,
                AUDIOSINK,
                A_V_REMOTECONTROLTARGET,
                ADVANCEDAUDIODISTRIBUTION,
                A_V_REMOTECONTROL,
                VIDEOCONFERENCING,
                INTERCOM,
                FAX,
                HEADSETAUDIOGATEWAY,
                WAP,
                WAP_CLIENT,
                PANU,
                NAP,
                GN,
                DIRECTPRINTING,
                REFERENCEPRINTING,
                IMAGING,
                IMAGINGRESPONDER,
                IMAGINGAUTOMATICARCHIVE,
                IMAGINGREFERENCEDOBJECTS,
                HANDSFREE,
                HANDSFREEAUDIOGATEWAY,
                DIRECTPRINTINGREFERENCEOBJECTSSERVICE,
                REFLECTEDUI,
                BASICPRINTING,
                PRINTINGSTATUS,
                HUMANINTERFACEDEVICESERVICE,
                HARDCOPYCABLEREPLACEMENT,
                HCR_PRINT,
                HCR_SCAN,
                COMMON_ISDN_ACCESS,
                VIDEOCONFERENCINGGW,
                UDI_MT,
                UDI_TA,
                AUDIO_VIDEO,
                SIM_ACCESS,
                PNPINFORMATION,
                GENERICNETWORKING,
                GENERICFILETRANSFER,
                GENERICAUDIO,
                GENERICTELEPHONY,
                UPNP_SERVICE,
                UPNP_IP_SERVICE,
                ESDP_UPNP_IP_PAN,
                ESDP_UPNP_IP_LAP,
                ESDP_UPNP_L2CAP,
                VIDEOSOURCE,
                VIDEOSINK
      };

      private static String[] ServiceClassIdStrings = {
                  "ServiceDiscoveryServerServiceClassID",
                "BrowseGroupDescriptorServiceClassID",
                "PublicBrowseGroup",
                "SerialPort",
                "LANAccessUsingPPP",
                "DialupNetworking",
                "IrMCSync",
                "OBEXObjectPush",
                "OBEXFileTransfer",
                "IrMCSyncCommand",
                "Headset",
                "CordlessTelephony",
                "AudioSource",
                "AudioSink",
                "A_V_RemoteControlTarget",
                "AdvancedAudioDistribution",
                "A_V_RemoteControl",
                "VideoConferencing",
                "Intercom",
                "Fax",
                "HeadsetAudioGateway",
                "WAP",
                "WAP_CLIENT",
                "PANU",
                "NAP",
                "GN",
                "DirectPrinting",
                "ReferencePrinting",
                "Imaging",
                "ImagingResponder",
                "ImagingAutomaticArchive",
                "ImagingReferencedObjects",
                "Handsfree",
                "HandsfreeAudioGateway",
                "DirectPrintingReferenceObjectsService",
                "ReflectedUI",
                "BasicPrinting",
                "PrintingStatus",
                "HumanInterfaceDeviceService",
                "HardcopyCableReplacement",
                "HCR_Print",
                "HCR_Scan",
                "Common_ISDN_Access",
                "VideoConferencingGW",
                "UDI_MT",
                "UDI_TA",
                "Audio_Video",
                "SIM_Access",
                "PnPInformation",
                "GenericNetworking",
                "GenericFileTransfer",
                "GenericAudio",
                "GenericTelephony",
                "UPNP_Service",
                "UPNP_IP_Service",
                "ESDP_UPNP_IP_PAN",
                "ESDP_UPNP_IP_LAP",
                "ESDP_UPNP_L2CAP",
                "VideoSource",
                "VideoSink"
      };

 /**
 * Useless default constructor.
 * @deprecated
 */
public BTServiceClass(){

 }

    /**
     * Returns the service class name corresponding to the short UUID, according to
     * the Bluetooth Assigned Numbers document.
     * @param shortUUID The short UUID to look up.
     * @return Service class name as <code>String</code>. <code>null</code>
     * if the shortUUID is not specified in the Bluetooth Assigned Numbers
     * document.
     */
    public static String serviceClassName(int shortUUID) {

        //let's search for our UUID.
        for (int i = ServiceClassIds.length -1 ; i >= 0; i--) {

            if(shortUUID == ServiceClassIds[i]) {

                return ServiceClassIdStrings[i];
            }
        }

        return null;
    }

}
```

BTUUIDTool.java

```
/*
 * BTUUIDTool.java
 *
 * Version 1.0
 *
 * 21. June 2004
 *
 * Copyright (c) 2004, Andre N. Klingsheim
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * Redistributions of source code must retain the above copyright notice, this
 * list of conditions and the following disclaimer.
 *
 * Redistributions in binary form must reproduce the above copyright notice,
 * this list of conditions and the following disclaimer in the documentation
 * and/or other materials provided with the distribution.
 *
 * Neither the name of the NoWires research group nor the names of its
 * contributors may be used to endorse or promote products derived from this
 * software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

package org.klings.wireless.BluetoothNumbers;

import javax.bluetooth.UUID;
/**
 * Translates 128-bit UUIDs to short UUIDs, returns the hex value of a short UUID
 * as String.
 *
 * @author  Andr&eacute; N. Klingsheim
 * @version 1.0
 * @since 1.0
 */
public class BTUUIDTool {

      private static final String BTBase = "00001000800000805F9B34FB";
      private static String preHex = "0x0000000";

      /**
       * Useless default constructor.
       * @deprecated
```

```
	 */
	public BTUUIDTool(){

	}

	/**
	 * Returns the short UUID value of a long UUID.
	 *
	 * @param uuid UUID to be converted to short UUID
	 * @return Short UUID. -1 if the UUID is not a valid short UUID, meaning
	 * it is not based on the Bluetooth base UUID,
	 * 00000000-0000-1000-8000-00805F9B34FB.
	 */
	public static int shortUUID(UUID uuid){

		if (uuid == null) return -1;

		String id = uuid.toString();
		int offset =id.length()-24;

		if (id.substring(offset).equals(BTBase)) {
			id = id.substring(0,offset);

		}else return -1;

		int result = -1;
		try {
			result = Integer.parseInt(id,16);
		}catch(NumberFormatException nfe){
			return -1;
		}

		return result;

	}

	/**
	 * Returns a zero padded hex value representing the short UUID. If you
	 * supply e.g. 1, you will get: 0x0001.
	 *
	 * @param shortUUID The short UUID to be represented as a
	 * <code>String</code>
	 * @return The short UUID hex value as <code>String</code>.
	 */
	public static String toHexString(int shortUUID){

		if (shortUUID < 0xFFFF){
			String hex = Integer.toHexString(shortUUID);
			hex = preHex.substring(0,6-hex.length()) + hex;
			 return hex;
		}else{
			String hex = Integer.toHexString(shortUUID);
			hex = preHex.substring(0,10-hex.length()) + hex;
			return hex;

		}
	}

}
```

BluetoothInfoCanvas.java

```
/*
 * BluetoothInfoCanvas.java
 *
 * Version 1.0
 *
 * 21. June 2004
 *
 * Copyright (c) 2004, Andre N. Klingsheim
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * Redistributions of source code must retain the above copyright notice, this
 * list of conditions and the following disclaimer.
 *
 * Redistributions in binary form must reproduce the above copyright notice,
 * this list of conditions and the following disclaimer in the documentation
 * and/or other materials provided with the distribution.
 *
 * Neither the name of the NoWires research group nor the names of its
 * contributors may be used to endorse or promote products derived from this
 * software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

package org.klings.wireless.j2me;

import javax.microedition.lcdui.Canvas;
import javax.microedition.lcdui.Font;
import javax.microedition.lcdui.Graphics;
import javax.microedition.lcdui.Ticker;
import javax.bluetooth.LocalDevice;

/**
 * BluetoothInfoCanvas retrieves and prints Bluetooth system properties
 * from the Java/Bluetooth enabled host device. The output is correctly
 * formated according to the screen-size of the device. The user can scroll
 * up and down to see all the relevant properties. Like any other
 * <code>Canvas</code>, <code>Commands</code> etc. can be added to the
 * BluetoothInfoCanvas.
 * Note that no methods are available for the BluetoothInfoCanvas. Just create
 * a new BluetoothInfoCanvas and show it to the user.
 *
 * @author  Andr&eacute; N. Klingsheim
 * @version 1.0
 * @since 1.0
 * @see javax.microedition.lcdui.Canvas
 */
public class BluetoothInfoCanvas extends Canvas {



      /* Different fonts for different types of text */
      private Font plain,bold;

    /*Size of canvas */
      private int canvasHeight, canvasWidth;

    /* Height of fonts */
      private int plainHeight,boldHeight;

    /* Keep track of where we are in the canvas */
      private int y = 0;

      /* Constants used when drawing string */
      private final int X = 1;
      private int anchor = Graphics.LEFT|Graphics.TOP;

      /* Offset used to keep track of which lines to draw */
    int baseOffset = 0;

      /* Keycodes. The user can scroll up and down in the canvas */
    int upKey = getKeyCode(UP);
    int downKey = getKeyCode(DOWN);

    /**
     * Constructs a new BluetoothInfoCanvas object.
     *
     */
      public BluetoothInfoCanvas() {
            super();

            canvasHeight = getHeight();
            canvasWidth = getWidth();

            /* Get height of fonts */
            plain=Font.getFont(Font.FACE_SYSTEM, Font.STYLE_PLAIN, Font.SIZE_MEDIUM);
            bold = Font.getFont(Font.FACE_SYSTEM, Font.STYLE_BOLD, Font.SIZE_MEDIUM);

            /* heights to compute where to draw. */
            plainHeight = plain.getHeight();
            boldHeight = bold.getHeight();

            setTitle("JABWT Information");
            setTicker(new Ticker("JABWT Info Canvas, by Klings @ "+
                  "http://www.klings.org/nowires --- "));
      }


      protected void paint(Graphics g) {

            /* Initialize the canvas */
            g.setColor(0xffffff);
```

```
        g.fillRect(0,0, canvasWidth,canvasHeight);

        /* We want black text */
        g.setColor(0x000000);
        g.setFont(plain);

        y=2;
        int i =0;

        /*
         * Set the offset to baseOffset. If base Offset is 0, all lines
         * will be drawn. If baseOffset is e.g. -1, the first two lines
         * will be skipped.
         */

        int offset = baseOffset;
        String temp = LocalDevice.getProperty("bluetooth.api.version");
        String temp2 = null;
        String[] content;

        if (temp == null){

              y += CanvasHelper.printString("Bluetoth system not available.",X,y,
                          anchor,plain,canvasWidth,g);

              y += CanvasHelper.printString(
                    "Some JABWT devices require that you turn on Bluetooth"+
                    "before starting Java Bluetooth applications. If Bluetooth"
                    +"is already turned on, your device may not support JABWT.",
                          X,y,anchor,plain,canvasWidth,g);
              return;
        }

        if (offset++ >=0){


              y += CanvasHelper.printString("JABWT version:",X,y,anchor,
                          bold,canvasWidth,g);

              y += CanvasHelper.printString(temp,X,y,anchor,plain,canvasWidth,g);

        }

        /* If the canvasHeight is exceeded, there is no need to draw more
         * information.
         */

        if (y > canvasHeight) return;

        if (offset++ >=0){
              temp = LocalDevice.getProperty("bluetooth.l2cap.receiveMTU.max");

              if (temp != null){

                    y += CanvasHelper.printString("L2CAP Max Receive MTU :",X,y,
                                anchor,bold,canvasWidth,g);
                    y += CanvasHelper.printString(temp + " bytes",X,y,anchor,
                                plain,canvasWidth,g);
```

```
            }
      }

      if (offset++ >=0){
            temp = LocalDevice.getProperty("bluetooth.master.switch");

            if (temp != null){

                  y += CanvasHelper.printString("Master/slave switch allowed:",
                              X,y,anchor,bold,canvasWidth,g);
                  y += CanvasHelper.printString(temp,X,y,anchor,plain,
                              canvasWidth,g);

            }
      }

      if (y > canvasHeight) return;

      if (offset++ >=0){
            temp = LocalDevice.getProperty(
                                          "bluetooth.sd.attr.retrievable.max");

            if (temp != null){

                  y += CanvasHelper.printString(
                        "Max service record attributes received:",X,y,anchor,
                              bold,canvasWidth,g);
                  y += CanvasHelper.printString(temp,X,y,anchor,plain,
                              canvasWidth,g);


            }
      }

      if (y > canvasHeight) return;

      if (offset++ >=0){
            temp = LocalDevice.getProperty("bluetooth.connected.devices.max");

            if (temp != null){

                  y += CanvasHelper.printString("Max connected devices:",X,y,
                              anchor,bold,canvasWidth,g);
                  y += CanvasHelper.printString(temp,X,y,anchor,plain,
                              canvasWidth,g);

            }
      }

      if (y > canvasHeight) return;
      if (offset++ >=0){
            temp = LocalDevice.getProperty("bluetooth.sd.trans.max");

            if (temp != null){

                  y += CanvasHelper.printString(
                        "Concurrent service discoveries:",X,y,anchor,bold,
                              canvasWidth,g);
```

```
                y += CanvasHelper.printString(temp,X,y,anchor,plain,
                        canvasWidth,g);


            }
        }

        if (y > canvasHeight) return;

        if (offset++ >=0){
            temp = LocalDevice.getProperty("bluetooth.connected.inquiry.scan");

            if (temp != null){


                y += CanvasHelper.printString(
                        "Inquiry scan during connection:",X,y,anchor,
                        bold,canvasWidth,g);
                y += CanvasHelper.printString(temp,X,y,anchor,
                        plain,canvasWidth,g);

            }
        }

        if (y > canvasHeight) return;
        if (offset++ >=0){
            temp = LocalDevice.getProperty("bluetooth.connected.page.scan");

            if (temp != null){


                y += CanvasHelper.printString(
                        "Page scan during connection:",X,y,anchor,
                        bold,canvasWidth,g);
                y += CanvasHelper.printString(temp,X,y,anchor,
                        plain,canvasWidth,g);


            }
        }
        if (y > canvasHeight) return;

        if (offset++ >=0){
            temp = LocalDevice.getProperty("bluetooth.connected.inquiry");

            if (temp != null){

                y += CanvasHelper.printString("Inquiry during connection:",
                        X,y,anchor,bold,canvasWidth,g);
                y += CanvasHelper.printString(temp,X,y,anchor,
                        plain,canvasWidth,g);

            }
        }


        if (y > canvasHeight) return;

        if (offset++ >=0){
            temp = LocalDevice.getProperty("bluetooth.connected.page");

            if (temp != null){

                  y += CanvasHelper.printString("Paging during connection:",
                              X,y,anchor,bold,canvasWidth,g);
                  y += CanvasHelper.printString(temp,X,y,anchor,
                              plain,canvasWidth,g);

            }
      }
}

protected void keyPressed(int keyCode) {

      if(keyCode == downKey && y > canvasHeight) {

            /*
             * Show one property less in the top of the canvas, which
             * gives one property more in the bottom of the canvas.
             */
            baseOffset--;
      }else if (keyCode == upKey && baseOffset < 0){

            /*
             * Show one property more in the top of the canvas, which
             * gives one property less in the bottom of the canvas.
             */
            baseOffset++;
      }

      repaint();
}

}
```

BluetoothServiceRecordCanvas.java

```
/*
 * BluetoothServiceRecordCanvas.java
 *
 * Version 1.0
 *
 * 21. June 2004
 *
 * Copyright (c) 2004, Andre N. Klingsheim
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * Redistributions of source code must retain the above copyright notice, this
 * list of conditions and the following disclaimer.
 *
 * Redistributions in binary form must reproduce the above copyright notice,
 * this list of conditions and the following disclaimer in the documentation
 * and/or other materials provided with the distribution.
 *
 * Neither the name of the NoWires research group nor the names of its
 * contributors may be used to endorse or promote products derived from this
 * software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */
package org.klings.wireless.j2me;

import java.util.Enumeration;

import javax.bluetooth.DataElement;
import javax.bluetooth.ServiceRecord;
import javax.bluetooth.UUID;
import javax.microedition.lcdui.Canvas;
import javax.microedition.lcdui.Font;
import javax.microedition.lcdui.Graphics;
import javax.microedition.lcdui.Ticker;

import org.klings.wireless.BluetoothNumbers.BTProtocol;
import org.klings.wireless.BluetoothNumbers.BTServiceAttributeId;
import org.klings.wireless.BluetoothNumbers.BTServiceClass;
import org.klings.wireless.BluetoothNumbers.BTUUIDTool;

/**
 * BluetoothServiceRecordCanvas prints the attributes of a Bluetooth
 * <code>ServiceRecord</code>. Only attributes which are set in the
 * <code>ServiceRecord</code>
 * will be printed. The user may scroll up and down in order to see the details
 * for all attributes. The most common attributes related to the Bluetooth
 * Service Discovery Profile (SDP) are shown. These are (with attribute
 * IDs, in the order they are printed by BluetoothServiceRecordCanvas):
 * <ul>
 * <li>0x0100, ServiceName</li>
 * <li>0x0101, ServiceDescription</li>
 * <li>0x0102, ProviderName</li>
 * <li>0x0000, ServiceRecordHandle</li>
 * <li>0x0003, ServiceID</li>
 * <li>0x0001, ServiceClassIDList</li>
 * <li>0x0004, ProtocolDescriptorList</li>
 * <li>0x0009, BluetoothProfileDescriptorList</li>
 * <li>0x0007, ServiceInfoTimeToLive</li>
 * <li>0x0008, ServiceAvailability</li>
 * <li>0x000A, DocumentationURL</li>
 * <li>0x000B, ClientExecutableURL</li>
 * <li>0x000C, IconURL</li>
 *
 * </ul>
 *
 * @author  Andr&eacute; N. Klingsheim
 * @version 1.0
 * @since 1.0
 */

public class BluetoothServiceRecordCanvas extends Canvas {

      /* service record to display */
    ServiceRecord sr = null;

    /* Keep track of where we are in the canvas */
    int y = 0;

    /* Different x values to indent lines */
    final int X1 = 2;
    final int X2 = 5;
    final int X3 = 8;

    /* Anchor for our text */
    final int anchor = Graphics.LEFT|Graphics.TOP;

    /* Different fonts for different types of text */
    Font plain,bold;

    /* Height of fonts */
    int plainHeight,boldHeight;

    /* Dimensions of canvas */
    int canvasHeight, canvasWidth;

    /* Offset used when there are more attributes than space in the canvas */
    int attrOffset = 0;

    /* Keycodes. The user can scroll up and down in the canvas */
    int upKey = getKeyCode(UP);
    int downKey = getKeyCode(DOWN);
    /*
     * The MIDlet can retrieve these URLs and open them in
     * a wap browser.
     */
    String clientExecutableURL = null;
      String documentationURL = null;

      /**
       * Constructs a new BluetoothServiceRecordCanvas object.
       *
       * @param record The ServiceRecord to display.
       */

      public BluetoothServiceRecordCanvas(ServiceRecord record) {
            super();

            this.sr = record;

            /* Fonts for Bold and Plain text */
        plain = Font.getFont(Font.FACE_SYSTEM, Font.STYLE_PLAIN, Font.SIZE_MEDIUM);
        bold = Font.getFont(Font.FACE_SYSTEM, Font.STYLE_BOLD, Font.SIZE_MEDIUM);

        /* font heights to compute where to draw. */
        plainHeight = plain.getHeight();
        boldHeight = bold.getHeight();

        /* Canvas dimensions */
        canvasHeight = getHeight();
        canvasWidth = getWidth();

        setTicker(new Ticker("Service Record Info Canvas, by Klings @ "+
                  "http://www.klings.org/nowires --- "));
      }


      protected void paint(Graphics g) {

            /* Initialize the canvas */
        g.setColor(0xffffff);
        g.fillRect(0,0, getWidth(),getHeight());

        /* We want black text */
        g.setColor(0x000000);
        g.setFont(plain);

        /* Start drawing two pixels from top of screen */
        y = 2;

        DataElement elm = null;

        //temp String
        String out = null;

        int shortUUID = 0;

        int offset = attrOffset;

        /* Get the serviceName */
        elm = (DataElement) sr.getAttributeValue(
                  BTServiceAttributeId.SDP_SERVICENAME);
        if(elm != null && elm.getDataType() == DataElement.STRING
                  && offset++ >=0)  {

            out = (String)elm.getValue();

            /* Print servicename */
            y += CanvasHelper.printString("Service name:",X1,y,anchor,
                        bold,canvasWidth-X1,g);
            y += CanvasHelper.printString(out,X2,y,anchor,plain,
                        canvasWidth-X2,g);
        }

        if (y > canvasHeight) return;

        /* Get the serviceDescription */
        elm = (DataElement) sr.getAttributeValue(
                  BTServiceAttributeId.SDP_SERVICEDESCRIPTION);

        if (elm != null && elm.getDataType() == DataElement.STRING
                  &&  offset++ >= 0) {

            out = (String) elm.getValue();

            /* Print serviceDescription */
            y += CanvasHelper.printString("Service description:",X1,y,
                        anchor,bold,canvasWidth-X1,g);
            y += CanvasHelper.printString(out,X2,y,anchor,plain,
                        canvasWidth-X2,g);
        }

        if (y > canvasHeight) return;


        /* Get the providerName */
        elm = (DataElement) sr.getAttributeValue(
                  BTServiceAttributeId.SDP_PROVIDERNAME);

        if (elm != null && elm.getDataType() == DataElement.STRING
                  &&  offset++ >= 0) {

            out = (String) elm.getValue();

            /* Print providerName */
            y += CanvasHelper.printString("Provider name:",X1,y,anchor,
                        bold,canvasWidth-X1,g);
            y += CanvasHelper.printString(out,X2,y,anchor,
                        plain,canvasWidth-X2,g);
        }

        if (y > canvasHeight) return;

        /* Get the serviceRecordHandle */
        elm = (DataElement) sr.getAttributeValue(
                  BTServiceAttributeId.SDP_SERVICERECORDHANDLE);

        if (elm != null && elm.getDataType() == DataElement.U_INT_4
                  &&  offset++ >= 0) {
```

```
        long var = elm.getLong();
        out = "0x" + Long.toString(var,16);


        /* Print serviceRecordHandle */
        y += CanvasHelper.printString("ServiceRecordHandle:",X1,y,
                    anchor,bold,canvasWidth-X1,g);
        y += CanvasHelper.printString(out,X2,y,anchor,
                    plain,canvasWidth-X2,g);
    }

    if (y > canvasHeight) return;

    /* Get the serviceId */
    elm = (DataElement) sr.getAttributeValue(
              BTServiceAttributeId.SDP_SERVICEID);

    if (elm != null && elm.getDataType() == DataElement.UUID
              &&  offset++ >= 0) {

        UUID var = (UUID) elm.getValue();
        out = "0x" + var.toString();

        /* Print serviceId */
        y += CanvasHelper.printString("ServiceId:",X1,y,anchor,
                    bold,canvasWidth-X1,g);
        y += CanvasHelper.printString(out,X2,y,anchor,
                    plain,canvasWidth-X2,g);

    }

    if (y > canvasHeight) return;

    /* Get the serviceClassIdList */
    elm = (DataElement) sr.getAttributeValue(
              BTServiceAttributeId.SDP_SERVICECLASSIDLIST);

    if (elm != null && elm.getDataType() == DataElement.DATSEQ
              &&  offset++ >= 0) {

        y += CanvasHelper.printString("ServiceClassIdList:",X1,y,
                    anchor,bold,canvasWidth-X1,g);

        /* elm should be a DATSEQ of UUIDs */
        DataElement elm2 = null;
        UUID uuid = null;

        try {
              Enumeration e = (Enumeration) elm.getValue();

              while(e.hasMoreElements()) {
                    elm2 = (DataElement) e.nextElement();

                    if (elm2.getDataType() == DataElement.UUID){
                          uuid = (UUID) elm2.getValue();
                          shortUUID = BTUUIDTool.shortUUID(uuid);
                          if(shortUUID != -1){
                                out = BTUUIDTool.toHexString(shortUUID) +", "+
                                BTServiceClass.serviceClassName(shortUUID);
                          }else{
                                out = "0x"+uuid.toString();
                          }

                          y += CanvasHelper.printString(out,X2,y,anchor,
                                      plain,canvasWidth-X2,g);

                    }
              }

        }catch(ClassCastException cce) {
              y += CanvasHelper.printString("Unpredicted object",X2,y,anchor,
                                plain,canvasWidth-X2,g);

        }

  }

  if (y > canvasHeight) return;

  /* Get the protocolDescriptorList */
  elm = (DataElement) sr.getAttributeValue(
              BTServiceAttributeId.SDP_PROTOCOLDESCRIPTORLIST);

  if (elm != null && elm.getDataType() == DataElement.DATSEQ
              &&  offset++ >= 0) {

        y += CanvasHelper.printString("ProtocolDescriptorList:",X1,y,
                    anchor,bold,canvasWidth-X1,g);

        /*
         * elm should be a DATSEQ of DATSEQ of UUID and
         *  optional parameters
         */
        DataElement elm2 = null;
        DataElement elm3 = null;
        UUID uuid = null;

        try {
              /* Get enumeration to the "outer" DATSEQ */
              Enumeration e = (Enumeration) elm.getValue();

              /* Iterate through the "outer" DATSEQ */
              while(e.hasMoreElements()) {

                    elm2 = (DataElement) e.nextElement();

                    if (elm2.getDataType() == DataElement.DATSEQ){

                          /* Get enumeration to the "inner" DATSEQ */
                          Enumeration e2 = (Enumeration) elm2.getValue();

                          elm3 = (DataElement) e2.nextElement();

                          /* The first element should be a UUID */
                          if (elm3.getDataType() == DataElement.UUID){

                                uuid = (UUID) elm3.getValue();
                                /* Get short UUID */
                                int id = BTUUIDTool.shortUUID(uuid);

                                if (id != -1){
                                      out = BTUUIDTool.toHexString(id) + ", " +
                                            BTProtocol.protocolName(id);
                                }else{
                                      out = "0x"+uuid.toString();
                                }

                                y += CanvasHelper.printString(out,X2,y,anchor,
                                            plain,canvasWidth-X2,g);


                                /*
                                 * If the protocol is L2CAP or RFCOMM, an
                                 * optional parameter is set, the PSM for
                                 * L2CAP or the channel number for RFCOMM.
                                 */
                                if ( (id == BTProtocol.L2CAP
                                            || id == BTProtocol.RFCOMM)
                                            && e2.hasMoreElements()) {

                                      elm3 = (DataElement) e2.nextElement();
                                      int type = elm3.getDataType();

                                      /*
                                       * The PSM or channel number is expected to
                                       *  be an int of some kind
                                       */
                                      if (type >= DataElement.U_INT_1
                                                  && type <= DataElement.INT_16){


                                            if (id == BTProtocol.L2CAP) {
                                                  out = "PSM: " + elm3.getLong();

                                            }else {

                                                  out = "Channel: "
                                                        + elm3.getLong();

                                            }

                                            y += CanvasHelper.printString(
                                                  out,X3,y,
                                                  anchor,plain,canvasWidth-X3,g);

                                      }

                                }//End check for protocols and elements in DatSeq
                          } //End check if elm3 is UUID
                    }//End check for Initial element == DatSeq
              }

        }catch(ClassCastException cce) {
              y += CanvasHelper.printString("Unpredicted object",X3,y,
                                anchor,plain,canvasWidth-X3,g);
            }
        }

        if (y > canvasHeight) return;

        /* Get the BluetoothProfileDescriptorList */
        elm = (DataElement) sr.getAttributeValue(
                  BTServiceAttributeId.SDP_BLUETOOTHPROFILEDESCRIPTORLIST);

        if (elm != null && elm.getDataType() == DataElement.DATSEQ
                  &&  offset++ >= 0) {

            y += CanvasHelper.printString("ProfileDescriptorList:",X1,y,
                        anchor,bold,canvasWidth-X1,g);

            /*
             * elm should be a DATSEQ of DATSEQ pairs with a UUID and
             *  a version number
             */
            DataElement elm2 = null;
            DataElement elm3 = null;
            UUID uuid = null;
            long version = 0;

            try {
                  /* Iterate through the "outer" DataElement sequence */
                  Enumeration e = (Enumeration) elm.getValue();


                  while(e.hasMoreElements()) {
                        elm2 = (DataElement) e.nextElement();

                        if (elm2.getDataType() == DataElement.DATSEQ){
                              /* Enumerate the "inner" DataElement sequence */
                              Enumeration e2 = (Enumeration) elm2.getValue();

                              /*
                               * This is a Dataelement pair.
                               * First DataElement is UUID.
                               */

                              elm3 = (DataElement) e2.nextElement();

                              if (elm3.getDataType() == DataElement.UUID){
                                    uuid = (UUID) elm3.getValue();

                                    shortUUID = BTUUIDTool.shortUUID(uuid);

                                    if (shortUUID != -1){
                                          out = BTUUIDTool.toHexString(shortUUID) +
                                          ", " +
                                          BTServiceClass.serviceClassName(shortUUID);
                                    }else{
                                          out = "0x" + uuid.toString();
                                    }

                                    y += CanvasHelper.printString(out,X2,y,anchor,
                                                plain,canvasWidth-X2,g);
                              }
                              /* The second DataElement is the version number,
                               * probably stored as an int.
                               */

                              elm3 = (DataElement) e2.nextElement();
                              int type = elm3.getDataType();

                              if (type >= DataElement.U_INT_1
                                          && type <= DataElement.INT_16){

                                    version = elm3.getLong();

                                    out = "Version";
                                    out += (version <= 0 ? " unknown":": "+ version);
                                    y += CanvasHelper.printString(out,X3,y,anchor,
                                                plain,canvasWidth-X3,g);

                              } //End version check
                        }//End check of "inner" DataElement sequence
                  }//End iteration through "outer" DataElement sequence

            }catch(ClassCastException cce) {
                  y += CanvasHelper.printString("Unpredicted object",X3,y,anchor,
                                    plain,canvasWidth-X3,g);

            }

        } //End BluetoothProfileDescriptorList


        if (y > canvasHeight) return;

        /* Get the serviceInfoTimeToLive */
        elm = (DataElement) sr.getAttributeValue(
                  BTServiceAttributeId.SDP_SERVICEINFOTIMETOLIVE);

        if (elm != null && elm.getDataType() == DataElement.U_INT_4
                  &&  offset++ >= 0) {

            long var = elm.getLong();
            out = Long.toString(var)+ " seconds";

            /* Print ServiceInfoTimeToLive */
            y += CanvasHelper.printString("ServiceInfoTimeToLive:",X1,y,
                        anchor,bold,canvasWidth-X1,g);
            y += CanvasHelper.printString(out,X2,y,anchor,plain,
                        canvasWidth-X2,g);
        }

        if (y > canvasHeight) return;

        /* Get the serviceAvailability */
        elm = (DataElement) sr.getAttributeValue(
                  BTServiceAttributeId.SDP_SERVICEAVAILABILITY);

        if (elm != null && elm.getDataType() == DataElement.U_INT_1
                  &&  offset++ >= 0) {
```

```
            long var = elm.getLong();
            out = Long.toString(var)+"/255";


            /* Print ServiceAvailability */
            y += CanvasHelper.printString("ServiceAvailability:",X1,y,
                        anchor,bold,canvasWidth-X1,g);
            y += CanvasHelper.printString(out,X2,y,anchor,plain,
                        canvasWidth-X2,g);
        }

        if (y > canvasHeight) return;

        /* Get the DocumentationURL */
        elm = (DataElement) sr.getAttributeValue(
                  BTServiceAttributeId.SDP_DOCUMENTATIONURL);

        if (elm != null && elm.getDataType() == DataElement.URL
                  &&  offset++ >= 0) {

            documentationURL = (String) elm.getValue();

            /* Print DocumentationURL */
            y += CanvasHelper.printString("Documentation URL:",X1,y,anchor,
                        bold,canvasWidth-X1,g);
            y += CanvasHelper.printString(documentationURL,X2,y,anchor,
                        plain,canvasWidth-X2,g);
        }

        if (y > canvasHeight) return;

        /* Get the clientExecutableURL */
        elm = (DataElement) sr.getAttributeValue(
                  BTServiceAttributeId.SDP_CLIENTEEXECUTABLEURL);

        if (elm != null && elm.getDataType() == DataElement.URL
                  &&  offset++ >= 0) {

            clientExecutableURL = (String) elm.getValue();

            /* Print clientExecutableURL */
            y += CanvasHelper.printString("Client Executable URL:",X1,y,
                        anchor,bold,canvasWidth-X1,g);
            y += CanvasHelper.printString(clientExecutableURL,X2,y,anchor,
                        plain,canvasWidth-X2,g);
        }

        if (y > canvasHeight) return;

        /* Get the iconURL */
        elm = (DataElement) sr.getAttributeValue(
                  BTServiceAttributeId.SDP_ICONURL);

        if (elm != null && elm.getDataType() == DataElement.URL
                  &&  offset++ >= 0) {

            out = (String) elm.getValue();
        /* Print Icon URL */
        y += CanvasHelper.printString("Icon URL:",X1,y,anchor,
                    bold,canvasWidth-X1,g);
        y += CanvasHelper.printString(out,X2,y,anchor,
                    plain,canvasWidth-X2,g);
    }
  }


  /**
   * Set a new <code>ServiceRecord</code> to display.
   *
   * @param record The <code>ServiceRecord</code> to display.
   */
  public void setServiceRecord(ServiceRecord record) {
      this.sr = record;
      clientExecutableURL = null;
      documentationURL = null;
      attrOffset = 0;
      repaint();
  }

  /**
   * Returns the Documentation URL for the Bluetooth service described by the
   * <code>ServiceRecord</code> currently displayed.
   *
   * @return The Documentation URL as a <code>String</code>.
   * <code>null</code> if the DocumentationURL attribute is not set in the
   * <code>ServiceRecord</code> currently displayed.
   */
  public String getDocumentationURL(){

    return documentationURL;
  }

  /**
   * Returns the Client Executable URL for the Bluetooth service described by the
   * <code>ServiceRecord</code> currently displayed.
   *
   * @return The Client Executable URL as a <code>String</code>.
   * <code>null</code> if the ClientExecutableURL attribute is not set in
   * the <code>ServiceRecord</code> currently displayed.
   */
  public String getClientExecutableURL(){

    return clientExecutableURL;
  }

  protected void keyPressed(int keyCode) {

    if(keyCode == downKey && y > canvasHeight) {

        /*
         * Show one attribute less in the top of the canvas, which
         * gives one attribute more in the bottom of the canvas.
         */

         attrOffset--;
    }else if (keyCode == upKey && attrOffset < 0){
            /*
             * Show one attribute more in the top of the canvas, which
             * gives one attribute less in the bottom of the canvas.
             */

            attrOffset++;
        }

        repaint();
    }

}
```

CanvasHelper.java

```
/*
 * CanvasHelper.java
 *
 * Version 1.0
 *
 * 21. June 2004
 *
 * Copyright (c) 2004, Andre N. Klingsheim
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * Redistributions of source code must retain the above copyright notice, this
 * list of conditions and the following disclaimer.
 *
 * Redistributions in binary form must reproduce the above copyright notice,
 * this list of conditions and the following disclaimer in the documentation
 * and/or other materials provided with the distribution.
 *
 * Neither the name of the NoWires research group nor the names of its
 * contributors may be used to endorse or promote products derived from this
 * software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

package org.klings.wireless.j2me;

import java.util.Vector;

import javax.microedition.lcdui.Font;
import javax.microedition.lcdui.Graphics;

/**
 * Formats <code>String</code>s so they can be printed in a
 * <code>Canvas</code> without exceeding the <code>Canvas</code> width.
 *
 * @author  Andr&eacute; N. Klingsheim
 * @version 1.0
 * @since 1.0
 */
public class CanvasHelper {


      /**
```

```
     * Useless default constructor.
     * @deprecated
     */
    public CanvasHelper(){

    }

    /**
     * Splits a <code>String</code>. Each fragment will not be longer
     * than maxWidth when printed with the <code>Font</code> font.
     *
     * @param s The <code>String</code> to chop.
     * @param font The <code>Font</code> used when calculating <code>String
     * </code> lengths.
     *
     * @param maxWidth The maximum width of the resulting strings
     * measured in pixels.
     *
     * @return <code>String</code> array containing one or more formatted
     * strings. <code>null</code> if the supplied <code>String</code> or <code>
     * Font</code> is <code>null</code>
     */
    public static String[] splitString(String s, Font font, int maxWidth){

          if(s == null || font == null) return null;

          /* Return String immediately if it does not need formatting */
          if(font.stringWidth(s)<maxWidth){

                String[] res = {s};
                return res;
          }


          maxWidth-= font.charWidth('i');

          /*
           * Compute avg. number of characters on a line by getting the
           * average between the 'i' and 'M';
           */
          int avgChars = maxWidth/
                ((font.charWidth('i')+font.charWidth('M'))/2);


          int offset1 = 0;
          int len = 0;
          boolean longer = false;
          boolean keepOn = false;
          Vector v = new Vector();

          while(s != null){

                /* The remaining String is short enough, so we are done */
                if(font.stringWidth(s)< maxWidth){
                      v.addElement(s);
                      s=null;
                      break;
                }
```

```
            /*
             *  Check that the String length is bigger than avgChars
             * to avoid nullpointer exception.
             */
            len = s.length();
            if(len > avgChars) len = avgChars;

            /* If the String is to short, make it longer.
             * Else make it shorter.
             */
            if (font.substringWidth(s,0,len)< maxWidth) {

                  len++;

                  while(font.substringWidth(s,0,len++) < maxWidth){}


                  v.addElement(s.substring(0,len-1));
                  s = s.substring(len-1);
            }
            else {

                  len--;

                  while(font.substringWidth(s,0,len) > maxWidth)len--;

                  v.addElement(s.substring(0,len));
                  s = s.substring(len);
            }

      }

      /*
       * Copy the results from the Vector to a String array
       * and return the results.
       */
      String[] result = new String[v.size()];

      for (int i = 0;i < result.length;i++){
            result[i] = (String) v.elementAt(i);
      }
      return result;
}

/**
 * Formats and prints a <code>String</code>, possibly over several lines.
 *
 * @param s The <code>String</code> to be printed.
 * @param x The x coordinate of the anchor point.
 * @param y The y coordinate of the anchor point.
 * @param anchor The anchor point for positioning the text.
 * @param f The <code>Font</code> to be used when drawing the text.
 * @param maxWidth Maximum width of the strings, measured in pixels.
 * @param g The <code>Graphics</code> object used when drawing text.
 * @return The number of pixels used to draw the text in the y direction,
 * given by: number of lines drawn * the <code>Font</code> height.
 * @see javax.microedition.lcdui.Graphics#drawString(String str,int x,
 * int y,int anchor)
 */
```

```
      public static int printString(String s, int x, int y, int anchor,
            Font f, int maxWidth, Graphics g){

            int yDelta = 0;
            int fontHeight = f.getHeight();
            g.setFont(f);

            /*
             * If the String does not need formatting, just print it.
             * Else, chop it and print the Strings from the returned array.
             */
            if (f.stringWidth(s) < maxWidth){
                  g.drawString(s,x,y,anchor);
                  yDelta = fontHeight;

            }else{
                  String[] content = splitString(s,f,maxWidth);

                  for (int i = 0; i<content.length;i++){
                        g.drawString(content[i],x,y+yDelta,anchor);
                        yDelta += fontHeight;
                  }
            }

            /* Return the amount of pixels we have moved in the y direction */
            return yDelta;

      }
}
```

Bibliography

[1]: M. S. Gast, *802.11 Wireless Networks*, First Edition, O'Reilly, 2002.

[2]: See `http://www.mobilemag.com/content/100/104/C2783/`

[3]: D. Gratton, *Bluetooth Profiles, The Definitive Guide*, First Edition, Prentice Hall, 2003.

[4]: See `http://www.sun.com/smi/Press/sunflash/2004-02/`

[5]: See `http://www.nokia.com/nokia/0,,32913,00.html`

[6]: B. Hopkings and R. Antony, *Bluetooth for Java*, First Edition, Apress, 2003.

[7]: Kumar et. al., *Bluetooth Application Programming with the Java APIs*, First Edition, Morgan Kaufmann, 2004.

[8]: See `http://forum.nokia.com`

[9]: See `http://sonyericsson.com/developer/`

[10]: See `http://groups.yahoo.com/group/JABWT/`

[11]: See `http://www.kjhole.com`

[12]: J. Bray and C. Sturman, *Bluetooth 1.1, Connect Without Cables*, Second Edition, Prentice Hall, 2002.

[13]: Bluetooth SIG, Specification of the Bluetooth System version 1.1, 2001.

[14]: See `http://www.bluetooth.org`

[15]: See `http://www.bluetooth.com`

[16]: See `http://qualweb.bluetooth.org`

[17]: C. Gehrmann, Bluetooth Security White Paper, 2002.

[18]: See `http://www.symbian.com`

[19]: See `http://java.sun.com/products/cdc/`

[20]: See `http://java.sun.com/j2me/`

[21]: See `http://www.jcp.org`

[22]: See `http://www.jcp.org/en/jsr/detail?id=30`

[23]: See `http://java.sun.com/products/cldc/`

[24]: Sun Microsystems, Inc., Connected, Limited Device Configuration. Specification Version 1.0a, 2000.

[25]: See `http://java.sun.com/products/midp/`

[26]: See `http://www.jcp.org/en/jsr/detail?id=37`

[27]: See `http://www.jcp.org/en/jsr/detail?id=118`

[28]: See `http://www.jcp.org/en/jsr/detail?id=82`

[29]: See `http://www.nowires.org`

[30]: See `http://wireless.klings.org`

[31]: See `http://java.sun.com/products/j2mewtoolkit/`

[32]: See `http://www.rococosoft.com`

[33]: See `http://www.netbeans.org`

[34]: See `https://www.cvshome.org`

[35]: See `http://www.borland.com/mobile/jbuilder/`

[36]: See `http://www.eclipse.org`

[37]: See `http://eclipseme.sourceforge.net`

[38]: J. Knudsen, *Wireless Java: Developing with J2ME*, Second Edition, Apress, 2003.

[39]: See `http://developers.sun.com/techtopics/mobility/`

[40]: See `http://wap.klings.org`

[41]: See `http://www.me4se.org`

[42]: See `http://www.bluez.org`