CHALMERS

Nomadic Device Integration For Automotive Applications

Master of science Thesis in the Programme Computer Science and Engineering

JOHAN BÖHLIN

Department of Computer Science and Engineering
Division of Computer Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden, 2008

Nomadic Device Integration For Automotive Applications

JOHAN BÖHLIN
©JOHAN BÖHLIN, 2008

Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Göteborg, Sweden 2008

Abstract

Mobile phones are becoming more and more sophisticated, and the current development tends to integrate all different functionalities like music, video, phone, internet, email, GPS, organiser and so on, into one portable device. When using a vehicle, it would be comfortable to stream music or place a call trough the cars infotainment system, without touching the mobile phone. The purpose of this master thesis has been to develop an API or gateway for easy integration of nomadic devices in a vehicle.

Required gateway API functions are functions for finding devices and services, and to connect and communicate with these wirelessly. Other preferable and required functions are functions for placing/answering/dialing calls and routing audio from the device to the gateway and further on, sending SMSes, accessing the internet, reading phone books etc. on a nomadic device.
The gateway communicates with a nomadic device using Bluetooth, which is a common communication standard available in many nomadic devices today, and which also has useful and well documented profiles.
Device search as well as functions on a nomadic device is accessible to other applications by an UDP-interface provided by the gateway.
The final gateway implements the Bluetooth serial, headset and handsfree profile and provides a dial-up internet access. It also implements non profile specific functionality like sending and reading SMSes, reading the phone book and receiving battery status.

Preface

This thesis is part of the Computer Science and Engineering Program at the Chalmers University of Technology, Gothenburg, Sweden, and has been issued by Volvo Technology in Gothenburg. The development has been carried out during the fall term of 2007 and the beginning of the spring term of 2008.

During my work I received a great amount of support from my supervisors at Volvo Technology, Frida Jonsson and Rasmus Nisslert.

I would also like to thank my examiner Arne Linde at the Department of Computer Engineering, Chalmers University Of Technology.

Gothenburg,
January 2008

Johan Böhlin

Contents

Definitions **9**

1 Introduction **10**

1.1 Document Overview . . . 10

1.2 Volvo Technology . . . 10

1.3 Background . . . 10

1.4 Description of task . . . 11

1.5 Project Goals . . . 11

2 NDGate - Nomadic Device Gateway **12**

2.1 System overview . . . 12

3 Design choice **13**

3.1 Gateway requirements . . . 13

3.1.1 Hardware compatibility . . . 13

3.1.2 Gateway functionality . . . 13

3.1.3 Device Connectivity . . . 15

3.1.4 Interface . . . 15

3.2 Device Connection Technology . . . 15

3.2.1 Alternatives . . . 16

3.2.1.1 Wireless USB or WUSB . . . 16

3.2.1.2 IrDA . . . 16

3.2.1.3 WLAN . . . 16

3.2.1.4 Bluetooth . . . 16

3.2.2 Best Alternative . . . 17

3.2.2.1 Bluetooth versus WLAN . . . 17

3.2.2.2 Bluetooth security . . . 18

3.3 Bluetooth Stack . . . 18

3.3.1 Overview . . . 19

3.3.1.1 Service Discovery Protocol (SDP) . . . 19

3.3.1.2 Logical Link Control and Adaptation Protocol (L2CAP) . . . 19

3.3.1.3 Radio Frequency Communication (RFCOMM) . . . 20

3.3.2 Profiles . . . 20

3.3.2.1 Serial Port Profile (SPP) . . . 20

3.3.2.2 Advanced Audio Distribution Profile (A2DP) . . . 20

3.3.2.3 Audio/Video Remote Control Profile (AVRCP) . . . 20

3.3.2.4 Dial-up Networking Profile (DUN) . . . 20

3.3.2.5 Headset Profile (HSP) . . . 20

3.3.2.6 Hands-Free Profile (HFP) 21
3.3.2.7 Object Exchange Protocol (OBEX) 21
3.3.2.8 Synchronization Profile (SYNC) 21
3.3.2.9 Video Distribution Profile (VDP) 21
3.4 Programming Language 21
3.5 Gateway Interface 21

4 Implementation **22**
4.1 Gateway overview 22
4.1.1 Services 23
4.2 Architecture and structure 23
4.3 Main class - Cmtapi 24
4.3.1 UDP interface 24
4.3.2 Commands 25
4.3.3 UDP-receiver thread 27
4.3.4 Command processing thread 27
4.3.5 Program configuration 28
4.4 Device and service discovery / finder service class - CbtFinder 28
4.4.1 Device discovery 29
4.4.2 Service discovery 29
4.4.3 Bonding/Pairing 29
4.5 RFCOMM connection class - CbtRfcomm 29
4.5.1 Connect function 30
4.6 L2CAP connection class - CbtL2cap 30
4.7 Serial port profile / serial service class - CbtSerial 31
4.7.1 AT-commands 31
4.7.2 Functions 32
4.7.2.1 General function procedure 32
4.7.2.2 Implemented functions 33
4.7.2.3 Incoming unsolicited AT-commands 33
4.7.3 TimerClass 34
4.8 Headset profile class / headset service - CbtHeadset 34
4.8.1 Profile layout 35
4.8.2 Profile initialization 35
4.9 Handsfree profile / handsfree service class - CbtHandsfree 36
4.9.1 Profile layout 37
4.9.2 Profile initalization 37
4.10 Internet Dial-up / dun service class - CbtDUN 38
4.11 Audio class - Caudio 40
4.11.1 Windows WinIn and WinOut API 40
4.11.1.1 WinIn - Reading/recording and sending audio 40
4.11.1.2 WinOut - Receiving and writing/playing audio 41
4.11.2 Audio format 41

5 Testing **42**
5.1 Raw AT commands 42
5.2 infoTainer - Infotainment Emulator 42
5.3 Tested devices 43

6 Results **45**

7 Discussions, Conclusions and Future developments **46**

7.1 Conclusions 46

7.2 Problems during development 46

7.3 Improvements 47

7.3.1 Services 47

7.3.2 Bluetooth stack issues 47

7.3.3 Non-implemented Bluetooth profiles/services 47

7.3.4 Future Bluetooth and Bluetooth profiles 47

References **49**

DEFINITIONS

Term	**Definition**
3G	Third generation of mobile phone standards and technology.
API	Application Programming Interface. A initial interface between an application and a hardware or other software.
ASCII	A character encoding based on the English alphabet.
GPRS	General Packet Radio Service. A Mobile Data Service. It provides data rates from 56 up to 114 Kbps.
GPS	Global Positioning System. A system of satellites and receiving devices used to compute positions on the Earth.
GUI	Graphical User Interface. A type of user interface which allows humans to interact with a computer and computer-controlled devices.
HFP	Hands Free Profile. Same as HSP, but with more functionality.
HMI	Human Machine Interface. The communication between the computer system and the humans using it.
HSP	HeadSet Profile. Bluetooth profile providing Bluetooth wireless simple call handling and audio routing.
Infotainment	In this context, a display showing both information and entertainment.
INI-file	A text file defining an initial variable value, one line per variable.
LAN	Local Area Network. A computer network covering a small geographic area.
ND	Nomadic Device. See below.
Nomadic Device	A hand-held wireless device such as a PDA or smartphone .
OEM	Original Equipment Manufacturer. The original manufacturer of a component for a product, which may be re-sold by another company.
PCM	Pulse-Code Modulation. A digital representation of an analog signal where the magnitude of the signal is sampled regularly at uniform intervals.
PPP	Point-to-Point Protocol. A data link protocol commonly used to establish a direct connection between two nodes.
PSM	Protocol/Service multiplexor. A Bluetooth L2CAP-channel service identifier.
RS-232	Recommended Standard 232. A standard for serial binary data signals connecting between data terminal equipments, used in computer serial ports.
SCN	Service Channel Number. A Bluetooth RFCOMM-connection service identifier.
SDK	Software Development Kit. A set of development tools allowing a software engineer to create applications for a certain software package, hardware platform or similar platform.
SIG	Special Interest Group. SIGs are volunteer-run discussion areas which focus on specific topics of interest to its members.
SPP	Serial Port Profile. Bluetooth profile for emulating serial ports over Bluetooth.
TCP	Transmission Control Protocol. One of the core protocols of the Internet protocol suite.
UDP	User Datagram Protocol. One of the core protocols of the Internet protocol suite.
WLAN	Wireless LAN.

Chapter 1

Introduction

This chapter introduces the reader to this document and the background for this thesis.

1.1 Document Overview

This document is a report for the master thesis ”Nomadic Device Integration For Automotive Applications” and further the development of a Bluetooth Gateway. The first part of this report is a system overview and requirement specification, followed by a technical overview, design choice and the actual implementation. The last part contains the result, conclusions and a discussion on possible future developments.

1.2 Volvo Technology

Volvo Technology (VTEC) is an innovation company developing new technologies and concepts for ”hard” as well as ”soft” products and processes within the transport and vehicle industry. VTEC is located at Lundbystrand and Chalmers Science Park in Göteborg, and at Volvo´s establishments in Lyon, France and Greensboro, USA [2].

1.3 Background

In the automotive industry, nomadic devices and off-board services are predicted to take market shares from the integrated systems offered by the OEMs, as well as introducing new functionality into the vehicles. For example, many customers already prefer mobile phones before an in-vehicle integrated phone and the same development can be expected in the area of navigation, media players etc.

Mobile phones are becoming more and more sophisticated, and the current development tends to integrate many different functions like music, video, phone, internet, email, GPS, organiser, etc. into one portable device. When using a vehicle, like a car, it would be comfortable to stream music or place a call trough the infotainment system of the car, without touching the mobile phone, and to control everything from the steering wheel or side panels, i.e. the mobile phone ought to be integrated into the vehicle and accessible in the in-vehicle HMI. There need to be a communication link between the nomadic device and a gateway in the car, and from this gateway a link between the gateway and the infotainment system. The gateway is an abstract layer/interface for accessing a nomadic device.

There are currently a number of possible approaches to making an integrated vehicle user interface for nomadic devices, but so far none have been agreed on as a standard. There are, however, some standards for communication with nomadic devices (i.e. Bluetooth profiles), and while waiting for a standard for the complete vehicle-ND integration, a detailed study focusing on the current possibilities would be very valuable.

1.4 Description of task

The goal of this master's thesis is to investigate possibilities for nomadic device integration in terms of present day available standards and communication technologies. Limits regarding the functionality available in integration using these standards should be identified and the impact of future technology or standards discussed.

In the evaluation of the available integration possibilities, an API, or gateway, for nomadic device integration should be developed. This API should allow for easy access to nomadic device functionality which could be used in research scenarios (simulations or demonstrations) concerning vehicle HMI solutions.

1.5 Project Goals

1. Investigation of available standards and communication technologies relevant for nomadic device integration in vehicles.
2. Identification of nomadic device functionality suitable for use in the vehicle using available standards.
3. Discussion on the impact future standards/technology might have on nomadic device integration.
4. Development of an API, or a gateway, supporting the functionality (or parts of the functionality) identified in 2.

Chapter 2

NDGate - Nomadic Device Gateway

This chapter is an overview of the gateway.

2.1 System overview

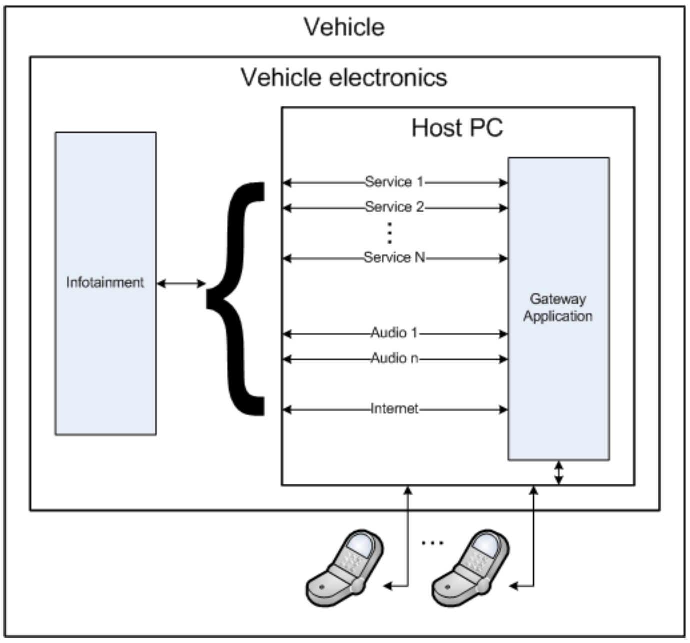


Figure 2.1: System overview

Figure 2.1 represents an overview of the system, and defines in which context the gateway can be used. The main task of the gateway is to communicate wirelessly with one or more nomadic devices in the local environment within the same vehicle. The gateway provides a interface with different services, in order to access functions in the gateway or in a connected nomadic device. A service is a set of functions as for example functions for managing devices connected to the gateway or for accessing information on a nomadic device. A typical user of the interface and the functions is a infotainment system, which will display device information and control connected devices. A vehicle driver interacts with a nomadic device using displays and navigation keys connected to the infotainment system in the vehicle.

Chapter 3

Design choice

This chapter lists the requirements of the gateway, available communication technologies, interfaces and programming languages, and discusses which one of each is the most appropriate to be used in the gateway.

3.1 Gateway requirements

3.1.1 Hardware compatibility

The gateway application should be able to run on a regular PC running Windows XP, using a standard consumer hardware dongle if necessary, and using as much free development tools, code bases and development kits as possible.

3.1.2 Gateway functionality

Functionality is a function within a nomadic device or a general function in the gateway.
Fundamental required functionality refers to the functionality which should be available in the gateway and accessible in the API. Preferred functionality should be implemented if time exists and optional services may be interested in the future and are not a part of this master thesis.
The gateway function requirements and their priority grouping are extracted from a requirement specification and a user case scenario document supplied by Volvo technology. These documents are not included in the report due to secretion.

Fundamental required functionality

What	**Description**
Find devices and services.	The gateway should be able to find devices and what services a device provides.
Establish a secure connection.	Device connections should be encrypted with authorization.
Answer/reject/hang up/place new calls.	The gateway should be notified about incoming calls and be able to answer or reject them, and also place new calls.
Route audio.	Voice call audio should be routed from the nomadic device to the gateway, and the other way around.
Read/receive/send SMS.	The gateway should be notified about new incoming SMS, be able to read them and send new SMS.

Table 3.1: Fundamental required functionality

Preferred functionality

What	**Description**
Read phone book.	Access the device phone book read contact entries.
Read last dialed/missed/received call list.	Read recent calls, missed calls and received calls lists.
Use a device as an internet gateway.	The gateway should be able to access internet using the nomadic device as an internet gateway.
Stream music from a device to gateway.	The nomadic device should be able to stream high quality music to the gateway.
Control music.	The gateway should be able to stop, pause, play, skip, read current song title currently played on the nomadic device.
Read/Write calendar.	Read and write a calendar events from a nomadic device.
Set ring level.	It should be possible to mute or set the ring volume on the device.

Table 3.2: Preferred functionality

Optional functionality

What	**Description**
Access GPS-data.	The gateway can access the GPS, if available, in a nomadic device and also receive map-data.
Control ND using voice (voice dialing).	Navigate in the mobile and dial using voice.
Stream/Control video.	Same as for music, but for video.

Table 3.3: Optional functionality

3.1.3 Device Connectivity

- The use of the nomadic device in the car should be as seamless as possible, and the need to force the user to connect a cable to the device prevents this. Therefore, a wireless connectivity is preferred.
- Low power consumption due to the limited battery in a nomadic device.
- Availability. The technology should be available in current mobile phones.
- Security. The technology should be secure enough to prevent unauthorized access.
- The minimum range should be 5 meters so passengers in the backseat can integrate their nomadic devices.
- It should be possible to have at least one device connected at a time, but multiple connection is preferable.
- There sholud be enough bandwidth for the services needed. Here is a list of bandwidth requirements for different kind of media streams:

  Voice audio: 64 kbit/s [29]

  Music audio: 288 kbit/s [29]

  DirectShow video (512x384, XviD, MP3) compressed: 1.1 Mbit/s

  2-channel stereo-quality audio compressed: 260 kbit/s

  2-channel CD quality audio (uncompressed): 1.4 Mbit/s [26]

  HDTV compressed (960x528, XviD, AC3): 2.2 Mbit/s

  DVD: 6 Mbit/s [27]

  HD-DVD: 36 Mbit/s [28]

  Blue-ray: 48 Mbit/s [28]

  For the services specified in table 3.1 and 3.2 starting at page 14, audio streaming is the most bandwidth consuming service which will require 260 kbits/s. So at least 500 kbit/s bandwidth is required to allow overhead and simultaneous services.

3.1.4 Interface

It should be possible to communicate with a nomadic device and provide the services/functions available on the device to other applications running in the environment of the vehicle. The interface should be accessible by applications running on other PCs or hardware, connected to the same network, outside the PC the gateway application is running on, or on the same PC. The interface should be independent of any programming language.

3.2 Device Connection Technology

This section discusses different technologies available to connect a nomadic device to the gateway in a vehicle.

3.2.1 Alternatives

There are a number of different ways to communicate with a nomadic device, either wired or wireless. Common current and future technologies are the Universal Serial Bus (USB) (wired and wireless), Firewire, IrDA, Bluetooth and WLAN.
As stated in the requirements, see section 3.1.3, the technology should be wireless and thereby the following overview contains only wireless technologies.

3.2.1.1 Wireless USB or WUSB

Wireless USB is USB but wireless, and is still considered to be a future technology. So far, only a few [3] devices have been certified to use WUSB and it's also not legal in all countries due to frequency licenses. It has a very high bandwidth [4] of 480 Mbit/s at distances up to 3 meters and 110 Mbit/s up to 10 meters. USB also has different device classes which define an expected behavior for a connected device, which removes the need for developing new device drivers, at the same time simplifying the data exchange [5].
The drawbacks of this technology is its unavailability, though it may be considered a good candidate in the future given its high bandwidth, its ability to provide high-quality video streaming, and the seamless integration to pre-existing wired USB-products.

3.2.1.2 IrDA

IrDA or Infrared Data Association [6] is a short range, low power communication standard using infrared light to exchange data. IrDA has a range of 1m down to 0.2 m, depending on power mode, and requires clear visible sight between the units communicating. Full-duplex is also impractical since the transmitter sees it own transmitted light. IrDA is a simple technology and it only defines the data transport, not any device classes/profiles. The short distance, lack of device classes/profiles and the free-sight requirement are drawbacks and make IrDA inconvenient for this kind of environment.

3.2.1.3 WLAN

Wireless LAN is widely used today, mostly for providing wireless internet access to stationary and portable laptops, but it is available in mobile phones like the Nokia N95 [7]. There are plenty of WLAN (IEEE 802.11) standards which operate at different frequencies and have different range and bandwidth. The most common standards today are IEEE 802.11g [8] with a maximum data rate of 54 Mbit/s (19 Mbit/s typical) and an indoor range of 35 m. The power consumption [9, 10] ranges from 10mW to 300mW depending on data rate and the distance between the nodes. WLAN has an inbuilt security technology called WPA, Wi-Fi Protected Access [11], providing authentication and encryption. WLAN is currently being developed and faster standards will soon be available, such as the IEEE 802.11n with a data rate up to 248 Mbit/s.

3.2.1.4 Bluetooth

Bluetooth is a communication standard designed for low-cost, low-power consumption communication, available in many mobile phones and hand-held devices [13, 14]. The current version, 2.0+EDR (Extended Data Rate), supports a maximum bandwidth of 3 Mbit/s (2.1 Mbit/s typical). There are different classes of Bluetooth with different range/maximum power consumption requirements. The most common one is class 2, with a range of 10 meters and a maximum power consumption of 2.5mW. Future standards are Bluetooth 3.0 which has a data rate of 480 Mbit/s, similar to Wireless USB. Bluetooth provides both authentication and encryption.

3.2.2 Best Alternative

Of the alternatives in section 3.2.1, only Bluetooth and WLAN are suitable.

3.2.2.1 Bluetooth versus WLAN

	Bluetooth 1.2	*Bluetooth 2.0 + EDR*	*WLAN 802.11b*	*WLAN 802.11g*
Bandwidth:	1 Mbit/s	3 Mbit/s	11 Mbit/s	54 Mbit/s
Range:	10 m (class 2)	10 m (class 2)	35 m	35 m
Power:	2.5mW	2.5mW	200mW	200mW

Table 3.4: Bluetooth versus WLAN bandwidth/range/power consumtion

Both Bluetooth and WLAN meet the bandwidth and distance requirements, and are also available in nomadic devices on the market today. The main different between the two technologies is the software required on the nomadic device and in the gateway to implement the requested services. After a WLAN connection has been established using WPA authentication, which is done on a lower level, regular internet sockets can be created where data is exchanged. To create such sockets and access objects, functions and data on a nomadic device, some client software is needed on the device. The software must to be certified by the phone manufacturer if the software needs to access functionality like answering phone calls, route audio, and so on. The certification process requires the need to buy certificates, which cost money and can take a long time to process. Using sockets, it is quite simple to implement streaming and other services using regular internet protocols, but there is much to do in developing the mobile application for accessing hardware. Also, almost the same amount of work is required in the gateway. Compared to WLAN, Bluetooth has different application profiles [15]. A profile defines the behavior according to which Bluetooth devices should communicate with each other for a specific type of service, i.e. the headset profile describes how audio and commands are sent between a headset and an audio gateway. Using the Bluetooth profiles simplifies the implementation by removing the requirement of additional software on the nomadic device, as long as the nomadic device support the Bluetooth profiles needed.

Table 3.5 lists the Bluetooth profiles which may be used for the respective functionality, see section 3.1.2 on page 13.

Service/Function	**Bluetooth profile**
Find devices/services.	Built in Service Discovery Profile (SDP).
Answer/reject/hang up/place new calls.	Serial (SPP), Headset (HSP), Hands Free (HFP).
Route audio ongoing call from phone to gateway.	HSP, HFP.
Read/receive/send SMS.	Not defined in any profile, but can work with SPP, HSP, HFP.
Read/write phone book.	Same as for SMS. Synchronization (SYNC).
Read last dialed/missed/received call list.	Same for SMS.
Use ND as an internet gateway.	Personal Area Networking (PAN), Dial-up Networking (DUN).
Stream music from ND to gateway.	Advanced Audio Distribution (A2DP).
Control music.	Audio/Video Remote Control (AVRCP).
Read/Write calendar.	SYNC.
Access GPS-data.	No profile.
Control ND using voice (voice dialing).	HSP, HFP.
Stream/Control video.	AVRCP.

Table 3.5: Bluetooth profiles suitable for each service/function

Bluetooth has a higher availability in current nomadic devices, and a lower power consumption and along with sophisticated profiles which could be used without additional software needed on the target device. Bluetooth was therefore selected as the technology to access nomadic devices wirelessly.

3.2.2.2 Bluetooth security

There are several potential security issues regarding the Bluetooth link between a nomadic device and the gateway. A hacker could act as an authenticated device, connect to the gateway and inject commands. This issue refers to the Bluetooth pairing vulnerability [16]. Another possibility is for the hacker to use bugs in the Bluetooth stack in the gateway to gain access to the network to which the gateway is connected [17]. The latter problem is solved by using a recent and up-to-date Bluetooth stack version where these known bugs are solved. According to Bluetooth SIG, the resolution to the fist problem is to use a long PIN-code and do the paring in a private location. The paring is only done once per device and on the initiative of the user trough the gateway (You can read mor about pairing in section 4.4.3). These security issues are also pointed out by [18].

3.3 Bluetooth Stack

For Windows XP, there are essentially two free Bluetooth stacks available, either Microsoft's own stack or the Broadcom Bluetooth stack [19]. There are SDK available for both stacks, though these are quite different [25]. Microsoft's stack uses C while Broadcom's stack is a set of C++. The Broadcom stack supports virtual COM ports while Microsoft's stack has socket support. The Broadcom stack is selected since it has a good documentation, example source code and since it is built on C++.

3.3.1 Overview

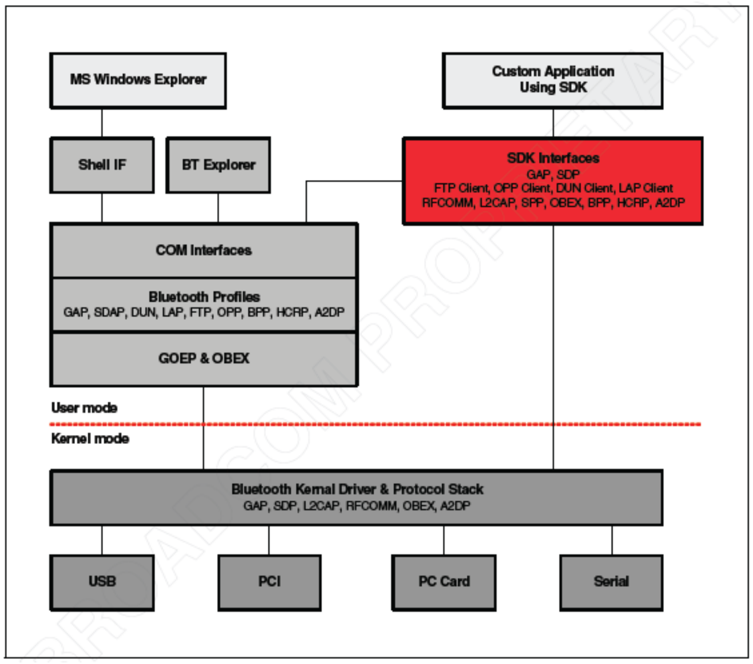

MS Windows Explorer
Custom Application Using SDK
Shell IF
BT Explorer
SDK Interfaces
GAP, SDP
FTP Client, OPP Client, DUN Client, LAP Client
RFCOMM, L2CAP, SPP, OBEX, BPP, HCRP, A2DP
COM Interfaces
Bluetooth Profiles
GAP, SDAP, DUN, LAP, FTP, OPP, BPP, HCRP, A2DP
GOEP & OBEX
User mode
Kernel mode
Bluetooth Kernal Driver & Protocol Stack
GAP, SDP, L2CAP, RFCOMM, OBEX, A2DP
USB
PCI
PC Card
Serial

Figure 3.1: BTW Stack Overview [19]

The Bluetooth stack consists of a kernel part and an user part. The kernel part contains both the driver for the Bluetooth hardware as well as fundamental Bluetooth specific protocol like SDP, L2CAP and RFCOMM.
As seen in figure 3.1, the user part provides SDK interfaces to Bluetooth specific protocols, as well as interface to some application Bluetooth profiles.

The following sections list some fundamental protocols.

3.3.1.1 Service Discovery Protocol (SDP)

SDP defines how devices discovers other devices and services, as well as how devices should announce their own services and capabilities. A device can act as either a client or as a server, or both. Each device has its own SDP database, containing a number of service records. A service record contains a universally unique service id (UUID), service version, parameters, port numbers and different kinds of information on how a client can connect to the service. The UUID is unique for each service or Bluetooth profile.

3.3.1.2 Logical Link Control and Adaptation Protocol (L2CAP)

L2CAP provides flow-control and retransmission of packets for upper level protocols and applications, like the TCP-protocol does in the internet protocol suite. The L2CAP-layer consists of L2CAP-channels, where each channel is identified with a channel identifier, CID. A device connects to another device with L2CAP using a specific CID received from a service record, see 3.3.1.1 at page 19. A PSM (Protocol/Service multiplexor) is assigned to each channel allowing for a L2CAP-connection to be created to specific services.

3.3.1.3 Radio Frequency Communication (RFCOMM)

RFCOMM is implemented on top of the L2CAP layer, and emulates the RS-232 protocol, with all its control and flow signals. As for the L2CAP-protocol, RFCOMM connections have different channel identifiers to differentiate each channel. Each RFCOMM-connection is identified with a SCN (Service Channel Number) allowing up to 60 simultaneous RFCOMM-connections.

3.3.2 Profiles

A Bluetooth profile, or application profile, is implemented on top of the L2CAP protocol or the RF-COMM protocol. A profile defines a behavior for a device and specifies how devices should communicate when acting as a service client or server.

Common Bluetooth profiles are described in the following sub-sections.

3.3.2.1 Serial Port Profile (SPP)

The SPP profile uses the RFCOMM-protocol to set up virtual serial ports between devices. SPP is like a wireless serial RS-232 connection over Bluetooth. The max data rate for this profile is 128 Kbit/sec.

3.3.2.2 Advanced Audio Distribution Profile (A2DP)

Profile for streaming stereo-quality audio between two Bluetooth devices. Commonly used to stream music between a music player, like a mobile phone, and a headset. Supports the audio formats SBC, MEPG-1,2 Audio, Mpeg-2,4 AAC and ATRAC. This profile only supports two-channel audio and are using L2CAP-channels as its barrier.

3.3.2.3 Audio/Video Remote Control Profile (AVRCP)

The AVRCP profile is a profile for controlling Audio/video-equipment, TV, media players and similar. The profile defines commands like pausing, stopping and starting a playback, and many others general and non-general commands. It also defines how to read and receive information on the current playing media. A typical application is a Bluetooth enabled remote control. AVRCP uses the L2CAP protocol.

3.3.2.4 Dial-up Networking Profile (DUN)

The DUN profile, when running as a server, emulates a general modem. When connected to the service as a client the modem can be used as an regular dial-up modem. The modem makes it possible to create an dial-up connection and to gain access to the internet. The Bluetooth device emulating the modem typically uses GPRS or 3G as its data barrier. The DUN profile uses the RFCOMM protocol.

3.3.2.5 Headset Profile (HSP)

The HSP profile is used for transferring call audio and controlling phone calls on a server Bluetooth device from a client device. The server device has access to a network and is able to create phone calls. From the client device, calls can be placed, hung up and answered. The microphone and speaker audio is also routed from the server to the client device. This profile is commonly used between a mobile phone and a headset, and the profile uses the RFCOMM protocol.

3.3.2.6 Hands-Free Profile (HFP)

This profile is similar to the HSP, but has more functions. The HFP profile supports multi-party calls, events like call status changes, in-band ring tones, incoming call notification, and more. This profile also uses the RFCOMM protocol. This profile is common in cars.

3.3.2.7 Object Exchange Protocol (OBEX)

OBEX is a protocol and not a Bluetooth profile, but is used over RFCOMM for transferring files/objects between Bluetooth enabled devices. OBEX also supports listing of file system directories.

3.3.2.8 Synchronization Profile (SYNC)

The SYNC profile is used for synchronization of object between devices, like phone book and calendar entries. This profile uses the OBEX protocol.

3.3.2.9 Video Distribution Profile (VDP)

This profile enables video streaming between Bluetooth enabled devices. The profile has a mandatory support for the H.263 codec, but can optionally support the MPEG-4 codec.

3.4 Programming Language

There are a numerous different programming languages to choose from. The general requirement is an object-oriented language and a language which easily can use the Bluetooth stack. Since the Broadcom stack is written in C++, this is obviously the programming language to choose since it is also object-oriented and a well documented language with a lot of help to be found on the internet.

3.5 Gateway Interface

The gateway application provides an interface to one or more nomadic devices. As the requirements states, see section 3.1.4 at page 15, it should be possible to communicate with a nomadic device and provide the services/functions available on the device to other applications running in the environment of the vehicle. The gateway application runs on a regular PC running Windows XP.
The interface should be be accessible by applications running on other PCs or hardware, connected to the same network, outside of the gateway PC. An internet transport protocol like UDP or TCP is a suitable transport layer for the interface protocol. UDP compared to TCP is easier to implement, has less overhead and is connectionless, but does not guarantee that a sent packet is received by the other end. List 3.5 outlines why UDP is a good interface protocol. :

- Requires less code for implementation.
- Small amount of packet overhead.
- The interface commands contains relative small amounts of data, like an SMS or phone book entries, and the traffic will be low. This leads to low network utilization and reduces the changes of a packet loss, all of which is suitable for UDP.

Chapter 4

Implementation

This chapter describes how the gateway is implemented.

4.1 Gateway overview

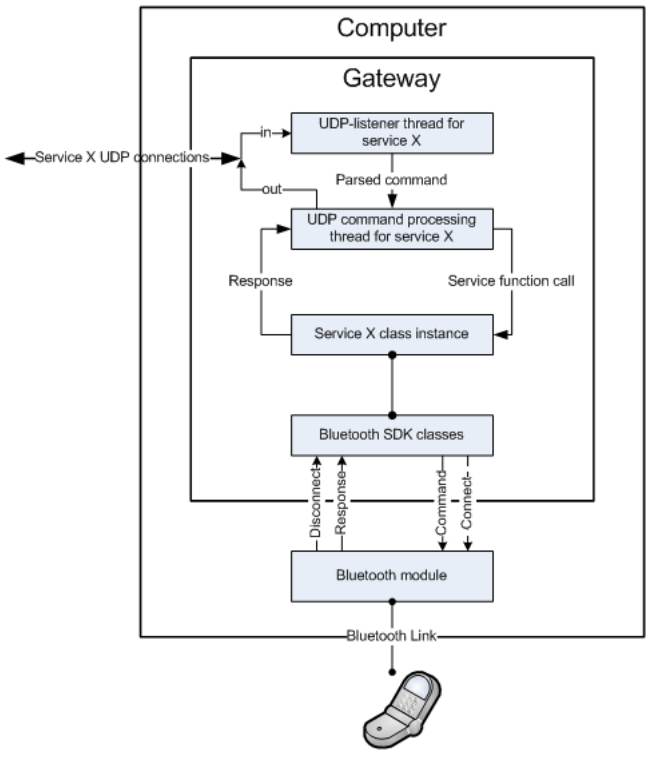


Figure 4.1: Gateway overview

Figure 4.1 shows an overview of the gateway. To interact with a device, UDP-packets are sent to and received from the gateway using the UDP-port number pair associated with the service one wants to use. Each UDP-packet is parsed to a command and processed where the corresponding class function is called. A class function uses lower level Bluetooth SDK-class function to send/receive/access data on a Bluetooth device.

4.1.1 Services

A service is a set of functions accessible over UDP, and its often associated with a Bluetooth profile. As stated earlier, each service has its own UDP-port number pair allowing for different services to be accessed at the same time. Table 4.1 lists the available services. All service classes exists but are not implemented, see section 7.3.3 Future development on page 47.

Service	**Description/Functionality/Implements**	**Implemented?**
finder	Search for devices and services, bond and und-bond of devices.	yes
serial	Serial port profile and general device function-alities.	yes
headset	Bluetooth headset profile.	yes
handsfree	Bluetooth handfree profile.	yes
dun	Dial-up internet.	yes
a2dp	Stereo quality audio streaming.	**no**
avrcp	Media player control.	**no**
obextftp	Directory listing and file transfer.	**no**
syncml	Synchronization of calendar and phone book entries.	**no**

Table 4.1: Services

4.2 Architecture and structure

The method applied throughout in the development is the division of codes into classes and functions, and the creation of base classes which can be extended by other classes. This method allows for a code which is easy to follow, but which above all makes debugging easier. Functionality are broken down in to simpler sub functions which make it easy to test each sub function for its validity and thus decrease the time it takes to find an possible error. On a different level this results in reduced code replication.

4.3 Main class - Cmtapi

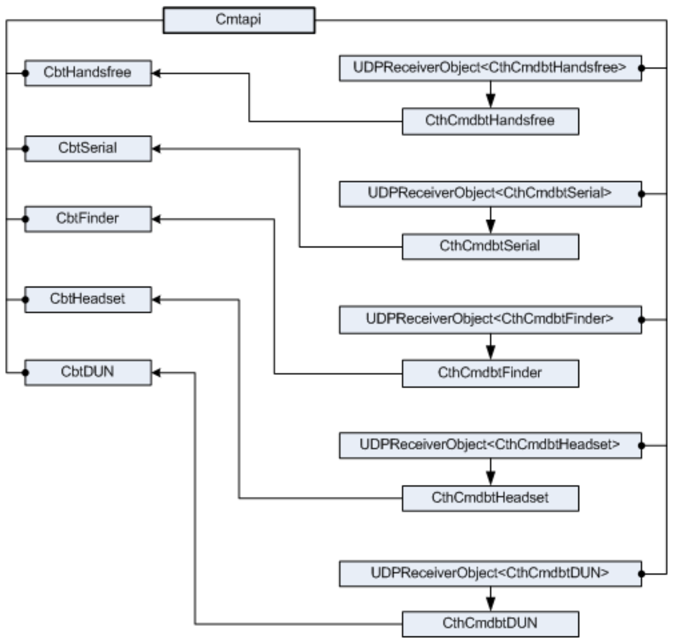

Cmtapi
CbtHandsfree
CbtSerial
CbtFinder
CbtHeadset
CbtDUN
UDPReceiverObject<CthCmdbtHandsfree>
CthCmdbtHandsfree
UDPReceiverObject<CthCmdbtSerial>
CthCmdbtSerial
UDPReceiverObject<CthCmdbtFinder>
CthCmdbtFinder
UDPReceiverObject<CthCmdbtHeadset>
CthCmdbtHeadset
UDPReceiverObject<CthCmdbtDUN>
CthCmdbtDUN

Figure 4.2: Cmtapi class

Figure 4.2 shows an overview of the main class Cmtapi. Upon construction this class creates a service class and an UDP-receiver thread for each service, where each UDP-receiver thread listens on different port-numbers. On a new incoming packet on UDP, the packet is parsed to a command and a new processing thread is created where the command is processed. In the processing thread, a proper service class function is called.

4.3.1 UDP interface

The interface is a simple command-response structure, where each sent command is followed by an error or an success response. The sent commands are in human readable ASCII-format, see section 4.3.2. Each service listens for commands on an UDP-port, and responds on another UDP-port. The UDP-port is used for different services, where the base port is configured as an configuration variable to the program. See section 4.3.5. Table 4.2 lists how the UDP-ports are set up.

Service	**Out to gateway**	**In from gateway**
finder	Base+1	Base+0
serial	Base+3	Base+2
headset	Base+5	Base+4
obexftp	Base+7	Base+6
syncml	Base+9	Base+8
avrcp	Base+11	Base+10
a2dp	Base+13	Base+12
dun	Base+15	Base+14
handsfree	Base+17	Base+16
Headset-audio	Base+101	Base+100
Handsfree-audio	Base+103	Base+102
A2DP-audio	Base+105	Base+104

Table 4.2: UDP port mapping

4.3.2 Commands

A command has three parts: A function name, zero or more arguments and a terminator.

The command is formatted in one of the two following ways:

For one or more arguments:

Function name	<space>	One or more arguments	terminator

Table 4.3: One or more arguments

For zero arguments:

Function name	terminator

Table 4.4: Zero arguments

The *function name* is the name of the function.
If there are no arguments, the terminator follows immediately after the function name.
The *terminator* consists of the carrier return, followed by the new line character, which is written \r\n, corresponding to the ASCII values 13 and 10, respectively.

If there is at least one argument, a space separates the function name and the arguments. The first character after the space is part of the first (or only) argument. Each argument is separated with the character sequence **{:}**, i.e. a left frizz-parenthesis, a colon and finally a right frizz-parenthesis. An argument can be zero or more characters. The last argument is followed by the *terminator*. An argument cannot contain the **{:}** sequence.

Command examples

functionName argument1{:}argument2{:}argument3\r\n	A command with three arguments.
functionName argument1\r\n	A command with one argument.
functionName argument1{:}{:}argument3\r\n	A command with three arguments, where argument2 is empty.
functionName\r\n	A command with no argument.
functionName \r\n	A command with one empty argument.
functionName {:}\r\n	A command with two empty arguments.
functionName {:}{:}\r\n	A command with three empty arguments.
<space>functionName\r\n	An **INVALID** command.
\r\n	An **INVALID** command.

Table 4.5: Command examples

Command responses
All commands sent are responded to with the same command structure, and with zero or more arguments. The function name for a response command has the string *Return* appended to the function name. Example: If the command *functionName argument1\r\n* is sent, a possible return argument could be *functionNameReturn\r\n* or *functionNameReturn argument1\r\n.*

Error responses
An error response is sent when a command produces an error. The error command is formatted as follow:

error	<space>	errValue{:}subErrValue{:}errStr	terminator

Table 4.6: Error command

Where the function name is *error*, and argument 1 and 2 are error type strings, see section *Error codes* in the user manual, and errStr is a human readable string of the error occurred, useful for debugging. An example of an error response:
error errInvalidArguments{:}subErrInvalidNumber{:}The number is invalid.\r\n

Lists
An argument could be a list. A list separates its elements using the | character (ASCII = 124).
Example list:

functionNameReturn element1|element2|element3||element5:argument2\r\n

Where the list is *element1|element2|element3||element5* with five elements, where element 4 is empty.

4.3.3 UDP-receiver thread

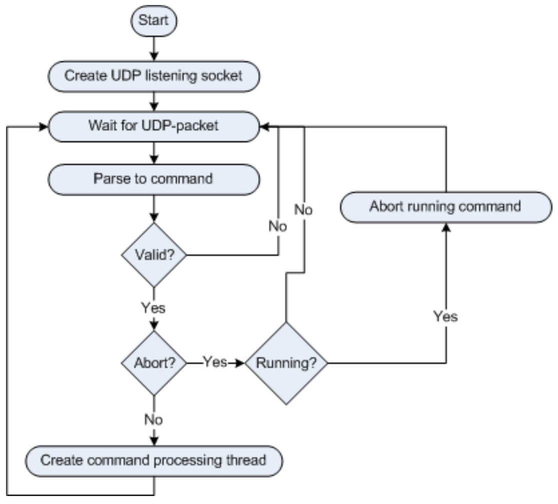


Figure 4.3: UDP-receiver thread

One UDP-receiver thread is created for each service, where each service listens on a different UDP-port number. The main loop of the thread is blocking until a new UDP-packet is received, where the packet is parsed into a command and validated. If no previous thread is in progress, a new command processing thread is created, with the command as an argument, see section 4.3.4. The UDP-receiver thread prevents a new command to be started when there is a running command, and allows for a command to be aborted.

4.3.4 Command processing thread

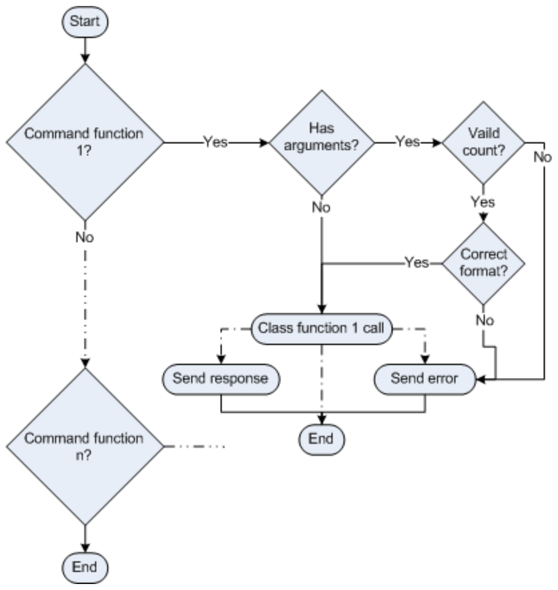


Figure 4.4: Command processing thread

Each service has a command processing thread. This thread has a set of statements checking for known function calls found in the command. When a function matches a statement, the corresponding service class function is called. If the class function needs arguments, the number and type of arguments are checked against the required arguments for the class function. The result from the class function call is handled and formatted into a new command, and is sent back as an UDP packet.

4.3.5 Program configuration

The gateway relies on a couple of configurable variables. The value for these variables is stored in a INI-file which is read when the gateway is started. Table 4.7 lists available configuration variables.

Variable	**Values**	**Default**	**Description**
udpBase	1-65535	9000	*The base UDP-port. See section 4.3.1*
udpDestinationHost	a string	127.0.0.1	*The destination address where udp-packets are sent on from the gateway.*
bluetoothModemComPort	1-30	3	*The COM-port number the Bluetooth modem is connected to. See section 4.10.*
logToFile	0-1	0	*Prints log messages to the NDGate.log file in the NDGate.exe directory. 0 to disable.*
logToConsole	0-1	1	*Prints log messages to console window. 0 to disable.*
handsfreeDevices	devices		*Device addresses in the hexadecimal format separated by comma.*
handsfreeScns	Scns		*The handsfree SCN for each device in handsfreeDevices. See 4.4.2 about the handsfree workaround.*

Table 4.7: Configuration variables

4.4 Device and service discovery / finder service class - CbtFinder

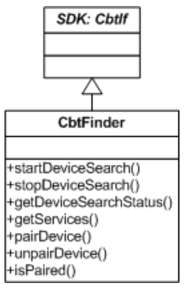


Figure 4.5: Device and service discovery class - CbtFinder

The CbtFinder class provides functions for searching for devices and services, as well as bonding to a device.

4.4.1 Device discovery

The device discovery searches for all Bluetooth enabled devices nearby. Only Bluetooth devices which have enabled discovery are found. A device search is initiated by calling a SDK-specific function and where each discovered device is reported back in a call-back function using the device unique hardware address of the device and an optional human readable device name.

4.4.2 Service discovery

Service discovery is issued per device. A device service inquiry returns all available Bluetooth profiles for a device, but only services supported by the gateway are returned. Each service has an optional service name, an universal unique identifier (UUID) and has either a SCN if the service is based on RFCOMM, or a PSM if the service uses the L2CAP-protocol. Read more about RFCOMM in section 3.3.1.3 and L2CAP in section 3.3.1.2 on page 19.
Bevause of a bug in the Broadcom Bluetooth stack, it is not possible to find the Bluetooth handsfree profile on a device. Since the profile is not found, the SCN for the profile is unknown and it is not possible to connect to the handsfree service on a device. The solution is to manually add the SCN for each device as an configuration variable to the gateway, see section 4.3.5 on page 28.

4.4.3 Bonding/Pairing

Before it is possible to connect to a service on a device, the device has to be paired, or bonded, with the gateway. This prevents unauthorized devices to access data on a device. The bonding is accomplished by issuing a bond function call to a specific device. This causes the user of this device to manually enter a PIN-number, which must be the same as the PIN-number supplied to the bond function as an argument. If the PIN-numbers match, the gateway and the the device are bonded allowing the gateway to create RFCOMM- or L2CAP-connections to the device.

4.5 RFCOMM connection class - CbtRfcomm

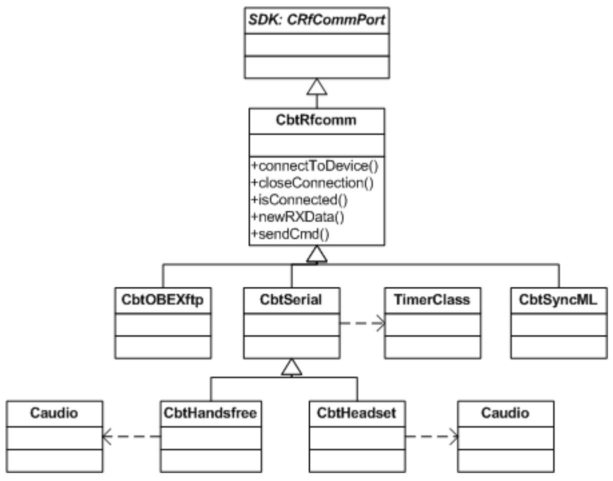


Figure 4.6: RFCOMM connection class - CbtRfcomm

The *CbtRfcomm* class implements functions for creating and removing a RFCOMM-connection to a specific service on a device using the device address and the SCN. This class contains a pure virtual function *newRXData* which is called when an open connection has new incoming data.

Because of its pure virtual function member, the class itself is abstract and cannot be used directly. All services using RFCOMM extends this class and implements the pure virtual function *newRXData*. Note that the two classes *CbtOBEXTftp* and *CbtSyncML* which extends *CbtRfcomm* are currently not implemented, see section 7.3.3 Future development on page 47.

4.5.1 Connect function

The connect function, *connectToDevice*, is synchronous. A call to this function blocks until a connection attempt is successful, times out or fails. The SDK CRfCommPort class connection function is non-blocking, and the class signals connection status change via a call-back. The *connectToDevice* function uses events to wait for a connection status change indicating a successful connection. Figure 4.7 explains the procedure.

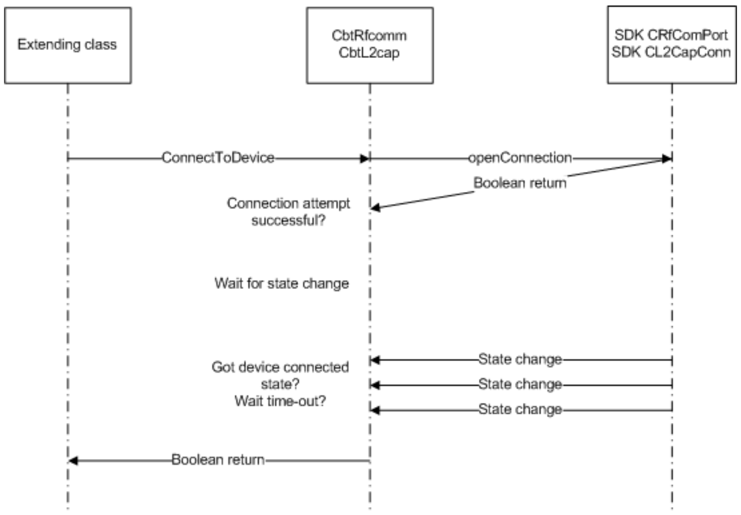


Figure 4.7: CbtRfcomm/CbtL2cap connection function call

4.6 L2CAP connection class - CbtL2cap

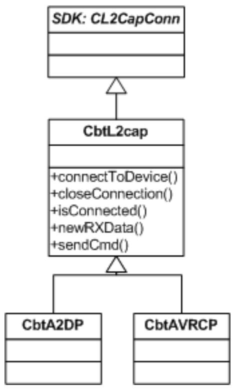


Figure 4.8: L2CAP connection class - CbtL2cap

The *CbtL2cap* class is similar to the *CbtRfcomm* class, but instead of a RFCOMM-connection, it provides functions for establishing and removing L2CAP-connections to a service on a device using an device address and a PSM. Like the *CbtRfcomm* class, this class is abstract and has a pure virtual new incoming data function *newRXData*. There are currently no implemented classes using the *CbtL2cap* class, see section 7.3.3 Future development on page 47. The connect function works in the same way as for the *CbtRfComm* class, see section 4.5.1 and figure 4.7.

4.7 Serial port profile / serial service class - CbtSerial

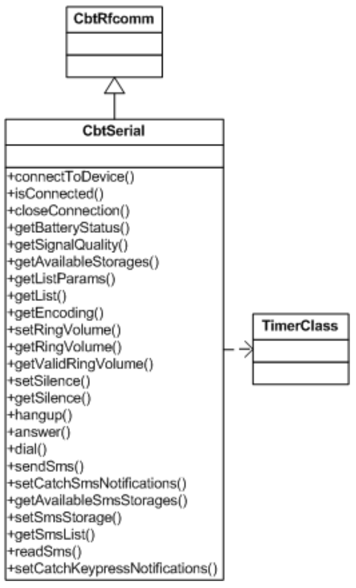


Figure 4.9: Serial AT-command class - CbtSerial

The serial class extends the *CbtRfcomm* class to connect to a serial port profile (SPP) on a device, and provides the serial service on UDP. The serial port profile is based on the RFCOMM protocol, and defines how a virtual serial port is created, but the data sent over the connection is application specific and is not defined in the SPP. However, all nomadic devices that have the SPP enabled as a server responds to AT-commands sent to the device, though it is nowhere defined which AT-commands a device must implement. This class implements common functions using AT-commands that most devices support [30, 31, 32, 33, 34].

4.7.1 AT-commands

AT-commands, or the Hayes command set, is a command language developed in 1977 for the Heyes Smartmodem 300 baud modem. A command is formed as a human readable string, and ends with the carrier return, which is written \r, corresponding to the ASCII values 13.
There are many different AT commands, and different vendors often have their own sub-command set, but the most general syntax is seen in table 4.8. All commands begins with the *AT* string, followed by a specific command. If the command is successful, the response is zero or more return values, each on a new line, followed by *\r\nOK\r\n*. If the command fails, the response is *\r\nERROR\r\n*. See table 4.9 for examples.

Command	**Description**
AT+cmd\r	**Execute** a command.
AT+cmd?\r	**Read** the current value for a command.
AT+cmd=?\r	**Test** if a command is available, and also query for which values that are valid.
AT+cmd=<*value*>\r	**Set** a value for a command.

Table 4.8: AT-command syntax

Command	**Response**	**Description**
AT+cmd1\r	\r\nOK\r\n	Execute the command *+cmd1*.
AT+cmd2?\r	+cmd: A\r\n\r\nOK\r\n	Read the value of *cmd2*, which in this case is A.
AT+cmd3?\r	+cmd: B\r\n+cmd: C\r\n\r\nOK\r\n	Read value of *cmd3*, which has two values B and C.
AT+cmd4=D\r	\r\nOK\r\n	Set *cmd4* to value D.
AT+cmd5=E\r	\r\nERROR\r\n	The command *cmd5* is not implemented, or the value E is invalid for this command.

Table 4.9: AT-command examples

4.7.2 Functions

Not all functions implemented are supported by all devices.

4.7.2.1 General function procedure

All functions work in the same way. The AT-command is written on the serial connection to the device, a waiting event is created and the input buffer is cleared. When either an \r\nOK\r\nor \r\nERROR\r\nis received, the event is raised and received data is processed, if any, and the function returns. If neither OK nor ERROR is received the event times out and the function returns.

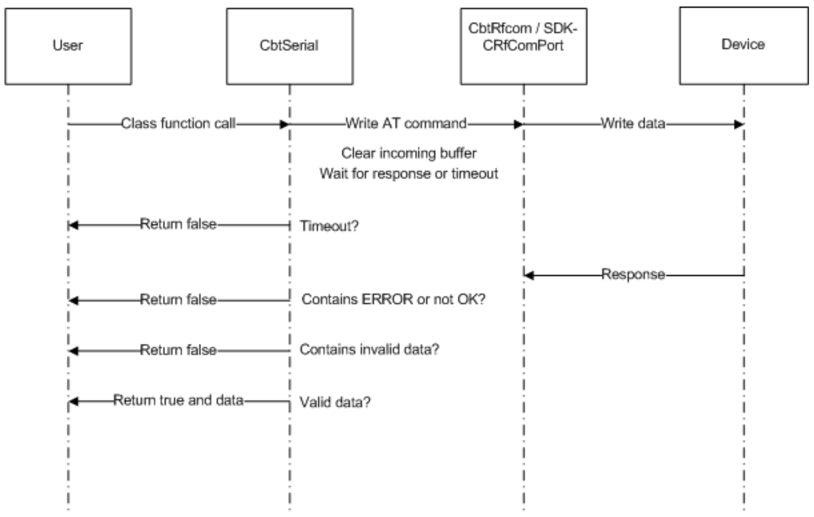


Figure 4.10: General function procedure

4.7.2.2 Implemented functions

Table 4.10 contains all the device functions the *CbtSerial* provides.

Command	Description
getBatteryStatus	Get battery level and status.
getSignalQuality	Get strength of the signal level of the GSM or UMTS-network.
getAvailableStorages	Get all available phone book storages. This includes unanswered calls, recent calls list, etc.
getListParams	Returns the start index and end index for entries in a phone book, as well as the maximum length of the name and number stored for a entry.
getList	Get all entries between two indexes from a phone book storage.
getEncoding	Get the current encoding for received text data in phone book entries and SMSes.
setRingVolume	Set the ring volume.
getRingVolume	Get the current ring volume.
getValidRingVolume	Get valid ring volume interval.
setSilence	Set the phone in soundless mode or not.
getSilence	Get silence mode status.
setCatchKeypressNotifications	Enable or disable user key press notification.
hangup	hang up an ongoing or incoming call.
answer	Answer an incoming call.
dial	Dial a number.
sendSms	Create and send a new SMS.
setCatchSmsNotifications	Enable or disable SMS notification.
getAvailableSmsStorages	Get available SMS storages for a SMS storage type.
setSmsStorage	Set the storage for a SMS storage type.
getSmsList	Get all SMSes in a SMS storage.
readSms	Read a SMS from a SMS storage.

Table 4.10: Implemented functions

Refer to the user manual for detailed functionality, parameters and return arguments.

4.7.2.3 Incoming unsolicited AT-commands

The functions in table 4.10 are all request-response functions. A command is sent to the device, and a result is expected. Another type of command are commands sent by the device without being requested by the gateway. The syntax for this format is the same as a response from a regular AT-query command, see table 4.8. The *newRXData* function in the *CbtSerial* parse incoming commands and try to match them with known commands, and then uses the UDP-connection directly to send the result out on UDP.

Command	**Sent onto UDP...**
newSMS	together with an index of a received SMS.
ring	every third second when there is an incoming call.
number	together with the caller's number for an incoming call.
keypress	together with a key and the key state when the user presses or releases a key on his/hers device.
event	when an event has changed its status. See handsfree profile section 4.9 on page 36.
newDevice	when a new device has been found during a device search.

Table 4.11: Unsolicited commands

4.7.3 TimerClass

An instance of the *TimerClass* is created upon construction of the *CbtSerial* class. The *TimerClass* is simply a thread periodically making a call-back. The *CbtSerial* class uses this class to periodically send a dummy AT-command to the connected device, which is needed to prevent the connection from dropping out due to inactivity.

4.8 Headset profile class / headset service - CbtHeadset

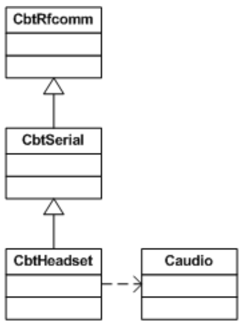


Figure 4.11: Headset profile class - CbtHeadset

The headset profile class, *CbtHeadset*, implements the Bluetooth headset profile (HSP), and provides the headset service onto UDP. The HSP relies on virtual serial ports and thereby extends the *CbtSerial* class. The profile itself does not define all functions available in the *CbtSerial* class, see table 4.10, but as the HSP uses a virtual serial port, its possible that these functions are accessible from the SPP-connection used by the HSP.
The *CbtHeadset* does not provides any new functionality except for the audio routing, but functions that are specific for the headset profile are listed in table 4.12. The audio routing routes audio from a device via the gateway out on UDP, and the other way around. More on how the Audio class is implemented is given in section 4.11.

Command	**Description**
hangup	Hangup an ongoing or incoming call.
answer	Answer an incoming call.
dial	Dial a number.

Table 4.12: Headset specific functions

4.8.1 Profile layout

The headset profile has two roles, to function as an audio gateway and a headset. The audio gateway is the gateway of the audio, both for input and output. This is typically a mobile phone or a smart phone. The headset is a device with one button for call handling, and a microphone and speaker for remote input and output of the audio in the gateway [23].

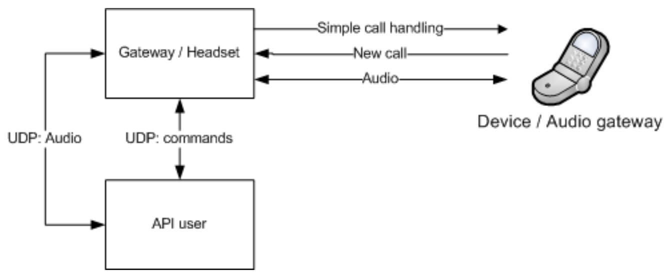


Figure 4.12: Headset profile overview

4.8.2 Profile initialization

After a successful connection to a handsfree audio gateway service, the connection has to be initialized. The initialization part for the HSP is just one step. After a successful initialization, the audio gateway listens for call handling commands from the headset, sends notification of new incoming call and creates an audio connection when a call is answered or placed. See figure 4.13.

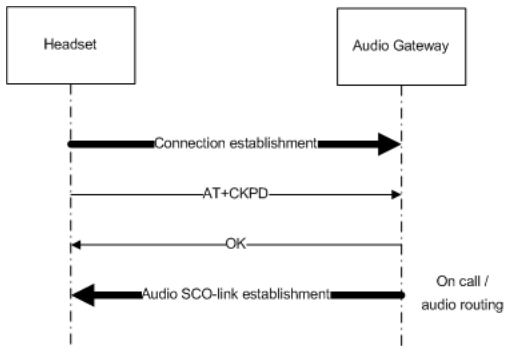


Figure 4.13: Headset profile initialization

4.9 Handsfree profile / handsfree service class - CbtHandsfree

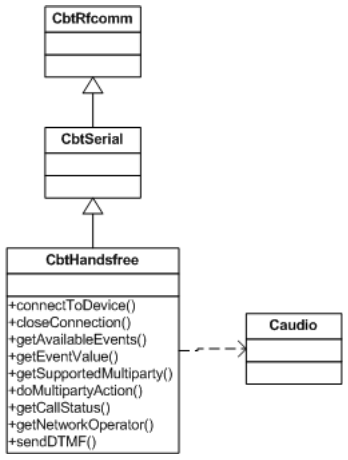


Figure 4.14: Handsfree profile class - CbtHandsfree

This class implements the Bluetooth handsfree profile (HFP). The same as for the HSP, see section 4.8, the HFP class extends the *CbtSerial* class and thus functions available in the *CbtSerial* class can be used in the handsfree service. Additional to the HSP, this profile supports and can handle calls with more than one party, like conference calls and three-way calls. The profile also supports events. A event is sent whenever a status holder in a device changes its value, where a status holder is one of the status holders in the list 4.13. Implemented events differs between devices.
Table 4.14 lists the specific HFP service functions both extended from the *CbtSerial* and new handfree only functions.

Name	**Description**
battchg	Battery charge level (0-5)
signal	Signal quality (0-5)
batterywarning	Battery warning (0-1)
chargerconnected	Charger connected (0-1)
service	Service availability (0-1) (Net contact status, 1 = Net contact)
sounder	Sounder activity (0-1) (Phone silent status, 1 = phone silent)
message	Message received (0-1)
call	Call in progress (0-1)
roam	Roaming indicator (0-1) (Home net status, 0 = Home Net)
smsfull	1: a short message memory storage in the MT has become full. 0: Memory locations are available
callsetup/call_setup	Bluetooth proprietary call set up status indicator. Possible values are as follows: 0) Not currently in call set up, 1) ongoing incoming call process, 2) ongoing outgoing call set up, 3) remote party alerted in an outgoing call
callheld	Indicates the status of any held calls on the audio gateway. 0 = No held calls. 1 = Call on hold.
vox	transmit activated by voice activity (0-1)

Table 4.13: Events

Command	**Description**
hangup	Hangup an ongoing or incoming call.
answer	Answer an incoming call.
dial	Dial a number.
getAvailableEvents	Get available device events.
getEventValue	Get the value of an event.
getSupportedMultiparty	Get available multiparty actions.
doMultipartyAction	Execute a multiparty action.
getCallStatus	Get a list of all ongoing/incoming/outgoing calls.
getNetworkOperator	Get the name of the network operator.
sendDTMF	Send DTMF tones during an ongoing call.

Table 4.14: Handsfree specific functions

4.9.1 Profile layout

Like the HSP, the handsfree profile has two roles, an audio gateway and a handsfree. The handsfree inputs and outputs the audio routed to and from the audio gateway where the audio gateway is routing the audio to and from the device [24].

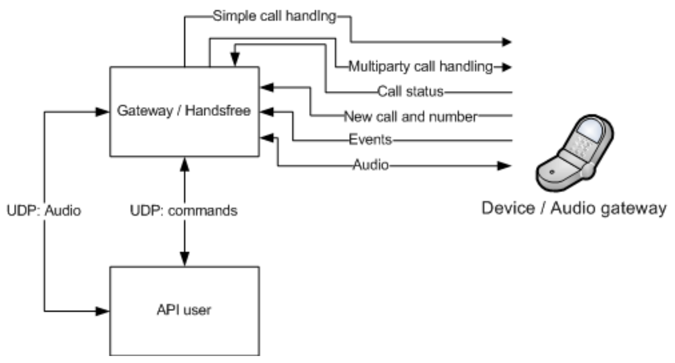


Figure 4.15: Handsfree profile overview

4.9.2 Profile initalization

The Handsfree profile has a five step initialization procedure which is mandatory before the connection can be used for call handling. The initialization includes specifying the capabilities of the handsfree to the audio gateway and vice versa, getting available events and the current value for the events, enabling event notification and getting the supported multi-party actions in the audio gateway. The connection is ready to be used when these steps are done. See figure 4.16.

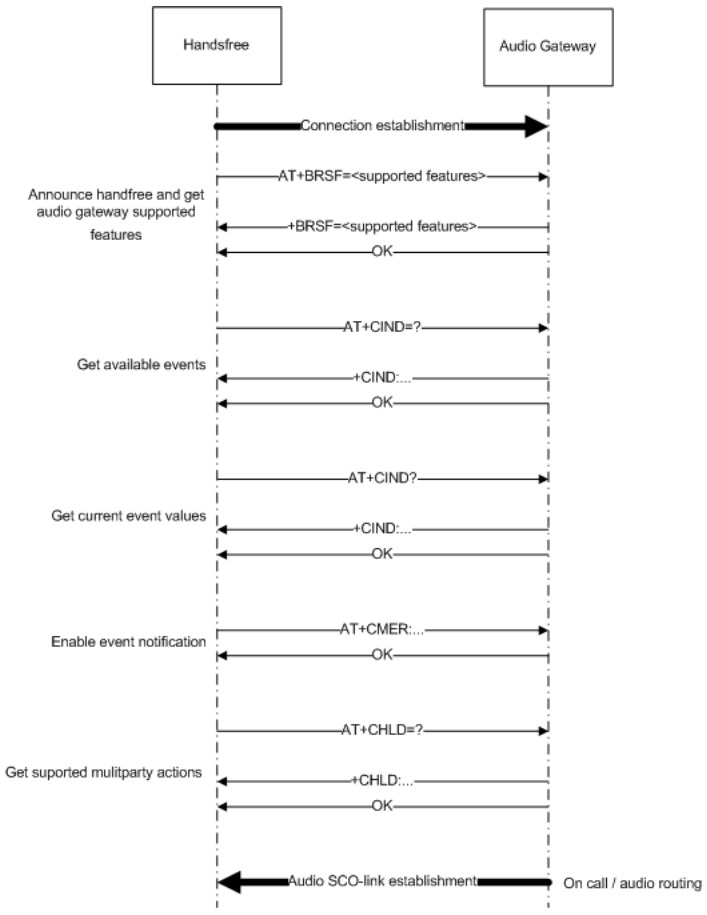

Handsfree
Audio Gateway
Connection establishment
AT+BRSF=<supported features>
Announce handfree and get audio gateway supported features
+BRSF=<supported features>
OK
AT+CIND=?
Get available events
+CIND:...
OK
AT+CIND?
Get current event values
+CIND:...
OK
AT+CMER:...
Enable event notification
OK
AT+CHLD=?
Get suported mulitparty actions
+CHLD:...
OK
Audio SCO-link establishment
On call / audio routing

Figure 4.16: Handsfree profile initialization

4.10 Internet Dial-up / dun service class - CbtDUN

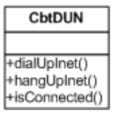


Figure 4.17: Internet Dial-up class - CbtDUN

The dun service class enables the access to the internet using a dial-up connection to a Bluetooth device acting as a modem. When an internet connection is created, the external applications uses the PC running the gateway as its internet gateway to get access to internet.

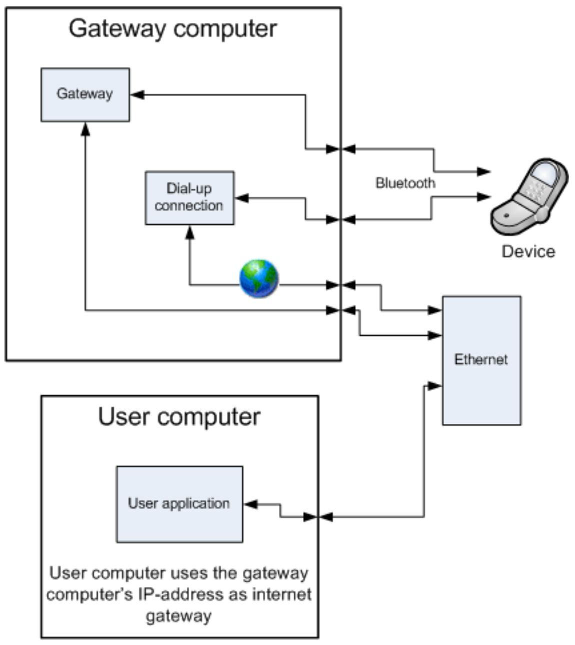


Figure 4.18: DUN overview

Due to limitations in the Broadcom SDK, the *CbtDUN* does not use any Bluetooth SDK APIs, but instead Windows API for establishing a dial-up connection. There are a couple of settings which have to be done in the Windows environment to enable DUN-support. The complete procedure is presented in section *Installation* in the user manual. A shorter presentation includes the following steps:

1. Install general Bluetooth modem drivers.

2. Add new dial-up connection to Windows, and name it exactly ”bluetooth”.

3. Set it to use the Bluetooth-modem as its modem.

4. Enable Internet Connection Sharing for the dial-up connection.

The Bluetooth-modem is a general modem for all Bluetooth devices, and thus each new Bluetooth device uses the same modem, with no need for a different modem or serial port. Instead the Bluetooth-modem points to a Bluetooth device and connects to this device and uses it as a modem uppon connection. The Widcomm/Broadcom driver stores information about which device the Bluetooth-modem should use in the Windows registry. The Windows registry is a directory which stores settings and options for the operating system.
The *dialUpInet* in this class is called to create an dial-up connection. This function first reads information about the device to be used as a modem, and then writes settings in the registry to point the Bluetooth-modem to this device. Native Windows API for dialing a dial-up connection is then called and an internet connection is established.

4.11 Audio class - Caudio

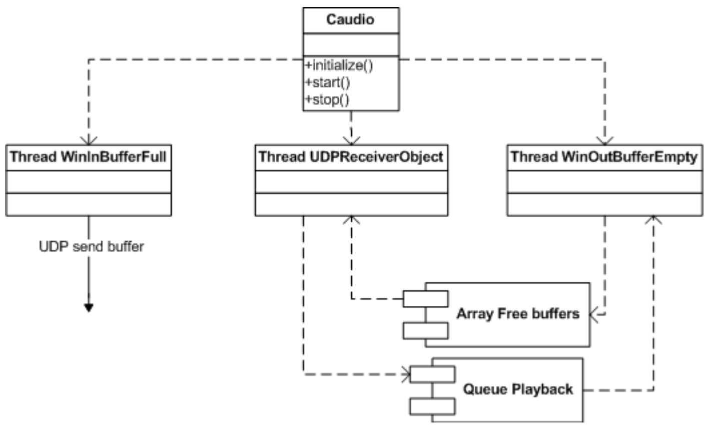


Figure 4.19: Audio streaming class - Caudio

The Caudio class is used to route audio from a Bluetooth device via the gateway and out on UDP, and vice versa. It reads and writes audio data using a local audio device, which in this case is the Bluetooth audio device, and sends and receives the same data using UDP where the other end of the UDP-connection does the same using its local audio device.

4.11.1 Windows WinIn and WinOut API

The native Windows API WinIn and WinOut is used to read and write data to an audio device. The constructor of the class searches among all local audio devices for an audio device with a device name containing the string "bluetooth", which is then opened for reading and writing.

4.11.1.1 WinIn - Reading/recording and sending audio

The WinIn API is used to read data from an audio device. The procedure involves the following two steps: 1) Adding an arbitrary number of buffers to the winIn internal buffer queue, and 2) starting the recording. WinIn uses the first buffer in the queue and starts recording/writing audio data to it. When the buffer is full, the buffer is removed from the queue and WinIn starts writing to the next free buffer. The full buffer is sent onto UDP and then placed in the queue again.

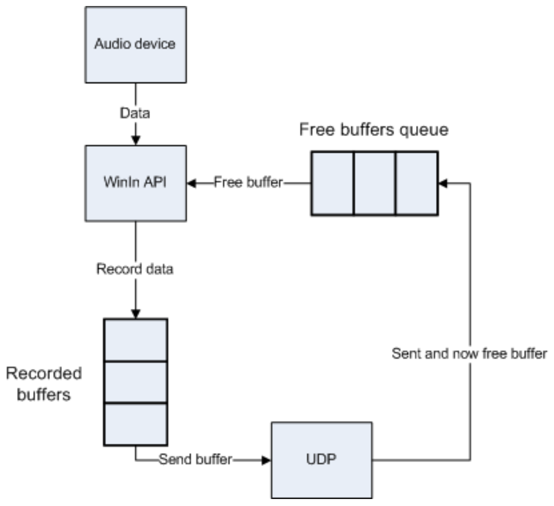


Figure 4.20: WinIn API buffer flow

4.11.1.2 WinOut - Receiving and writing/playing audio

The WinOut API is used to write data to an audio device. As same as for the WinIn, WinOut also has a internal queue, but WinOuts queue is a playback queue. Every new audio buffer packet received on UDP is added to the playback queue. When WinOut is done reading/playing a buffer, the API proceeds to the next buffer in queue.

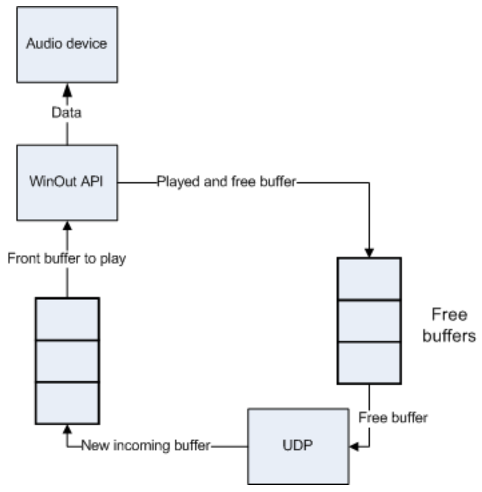


Figure 4.21: WinOut API buffer flow

4.11.2 Audio format

The audio is sampled using the standard PCM audio format with a sampling rate of 8k and 16 bits per sample and in mono. These settings are satisfying for streaming voice, but not for music.
Each packet sent onto UDP is raw audio data, and can be added directly to a WinOut playback queue, given that the WinOut uses the same Audio format settings.

Chapter 5

Testing

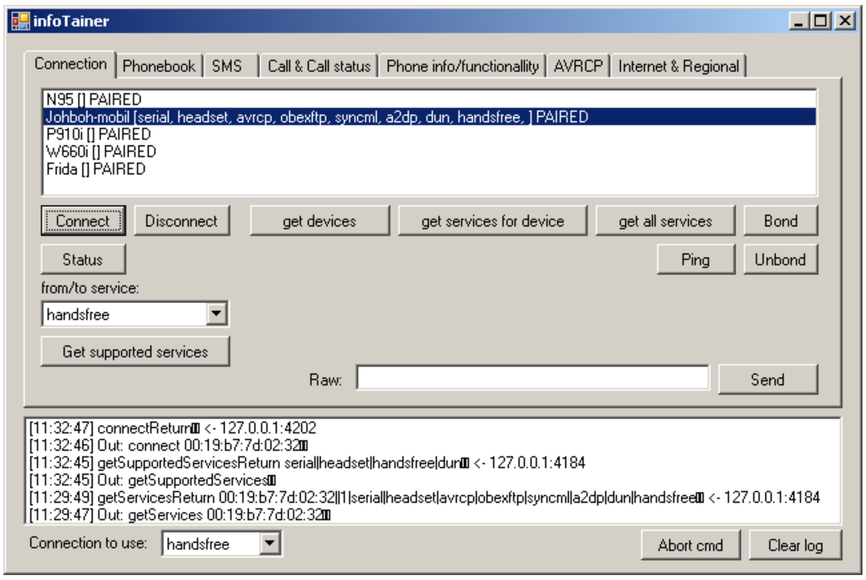
Throughout the development of the gateway the testing has been a continuous process, with a complete final testing. All functions in the user manual have been tested to see if they follow the expected behavior as specified in the manual.

5.1 Raw AT commands

Wherever it was possible to connect to a Bluetooth profile using The Windows HyperTerminal application, as it was with the serial, headset and handfree profile, HyperTerminal was used to send raw AT commands to a connected device. When sending raw AT commands directly to a device, one can observe the result for a command and the behavior of a device, which is helpful when implementing command parsers and formatting a command.

5.2 infoTainer - Infotainment Emulator

The infoTainer is a C# application I developed to emulate a potential infotainment system. It implements all available functions implemented in the gateway.



```
N95 [] PAIRED
Johboh-mobil [serial, headset, avrcp, obexftp, syncml, a2dp, dun, handsfree, ] PAIRED
P910i [] PAIRED
W660i [] PAIRED
Frida [] PAIRED
```

```
[11:32:47] connectReturn▯▯ <- 127.0.0.1:4202
[11:32:46] Out: connect 00:19:b7:7d:02:32▯▯
[11:32:45] getSupportedServicesReturn serial|headset|handsfree|dun▯▯ <- 127.0.0.1:4184
[11:32:45] Out: getSupportedServices▯▯
[11:29:49] getServicesReturn 00:19:b7:7d:02:32||1|serial|headset|avrcp|obexftp|syncml|a2dp|dun|handsfree▯▯ <- 127.0.0.1:4184
[11:29:47] Out: getServices 00:19:b7:7d:02:32▯▯
```

Figure 5.1: infoTainer - Infotainment Emulator

5.3 Tested devices

Table 5.1 on page 44 lists Bluetooth devices tested along with the functionality that works on each device.

Function	*Nokia 6233*	*SE P910i*	*SE W660i*	*SE P1i*
Find device	Y	Y	Y	Y
Find services	Y	Y	Y	Y
Bond	Y	Y	Y	Y
Unbond	Y	Y	Y	Y
Connect serial	Y	N	Y	N
Connect headset	Y	N	Y	Y
Connect handsfree	Y	Y(1)	Y	Y
Connect dun	Y	x	Y	Y
getBatteryStatus	Y	N	Y	Y
getSignalQuality	Y	N	N	Y
getAvailableStorages	Y	N	Y	Y
getListParams	Y	N	Y	Y
getList	Y	N	Y	Y
getEncoding	Y	Y	Y	Y
setRingVolume	N	N	Y	Y
getRingVolume	N	N	Y	Y
getValidRingVolume	N	N	Y	Y
setSilence	N	N	Y	Y
getSilence	N	N	Y	Y
setCatchKeypressNotifications	N	N	Y	Y
hangup	Y	Y	Y	Y
answer	Y	Y	Y	Y
dial	Y	Y	Y	Y
sendSms	Y	N	Y	Y
setCatchSmsNotifications	N	N	Y	Y
getAvailableSmsStorages	N	N	Y	Y
setSmsStorage	N	N	Y	Y
getSmsList	N	N	Y	Y
readSms	N	N	Y	Y
New SMS not.	N	N	Y	Y
New call not.	Y	Y	Y	Y
Incoming No. not.	Y	Y	Y	Y
Keypress not.	N	N	Y	Y
Route audio	Y	x	Y	Y
getAvailableEvents	Y	x	Y	Y
getEventValue	Y	x	Y	Y
getSupportedMultiparty	Y	x	Y	Y
doMultipartyAction	Y	x	x	Y
getCallStatus	Y	x	Y	Y
getNetworkOperator	Y	x	Y	Y
sendDTMF	Y	x	Y	Y

Table 5.1: Device vs functionality

(1) Disconnected after some time.

(x) Not tested. Coresponding profile not available.

Chapter 6

Results

A gateway communicating wirelessly using Bluetooth with nomadic devices has been developed. It provides an interface/API to a Bluetooth device on UDP and implements the Bluetooth serial, headset and handsfree profile. It supports device and service search, dial-up internet connection and non-profile specific functionality like sending and reading SMSes, reading the phone book and receiving battery status. Not implemented are music streaming (A2DP), media player control (AVRCP), Directory listing and file transfer (OBEXFtp) and synchronization (SYNC).

Chapter 7

Discussions, Conclusions and Future developments

7.1 Conclusions

The gateway works satisfyingly with respect to a couple of Bluetooth stack issues. The UDP-interface with its commands is consistent and straight forward to use. The command-response time is fast and depends only on the speed of the connected device. The bottleneck in the gateway is the Broadcom Bluetooth stack which is not as stable as one would like. Device searching can also take up to two minutes which may be a bad user experience for a vehicle driver integrating his device to the vehicle.
Functionality can be guaranteed with current and future nomadic devices implementing a Bluetooth profile, like the headset or handsfree profile. Device manufacturers ”must” follow the specification if they implement a profile. A number of functions like reading and sending SMSes and reading phone book entries are not defined in a profile, and it is up to the device manufacturer to decide if they want to implement them or not, and it is not possible to know exactly which functions a device supports until they have been tested on the actual device. This is not good since the answer to the question ”Will my phone work?” is ”It might.”.
The gateway is developed for conceptual projects only. When used in a bulk produced car, the Media Oriented Systems Transport (MOST) [35] networking standard commonly used for media transport in cars, is probably replacing UDP.

7.2 Problems during development

During the development of the gateway there has actually been only one major problem regarding the audio routing implementation. There where no examples on the internet and the Bluetooth stack documentation did not contain any information on how the audio data should actually be exchanged with the Bluetooth device. After numerous mails to the very taciturned and slow support, I eventually figured this out. The solution was to use the Windows WinIn and WinOut API, se section 4.11 at page 40. It did however delay the development since the audio routing is such a fundamental part in this thesis.

7.3 Improvements

7.3.1 Services

The DUN-service is not very nicely implemented, see section 4.10 at page 38. To establish a Dial-up connection the Windows registry must be modified and the service relies on a number of settings. This was the simplest and fastest solution. Implementing the PPP-protocol, which is used in a dial-up connection, and connecting to the dun profile on a device directly would be a nicer and more robust solution.

7.3.2 Bluetooth stack issues

The Broadcom Bluetooth stack is not 100% stable. Sometimes, after a period of usage, the stack needs to be rebooted. Also, it does not always work when routing audio. Sometimes the audio is quite crappy, and is sometimes routed in only one direction. This is the case both when using the gateway implementation, as well as the native Bluetooth profiles in Windows XP.
The stack also fails when doing service discovery and searching for the handsfree profile. The stack fails to find it, even tough it is present. Broadcom knows about this problem, but at the moment does not have any solution.

7.3.3 Non-implemented Bluetooth profiles/services

The Advanced Audio Distribution (A2DP), Audio/Video Remote Control (AVRCP), Synchronization (SYNC) and File transfer (OBEX/FTP) Bluetooth profiles are not implemented, but there are classes for them. The reason they are not implemented is primarily the lack of time, but also due to lack of documentation. Two documents, not available for free (each costing 3000SEK from `1394ta.org`), were required to know the full specification for the AVRCP profile.

7.3.4 Future Bluetooth and Bluetooth profiles

There are currently draft specification for an improved Bluetooth handsfree profile [21]. The new version includes push-to-talk and Voice over IP services, and it may be preferable to implement these functionalities, specially when network operators start to support the Push-to-talk [22] feature.
A new profile scheduled to be adopted by the beginning of the fourth quarter of 2008 is the Global Navigation Satellite System (GNSS) Profile [21]. The purpose of this profile is to specify how a Bluetooth enabled GPS and a Bluetooth device should communicate to exchange positioning data. This profile can be useful to make use of a built in GPS-receiver in a nomadic device for vehicle navigation.
The next generation Bluetooth, version 3.0 [36], is under development and will adopt the Ultra-Wideband (UWB) radio technology. This will increase the bandwidth from a current maximum of 3 Mbit/s to 480 Mbit/s, which is more than enough for streaming high-definition (HD) video over Bluetooth. This may be useful in a couple of years when nomadic devices can play HD-movies and there is a wide flat-screen in the backseat of every car.

List of Figures

2.1 System overview 12

3.1 BTW Stack Overview [19] 19

4.1 Gateway overview 22
4.2 Cmtapi class 24
4.3 UDP-receiver thread 27
4.4 Command processing thread 27
4.5 Device and service discovery class - CbtFinder 28
4.6 RFCOMM connection class - CbtRfcomm 29
4.7 CbtRfcomm/CbtL2cap connection function call 30
4.8 L2CAP connection class - CbtL2cap 30
4.9 Serial AT-command class - CbtSerial 31
4.10 General function procedure 32
4.11 Headset profile class - CbtHeadset 34
4.12 Headset profile overview 35
4.13 Headset profile initialization 35
4.14 Handsfree profile class - CbtHandsfree 36
4.15 Handsfree profile overview 37
4.16 Handsfree profile initialization 38
4.17 Internet Dial-up class - CbtDUN 38
4.18 DUN overview 39
4.19 Audio streaming class - Caudio 40
4.20 WinIn API buffer flow 41
4.21 WinOut API buffer flow 41

5.1 infoTainer - Infotainment Emulator 42

List of Tables

3.1 Fundamental required functionality 14
3.2 Preferred functionality 14
3.3 Optional functionality 14
3.4 Bluetooth versus WLAN bandwidth/range/power consumtion 17
3.5 Bluetooth profiles suitable for each service/function 18

4.1 Services 23
4.2 UDP port mapping 25
4.3 One or more arguments 25
4.4 Zero arguments 25
4.5 Command examples 26
4.6 Error command 26
4.7 Configuration variables 28
4.8 AT-command syntax 32
4.9 AT-command examples 32
4.10 Implemented functions 33
4.11 Unsolicited commands 34
4.12 Headset specific functions 35
4.13 Events 36
4.14 Handsfree specific functions 37

5.1 Device vs functionality 44

Bibliography

[1] pcmag.com encyclopedia, http://www.pcmag.com/encyclopedia_term/0,2542,t=nomadic+device&i=55973,00.asp

[2] Volvo Technology, http://www.volvo.com/group/global/en-gb/Volvo+Group/our+companies/volvotechnologycorporation/

[3] The Register, Bill Ray, July 24, 2007, http://www.theregister.co.uk/2007/07/24/first_wireless_usb/

[4] Aiello, Roberto; Batra, Anuj: ”Ultra Wideband Systems: Technologies and Applications”, chapter 9. Newnes, 2006.

[5] USB.org - Approved Class Specification Documents, http://www.usb.org/developers/devclass_docs#approved

[6] Dowla, F; Moring, J T.: ”Handbook of RF and Wireless Technologies”, page 9. Newnes, 2004.

[7] Nokia N95, http://www.forum.nokia.com/devices/N95

[8] Garg, Vijay K.: ”Wireless Communications Networking”, Chapter 21. Morgan Kaufmann, 2007.

[9] Broadcom WLAN module, http://www.broadcom.com/collateral/pb/4326-PB02-R.pdf

[10] Tech Blog, Abdul Aziz, April 21, 2004, http://tinyurl.com/ysx74b

[11] Carpenter, Tom: ”Wireless# Certification Official Study Guide”, Chapter 11. McGraw-Hill Osborne, 2006.

[12] Bluetooth.com, How Bluetooth technology works http://www.bluetooth.com/Bluetooth/Technology/Works/

[13] Bluetooth.com, Bluetooth enabled mobile phones, http://www.bluetooth.com/Bluetooth/Connect/Products/Product_Listing.htm?ProductCategory=17

[14] Bluetooth.com, Bluetooth enabled handheld devices, http://www.bluetooth.com/Bluetooth/Connect/Products/Product_Listing.htm?ProductCategory=10

[15] Bluetooth.com, Available Bluetooth profiles, http://bluetooth.com/Bluetooth/Learn/Works/Profiles_Overview.htm

[16] Bluetooth.com, Bluetooth Security Questions and answers, http://tinyurl.com/25d7k4

[17] Darkreading.com - Bluetooth Security Worse Than WiFi, Kelly Jackson Higgins, Januari 10, 2007 , http://www.darkreading.com/document.asp?doc_id=114424

[18] Garg, Vijay K.: ”Wireless Communications Networking”, Chapter 19.7. Morgan Kaufmann, 2007.

[19] Broadcom Bluetooth Stack, `http://www.broadcom.com/products/bluetooth_support.php?source=top`

[20] Broadcom Bluetooth Stack Programmer's guide, `http://www.broadcom.com/products/bluetooth_support.php?source=top`

[21] Bluetooth.org, Bluetooth profile enhanchments, `https://www.bluetooth.org/apps/content/?doc_id=47337`

[22] SizeIT AB, Push-to-talk, `http://www.pushtalk.se/`

[23] Bluetooth.com, Bluetooth Headset profile (HSP) specification v1.2, `http://tinyurl.com/257p2a`

[24] Bluetooth.com, Bluetooth Handsfree profile (HFP) specification v1.5, `http://tinyurl.com/yow2ss`

[25] Experiencing This Mysterious Bluetooth Stack Programming , Alex Gusev, October 13, 2006, `http://www.developer.com/ws/other/print.php/3637741`

[26] Haines, Russ: "Digital Audio ", Page 313. The Coriolis Group, LLC, 2001.

[27] Taylor, Jim: "Everything You Ever Wanted to Know about DVD", Chapter 1. McGraw-Hill Professional Publishing, 2004.

[28] Blu-ray vs. hd-dvd, Rich Teer, August 23, 2006, `http://richteer.blogspot.com/2006/08/blu-ray-disc-vs-hd-dvd.html`

[29] Kammer, David; McNutt, Gordon; Senese, Brian; Bray, Jennifer: "Bluetooth Application Developer's Guide: The Short Range Interconnect Solution", Chapter 9. Syngress, 2002.

[30] AT Command Set For Nokia CDMA Products, March 14, 2005, `http://tinyurl.com/23eg79`

[31] AT Command Set For Nokia GSM And WCDMA Products, July 1, 2005, `http://tinyurl.com/3yaxww`

[32] AT Command Set For Nokia GSM Products, 2000, `http://tinyurl.com/yufhpr`

[33] AT Command for Sony Ericsson phones, December 2007, `http://developer.sonyericsson.com/getDocument.do?docId=65054`

[34] GSM 07.07 AT command set for GSM Mobile Equipment, Devember 1999, `http://www.etsi.org`

[35] MOST Cooperation, `http://www.mostcooperation.com/home/index.html`

[36] Ryan M. Steele, Introduction to Bluetooth, `http://www.crutchfieldadvisor.com/ISEO-rgbtcspd/learningcenter/home/bluetooth.html?page=3`