

MASTER'S THESIS

File Transfer Using Bluetooth

MARCUS CARLSSON
ERIK HOLMBERG

MASTER OF SCIENCE PROGRAMME

Department of Computer Science and Electrical Engineering
Division of Computer Science and Networking

2004:088 CIV • ISSN: 1402 - 1617 • ISRN: LTU - EX - - 04/88 - - SE

FILE TRANSFER USING BLUETOOTH

DESIGNING AND IMPLEMENTING A BLUETOOTH
APPLICATION ACCORDING TO THE BLUETOOTH
SPECIFICATION

Version 2

Marcus Carlsson, Luleå University of Technology

Erik Holmberg, Uppsala University

March 1, 2004

ABSTRACT

Bluetooth is one of the standards for wireless communication. It is expected that the number of services provided over Bluetooth links will rapidly increase during the next few years. This puts hard demands on the Bluetooth specification, to preserve the interoperability between services offered from different manufactures. It's the Bluetooth profile specification that states the requirements on a Bluetooth application.

This thesis investigates one of the existing application profiles, the file transfer application profile and the requirements for the implementation of that profile. The file transfer profile requires some functionality from the underlying profiles and dictates the way that they should be implemented. This thesis will also bring up the different layers in the Ericsson Bluetooth PC Reference Stack.

The Graphical user interface (GUI) of the file transfer applications is designed like a common FTP application. The requirements of the file transfer application are the ability to browse a remote folder structure and to push and pull files. The server decides if the user has read-only or read and write permissions in the shared folders. The server also has the ability to decide which folders a user gets access to.

The file transfer profile depends on several underlying profiles and protocols. Two profiles handle discoverability and connection establishment. For the transferring of files the object exchange protocol is used. This protocol is based on the IrOBEX protocol, a protocol used for Infrared data communication. A complete file transfer must implement all these profiles and protocols.

The application implemented is a client/server file transfer application. It fulfills all the mandatory requirements of the profile specification. It is really a set of five different programs, three running on the server and two on the client. On each there is a security application, which handles discovery and connection establishment. There is one client and one server application, which handle the communication and file browsing. The fifth application is a tool to configure which files that should be shared and to whom.

Preface

This Master Thesis work was done at Ericsson Microelectronics AB at the department of Application Development in Kista during the winter 2000-2001. The thesis concludes the Master of Science study in Electrical Engineering, with a major in Computer Communications, at Luleå University of Technology and Technical Physics at Uppsala University School of Engineering.

Our gratitude goes out to our supervisors Henrik Arfwedson and Johan Meivert at Ericsson Microelectronics who have helped us with support and knowledge during the whole project. We would also like to thank our examiners Bengt Lennartsson at Luleå University of Technology and Ander Svårdström at Uppsala University School of Engineering.

The first version of this report was written by Erik Holmberg and Marcus Carlsson in March 2001. Since then much has happened concerning wireless communication. Companies have developed new Bluetooth hardware and software. Even the Bluetooth specification has changed several times. This means that some information in this report might be out of date.

This report exists in two versions. One version of the report was submitted and accepted by Uppsala University in 2001. Since then the report has been restructured and rewritten by Marcus Carlsson so that it could be used at Luleå University of Technology in 2004.

TABLE OF CONTENTS

1	INTRODUCTION.....	4
1.1	BACKGROUND	4
1.2	PARTICIPANTS.....	4
1.3	REPORT STRUCTURE.....	4
2	PROJECT DESCRIPTION.....	5
2.1	OBJECTIVE	5
2.2	SPECIFICATION.....	5
2.3	LIMITATIONS AND RISKS.....	5
3	THE BLUETOOTH SYSTEM	7
3.1	GENERAL INFORMATION ABOUT BLUETOOTH.....	7
3.2	THE BLUETOOTH MODULE.....	8
3.3	THE HOSTSTACK	10
4	OUR WORK.....	12
4.1	WORKING PROCEDURE.....	12
4.2	HARDWARE	13
4.3	SOFTWARE.....	14
4.4	APPLICATION.....	17
4.5	DESIGN.....	21
4.6	IMPLEMENTATION	26
5	DISCUSSION	35
5.1	WORKING PROCEDURE.....	35
5.2	HARDWARE	35
5.3	SOFTWARE.....	35
5.4	APPLICATION.....	36
5.5	THE FUTURE	37
6	ABBREVIATIONS AND DEFINITIONS.....	38
7	REFERENCES	39
8	TOOLS.....	40

1 INTRODUCTION

In the fall 2000 two students, Marcus Carlsson and Erik Holmberg, started working with their master thesis Ericsson Microelectronics AB¹ at the department of Application Development.

The project objective was to develop an application implementing a higher layer of the Bluetooth specification. The project was done with focus on wireless communication.

The result of this project is an application implementing the Bluetooth File Transfer Profile. The application consists of two parts, a server and a client. The server accepts incoming connections and shares files. The client application allows the user to search for available servers and connect. Files and folder structures can then be transferred between the client and server applications.

1.1 BACKGROUND

Wouldn't it be great to make every minute count? If you could conveniently connect to your corporate network or send/receive email while waiting in the doctor's office, having your car serviced, or even traveling around the globe? Imagine having your mobile PC and PDA automatically synchronize your address list and calendar as soon as you walk through your office door. Imagine never again having to wrestle with cables to connect electronic devices. With Bluetooth wireless technology, these features, and many more, will soon become routine functions of mobile computing.

The authors of this report believe that wireless communication will be used more and more in the future. Bluetooth is one of the technologies competing to become one of the standards for wireless communication. With this project the Bluetooth specification will be studied and investigated. Furthermore an implementation of a Bluetooth application will be made according to the specification.

1.2 PARTICIPANTS

The participants in this thesis project are the students Marcus Carlsson and Erik Holmberg and the supervisors Henrik Arfwedson and Johan Meivert on Ericsson Microelectronics.

1.3 REPORT STRUCTURE

This report is divided into four chapters. The project description part gives a thorough description of the objective, goal and limitations of the thesis work.

The Bluetooth System section contains a short description of the Bluetooth hardware and software. This can be skipped by readers already familiar with the Bluetooth architecture.

Our work is a chapter that describes all the steps done in this thesis work. It starts with the planning of the work and finishes with finished application. It describes the design and development phases.

In Section 5 the experience gained and thoughts about this thesis work are discussed.

¹ Ericsson Microelectronics was bought by Infineon Technologies in 2002.

2 PROJECT DESCRIPTION

This section describes what should be achieved as well as the boundaries of the thesis project.

2.1 OBJECTIVE

The main objective of this thesis is to study and investigate the Bluetooth specification. This is achieved by the development of an application implementing a Bluetooth profile. The application should implement one of the application profiles in the Bluetooth Specification 1.0 b. [3]. The application chosen is Network Neighborhood which implements the File Transfer Profile. This profile specifies a client-server type of application. The server makes itself available to other devices, accepts incoming connections and allows basic file transfer operations. The client should find devices with the corresponding service, make a connection and allow the user to push, pull and manipulate files on the server. The application should be able to interoperate with other applications following the specification.

The scope of the thesis is limited to the File Transfer profile. However the File Transfer Profile is highly dependent on the underlying layers in the stack and profiles. Thus many features which are required in lower profiles are also discussed thoroughly in this report.

2.2 SPECIFICATION

This section will try to give an early requirement specification for the resulting application. The requirements will become more and more specific during the project since an important part of the work is to find the system requirements. Some parameters experienced as important for the final products are listed below.

The application shall:

- Discover and identify other devices
- Find desired services in other devices
- Connect to and communicate with a device implementing the same profile
- Present an “easy to use” graphical user interface

2.3 LIMITATIONS AND RISKS

Since this project is a development project there are a couple of uncertainty parameters that should be considered.

2.3.1 LIMITATIONS

A number of Bluetooth stacks were available when this thesis work started. However few are certified and even fewer are tested for interoperability. It might turn out that two devices running

stacks from different vendors can't communicate at all. Hopefully the functionality can be shown using the same stack on both devices.

2.3.2 RISKS

The Bluetooth technology² is a new and unexplored area. The Bluetooth specification 1.1 [Ref. 1] is determined the same week as writing this, so it is not certain that this thesis work will result in anything more than an investigation and a research report. The project may take longer to complete than the scheduled 20 weeks and therefore there is a possibility that we will have to finish off even before we have a functioning application.

² This report was first written in 2001 when the Bluetooth Technology was new.

This section will discuss the corner stones, which are used as a foundation for a Bluetooth application. It will also give the reader some background information about Bluetooth. The Bluetooth hardware figures and information are mainly from the Ericsson Bluetooth Module Irma C, but other solutions have similar architectures.

3.1 GENERAL INFORMATION ABOUT BLUETOOTH

Bluetooth is a wireless personal area network (PAN) technology from the Bluetooth Special Interest Group, (www.bluetooth.com), founded in 1998 by Ericsson, IBM, Intel, Nokia and Toshiba. Bluetooth is an open standard for short-range transmission of digital voice and data between mobile devices (laptops, PDAs, phones) and desktop devices. It supports point-to-point and multipoint applications. Bluetooth provides up to 723 kb/s data transfer within a range of 10 m and up to 100 m with a power amplifier. Unlike IrDA, which requires that devices are aimed at each other (in line of sight), Bluetooth uses omnidirectional radio waves that can transmit through walls and other non-metal barriers. Bluetooth transmits in the unlicensed 2.4GHz band and uses a frequency hopping spread spectrum (FHSS) technique that changes its signal 1600 times per second. If there is interference from other devices, the transmission does not stop, but its speed is downgraded. The name Bluetooth comes from King Harald Blatand (Bluetooth) of Denmark.

The Bluetooth Specification is developed by the Bluetooth Special Interest Group (SIG). Their goal is to specify rules for the radio and data communication over Bluetooth. A layered protocol stack referred to as the “Bluetooth protocol architecture” has been developed. The architecture dictates how communication should be performed at all levels from radio up to high-level application protocols. This would allow applications written in conformance to the Specification to interoperate with each other.

Bluetooth was originally designed to eliminate cables between remote devices in a short range, e.g. between a digital camera and a cellular phone. Thus it was implemented to be cheap and have low power consumption.

The interest in the Bluetooth technology grew and more companies joined the SIG. As people understood that Bluetooth would be a new fairly large standard for wireless communication, this resulted in a great demand from the market for Bluetooth products and OEM products for use in customer products.

The ultimate objective of the Bluetooth specification is to allow applications, written in a manner that is conformant to the specification, to interoperate with each other. To achieve this, matching applications in remote devices, e.g. client/server, must run over identical protocol stacks. That is on a large scale what the Bluetooth stack is, a number of protocols following a standard. This does not mean that all Bluetooth devices should be able to send and receive data to/from every other Bluetooth device, but it does mean that a Bluetooth device from Toshiba running OSE shall be able to speak with another device from Ericsson running Linux.

A Bluetooth system today usually consists of two physically separated parts. One is the Bluetooth module and the other one is the host, e.g. the PC or the embedded system that is going to use the transferred information. These two parts communicate with each other over UART or USB using the Host controller interface, HCI, protocol. Thus the Bluetooth stack is divided in two parts.

The stack which resides inside the host, e.g. PC or PDA, is referred to as the hoststack. More information about Bluetooth can be found in [7].

3.2 THE BLUETOOTH MODULE

The module itself is a small embedded system on a small PCB and it consists of a number of parts; the Baseband (the processing unit), the Flash memory, a 13MHz crystal, and a radio module. The module has not itself an internal antenna; such a device must be added externally. The Ericsson Irma C module layout is depicted in Figure 1.

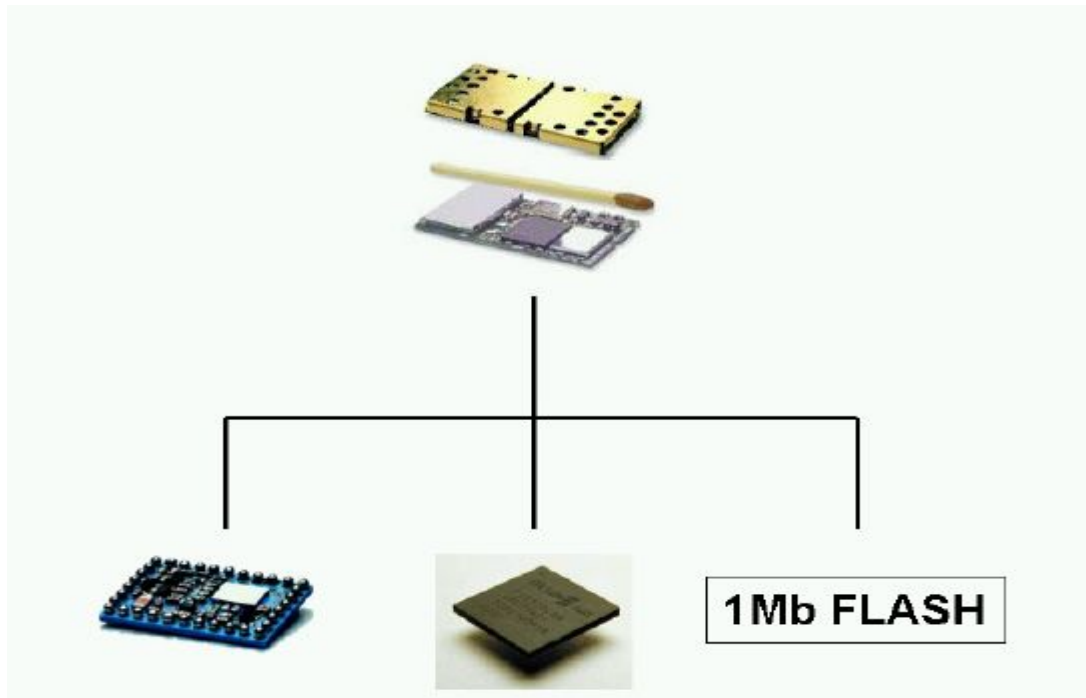


Figure 1, The Bluetooth module and a match

The baseband chip inside the module is used to control the radio traffic. Watched from inside the baseband chip, it processes commands from the host, establishes links to other Bluetooth devices and sends commands, status and data over the radio link. It also, of course, receives data and commands from other devices and handles error correction, encryption and retransmissions. The baseband chip, depicted in the mid/bottom of Figure 1, encapsulates an Arm7 processing unit, 56k RAM, different communication interfaces and 8 address pins that can be set to be GP I/O pins if one wishes to. It has also some Bluetooth specific hardware where some of the functionality resides [ref.2]

An external flash memory of 8MBit (1Mb) is included in the module. The Bluetooth firmware resides inside this memory. A flash memory is used today³, even though it is much more expensive than a ROM, this is probably mainly because of the constant upgrades of the firmware under the development phase that Bluetooth module undergoes today. In the future the flash will most certainly be replaced with a ROM.

³ In 2003 and 2004 this is still true even though some vendors have changed to ROM.

The Bluetooth radio module is in fact a module in itself. It is a short-range microwave frequency radio transceiver for Bluetooth communication. The module is designed to operate in the globally available ISM frequency band, 2.4 -2.5 GHz. Fast frequency hopping (1600 channel hops/s) with 79 channels available (2.402 to 2.480 GHz) and a maximum TX & RX bit rate of 1 MBit/s exploits the maximum channel bandwidth allowed in the unlicensed ISM band. Figure 2 shows the Ericsson radio module.



Figure 2, The Bluetooth Radio

The Bluetooth module includes firmware for the Host Controller Interface (HCI), and the link manager (LM) and the baseband operating system; in Ericsson module it is OSE delta. The firmware (FW) resides in the flash. The protocol stack in the firmware is fully specified in the Bluetooth specification since it is absolutely necessary that the information sent runs over identical protocol stacks. The protocol stack in the firmware consists of two layers, the HCI and the Link Manager Protocol layer seen in Figure 3. The Link Controller is a hardware block which controls the radio.

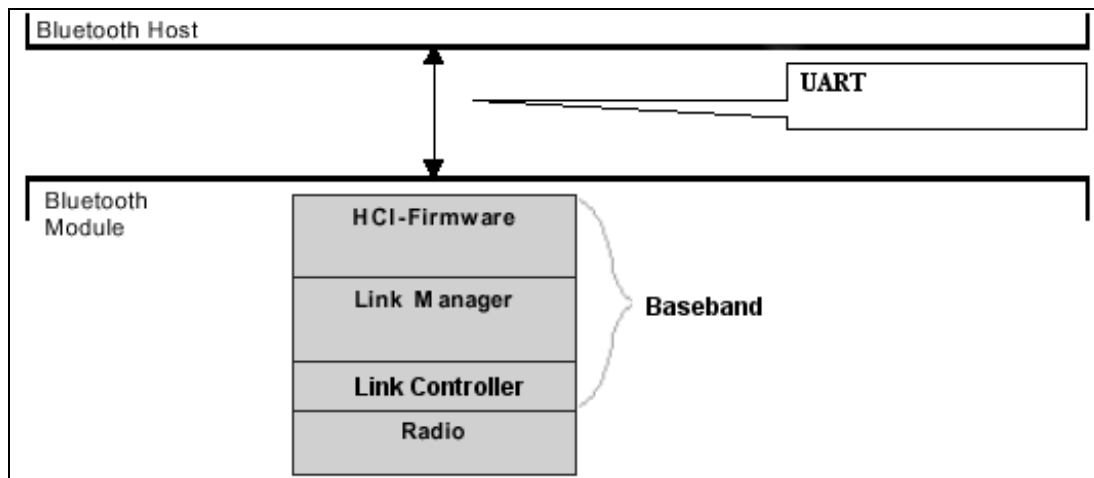


Figure 3, The Controller Bluetooth stack

The HCI layer can be seen as a "Bluetooth language". It is a set of instructions used for communication between a Host and a Host Controller (e.g. a Bluetooth module and a PC). The HCI protocol layer has a set of standardized commands and signals but can also implement producer specific commands. The HCI layer communicates with either the host's HCI layer through a hardware interface driver or it communicates with the Link Manager.

The Link Manager in each Bluetooth module can communicate with another Link Manager in another Bluetooth module using the Link Manager Protocol (LMP), which is a peer-to-peer protocol. The Link Manager assembles data and control messages sent over the air. The Link Manager exchanges information with the radio module through the link controller.

3.3 THE HOSTSTACK

This is on a large scale a summary of the different protocols and criteria for Bluetooth products significant for the basic understanding of Bluetooth and our system. More information on this topic is found in the Bluetooth specification [2] and [3].

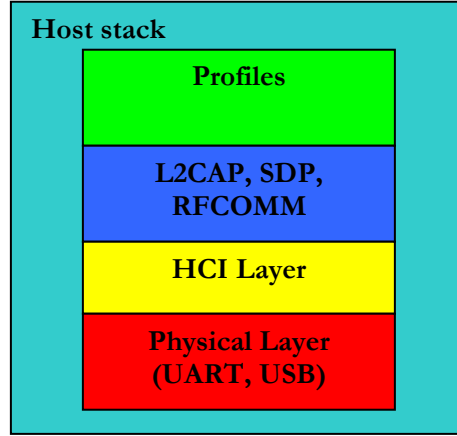


Figure 4, The hoststack

In Figure 4 the basic layers of a hoststack are depicted. The HCI layer is the lowest layer in the hoststack (and the highest layer in the host controller stack) and it is the one that communicates with the Bluetooth device over a serial connection. HCI is essentially a standardized communication language between host and host controller. The Host Control Transport Layer (i.e. physical bus, for example UART) driver provides the HCI layers with the ability to exchange information with each other. The objective of the HCI-UART Transport Layer is to make it possible to use the Bluetooth HCI over a serial interface between two UART's on the same PCB.

HCI is a standardized Bluetooth "command language" and is an abbreviation for Host Controller Interface. There are four kinds of HCI packets that can be sent; i.e. HCI Command Packet, HCI Event Packet, HCI ACL Data Packet and HCI SCO Data Packet and all the packages have different packet structures. The command packet is sent from the host to the host controller and is used for the set up of the Bluetooth device and for the connection between the units. The event packet is sent from the host controller to the host to confirm that a command has been completed. The data packet is used to exchange data between the host and the host controller. The HCI layer does not provide the ability to differentiate the four HCI packet types. Therefore, if the HCI packets are sent via a common physical interface, a packet indicator has to be added. The HCI packet indicator shall be sent immediately before the HCI packet. All four kinds of HCI packets have a length field, which is used to determine how many bytes are expected for the HCI packet. When an entire HCI packet has been received, the next HCI packet indicator is expected for the next HCI packet. Over the UART Transport Layer, only HCI packet indicators followed by HCI packets are allowed.

The Service Discovery Protocol, SDP is a crucial part of the Bluetooth framework. Finding the service wanted is fundamental for all the usage models. The SDP provides a means for applications to discover which services are available and to determine the characteristics of those available services using an existing L2CAP connection. After that, an appropriate separate connection between two or more Bluetooth devices can be established using information obtained via SDP. The service discovery application does not make use of SDP as a means for accessing a service, but rather as a means for informing the user of a local device about the services available on remote devices. For

example a Bluetooth enabled phone wants to connect to a PC and send a picture. It connects and asks, through SDP, if the PC supports the File Transfer Profile. If the PC does, the phone can use the parameters retrieved through SDP and send the picture.

L2CAP is an acronym for Logical Link Control and Adaptation Layer Protocol. L2CAP is based around the concept of "channels". A channel is a logical link between two L2CAP layers on physical devices, which means that it handles multiplexing. A channel identifier (CID) refers to each end-point of an L2CAP channel. An L2CAP implementation exposes the outgoing maximum transfer unit (MTU) and segments higher layer packets into 'chunks' that can be passed to the Link Manager via the Host Controller Interface (HCI). On the receiving side, an L2CAP implementation receives 'chunks' from the HCI and reassembles those chunks into L2CAP packets using information provided through the HCI from the packet header. Compared to wired physical media, the data packets defined by the Baseband Protocol are limited in size. Exporting a maximum transmission unit (MTU) associated with the largest Baseband payload (341 bytes for DH5 packets) limits the efficient use of bandwidth for higher layer protocols that are designed to use larger packets, but L2CAP is designed to have as little overhead as possible to minimize this effect. Large L2CAP packets must be segmented into multiple smaller Baseband packets prior to their transmission over the air. Similarly, multiple received Baseband packets may be reassembled into a single larger L2CAP packet. The Segmentation and Reassembly (SAR) functionality is absolutely necessary to support protocols using packets larger than those supported by the Baseband. L2CAP permits higher level protocols and applications to transmit and receive L2CAP data packets up to 64 kB in length. If for example a big IP packet comes in it is segmented into smaller packets to be able to be sent over the radio link.

RFCOMM is a transport protocol, with additional provisions for emulating RS-232 serial ports. The protocol is based on the ETSI standard TS 07.10. RFCOMM is a simple transport protocol, with additional provisions for emulating the 9 circuits of RS-232 (EIA/TIA-232-E) serial ports. The RFCOMM-protocol supports up to 60 simultaneous connections between two BT devices. For the purposes of RFCOMM, a complete communication path involves two applications running on different devices with a communication segment between them. RFCOMM is able to transmit/receive data packets of up to 32 kB over such a link.

The Bluetooth profiles are standards and references developed by the Bluetooth SIG for implementation of different usage models. This will secure that a product designed to be able to connect to a file transfer server always will be able to connect to one, independently of who has designed it. The usage models describe APIs and functionality that has to be implemented in the products to get this desired compatibility. The profiles below are just examples of some of the most widely spread profiles. Only a few profiles will be described later in this document. The rest can be found in the Bluetooth specification [2] and [3].

- File transfer profile – Describes the actions and requirements that have to be supported by an application when browsing, retrieving and sending files between Bluetooth devices. Found in PCs and PDAs.
- Headset profile – Specifies the interface that is used for communication between phone and a headset. Describes the possible user actions as well as commands used between phone and headset. Used in Mobile Phones.

This section will describe our work and the steps to achieve the objective in this thesis work.

4.1 WORKING PROCEDURE

This section is used to describe the working procedure that was used to achieve a functioning file transfer application. The actions will be presented in chronological order.

The thesis work started in the fall 2000 with a very vague description of the objective. The goal was to have an application implementing a profile. This had to be refined to a more specific description. A lot of choices had to be made. Which hardware, Bluetooth stack and profile should be used? To answer these questions more knowledge about Bluetooth was needed. Step one in the working procedure was therefore to study the Bluetooth specification.

During step one many ideas for applications arose. The next step would be to use the new knowledge and decide on one application. To do that Bluetooth hardware, software and host platform had to be taken into account. The Ericsson Irma C Bluetooth module had already been selected. But could the available Bluetooth stacks provide the functionality needed by the application? A comparison of the available stacks was therefore made.

The next step was the design of the application. Since Bluetooth is designed as a layered architecture, the profiles are very dependent on the protocols below. Therefore the chosen hardware and stack had to be studied closer.

After the design phase the implementation phase started. The application was developed using 3 steps.

- Achieve communication between the Bluetooth stack and the module.
- Achieve communication between two Bluetooth devices, using stack and application.
- Achieve profile and application communication.

In the last step a whole protocol will be implemented. The last step will therefore require most of the time and effort.

4.2 HARDWARE

When selecting Bluetooth hardware the natural selection was the Irma C from Ericsson Microelectronics. This choice was of course influenced by the fact that the thesis work was done at Microelectronics. However the alternatives out on the market weren't that many. The Irma C was the second generation Bluetooth chip from Ericsson and had already been proven in consumer products. Thus it was a good choice. When working with the Irma C module the development board Ericsson Bluetooth Starter Kit, EBSK, was used. For more information see [5] and [6].

The EBSK consists of two PCBs called Motherboard and Daughterboard, which are connected together, see Figure 5 below.

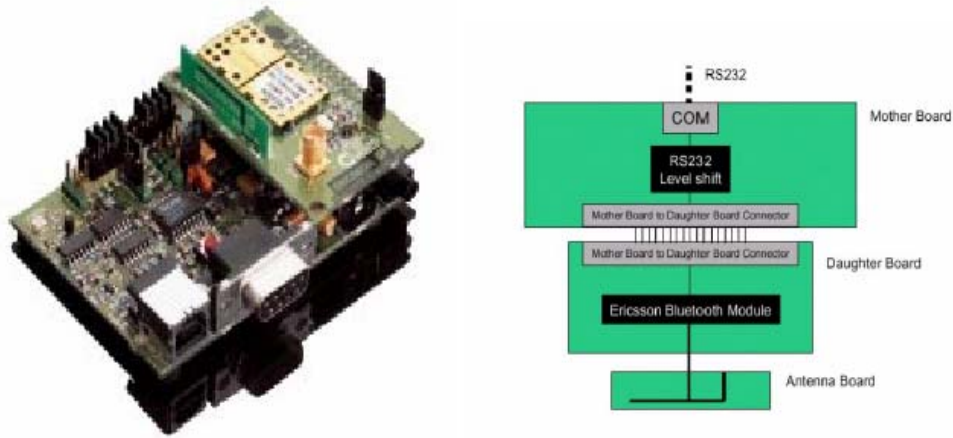


Figure 5, Ericsson Bluetooth Starter Kit and the EBSK block diagram

The EBSK connects to the host using an ordinary 9-pin D-sub serial port or USB. The Motherboard carries power circuits, power filters, start-up circuitry, voice codec and interfaces. The Daughterboard, which is plugged on the Motherboard, is replaceable making it easy to upgrade.

The daughter board carries the Ericsson Bluetooth module. In addition to this it also has a RF switch to choose to either use the onboard antenna or an external antenna connected to the SMA connector. The Daughterboard has 22 connectors drawn out on the PCB, but the baseband chip has many more. The reason for this is that these pins on the daughter board are the only ones drawn out on the module, which in turn also is a PCB with a number of components on it. Communication is carried out using the module's built-in high-speed USB (Universal Serial Bus), UART or PCM interface (for voice). The layout of the daughter board can be seen in Figure 6.

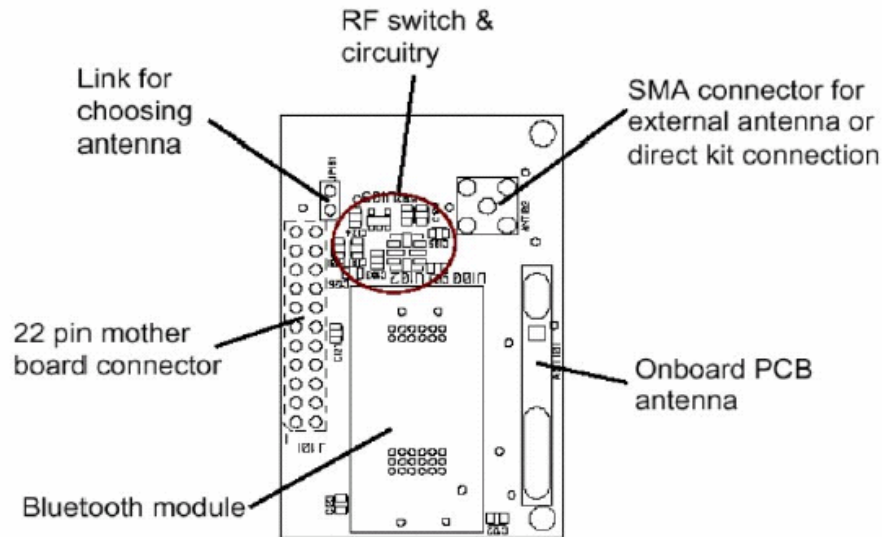


Figure 6, EBSK Daughterboard layout

When selecting the host platform the natural selection was a Desktop and Laptop computer. The desktop could illustrate an access point which was immobile. While the laptop could move around and still connect to the desktop. Most laptops and desktops already have a UART available for the communication with the Bluetooth Module

4.3 SOFTWARE

On the Host computers the operating system was selected to Windows NT or Windows 98. These two operating systems are⁴ used on a large part of all desktops and laptops. The decision was also based on the fact that appropriate development tools already were available for these platforms. The following section describes the process of selecting a stack for these platforms. After that the selected stack is described in more detail.

4.3.1 SELECTING STACK

Many companies have developed their own Bluetooth stacks, however many of these are for specific platforms. Since the target platform was a PC running windows a generic stack or one for that specific platform was needed.

The other property that was important was stability and interoperability. Since so few stacks has been qualified⁵ it's very hard to determine if they will work together. However the qualification proves that they work against a reference system.

⁴ Windows NT and Windows 98 were popular in 2000/2001.

⁵ Several stacks have been qualified since 2000

In Table 1 below is a short list of some of the stacks that were part of the evaluation when the thesis work started. Only the ones that were possible as platforms will be discussed.

Company	Operating System	Qualified	Free of charge
Ericsson	Generic (or windows)	Yes	Free for Ericsson
Axis	Linux (source)	No	Yes
Enea	OSE	Yes	No
Widcomm	GKI (generic)	Yes	No
IBM	Linux	No	Yes
Iar Systems	Generic	No	Yes (for eval.)

Table 1. Bluetooth stacks

The Ericsson stack is available both as generic Ansi C code and as a PC reference stack. The stack contains all layers needed to build end-user applications. All layers are made according to the specification and are qualified. The stack is already used in several customer oriented projects. Another positive aspect with the Ericsson stack is that it has been tested with the Irma C module.

The axis stack is available as source code. However it is not qualified. Another drawback is that it does not include the SDP layer. SDP is needed in all applications implementing a profile. Therefore it is not a candidate.

The stack from Widcomm seems to be very complete and flexible. It is completely qualified and is available in source. Several Companies are already using this stack for customer products. The drawback is that it isn't free of charge.

Iar System has developed a generic stack. However at the time it lacks many of the needed features. The SDP part is very limited and does only support a few predefined profiles.

The choice of stack was made with respect to stability and adaptation to the target platform. The hoststack chosen is the Ericsson stack, more detailed information about it follows below.

4.3.2 THE ERICSSON STACK

The Ericsson Bluetooth hoststack is built in a layered architecture where each layer can use the ones below. Each layer encapsulates its own data and other layers are not polluted except at the interfaces. It has been designed with portability in mind and also time-to-market in producing any new applications. This is achieved by a cradle architecture shown in Figure 7 below. Three interface layers are used to create static boundaries between the stack and the applications, the OS and the communication ports. The virtual OS interface separates the protocols from OS dependent features and the serial interface layer encapsulates the port drivers. These two layers ensure easy portability since only the HCI driver and the OS adaptation have to be implemented on a new platform. The application programmer's interface provides a static interface for applications and other layers above the hoststack. It gives access to the interfaces of all layers in the stack. Since the hoststack has static

interfaces on all sides it is easy to improve the protocols without having to make changes in the applications. More information about this stack can be found in [4].

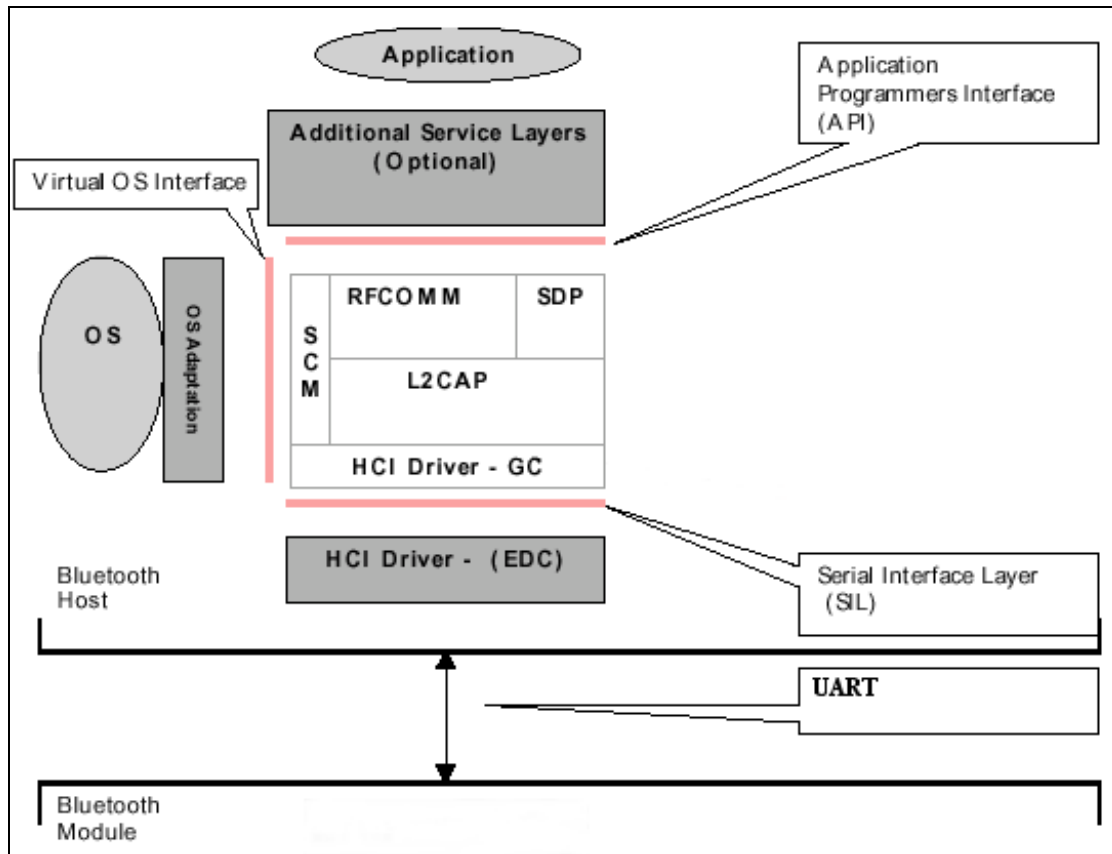


Figure 7, The Ericsson hoststack

There are five main layers in the Ericsson hoststack.

SCM - Stack Connection Manager – The Stack Connection Manager handles and administrates the Bluetooth baseband connection. It is used by an application to create and take care of data and voice links independent of other applications. This enables several applications on top of the stack communicating over different channels but on the same link, with the local routing handled within L2CAP.

RFCOMM - is a transport protocol, which emulates a RS-232 serial port. It provides means for data communication with flow control to higher level layers. These layers can be end-user applications as well as higher layer protocols.

SDP - Service Discovery Protocol - Discovery of services is a crucial part of the Bluetooth framework. Service discovery is fundamental for all the usage models. The SDP provides a means for applications to discover which services are available and to determine the characteristics of those available services using an existing L2CAP connection. After that, an appropriate separate connection between two or more Bluetooth devices can be established using information obtained via SDP. The service discovery application does not make use of SDP as a means of accessing a service, but rather as a means of informing the user of a Local Device about the services available to his/her device by (and possibly via) Remote Device(s).

L2CAP - Logical Link Control and Adaptation Protocol - provides connection-oriented and connectionless data services to upper layer protocols with protocol multiplexing capability, segmentation and reassemble operation, and group abstractions. L2CAP permits higher level protocols and applications to transmit and receive L2CAP data packets up to 64 kB in length.

HCI Driver - Host Controller Interface - The HCI Interface establishes the communication between the stack and the HCI Firmware in the Bluetooth module connected to the equipment running the stack. The HCI Interface ensures that the stack can run over a generic hardware. The HCI Interface is standardized by the Bluetooth SIG. The HCI driver, as developed in this software, is split into a Generic Component and an Environment Dependent Component.

If a layer wants to make use of the services of another layer it has to register itself to it. This can be done with the request register function that is present in every API of every layer. If this is done the two layers have an interface with each other. If a layer doesn't want to make use of another layer anymore, it can call the request de-registration function.

The communication between layers is done by function calls and messages. There are two different communication messages, control and data. Control messages are used to exchange information between layers. Data messages are used to send data through the stack. At the sending side every layer adds its heading to the data and sends it down. On the receiving side the header is peeled off for each layer on its way up to the receiving layer. Communication from a higher layer or an application to a lower layer is done by a call to an API-function in the lower layer. When a lower layer communicates with a higher layer interprocess messages are used.

4.4 APPLICATION

When both the hardware and software platforms were chosen, there were still a variety of possible applications. Most applications using Bluetooth could use the host hardware at least as development platforms. And the Ericsson stack is the most versatile on the market. The selection of applications was limited to the following three:

- Lan Access – This application implements the LAN Access profile. It consists of two sides, LAN access point and LAN access terminal. This application shows the usefulness and advantages with wireless technology. However it requires a lot of non-Bluetooth specific programming. The stack would have to be connected to the Windows NT or Windows 98 IP stack somehow.
- Wireless speakers – These two applications would stream audio from a server part to a client part. This application looked as the most fun and useful application. However no profile is specified for streaming audio.
- File Transfer – This application implements the file transfer profile. It allows a client to browse, send and retrieve files on a server. This application shows the strengths of Bluetooth, transmission of files without any wires. The application is also a direct implementation of a profile.

The selection was made with the following arguments:

- The thesis project objective is to implement a complete Bluetooth application. This requires that the application complies with a profile.

- The application should visualize the advantage of having no wires.
- The effort should be focused on the Bluetooth protocols and profiles.

The application that complied best to these demands was the File Transfer application. The decision was therefore to make a Bluetooth Server and Client.

4.4.1 FILE TRANSFER PROFILE

This section will give a short introduction to the profiles used in the file transfer implementation. For more detailed information, see [3]. The upper two profiles, the file transfer profile and the generic object exchange profile are discussed in more detail, while the other profiles used are discussed more briefly, just to give an idea of their purpose.

File Transfer Profile

The file transfer profile defines all the requirements for the protocols and procedures that shall be used by the application providing the file transfer usage model. The most common devices using this profile are PCs, notebooks and PDAs. The scenarios covered by this profile are the following:

- Browse an object stored on a remote device. This can for example be the file system of a PC. Browsing involves viewing objects and navigating the folder hierarchy.
- Transfer objects between two devices. This can for example be copying files and folders from one PC to another.
- Manipulate objects on a remote device. This includes deleting old objects and creating new ones.

The server has to be set into file transfer mode. This mode enables a client to perform file transfer operations with the server. The server should set the device in limited discoverable mode. The file transfer service should also be recorded in the SDDB (Service Discovery Data Base).

The client has to be able to select the server from a list of possible servers, and set up a connection. It should know how to display the server's folder hierarchy, including the files in the folders, and move through the server's folder hierarchy to select the current folder. The current folder is where items are pulled and/or pushed. The client has to be able to perform file transfer operations to and from the server.

Many of the functionalities required by the file transfer profile are supported in profiles or protocols below. For example all the file transfer operations used in the file transfer profile are specified in a profile called Generic Object Exchange Profile, described below.

In Figure 8 the Bluetooth profile structure and the dependencies of the profiles are depicted. A profile is dependent upon another profile if it reuses parts of that profile, by referring to it implicitly or explicitly. Dependency is also illustrated; a profile has dependencies on the profile(s) in which it is contained, directly and indirectly. For example, the Object Push profile is dependent on Generic Object Exchange, Serial Port, and Generic Access profiles.

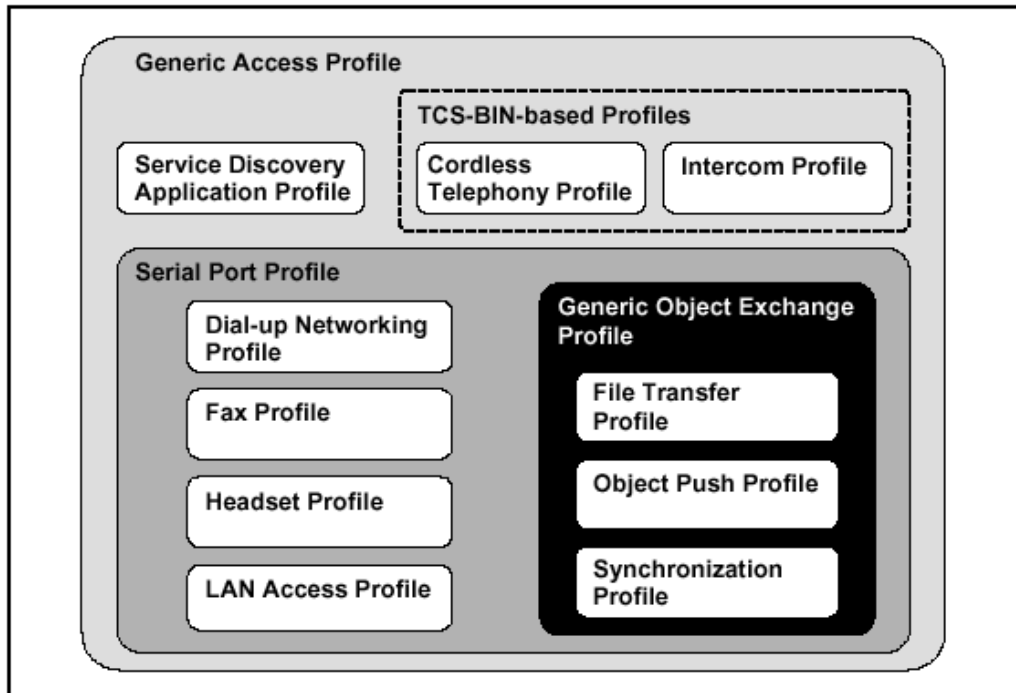


Figure 8. The Bluetooth profile structure and dependencies

Generic Access Profile (GAP)

The purpose of the Generic Access Profile is:

- To introduce definitions, recommendations and common requirements related to modes and access procedures to be used by transport and application profiles.
- To describe how devices are to behave in standby and connecting states in order to guarantee that links and channels always can be established between Bluetooth devices, and that multi-profile operation is possible. Special focus is put on discovery, link establishment and security procedures.
- To state requirements on user interface aspects, mainly coding schemes and names of procedures and parameters, needed to guarantee a satisfactory user experience.

Serial Port Profile (SPP)

The Serial Port Profile defines the protocols and procedures that shall be used by devices using Bluetooth for UART RS232 (or similar) serial cable emulation. The scenario covered by this profile deals with legacy applications using Bluetooth as a cable replacement, through a virtual serial port abstraction (which in itself is operating system-dependent).

Only one connection at a time is dealt with in this profile, thus only point to point configurations are considered. However this shouldn't be a limitation, since multiple executions of this profile should be able to run concurrently in the same device.

The use of security features such as authorization authentication and encryption is optional. But the support for authentication and encryption is mandatory.

Bonding is not explicitly used in this profile. Thus support for bonding is optional. RFCOMM is used to transport the user data, modem control signal and configuration commands. Only connection oriented channels shall be used, thus broadcast will not be used in this profile.

Generic object exchange profile (GOEP)

This profile defines the requirements for Bluetooth devices necessary for the support of the object exchange usage models. The usage model can be, for example, Synchronization, File Transfer, or Object Push. The requirements are expressed by defining the features and procedures required for interoperability between Bluetooth devices in the object exchange usage models. The features that the generic object exchange profile provides for the application profiles are the following, establishing of an object exchange session, pushing data and pulling data.

The GOEP profile is based on the IrOBEX protocol. IrOBEX stands for IrDA object exchange and IrDA stands for infrared data communication. Often OBEX is used for short. One of the most basic and desirable uses of the IrDA infrared communication protocols is simply to send an arbitrary "thing", or data object, from one device to another, and to make it easy for both application developers and users to do so.

OBEX is a compact, efficient, binary protocol that enables a wide range of devices to exchange data in a simple and spontaneous manner. OBEX is being defined by members of the Infrared Data Association to interconnect the full range of devices that support IrDA protocols. It is not, however, limited to use in an IrDA environment. The OBEX protocol can be compared with the HTTP protocol, but it doesn't have all the features and options. A selection of the OBEX commands is presented below, for more information, see [3] and [1].

- Connect - This operation initiates the connection and sets up the basic expectations of each side of the link.
- Disconnect - This command signals the end of an OBEX session.
- Put - The put operation sends one object from the client to the server.
- Get - The Get operation retrieves an object from the server.
- Abort - This operation aborts any ongoing operation.
- Setpath - The setpath operation is used to change the current directory on the remote device.

4.5 DESIGN

An early choice in the thesis work was to use the Ericsson host stack. Together with the Ericsson PC reference stack a few test examples were included. One of the applications was a chat program. The program searches for other devices and presents the result to the user. The user selects the desired device and a connection is created between the two Bluetooth devices. User written text messages can then be exchanged between the server and the client using the created connection. The connection is established according to the two lower profiles, generic access and serial port. The file transfer profile is dependent on both these profiles, which made the chat application a perfect example to study. The Bluetooth chat was broken down and compared to the Bluetooth specification, tracing the procedure to connect and send messages. Many ideas to the basic Bluetooth design came from the test application.

Many of the features required in the application are specified in the file transfer profile. This helped in the design a great deal. A choice was made to only implement the mandatory functionalities described in the profile.

The whole File Transfer application consists of two parts, a server and a client side. A lot of the Bluetooth specific functionality will be the same for the server and the client. The list below describes these common objects.

1. Initilization of the Bluetooth stack and the Module.
2. A security module handling the Bluetooth connections.
3. A Bluetooth communication part which sends and receives messages from the stack.
4. An OBEX module which can decode and generate messages.
5. A module for the profile level authentication

The OBEX and authentication module are really big parts of this application therefore they will be described in their own sections below.

4.5.1 SECURITY APPLICATION

An early design choice is to do the initialization and handle security from the security application. This is a logical choice since the security application is necessary for any other Bluetooth functionality. The Security application should therefore initialize the stack and prepare it for further communications. The security applications are named Server Security and Client security, since some functionality differs.

On startup the Security application loads the stack and establishes a communication path to it. It then sets the device name in the Bluetooth module. The device name can be requested by remote devices as a compliment to the device address. The application registers as security handler with the SCM, which means that it gets all the messages concerning links and connections. The security application handles the discoverability and connectability towards other devices. The security on the link is also handled here. However in our application, neither encryption nor authentication is used at this level.

After the initialization the functionality starts to differ between the server and client security. On the server side it answers to inquiry and allows a remote device to connect and establish an ACL link. The client side remains hidden from all other devices. The Client also allows searching for possible servers and presents this list to the user. The search only looks for devices that support the OBEX file transfer service. The user can then select which server to connect to.

4.5.2 SERVER APPLICATION

The server application registers a file transfer service and waits for incoming connections. The GUI for this can be very simple. Something like an icon in the taskbar just to show that the server is running and perhaps some status info. A client needs to connect to the server on two levels RFCOMM and OBEX. The RFCOMM commands, which are quite few, are understood directly by the application. The OBEX commands are decoded and generated using the OBEX module, described below. When a client connects the server needs to authenticate the user and permit the user to see the folders allowed. For the authentication the authentication module, called OurMD5, is used. The authentication module is described in a Section 4.5.5 below. For the management of users and file permissions a GUI is needed. The solution to this is a standalone application called the sharing application described further down. The class diagram for the server can be seen in Figure 9.

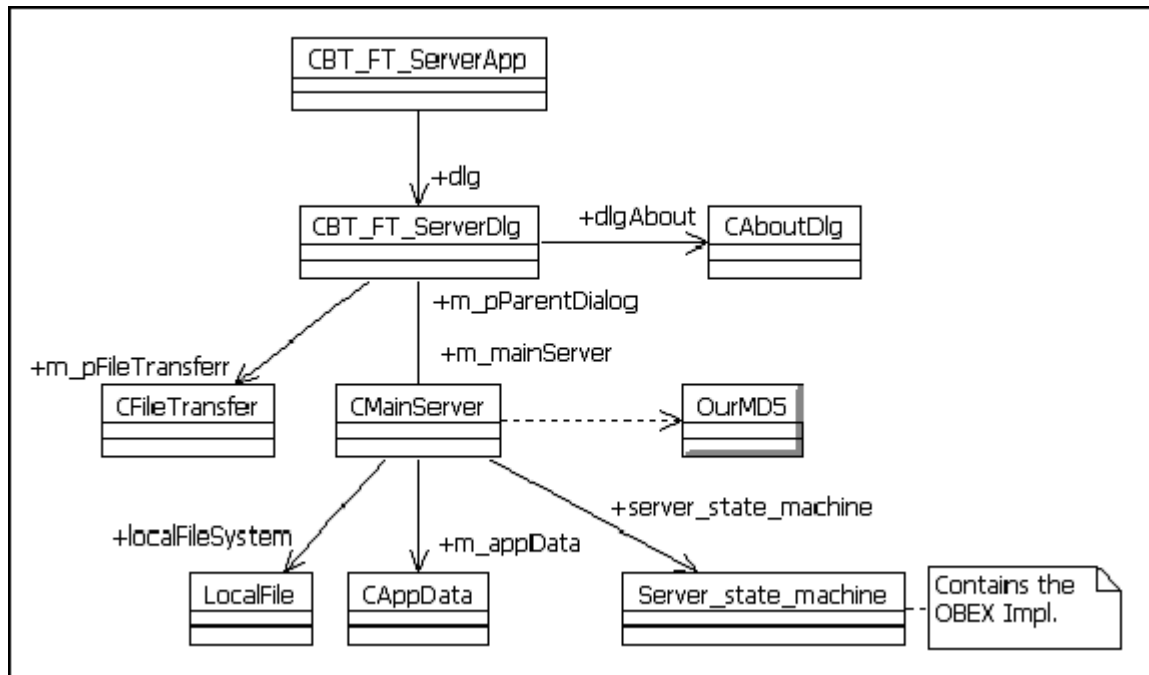


Figure 9. Server application class diagram

The main server dialog has a class `CMainServer` which handles the Bluetooth communication. `LocalFile` is used to read, write and list folders from the local file structure. `CAppData` is the structure holding the configured settings for file transfer users and permissions. The `Server_state_machine` handles the generation and decoding of the OBEX commands.

The sharing application is only needed on the server side. It allows the user to manage the file transfer users allowed to logon to the server. Using the sharing application the visible folders for each user and the read/write permissions can be configured. These settings are then saved to a configuration file which is used by the server. The class diagram can be seen in Figure 10.

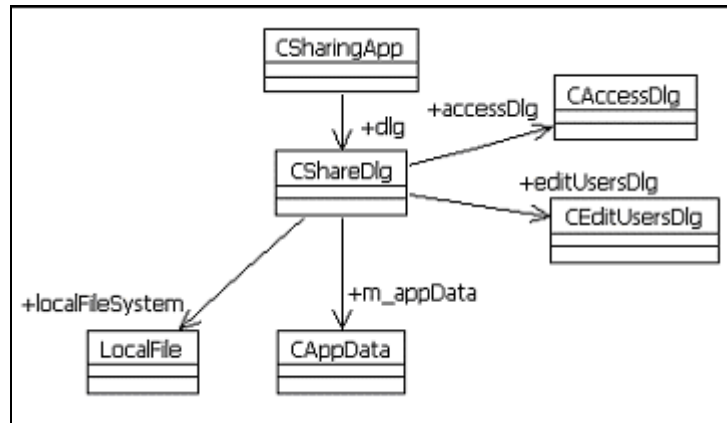


Figure 10. Class diagram for sharing application

The sharing application consists of three dialogs. The main dialog presents a view of the local file system, handled by LocalFile. There is also a view of the folders shared. The dialog CAccessDlg is used to edit which users should have writing or reading permissions to the folders. CEditUserDlg is a dialog used to edit the user database. The sharing information is stored in the CAppData object, which is stored to a file.

4.5.3 CLIENT APPLICATION

In the client application much more user interaction is needed. The idea to the GUI came from a regular internet file transfer protocol program, an FTP program. An FTP program provides the user with the options connect, disconnect, remove, put, get, list, and abort. These commands are very similar to the ones specified by the file transfer profile. No connect command is needed since the link level connection is handled by the client security application. The GUI of the client is very familiar to a person who has used a common FTP application, but it is also made as intuitive as possible for new users.

When the client is started it retrieves the handle to the transport layer from the client security application. It then initiates the OBEX connection. Once the connection is established the basic idea is to show both the local and the remote file system at all times. The user can then select one or more files and send or retrieve them from the server. There should also be a window that shows the user the status of the ongoing actions.

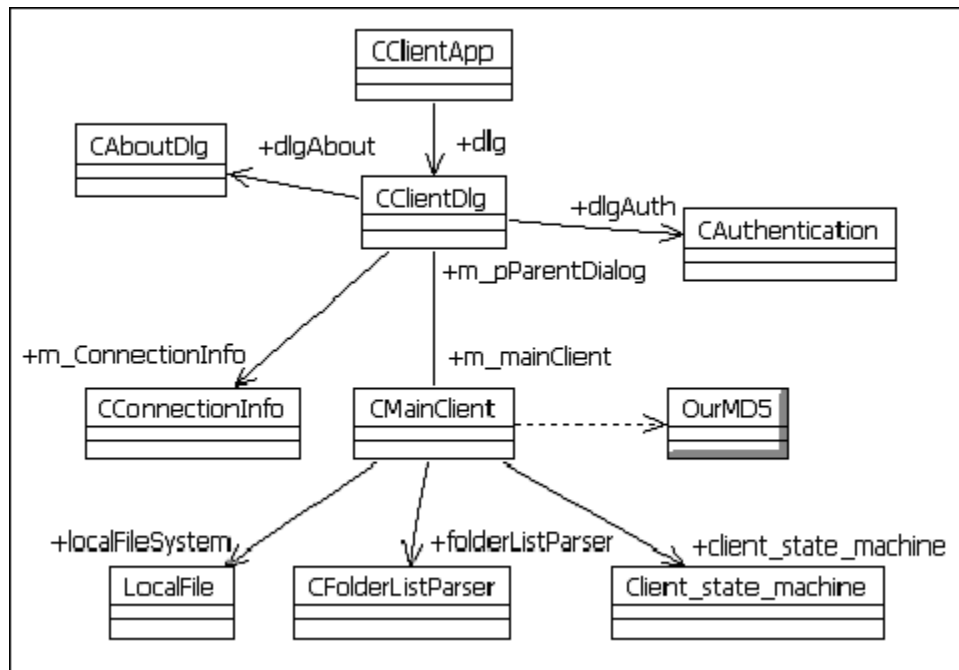


Figure 11. Class diagram for Client application

The client design is similar to the design of the server, but the functionality differs. The class diagram is shown in Figure 11. The CClientDlg has a lot of user interaction. It is the main GUI for this application. When the CClientDlg starts it creates a RFCOMM connection to the remote device and stores that information in the CConnectionInfo object. When the connection is established the user is queried whether OBEX authentication should be used or not. CAuthentication is a dialog doing that. If authentication is activated the authentication module, called OurMD5, is used. It then uses CMainClient to create an OBEX connection. CMainClient has one instance of Client_state_machine, which parses incoming OBEX packets and puts together outgoing ones. The Client_state_machine uses the OBEX module, described below, to generate and decode OBEX packets. LocalFile does all the filesystem operations, when the user pulls or pushes a file. CFolderListParser is a simple XML parser, which translates folder listing objects into a list in the GUI.

4.5.4 OBEX MODULE

OBEX is the module that speaks the language used when transferring files or browsing folders. OBEX has to keep track of all ongoing actions. This means that a packet generator/decoder and a state machine are needed.

When designing the OBEX module two alternatives are possible. OBEX can be written as a layer or as an independent module. Both options were considered in the design process.

OBEX layer

If implementing OBEX as a layer all Bluetooth traffic would go through it. A new API would have to be specified for all Bluetooth events and commands. The OBEX layer would also have to be

a separate process. One serious drawback of the layer architecture is that OBEX would be very dependent of the stack.

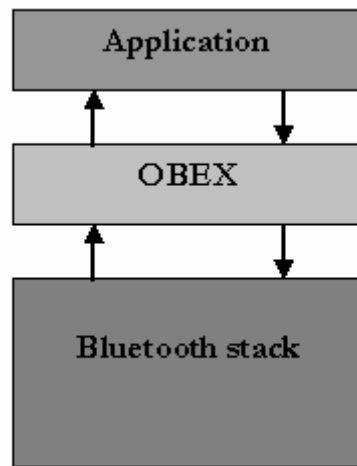


Figure 12. OBEX layer design

OBEX Generator/Decoder

If OBEX is implemented as a module used only to generate and decode messages it would be more independent of the stack. If file transfer or any other profile using OBEX was to be implemented on any other stack this module could be easily reused. When using this design all Bluetooth traffic not concerning OBEX is handled directly in the application. The OBEX module design makes it easy to add support for several simultaneous connections. Just use an instance of OBEX for each connection. It was decided that this design alternative was the best.

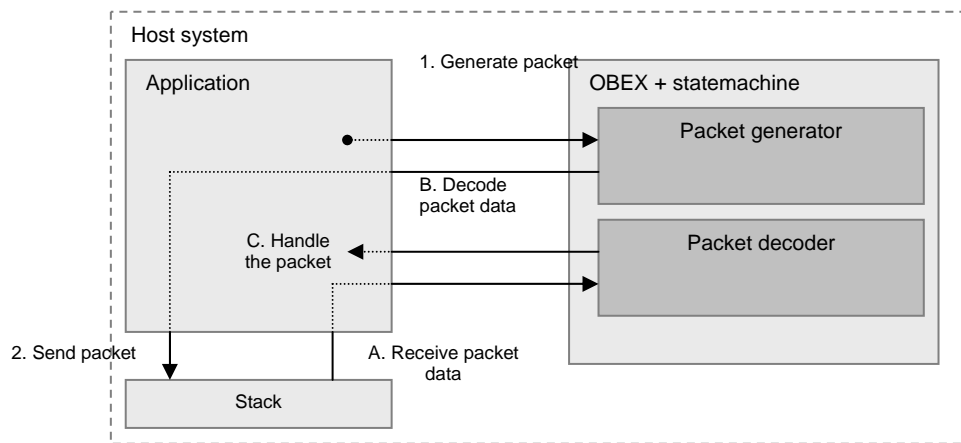


Figure 13. OBEX generator/decoder architecture

OBEX is called by the Main objects to handle and parse the incoming messages and to create the outgoing messages. Both on the client side and on the server side, OBEX consists of a state machine that parses the incoming messages. Depending on the current state that the state machine is in, it only allows some operations. The return value from the state machine tells Main what kind of message it was, if the message was fully understood. Main can then decide the next action and the next message that should be sent.

To ensure that OBEX is system independent it has nothing to do with reading and writing from and to files. OBEX just indicates where to read in an incoming message and where to put the content of a file in an outgoing message. The same procedure is used when assembling folder listings.

4.5.5 AUTHENTICATION

The file transfer profile uses optional authentication on a profile level. Since no link security is used this higher level authentication is chosen instead. This choice is made because higher level authentication is used in the profile which is more in the scope of this thesis work.

The authentication uses the fact that both sides must know the same password. The client uses the authentication module to calculate a checksum of the password and a nonce. It then sends the nonce used and the checksum to the server side. The server then does the same calculation and compares the checksums to see if the client password was correct. This way the password is never sent over the air. The nonce is a random 16 byte string and should never be used more than once. To calculate the checksum a MD5 algorithm is used.

4.6 IMPLEMENTATION

The whole implementation of the applications was divided into different steps. Each step followed a circular process where the application got better and better. Features and functionality were added to the application during the implementation phase. The applications are written using C++ and the development environment is Microsoft Visual Studio. During the testing and debugging phase mistakes were rooted out. This continued until the application was stable and the next step in the implementation process could be taken. During the debugging and testing tracing tools are used. It is possible to trace the communication both at the UART interface and in the air. The implementation process is described in Figure 14.

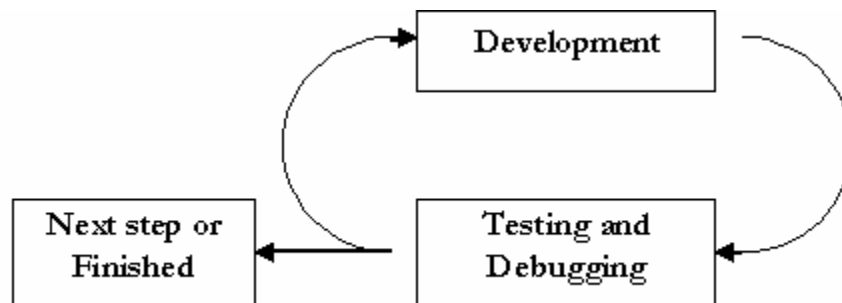


Figure 14, Implementation process

The steps come from the working procedure, see Chapter 4.1. In the actual implementation phase these three steps were broken down to even smaller parts.

4.6.1 SERVER SECURITY

When the Server security application starts it initializes the stack. It registers as a user of the desired protocols and starts receiving events from the stack. It then configures the interface to UART and selects settings.

The Server security application will handle creation of connections and the server's availability to other devices. It is a single process message handler. The Server security has a simple user interface which presents some information about the incoming messages from the stack in a dialog window. It should also start the server application either automatically or by user interaction.

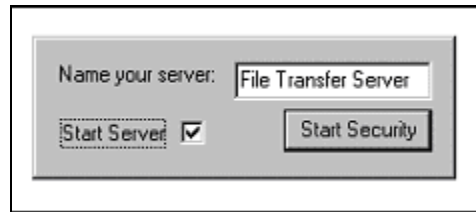


Figure 15. Security server Dialog 1

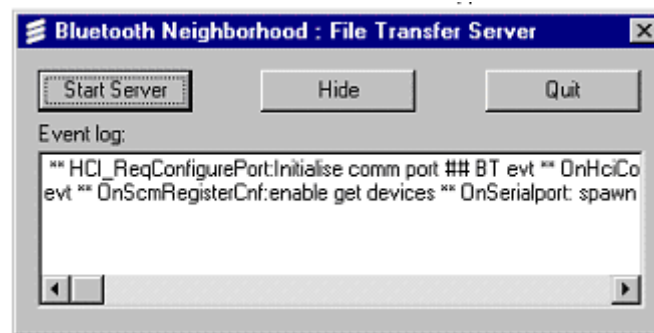


Figure 16. Server security, main dialog

This application consists of two dialogs. When the application is started the window in Figure 15 pops up. In this window you can specify a name for your server. This name will be set as device name in the Bluetooth module. It can be requested by remote devices as a compliment to the device address. In Figure 16 the main window of the application is shown. In this window all Bluetooth events will be logged and the server can be started. Since no interaction is needed the window can be minimized to the system tray with the Hide button. This application registers as security handler with the SCM, which means that it gets all the messages concerning links and connections. The security application handles the discoverability and connectability towards other devices. The security on the link is also handled here. However in our application, neither encryption nor authentication is used at this level. It answers to inquiry and allows a remote device to connect and establish an ACL link.

This application has nothing to do with the security in the meaning of which folders a client can access, which files he can read or write in or which password a user has. This application just starts a server with an OBEX file transfer service that a client can access. The parts that control the accessing of folders are the server and sharing applications.

4.6.2 CLIENT SECURITY

The Client security application initializes the communication with the stack on start-up. It then directly starts using the Bluetooth functionalities. The Client security application presents possible file transfer servers to the user to connect to. It takes care of the device search. It finds all devices within range and displays them to the user. Then it checks whether the selected device is a file transfer server, if so the user can connect to it. The graphical user interface is shown in Figure 17.

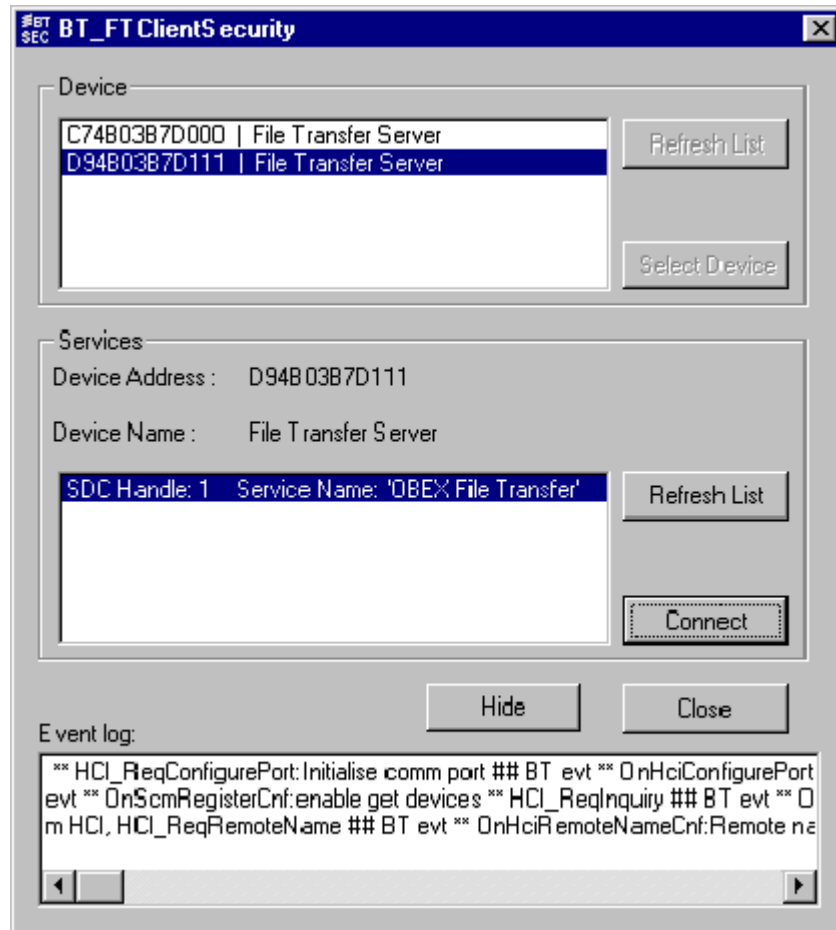


Figure 17. Client Security dialog

When initializing the stack the security application registers to receive events from the stack. It then configures the interface to UART and selects settings. When the initialization is done an inquiry is performed and all devices within range answer with their device address. It then requests the names from all discovered devices. This information is presented in the upper list, which will contain both the Bluetooth address and a device name. If the user wants to update the device list, for example if a new device enters the area or if an old device leaves the area, the refresh button is used. To see the services offered by a device, simply highlight the device in the device list and press the Select Device button. The selection initiates a service discovery procedure. The services that the device offers will be displayed in the service field. Note that only services of OBEX file transfer type will be listed. Also in the service field there exists a Refresh button, used to update the list of services on the currently selected device. If the connect button is pressed, an ACL connection will be established with the remote device and the client application is started.

Using the Server Security application and the Client security application the stack is initialized and communication with the Bluetooth module can be started. The security applications also allow communication between different Bluetooth devices. The client can request the name and which services another device supports. The security applications have established a link layer connection. Using this connection higher layer communication can be achieved.

4.6.3 SERVER

The Server is started by the Server Security application when an incoming connection is established. The link layer connection parameters are passed as arguments by the security application. The Server registers to the stack, and thereby starts a subscription for Bluetooth messages. It then waits for an incoming RFCOMM connection. When the connection is established the mainServer is initialized.

When the mainServer is started the application data, containing sharing settings and user permissions, are read from the file assembled by the Sharing application. An incoming message from the stack arrives at the message handler in the dialog. If it is an OBEX message it is passed to the mainServer. The message is passed on to the OBEX module, where it is interpreted and a code is returned to mainServer. The OBEX module is described in Section 4.6.6. The mainServer checks that the necessary conditions, such as writing and reading permissions, are correct and carries out the command. If the command involves any file or folder manipulation it is sent to LocalFile. The response message is put together by OBEX module and sent to the client by mainServer.

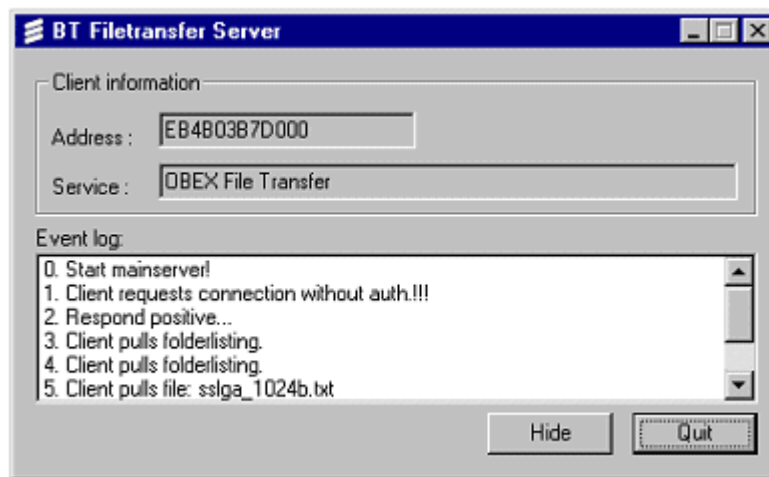


Figure 18. Server Dialog

Figure 18 shows the GUI of the server. There is one field in the window that logs the actions performed by the client. In the other field information about the user is presented. The information is the Bluetooth address and the service that is used. The Hide button allows the user to minimize the window to the system tray.

4.6.4 SHARING

The sharing application is a stand alone program. Figure 19 and Figure 20 show the dialogs of the application. Figure 19 shows the main window. This window is used to control which folders to represent in the root folder of the server. With the sharing application the server side can add new and remove old users. This is done in the edit user window shown to the right in Figure 20. The access to read and write is edited in the Access permissions window, left picture in Figure 20. There is one user called guest, who doesn't require a password. Therefore no authentication is done for a guest user. If a user name and a password are used by the client, the authentication procedure is used.

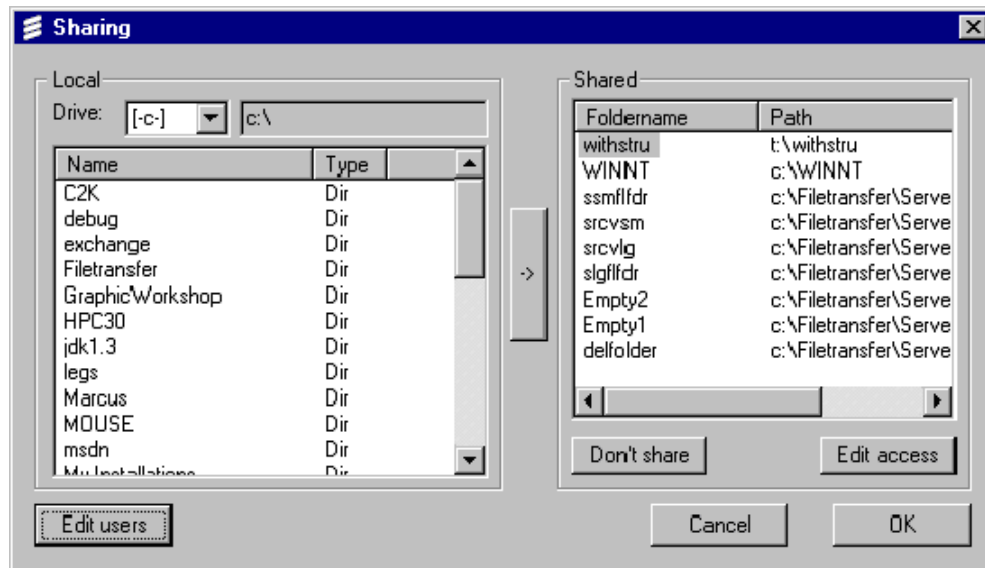


Figure 19. Sharing main dialog

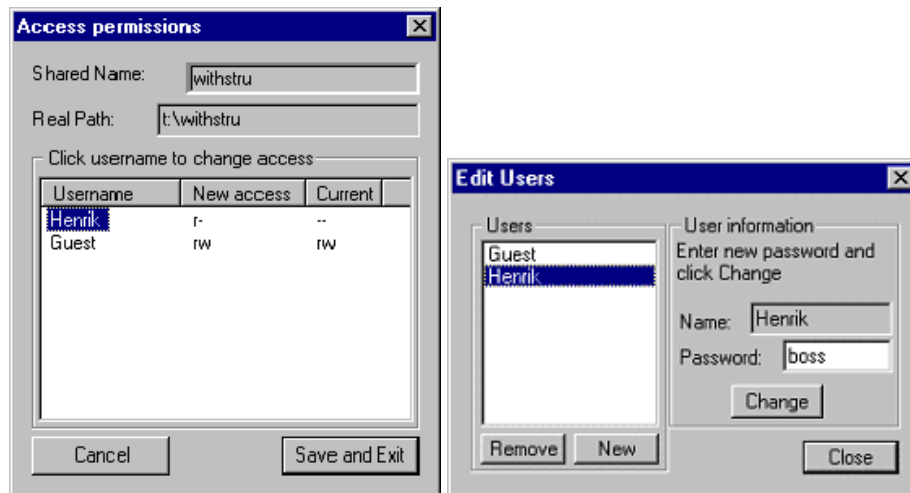


Figure 20. Sharing permission dialog and user dialog

The information written in the sharing application is stored in a file. The file is used by the server in the OBEX connection procedure to authenticate a user. Both the username, password and the permissions is visible in clear text in the file.

4.6.5 CLIENT

The Client application is started by the Client security application and the ACL link parameters are received as arguments. It then immediately tries to connect to the server, first on RFCOMM level and then on OBEX level using the existing link. To be able to make the OBEX connection it needs to know if the user wants to be anonymous or use a specific username and password. MainClient consists of two state machines and a method to send messages. One of the state machines controls the action performed by the user in the GUI. An action to put a file will cause MainClient to call a method in OBEX that creates a put message, and LocalFile is used to read from the file. When the message is created, MainClient will send it to the server. An incoming message will be passed to OBEX via MainClient. The second state machine will interpret the return value from OBEX and take the right measure. The main difference between mainClient and the mainServer is that mainClient also gets user interaction events, since all communication with the server is initialized by the user on the client side.

When the user changes folder on the remote device, a folder listing object is returned. This contains a description of the content in the new folder. The FolderListParser interprets the content of the folder listing object. Immediately when a tag with a file or a folder is found the content is presented to the user in the GUI. The folder listing objects are expressed as an object of the Extensible Markup Language (XML).

The LocalFile class is a part of the Sharing, Client and Server applications. It handles the reading and writing from and to files, the creation of folder listing objects and it also keeps the position of the current folder in mind. The current folder is an indicator to keep track of where in the folder structure the application is. The necessary information, for example the name of the file to open, the size that can be put in a message and where to put it, is announced by the class using it.

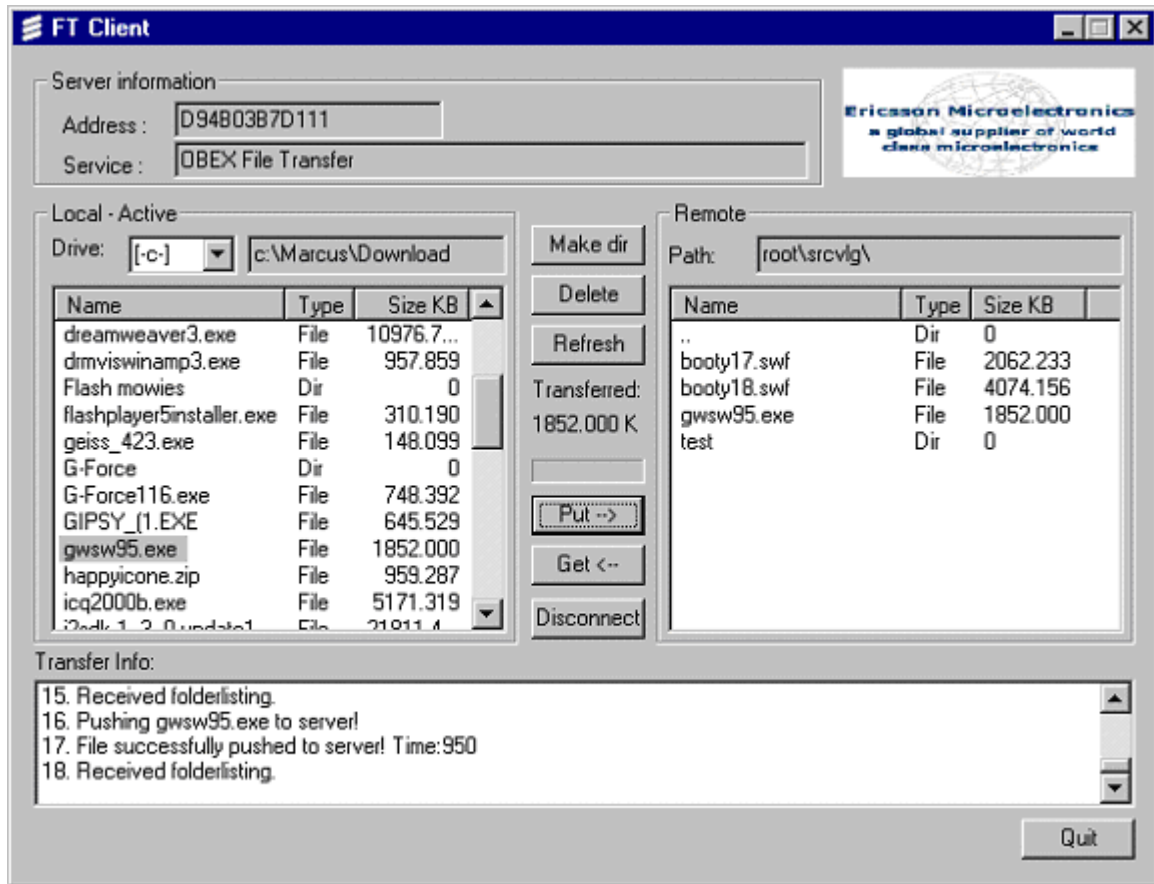



Figure 21. Client dialog

The graphical user interface allows the client to perform different actions on the server and it presents the results of the user actions. Figure 21 shows how the GUI looks. The GUI of the client is where all the user interaction will take place. The upper left corner contains the server information. The address is the Bluetooth device address and the service field contains the service that the server offers.

The information about the local device consists of three fields. The fields inform the user about the current drive, where in the folder structure the user is and the content of the current folder.

The information about the remote device contains information of where in the folder structure the current folder is and the contents of the current folder.

Between the information lists about the devices some buttons indicate the different actions the user can perform. The Make dir button creates a folder on the selected device. The new folder will be located as a child folder of the current folder. The Delete button will delete the highlighted file. Our version does not delete folders. Before the selected file will be deleted a security measure has been taken. Before the delete action takes place, the user has to confirm the deletion in a message box. This is a common security measure. The Refresh button is used to update the folder structure both on the client and on the server side. If changes are made in the folder structure of the client or if the server decides to share more folders, the user has to update the window by pressing the Refresh button. When a file is transferred the progress bar indicates the proportion of the file that has been transferred. The Put → button will send the highlighted file from the client to the current folder of

the server, and the Get  button will send the highlighted file on the server to the current directory of the client. When an action is performed, for example a get request, all buttons will be disabled and an Abort button will be enabled. The Abort button will terminate the current action and set the client to wait for a user action and the server to wait for an incoming message. The kind of message doesn't matter. The disconnect message will end the current OBEX session and a user can logon with a different username and get access to other folders. The disconnect action should always be available.

4.6.6 OBEX MODULE

The OBEX module is a direct translation of the mandatory features in the Bluetooth specification. It is implemented as a module which generates and decodes OBEX messages. All actions are function driven and the results are returned as pointers. Everything is executed in the function caller context.

Before any messages can be sent or received the OBEX module needs to be initialized. The maximum OBEX packet size must be specified since it depends on the maximum transmission unit of the transport layer. The initialization also starts the state machine. The state machine keeps track of all the ongoing command sequences. It is the state machine that checks which commands are allowed both for incoming and outgoing traffic. Since it had to check for erroneous communication initiated both by the local and remote sides it became quite complex.

Generating messages

When generating OBEX packets functions available in the OBEX API are used. The OBEX module provides a function named **Create_xxxx(...)** for every OBEX message that can be sent. For example `Create_put_message()`. The OBEX module then generates a complete OBEX message. The length of the constructed message is given as a return parameter. Since the OBEX module is completely independent of the operating system and transport layers some things must be done by the application. In these cases the create-functions construct all the headers, length fields and make space for the payload. Then a public pointer is set to indicate where the application should copy the data to. This procedure is used for packets containing file data or folder listing objects. The memory buffer used is allocated by the application; the OBEX module will never allocate memory. This way the application decides on the number of buffers that should be used and has full control of the memory consumption.

Decoding messages

When decoding packets the received message is passed to a decode function. This function returns a specific message code for every type of OBEX packet that can be received. The application can then take the correct step. If data is received a public pointer is used to pass it to the application. The application must make sure that the data is handled before the message is freed.

Debugging

During the development and debugging of the OBEX module the air trace instrument is used. The instrument is from the company CATC and is called Merlin. Merlin is basically a Bluetooth radio that listens to ongoing communication between two Bluetooth devices. The Merlin software can then

decode the different packages and show the payload. This helped a lot during the debugging phases, when rooting out packet errors. This showed that the OBEX packets were correct between the Server and the Client. Otherwise packet errors made on one side could have been corrected on the other side. The Merlin tester was also used for interoperability tests of the implemented OBEX protocol.

5 DISCUSSION

This thesis work is a learning process from the beginning to the end. Large quantities of information and specifications had to be comprehended to achieve the goals of the project. This information is not only Bluetooth specific. A lot of information regarding communication, stack architectures and programming is also acquired.

5.1 WORKING PROCEDURE

The work was mainly conducted according to the working procedure planned in the beginning of the thesis work. Some small deviations were made due to the problems with the hardware and the hoststack. If the hardware problems had been known earlier more time would have been spent on the USB transport layer in an earlier phase of the project.

We are convinced that the working procedure to achieve a fully functioning Bluetooth application was the right way to go. Through it we had the opportunity to handle small parts of the difficulties in a sequential process that led to the final system.

5.2 HARDWARE

It is always good to start a development on PC platforms. A PC has plenty of resources and tools to get the development started. However one big problem was discovered when the application started to send data over the UART. The problem occurred at UART speeds as low as 57600 bits per second and it got worse at higher speed. The UART started losing bytes of data when sending files. This corrupted the communication at different levels depending on which bytes were lost. The problem occurred frequently on both desktops and laptops, but it was much worse on the laptops. It proved to be a known problem. The UART flow control on the PC's weren't fast enough. When the module signaled STOP the PC didn't stop fast enough. The problem is discussed further in Chapter 5.3.

A solution to this problem would have been to implement the application using the USB interface. However the first hoststack didn't have the USB transport layer implemented. And when the second stack, with USB support, was released the thesis work was almost done.

5.3 SOFTWARE

Ericsson stopped the development of the PC reference stack at the end of the thesis work. Two versions were released and none of those were stable. Sometimes it hanged the whole host system. The solution would have been to change hoststack vendor or implement a new hoststack based on the generic solution from Ericsson. That would however consume too much time and effort for this thesis work.

A flow control problem in the UART caused a lot of weird problems in the stack, in the application and OBEX. Since the error could occur at any level in the stack or application it was hard to find. A lot of debugging was done trying to find these errors before the cause was found. The real cause of these errors comes from a flaw in the Bluetooth specification. When Bluetooth was designed the only likely cause of data corruption considered was the radio link. Therefore error control, error correction and resending are only done on the radio level. An error that occurs before the packet is sent or after it is received can cause trouble anywhere in the stack or application. It doesn't help to have error correction in the application since errors can still cause the stack

communication to fail. Possible solutions would instead be to have error control and resending over the UART layer or in the lowest layer of the hoststack.

5.4 APPLICATION

Many problems and issues during the development arose during the development of the GUI. These issues are not directly related to the Bluetooth communication. However a lot of experience regarding the interaction needed for Bluetooth communication was gained when solving these issues.

Even though most of the Bluetooth communication is handled by the OBEX module it is still a heavy load on the application developer. The fact that the hoststack functionality is generic and very unlimited makes it more complicated to use. The advantage is that the stack is very flexible and adaptive.

Perhaps the file transfer profile should have been developed as an easy to use module instead. In that case only a small API would be presented to the application. This would limit the amount of Bluetooth communication that has to be handled in the application. It would also make the file transfer profile more reusable.

The information that is written in the sharing application is stored in a file. The file that contains the user names and the passwords is not encoded, because this wasn't a main purpose of the thesis. The user has to keep malicious people away from his computer.

When investigating potential problems and testing the applications the air Bluetooth analyzer Merlin is used. By learning how to use this instrument and utilize it in the development process a lot of issues are discovered and solved. It also gives a hint of how well the application will interoperate with other applications.

5.4.1 AMBIGUITIES

During the implementation it became obvious why the qualification process for Bluetooth products is needed. It wasn't always that easy to understand what the specification stated. During the implementation of the profiles some ambiguities came up.

The first ambiguity is in the IrOBEX protocol. It involves the creation of a new folder. According to IrOBEX a new object is created with a put request with an empty end of body header. However in the section that involves the folder browsing service, a new folder is created with a setpath request. The file transfer file follows the last procedure. It states that a new object, in this case a folder, is created with a setpath request with the name of the folder in the name header. The get request with an empty name header would be better to use because it would be found faster. The whole packet doesn't have to be scanned to find out if it exists.

According to IrOBEX sending a get command with an empty name header is used to pull the contents of the current folder. The file transfer profile on the other hand states that a get command without a name header is sent to retrieve the contents of the current folder.

IrOBEX has a feature that is stated more consistent in The Bluetooth profile specification. This feature is to retrieve the content of a child folder. According to IrOBEX you should send a get request with the name of the child folder in a name header and a type header that specifies the folder object type. The file transfer profile tells you to first send a setpath request to set the current folder to the root folder. Then send a get request to retrieve the content of the current folder. The way that it's stated in GOEP is more consistent and similar to the other operations.

5.5 THE FUTURE

A natural future development for this project is to change the transport layer used. Some companies have already implemented a UART protocol that uses error control and resendings. However Ericsson has no such plans⁶. USB would be a better alternative for this application. That would increase speed and hopefully stability.

The next step would be to enable multi point communications. As it is now, only one client can connect at the time. The possible number of use cases increases if the application could handle several connections. The Bluetooth module and stack can already handle this. It would only require some small changes in the Server security part and that a server was started for each connection.

⁶ Atleast no such plans existed 2001.

6 ABBREVIATIONS AND DEFINITIONS

Abbreviation	Description
ACL	Asynchronous Connection-less Link
API	Application Programmers Interface An interface presented to the application programmer.
ASCII	American Standard Code for Information Interchange
Bluetooth SIG	Bluetooth Special Interest Group
BPS	Bits Per Second
BT	Bluetooth
CID	Channel Identifier
EBSK	Ericsson Bluetooth Starter Kit
ETSI	European Telecommunications Standards Institute
FTP	File Transfer Protocol
GAP	Generic Access Profile
GOEP	Generic Object Exchange Profile
GP I/O	General Purpose Input / Output
GUI	Graphical User Interface
HCI	Host Controller Interface. <i>Bluetooth</i> protocol layer that provides a command interface with the baseband controller
IrDA	Infrared Data Association
L2CAP	Logical Link Control and Adaptation Protocol <i>Bluetooth</i> protocol layer that provides protocol multiplexing and packet segmentation and reassembly
LMP	Link Manager Protocol
MTU	Memory Transfer Unit
OBEX	Object Exchange
PC	Personal Computer
PDA	Personal Digital Assistant
PP	Point-to-Point Protocol
RF	Radio Frequency
RFCOMM	Serial Cable Emulation Protocol <i>Bluetooth</i> protocol layer that provides serial port emulation and Multiplexing
ROM	Read Only Memory
SCM	Security Connection Manager
SCO	Synchronous Connection-Oriented Link
SDP	Service Discovery Protocol <i>Bluetooth</i> protocol layer that provides access to descriptions and attributes of <i>Bluetooth</i> services
SIG	Special Interest Group
SPP	Serial Port Profile
UART	Universal Asynchronous Receiver Transmitter The electronic circuit that makes up the serial port.
USB	Universal Serial Bus
XML	Extensible Markup Language

7 REFERENCES

- [1] IrDA Object Exchange Protocol V1.2 (1999-03). Infrared Data Association. www.irda.org.
- [2] Specification of the Bluetooth System, Core, V1.0 B (1999-12). Special Interest Group. www.bluetooth.org.
- [3] Specification of the Bluetooth System, Profiles, V1.0 B (1999-12). Special Interest Group. www.bluetooth.org.
- [4] Users Manual – Bluetooth PC Reference Stack by Ericsson, version R1a (2000-04). Ericsson Bluetooth Starter Kit.
- [5] Bluetooth Architecture Overview, 2000-06-26. Ericsson Bluetooth Starter Kit Documentation.
- [6] Ericsson Bluetooth Starter Kit Introductory Slides, 1999-09. Ericsson Bluetooth Starter Kit Documentation.
- [7] General Bluetooth information and links. www.bluetooth.org, 2000-2001.
- [8] Brian W. Kernighan, Dennis M. Ritchie, The C Programming Language, Second Edition, 1988.
- [9] Jon Bates, Tom Tompkins, Using Visual C++ 6, 1998.
- [10] Larry L. Peterson, Bruce S. Davie, Computer Networks, 1996

Microsoft Visual Studio 6.0

Sys Internals PortMon

CATC Merlin Bluetooth Tracer With Software

Ericsson Microelectronics PC reference stack

Ericsson Microelectronics Ericsson Bluetooth Starter Kit, EBSK

Microsoft Office

Jasc Paintshop Pro 7

APPENDIX

OBEX IMPLEMENTATION

This section will describe the packet exchange between the client and the server. The packets exchanged are the packets that we have designed and implemented. This section gives a deeper understanding of the implementation. Examples are given for all packets used. All values are hexadecimal values. The only exception is the content of a few headers such as name, user-id and the body header. The size of the field and header are depicted above each packet. Below each packet an explanation is given.

CONNECT

This section will bring up all the packets that use the opcode for connect. The first example shows a connect request without authentication. Figure 1 shows the packet flow. The request and response packets are shown in Table 1 and Table 2.

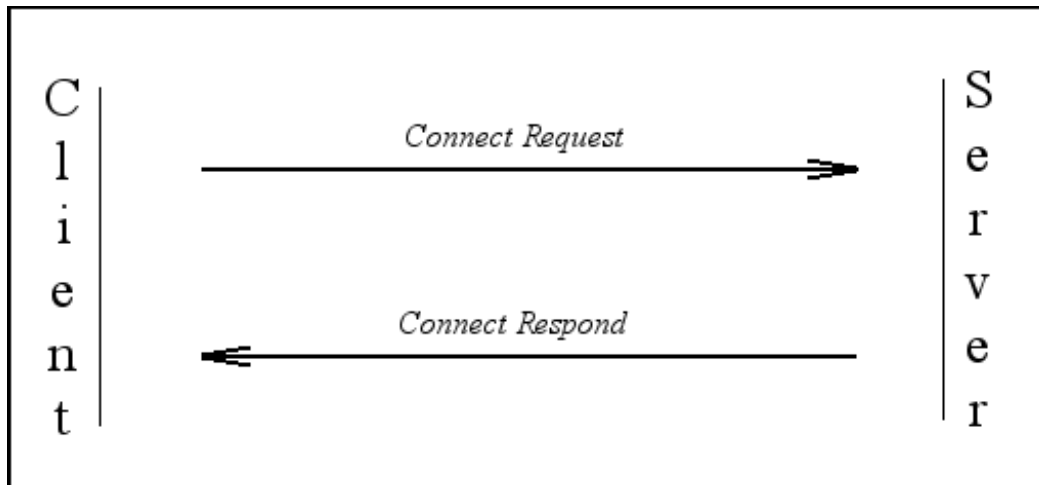


Figure 1. Establishment of OBEX session without authentication

Byte 0	Byte 1-2	Byte 3	Byte 4	Byte 5-6	Byte 7	Byte 8-9	Byte 10-25
0x80	0x001A	0x10	0x10	0x007F	0x46	0x0013	0xF9EC7BC4953C11D2984E525400DC9E09

Table 1. Connect request packet

Byte 0	Byte 1-2	Byte 3	Byte 4	Byte 5-6	Byte 7	Byte 8	Byte 9-12	Byte 13	Byte 14-15
0xA0	0x001F	0x10	0x10	0x007F	0xCB	0x01	0x00000001	0x4A	0x0013

Byte 16-31
0xF9EC7BC4953C11D2984E525400DC9E09

Table 2. Connect response packet

This example shows the procedure to establish an OBEX session and make use of the authenticate challenge option. The first request can either be the packet described in Table 1, or if the client initiates the authenticate challenge, the first packet will be the one described by Table 3. Independently of the connect request the response will look like the packet depicted in Table 4. The server includes the authenticate challenge header and the response code for unauthorized in the

response. The second connect request (Table 5) includes the authenticate response header, as response for the authenticate challenge header in the request. It also includes an authenticate challenge header to authenticate the server. The second and last response (Table 6) includes the authenticate response header, the response code is success if the server accepted the request.

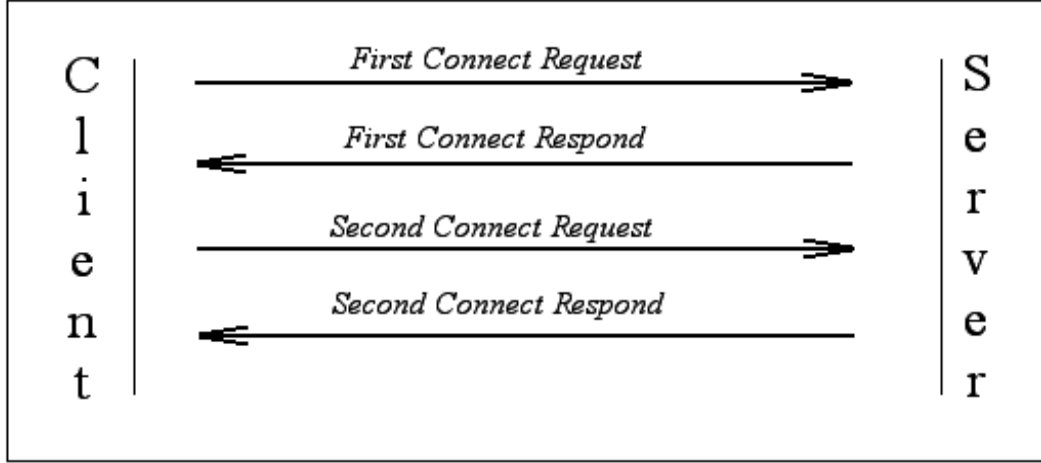


Figure 2. Connect with authenticate challenge

Byte 0	Byte 1-2	Byte 3	Byte 4	Byte 5-6	Byte 7	Byte 8	Byte 9-24
0x80	0x006C	0x10	0x10	0x007F	0x46	0x0013	0xF9EC7BC4953C11D2984E525400DC9E09

Byte 25	Byte 26-27	Byte 28	Byte 29	Byte 30-46	Byte 47	Byte 48	Byte 49
0x4d	0x0018	0x00	0x10	-----	0x02	0x01	0x00

Table 3. Connect packet with authenticate challenge

Byte 0	Byte 1-2	Byte 3	Byte 4	Byte 5-6	Byte 7	Byte 8-9	Byte 10	Byte 11	Byte 12-27	Byte 28	Byte 29
0xC1	0x0021	0x10	0x10	0x007F	0x4d	0x001A	0x00	0x10	0xdigest string	0x00	0x01

Byte 30	Byte 31	Byte 32
0x02	0x01	0x00

Table 4. First connect response packet

Byte 0	Byte 1-2	Byte 3	Byte 4	Byte 5-6	Byte 7	Byte 8-9	Byte 10-25
0x80	0x0060	0x10	0x10	0x007F	0x46	0x0013	0xF9EC7BC4953C11D2984E525400DC9E09

Byte 26	Byte 27	Byte 28	Byte 29	Byte 30-45	Byte 46	Byte 47	byte 48	byte 49	Byte 50-51	Byte 52	Byte 53
0x4D	0x0018	0x00	0x13	-----	0x02	0x01	0x00	0x4e	0x003D	0x00	0x13

Byte 54-69	Byte 70	Byte 71	Byte 72-76	Byte 77	Byte 78	Byte 79-94
-----	0x01	0x05	Erik\0	0x02	0x13	-----

Table 5. Second connect request

Byte 0	Byte 1-2	Byte 3	Byte 4	Byte 5-6	Byte 7	Byte 8-11	Byte 12	Byte 13-14	Byte 15-30
0xA0 0xC1	0x004C	0x10	0x1 0	0x007F	0xCB	0x00000001	0x4A	0x0013	0xF9EC7BC4953C11D2 984E525400DC9E09

Byte 31	Byte 32-33	Byte 34	Byte 35-40	Byte 41	Byte 42	Byte 43-58
0x4E	0x0027	0x00	Digest_rsp	0x02	0x10	Nonce_rec

Table 6. Second connect response packet

SETPATH

The setpath request is used to browse the remote folder structure and to create new folders. Figure 3.13 shows the packet exchange during a setpath request. A setpath request has to fit in one single packet. The tables below describe the different packets that make use of the opcode for setpath. The first is setpath forward, used to set the current folder to the child folder. The request and response are shown in Table 7 and Table 8.

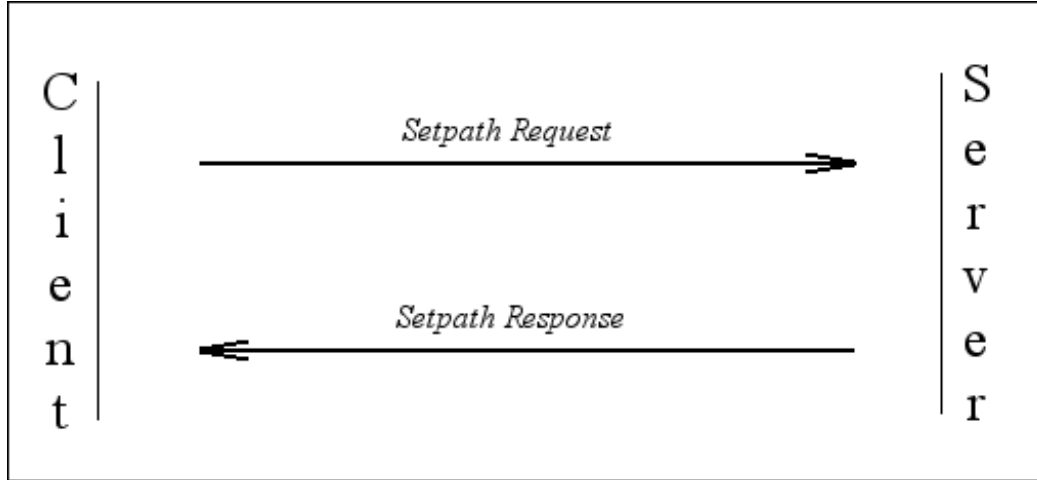


Figure 3. Setpath request

Table 7 shows a request to set the current folder on the server to a child folder with the name Arfwedson, Table 8 shows the response packet. 0xC4 is returned if the folder doesn't exist.

Byte 0	Byte 1-2	Byte 3	Byte 4	Byte 5	Byte 6-9	Byte 10	Byte 11-12	Byte 13-33
0x85	0x0021	0x02	0x00	0xCB	0x00000001	0x01	0x0017	Arfwedson\0

Table 7. Setpath forward

Byte 0	Byte 1-2
0xA0	0x0003
0xC4	

Table 8. Response packet for setpath forward

Table 9 shows a request to set the current folder on the server back to the parent folder. No name header is required. Table 10 shows the response packet. The response code is 0xC4 if the current folder is the root folder.

Byte 0	Byte 1-2	Byte 3	Byte 4	Byte 5	Byte 6-9
0x85	0x000A	0x03	0x00	0xCB	0x00000001

Table 9. Setpath backward

Byte 0	Byte 1-2
0xA0	0x0003
0xC4	

Table 10. Response packet for setpath forward

Table 11 shows a request to set the current folder on the server to the root folder, Table 12 shows the response packet.

Byte 0	Byte 1-2	Byte 3	Byte 4	Byte 5	Byte 6-9	Byte 7	Byte 8
0x85	0x000A	0x02	0x00	0xCB	0x00000001	0x01	0x0003

Table 11. Setpath root

Byte 0	Byte 1-2
0xA0	0x0003

Table 12. Response packet for setpath root.

Table 13 shows a request to create a folder with the name ERICSSON, and the response packet is shown in Table 14.

Byte 0	Byte 1-2	Byte 3	Byte 4	Byte 5	Byte 6-9	Byte 7	Byte 8	Byte 9-30
0x85	0x000A	0x00	0x00	0xCB	0x00000001	0x01	0x0003	ERICSSON\0

Table 13. Create folder

Byte 0	Byte 1-2
0xA0	0x0003

Table 14. Response packet for create folder

GET

The get operation is used to get the content of a file or to get a folder-listing object. The get operation can also be used to get the content of a folder (all files and folders and the containing content). The get operation of a folder is optional and is not supported by this version. Figure 4 illustrates the packet exchanges if the request and the response fit in a single packet. Figure 5 illustrates when the request fits in a single packet, but the response packet doesn't fit in a single packet.

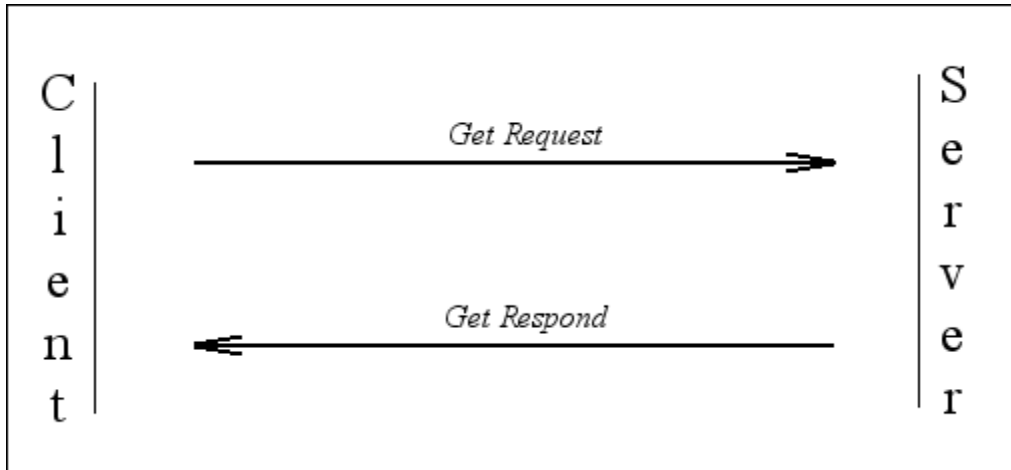


Figure 4. Get request

Table 15 shows the request to get the file Marcus.txt from the server. No type header is used. The response packet is depicted in Table 16. The get request can also concern a folder-listing object. That case is illustrated in Table 17. A type header must be used but a name header isn't sent. The response packet is shown in Table 18.

Byte 0	Byte 1-2	Byte 3	Byte 4-7	Byte 8	Byte 9-10	Byte 11-32
0x83	0x001F	0xCB	0x00000001	0x01	0x0017	Marcus.txt\0

Table 15. A request to get the file Marcus.txt

Byte 0	Byte 1-2	Byte 3	Byte 4-7	Byte 8-108
0xA0	0x006B	0x48	0x0068	H...s.

Table 16. The response message for the request to get the file Marcus.txt, file size is 101 bytes

Byte 0	Byte 1-2	Byte 3	Byte 4-7	Byte 8	Byte 9-10	Byte 11-32
0x83	0x001F	0xCB	0x00000001	0x42	0x0017	Xobex/folderlisting\0

Table 17. A request to get a folder listing of the server's current folder

Byte 0	Byte 1-2	Byte 3	Byte 4-7	Byte 8-108
0xA0	0x0069	0x48	0x0066	<...>

Table 18. The response message for the request to get the folder-listing object of the current folder, the size of the folder-listing object is 99 bytes.

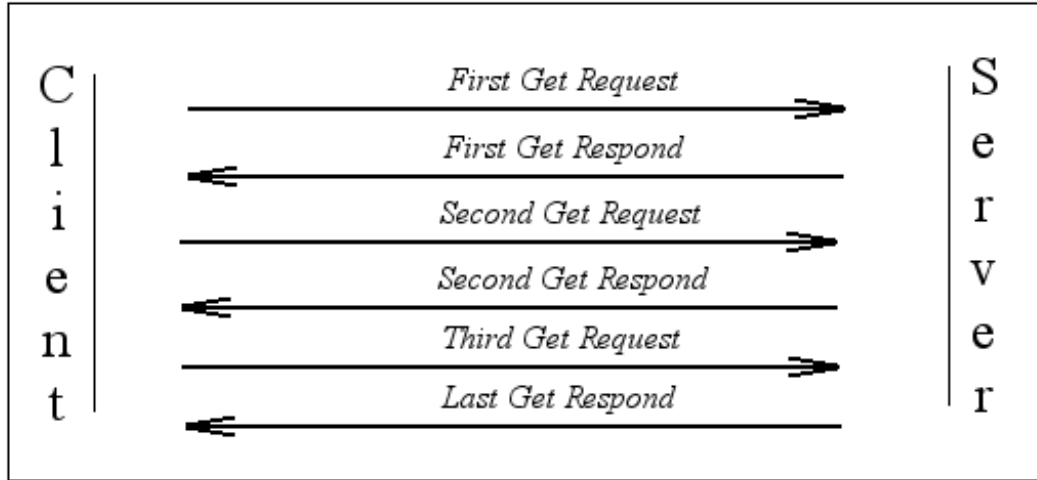


Figure 5. Get request

Suppose that the file Marcus.txt will be transferred. The size of the file is 500 bytes and the maximum OBEX packet length is 255 bytes. The first packet has the same look as the packet shown in Table 15. The response packet, illustrated in Table 19, will now contain the response code for continue, not success. Since the client gets the response code for continue, he knows that the server has more to send. He will ask the server for more. The second request packet is depicted in Table 20. The second response depicted in Table 21 will also transfer 259 bytes. Totally 498 bytes has been transferred now, two bytes left to transfer. The last response packet, Table 23, will transfer those two bytes. This packet will have the final bits set. The third request, illustrated in Table 22, has the same appearance as the second request packet.

Byte 0	Byte 1-2	Byte 3	Byte 4-5	Byte 6-254
0x90	0x00FF	0x48	0x00FC	...

Table 19. First response packet, final bit not set. This response will transfer 249 bytes of the requested file.

Byte 0	Byte 1-2
0x83	0x0003

Table 20. Second request packet, note that this packet contains neither a name header nor a connection Id header

Byte 0	Byte 1-2	Byte 3	Byte 4-5	Byte 6-254
0x90	0x00FF	0x48	0x00FC	...

Table 21. Second response packet, final bit not set, even this packet transfers 249 bytes

Byte 0	Byte 1-2
0x83	0x0003

Table 22. Third request packet, note that this packet contains neither a name header nor a connection Id header

Byte 0	Byte 1-2	Byte 3	Byte 4-5	Byte 6-7
0xA0	0x0008	0x49	0x0005	.

Table 23. Last response packet, final bit set. This last response will transfer the last two bytes

PUT

The put operation is used to put objects to a server. The object can for example be a file. The operation to put folders isn't supported by our version. The put operation is also used to delete files and folders. However deletion of folders isn't supported by our version. Figure 6 shows the packet exchange when both the request and the response fit in one single packet.

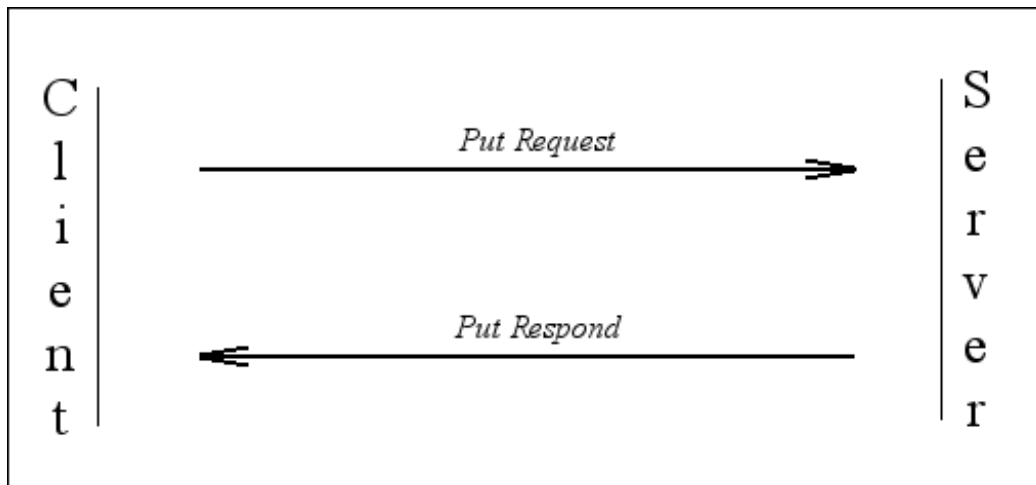


Figure 6. Illustration of the put procedure when the request fits in one packet.

In Table 24 the put packet is illustrated. The client requests the server to put the file JonasBredband.txt to the server. The size of the file is 155 bytes. Table 25 shows the response packet. The request to delete a file is shown in Table 26. In this case the file is Marcus.txt, no body header is used. The response packet for the delete request is shown in Table 27.

Byte 0	Byte 1-2	Byte 3	Byte 4-7	Byte 8	Byte 9-10	Byte 11-36	Byte 37	Byte 38-39	Byte 40-194
0x82	0x002B	0xCB	0x00000001	0x01	0x0024	JonasBredband.txt\0	0x49	0x000B	...

Table 24. A request to put the file JonasBredband.txt, the content of the file is in the body header

Byte 0	Byte 1-2
0xA0	0x0003

Table 25. The successful response packet for a request to put the file JonasBredband.txt

Byte 0	Byte 1-2	Byte 3	Byte 4-7	Byte 8	Byte 9-10	Byte 11-22
0x82	0x00	0xCB	0x00000001	0x01	0x0019	Marcus.txt\0

Table 26. A request to delete the file Marcus.txt

Byte 0	Byte 1-2
0xA0	0x0003

Table 27. The successful response packet for a request to delete the file Marcus.txt

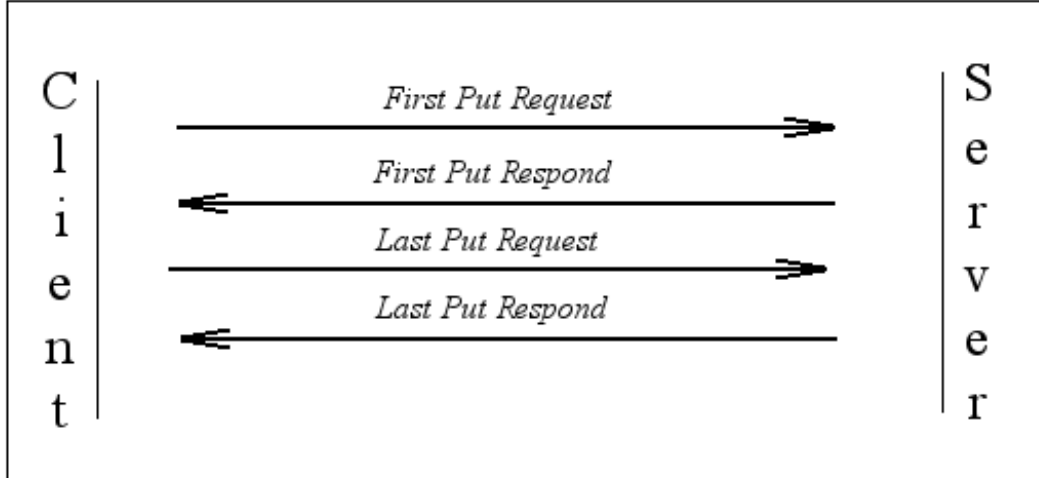


Figure 7. File put of a file that doesn't fit in one OBEX packet.

If a file doesn't fit in one OBEX packet the final bit will not be set in the request. Figure 7 shows an example of this case. The file Meivert.txt should be transferred from the server to the client. The size of the file is 250 bytes. Suppose that the maximum OBEX packet length is 255 bytes. Table 28 shows the first request packet. 215 bytes of the file will be sent in the first packet. The first response will contain the response code for continue, to indicate to the client to send more. The first response packet is shown in Table 29. The second, and last request has the final bits set, Table 30 illustrates this. The last response is illustrated Table 31.

Byte 0	Byte 1-2	Byte 3	Byte 4-7	Byte 8	Byte 9-10	Byte 11-36	Byte 37	Byte 38-39	Byte 40-254
0x02	0x00FF	0xCB	0x00000001	0x01	0x0019	Meivert.txt	0x48	0x00D7	...

Table 28. A request to put the file Meivert.txt, the content of the file is in the body header

Byte 0	Byte 1-2
0x90	0x0003

Table 29. The response packet for the first request

Byte 0	Byte 1-2	Byte 37	Byte 38-39	Byte 40-254
0x82	0x0029	0x49	0x0026	...

Table 30. The second request, the content of the file is placed in the body header

Byte 0	Byte 1-2
0xA0	0x0003

Table 31. The last response packet, final bit set

ABORT

The abort action is used to terminate the current operation, for example a put action. An Abort message sequence is shown in Figure 8. Both the request and the response must each fit into one single packet. Table 32 illustrates the request packet and Table 33 illustrates the response packet.

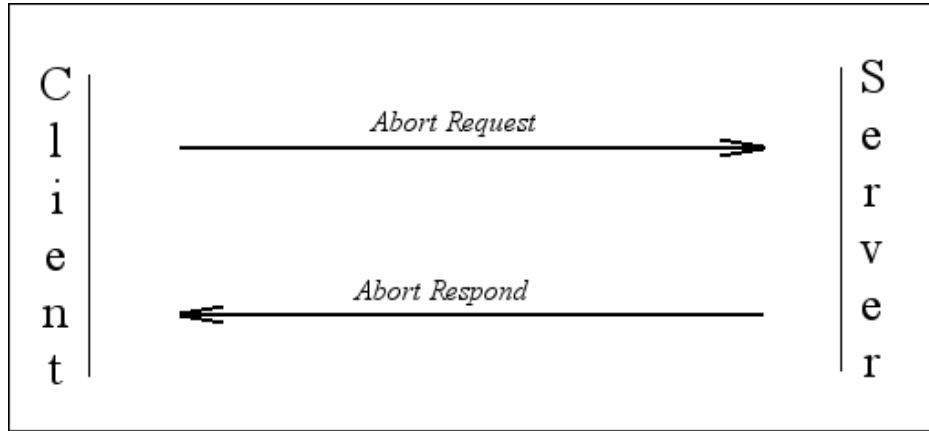


Figure 8. The Abort request

Byte 0	Byte 1-2	Byte 3	Byte 4-7
0xFF	0x0008	0xCB	0x00000001

Table 32. Format of the abort request

Byte 0	Byte 1-2
0xA0	0x0003

Table 33. Format of the abort response

DISCONNECT

The disconnect operation is used to end the current OBEX session. Both the request and the response must fit in one single OBEX packet. The Disconnect Message sequence is shown in and the packets in

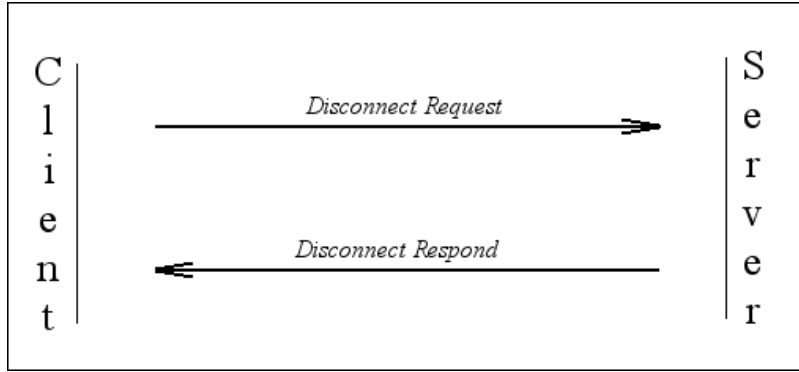


Figure 9. Disconnect request

Byte 0	Byte 1-2	Byte 3	Byte 4-7
0x81	0x0008	0xCB	0x00000001

Table 34. Format of the abort request

Byte 0	Byte 1-2
0xA0	0x0003

Table 35. Format of the abort response