Masterarbeit

Design and Implementation of a Demonstration Software for NFC

Gerald Hollweger

Institut für Technische Informatik
Technische Universität Graz

Graz University of Technology

Begutachter: Ass.-Prof. Dipl.-Ing. Dr. techn. Christian Steger
Betreuer: Ass.-Prof. Dipl.-Ing. Dr. techn. Christian Steger

Graz, im Mai 2013

Danksagung

Mein besonderer Dank gilt meinen Eltern Gertraud und Hermann für die außerordentliche Unterstützung während meiner gesamten Ausbildungszeit.

Weiters möchte ich mich auch bei Herrn Ass.-Prof. Dipl.-Ing. Dr. techn. Christian Steger für sein Engagement bedanken, der mich in zeitkritischen Phasen selbst in seiner Freizeit, sowohl am späten Abend als auch am Wochenende, unterstützt hat. Ferner möchte ich mich bei Herrn DI Dr. techn. Martin Schmid bedanken.

Für die geduldige Unterstützung möchte ich auch meinem Betreuer in der Firma NXP (ehemals Philips Semiconductors) in Gratkorn, Herrn DI (FH) Markus Harnisch recht herzlich danken. Mein Dank gilt zudem Herrn DI Andreas Mühlberger, der mir die Möglichkeit einer Firmen-Arbeit geboten hat, durch die ich viel wertvolle Praxis-Erfahrung sammeln konnte.

Mein Dank gilt auch den Mitarbeitern der TU Graz, die vorwiegend eine sehr nette Ausbildungsatmosphäre schaffen. Stellvertretend für die vielen zuvorkommenden Personen in den unzähligen Serviceeinrichtungen der TU Graz möchte ich Frau Astrid Brodtrager und Herrn Robert Binder anführen.

Graz, im Mai 2013 GERALD HOLLWEGER

Legal notes

Kurzfassung

Aufgrund des großen und wachsenden Marktes für drahtlose Kommunikation werden immer neue Technologien vorgestellt. Um die Verbreitung und Akzeptanz eines Standards zu fördern, setzen viele Firmen – darunter auch IC-Hersteller – auf offene Standards. Diese damit einhergehende Öffnung des Marktes für Mitbewerber und die Schaffung einer Konkurrenzsituation ist oft ein Erfolgsgeheimnis für die Durchsetzung eines Standards am Markt. Um sich eine führende Marktposition zu sichern, ist es für die Hersteller daher wichtig, ein neues Produkt schnell präsentieren und den Kunden vorstellen zu können.

Near Field Communication (NFC) ist ein auch mit vielen Smart Card Systemen kompatibler kontaktloser Übertragungsstandard, der nicht zuletzt durch den Smartphone-Boom immer beliebter wird.

Ziel dieser Arbeit ist, eine Software mit grafischer Benutzeroberfläche zu erstellen, die es einerseits Kunden und Produktmanagern erlaubt, einen NFC IC auf einer Demonstrationsplatine zu erproben und andererseits Entwicklern die Möglichkeit zu geben, den möglichst vollen Funktionsumfang mit wenig zusätzlichen Aufwand für die Entwicklung einsetzen zu können.

Im Zuge der Recherchen wurden andere Evaluierungskits untersucht, weiters sollte auch auf Benutzerfreundlichkeit der Software Wert gelegt werden.

Abstract

Due to the large and still growing market for wireless communication, new technologies are presented continuously. To support pervasiveness and acceptance of such a technology, a lot of companies – including semiconductor manufacturers – develop open standards. The accompanying opening of the market for competitors is sometimes the key to establishment in the market. To assure a leading position in the market, it is very important for a manufacturer to present a product very early and introduce it to potential customers.

Near Field Communication (NFC) is a contactless communication standard, which is compatible with several smart card systems. It is becoming more and more popular, thanks in part to the boom of smartphones.

The aim of this thesis is to develop a software with a graphical user interface, that on the one hand allows a customers and product managers to evaluate a NFC IC on a demonstration board and, on the other hand, allows engineers to use almost the full functionality without investing a great deal of time and effort in development.

In the course of investigation other evaluation kits were analysed, furthermore, focus on the usability of the software was also important.

Contents

List of Figures

List of Tables

List of Abbreviations

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange, is a character encoding based on the English alphabet.
BAL	Bus Abstraction Layer
BFL	(NXP) Basic Function Library (cp. RCL)
BRAN	Broadband Radio Access Networks
CAD	Computer Aided Design *or* Card Acceptance Device
CCITT	International Telephone and Telegraph Consultative Committee (from the French name “Comité consultatif international téléphonique et télégraphique”), see ITU-T
CDMA	Code Division Multiple Access
CEPT	Conférence Européenne des Administrations des Postes et des Télécommunications [13]
CHM	Compiled HTML Help, also Compressed HTML Help or Compiled Help Module(s)
CWUSB	Certified Wireless USB
DAL	Driver Abstraction Layer Driver
DECT	Digital Enhanced (formerly European) Cordless Telecommunications
DFSS	Distributed Frequency Spread Spectrum
DID	Device ID
DIDi	Initiator Device ID
DIDt	Target Device ID
DoS	Denial of Service
DSSS	Direct-Sequence Spread Spectrum
EAN	International Article Number (formerly European Article Number)
EAS	Electronic Article Surveillance (Electronic Anti-theft System)
EIRP	Equivalent Isotropically Radiated Power/Effective Isotropic Radiated Power
ERP	Effective Radiated Power/Equivalent Radiated Power
ETSI	European Telecommunications Standards Institute [25]
EULA	End User License Agreement
FDD	Frequency Division Duplex
FHSS	Frequency-Hopping Spread Spectrum
GPL	GNU General Public License [34]
GUI	Graphical User Interface
HF	High Frequency
HAL	Hardware Abstraction Layer
HCI	Human-Computer Interaction/Interface
HIPERLAN	High Performance Radio LAN
HIPERMAN	High Performance Radio MAN
HLL	High Level Language
HTML	Hypertext Markup Language

S^2C	SigIn-SigOut-Connection, a general purpose interface between NFC and a secure IC
IEEE	Institute of Electrical and Electronics Engineers [58]
IC	Integrated Circuit
ICC	Integrated Circuit Card
ID	Identification number
ISM	Industrial, Scientific and Medical (radio band)
ISO	International Organization for Standardization [48]
ITU	International Telecommunication Union [68]
ITU-T	ITU Telecommunication Standardization Sector; prior to 1992 it was known as the CCITT
LAN	Local Area Network
LSB	Least Significant Bit/Byte
MAC	Medium Access Control Layer
MAN	Metropolitan Area Networks
MNO	Mobile Network Operator
MSB	Most Significant Bit/Byte
NAD	Node Address
NDEF	NFC Data Exchange Format
NFC	Near Field Communication
NFCID	NFC Identifier
NFCIP	Near Field Communication – Interface and Protocol
OFDM	Orthogonal Frequency Division Multiplexing
OFDMA	Orthogonal Frequency Division Multiple Access
OS	Operating System
OSI model	Open Systems Interconnection (Basic Reference) Model
OTA	Over-the-Air
PAL	Protocol Adaptation Layer
PAN	Personal Area Network
PCB	Printed Circuit Board
PCD	Proximity Coupling Device (ISO/IEC 14443)
PDF	Portable Document Format
PDU	Protocol Data Unit
PHY	Physical Layer
PICC	Proximity Integrated Circuit Card (ISO/IEC 14443)
POS	point-of-sale
QoS	Quality of Service
RCL	(NXP) RF Component Library (former Reader Component Library, cp. BFL)
RF	Radio Frequency
RFID	Radio Frequency Identification
RTD	Record Type Definition
SAM	Secure Access Module
SAR	Specific Absorption Rate
SIG	(Bluetooth) Special Interest Group
SIM	Subscriber Identity Module
SOFDMA	Scalable OFDMA
SoHo	Small Office, Home Office
SWP	Single Wire Protocol, standard for an interface between a NFC chipset and a SIM
TDD	Time Division Duplex
TDMA	Time Division Multiple Access

TTA	Telecommunications Technology Association (of Korea) [50]
TTC	Telecommunications Technology Committee (Japan) [49]
UPC	Universal Product Code
VM	virtual machine
WECA	See Wi-Fi
WiBro	Wireless Broadband
Wi-Fi	Brand name owned by the Wi-Fi Alliance [3], primary named “WECA” (Wireless Ethernet Compatibility Alliance)
WirelessMAN	See WiMAX
WiMAX	Worldwide Interoperability for Microwave Access
WLAN	Wireless LAN
WPAN	Wireless Personal Area Network

For further NFC specific abbreviations see also ISO/IEC 18092.

Chapter 1

Introduction

Due to the fast development of semiconductor technology and decreasing prices the number of high-performance portable devices sold is rising very fast. This establishes a new range of applications. Most of these applications need to communicate with their environment, so there is also a need for communication capabilities. Commonly, cables are an unacceptable constraint, hence a lot of data transfers are carried out via wireless communication channels.

There is already a broad range of wireless communication standards like WLAN, Bluetooth, ZigBee etc. but usually each of these technologies fits better for a particular application. E.g. wireless LAN has a high operating range and data rate but consumes quite a lot of energy, so it is ineligible for some mobile devices, where the battery capacity is limited. There is also a huge difference in the setup time of a communication, the protocol overhead, security considerations and much more.

Lots of new technologies and protocols are developed for the growing market, both open and proprietary. Some of them are never used in mass products, others disappear after a short time. Establishment on the market can be influenced by several reasons. Backwards compatibility to existing infrastructure is an advantage of introducing a new product to the market and also allows a gradual migration. Free, open standards can also contribute to acceptance because the dependency on a single company can be reduced. But also non-technical facts can be reasons for acceptance, like marketing. Or simply a powerful company equipping a product with a large market share with the technology can be very attractive for other companies who then benefit from the situation. Furthermore licensing issues, pending patents and regulatory restrictions influence the potential of widespread, worldwide usage.

Contactless cards are already commonly used for some applications, for instance ticketing in public transport. The advantages of this technology are reliability against environmental influences (no corrosion of electric contacts), security features (much better than magnetic stripes), the fact they can be reused and a low price.

Near Field Communication (NFC) is a relatively young wireless short-range communication standard based on these advantages – evolved from a combination of contactless identification and interconnection technologies, jointly developed by Royal Philips Electronics (now NXP Semiconductors [27]) and Sony Corporation [18]. It is compatible with a large, already existing infrastructure for public transport, access control, payment and lots of other applications.

The aim of the thesis was to develop a easy-to-use Windows application with an intuitional graphical user interface, to demonstrate the potential of the NXP PN511 (code name *Joiner*) IC in combination with an evaluation board. It should be easy to use for simple demonstrations in marketing but also have the potential to be used by engineers in development – which means it should have a good usability for the different users.

This thesis has been conducted by the Institute of Technical Informatics, Graz University of Technology in cooperation with NXP Semiconductors Austria GmbH Styria (former Philips Semiconductors).

1.1 Motivation

Semiconductor manufacturers often provide hardware and a software library for easy integration in products. But before a new technology can break into a market, it will be evaluated and compared with other technologies to work out their advantages and disadvantages. For demonstration purposes and customer trials an easy-to-use tool with a graphical front-end is very useful. Using this, the new technology can be evaluated without writing code or the need for a software development environment. For open standards, providing such a tool can be an advantage over a competitor, reduce the time to market and help to achieve a leading position in the market.

The target of the thesis was to implement a stand-alone tool with a graphical front-end, which integrates the software library. This tool should show the functionality of the NFC IC and reflect the interface of the library. Furthermore it should not restrict the functionality of the IC too much, in order that it can also be used by engineers for development and debugging. Experience of using lots of computer software, awareness of their problems and the attempt to also place value on usability rather than only functionality should help to create better software.

1.2 Outline

Chapter 2 gives an overview of existing wireless communication standards, Smart Cards, NFC, introduces some existing and possible applications and points out security and privacy issues. It also gives a background to user interface design and presents some other evaluation kits.

Chapter 3 shows the design and the software architecture and Chapter 4 deals with the actual implementation of the application.

Chapter 5 introduces how to test applications with GUIs and how to develop software which can be tested easily.

Chapter 6 shows how to set up the evaluation kit and the software and provides a quick-start guide with some examples.

Chapter 7 summarises the outcome and gives an overview about possible future work.

Chapter 2

Theory and related work

This chapter deals with the basics of NFC and other wireless communication technologies, advantages, disadvantages and its applications. Later the challenges of designing a graphical user interface will be examined. Finally, some other evaluation kits are introduced.

2.1 Wireless communication standards

2.1.1 Introduction

The communication needs of mobile devices and their users have grown exponentially because of the proliferation of such communication and consumer electronics devices.

A lot of different wireless communication technologies have been developed and improved in the last years. Those technologies have a lot of different properties, advantages and disadvantages. Depending on the application, a particular standard may fit better than another. Some properties of different technologies are:

- Information carrier: light, radio waves, magnetic field
- Range: low – high
- Data rate
- Transmission synchronisation: synchronous/asynchronous
- Operating frequency
- Bandwidth
- Modulation scheme
- Power consumption
- Antenna size
- Network type: ad-hoc, peer-to-peer, infrastructure mode
- Protocols

- Reliability
- Complexity
- Costs

2.1.2 Explanation of some technical terms

Ad-hoc network: is a network connection method which is most often associated with wireless devices. The connection is established for the duration of one session and requires no base station.

Peer-to-peer network: uses diverse connectivity between participants in a network rather than conventional centralised topologies.

Infrastructure network: a managed wireless network, in which a special node known as an "access point" manages communication among other nodes.

Bridge: an electronic device used to connect two computer or telephone network segments.

Repeater: is an electronic device that receives a signal and retransmits it amplified. So the range of the system is extended transparent for the user.

EMI: electromagnetic interference/influence, also called **RFI** (radio frequency interference). This treats the interrelationship of electronic devices – emission of electromagnetic radiation as well as sensitivity against those radiation.

EIRP: equivalent isotropically radiated power or effective isotropic radiated power is the product of the power fed into the antenna and the antenna gain, relatively to the theoretical isotropic antenna. This value helps to compare different antennae.

2.1.3 Frequency allocation, license provisions, regulations

To avoid unintentional interferences of electronic radio systems, prevent disturbances and guarantee operation of other systems, some rules need to be observed.

Examples of what can be prescribed for particular applications:

- Allowed frequency band
- Channel bandwidth
- Frequency tolerance
- Maximum frequency shift (for angle modulation)
- Limits for spurious radiation
- Maximum duty cycle. This is the proportion of time during which a component, device, or system is operated, related to a defined time.
- Maximum duration of a transmission cycle

- LBT (Listen Before Talk): Transmitter are required to monitor a channel before transmitting to avoid collisions
- Maximum ERP
- Maximum EIRP
- Maximum spectral power density, with or without using spread spectrum techniques
- Maximum transmitter output power (carrier power)
- Maximum magnetic field strength
- Modulation scheme
- Restriction for antennae (physical dimensions, bendable etc.)

Regulation authorities and standardisation organisations

ITU, the International Telecommunication Union [68], is an international organisation established to standardise and regulate international radio and telecommunications. Its main tasks include standardisation, allocation of the radio spectrum, and organising interconnection arrangements between different countries.

ETSI, the European Telecommunications Standards Institute [25] is an independent, non-profit standardisation organisation in the European telecommunications industry, with worldwide projection.

CEPT (Conférence Européenne des Administrations des Postes et des Télécommunications), the European Conference of Postal and Telecommunications Administrations [13] is an umbrella organisation for cooperation of the regulation authorities in Europe.

FCC, the Federal Communications Commission is an independent United States government agency, directly responsible to Congress. The FCC was established by the Communications Act of 1934 as the successor to the Federal Radio Commission (FRC) and is charged with regulating interstate and international communications by radio, television, wire, satellite and cable. The FCC's jurisdiction covers the 50 states, the District of Columbia, and U.S. possessions.

Health concerns

Due to omnipresence of electromagnetic radiation in our everyday life the term "Electromagnetic Pollution" has been coined. A lot of research has been done in this area, but even today it is still not certain whether all influences have been identified.

One known effect is the warming of tissue by microwave exposition. Apart from heat exhaustion no further risks acknowledged. For reasons of precaution some threshold values are defined. For instance the Specific Absorption Rate (SAR) for mobile phones. The SAR is a measure of how much radio frequency energy is absorbed by the body when exposed to electromagnetic field. Its unit is W/kg, averaged over a defined mass of tissue.

Other effects are often not confirmed and a lot of studies are not reliable. But due to proliferation of radio wave exposition and the fact long-term risks cannot be eliminated it is important to research this topic.

Lots of other regulations do not exist because of health concerns rather than electromagnetic interference with other devices.

2.1.4 Radio Frequency Identification (RFID)

Radio frequency identification is a technology for contactless reading from a so-called transponder or tag with the aid of a reader (resp. transceiver). RFID tags can be classified into three general types: *Passive*, *semi-passive* (also known as battery-assisted), or *active*.

Passive transponders have no own power source, they are powered by the reader's RF field. The transmission of the information is performed by load modulation of the RF carrier signal.

Active transponders have an own power source and can so communicate over a higher distance. Also a IC with more power-consuming computing power can be used.

Semi-passive transponders have an own power source for the integrated circuit but the response signal is transmitted in the same way as passive transponders do.

Depending on the application, various designs are available. From simple, small transponders for identification purposes, which only returns a unique serial number; more sophisticated ones for smart cards, where data can be stored and processed; to active transponder modules in aircraft which can, for instance, return the identification and details about the current heading.

There are various applications for this technology, for instance

- Time registration and access systems
- Ticketing and payment systems
- Identity cards
- Libraries
- Protection against forgeries
- Electronic article surveillance to prevent shoplifting
- Storing keys or passwords (these appliances are sometimes also called tokens)
- Car immobilizers
- Tracking items in automated warehouses, airports and so on
- Labelling animals
- Toll collection systems
- Identifying aircraft
- Telemetry
- ...

There are a lot of advantages of this system compared with systems like barcodes. For instance, every transponder can hold an unique ID, which can be assigned to a product serial number. Unlike a barcode, where the number of coded digits are limited on a certain dimension. Furthermore also information can be stored in the transponder, e.g. the date of sale of a product, which is very convenient for warranty handling. But there are also some trade-offs: This technology is more expensive (especially relevant for labelling cheap products in stores), it cannot be easily printed on a product wrapping like barcodes, RF transmission can be a problem with metal covers. So, despite the great advantages, another technology can be more eligible, depending on the application.

For further information about radio frequency identification, see the RFID handbook [26].

2.1.5 Wireless Local Area Network (WLAN)

Wireless LAN (also written as W-LAN, WLAN) are standardised by IEEE 802.11/b/g/n and operates in the globally available, license-free 2.4 GHz ISM band, IEEE 802.11a/h/n between 5.15–5.725 GHz.

With data rates of 2–300 Mbps and a range of approximately 30–100 m it is ideal for connecting computer equipment and peripherals.

To improve interoperability of wireless local area network products of several manufacturers based on the IEEE 802.11 standards, six industry leaders came together on 3rd August 1999 to form a global, non-profit organization, called “WECA” (Wireless Ethernet Compatibility Alliance), with the goal of driving the adoption of a single worldwide-accepted standard for high-speed wireless local area networking. On 1st October 2002 the organization was renamed to the **Wi-Fi Alliance** [3]. Today the alliance has more than 300 members from more than 20 countries.

Figure 2.1: Wi-Fi certified logo

The **Wireless LAN Association** [7] was founded as a non-profit educational trade association, comprised of the thought leaders and technology innovators in the local area wireless technology industry. Through the vast knowledge and experience of sponsor and affiliate members, WLANA provides a clearinghouse of information about wireless local area applications, issues and trends and serves as a resource to customers and prospects of wireless local area products and wireless personal area products and to industry press and analysts. It was taken over by **CWNP, Inc.** [62], founded in 1999, the IT industry standard for vendor neutral enterprise Wi-Fi certification and training, focused on 802.11 wireless networking technologies.

2.1.6 High Performance Radio Local Area Network (HIPERLAN)

HIPERLAN is a Wireless LAN standard. It is an European alternative for the IEEE 802.11 standards. It is defined by the European Telecommunications Standards Institute by the BRAN project (Broadband Radio Access Networks) [24].

2.1.7 IEEE 802.16 WirelessMAN

The IEEE 802.16 Working Group on Broadband Wireless Access Standards develops standards and recommended practices to support the development and deployment of broadband Wireless Metropolitan Area Networks (MANs) – large computer networks, optimised for a larger geographical area than is a LAN, ranging from several blocks of buildings to entire cities. IEEE 802.16 is a unit of the IEEE 802 LAN/MAN Standards Committee [43], the premier transnational forum for wireless networking standardization.

The purpose of the WiMAX Forum [33] is to promote the IEEE 802.16 wireless networking standard and interoperable standards by, among other things: (1) encouraging manufacturers of broadband wireless access products to achieve a high degree of interoperability among all products employing the standard; and (2) promoting through a number of means the widespread adoption and use of products employing the IEEE 802.16 standard. WiMAX is sometimes used as synonym for WirelessMAN.

WirelessMAN is a telecommunications technology aimed at providing wireless data over long distances in a variety of ways, from point-to-point links (IEEE 802.16-2004 Standard for Local and metropolitan area networks Part 16: Air Interface for Fixed Broadband Wireless Access Systems; unofficially sometimes called IEEE 802.16d) to full mobile cellular type access (IEEE 802.16e-2005 Mobile WirelessMAN Standard for Local and metropolitan area networks Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems Amendment for Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands).

A lot of attention was drawn to high data rates and a very short latency to improve Quality of Service (QoS). In contrast to 802.11, the 802.16 MAC uses a scheduling algorithm: the base station acts as master and allocates an access slot for the subscriber.

WiBro (Wireless Broadband) [71] is a quite similar development of the South Korean telecommunications industry, phase 1 was standardised by the Korean Telecommunications Technology Association [50] by the end of 2004. In November 2004, the compatibility of WiBro to WiMAX was agreed by representative of the companies Intel and LG Electronics. By the end of 2005, WiBro was standardised as IEEE 802.16e, mobile WiMAX, by the ITU.

2.1.8 High Performance Radio Metropolitan Area Network (HIPERMAN)

HIPERMAN stands for High Performance Radio Metropolitan Area Network and is a standard created by the European Telecommunications Standards Institute Broadband Radio Access Networks (BRAN) group [24] to provide a wireless network communication in the 2–11 GHz bands. HIPERMAN is an alternative to WiMAX/WiBro.

2.1.9 Bluetooth

Bluetooth is an in the 1990s originally from Ericsson [23] developed industry standard.

In 1994, Ericsson Mobile Communications launched an initiative to study low-power, low-cost radio interfaces between mobile phones and their accessories. A radio-access solution was sought because it would eliminate the need for cables with lots of different connectors and overcome line-of-sight restrictions. It was also recognised that a low-cost solution would usher in wireless connectivity for a multitude of new applications and give rise to a host of associated components and devices.

Bluetooth was envisioned to be an open, global standard. To achieve a critical mass and to promote a joint worldwide standard, Ericsson approached IBM, Intel, Nokia and Toshiba with its concept. In May 1998, the Bluetooth Special Interest Group (SIG) was announced.

The name Bluetooth is a hommage to Harald I. Bluetooth Gormson (Danish: Harald Blåtand, Old Norse: Haraldr blátönn, Norwegian: Harald Blåtann), King of Denmark and Norway in the late tenth century. He is known for his unification of previously warring tribes from Denmark (including now Swedish Scania, where the Bluetooth technology was invented) and Norway. Bluetooth likewise was intended to unify different technologies of different manufacturers, such as computers and mobile phones. The name may have been inspired less by the historical Harald Blåtand than the loose interpretation of him in 'The Long Ships' by Frans Gunnar Bengtsson, a Swedish Viking-inspired novel, where he is referred to as Harald Bluetooth. The Bluetooth logo merges the old Nordic runes analogous to the modern Latin H and B: Long-branch hagall and bjarkan from the Younger Futhark runes forming a bind rune[1].

Bluetooth®

Figure 2.2: The Bluetooth logo

The Bluetooth Special Interest Group (SIG) [65] is the body that oversees the development of Bluetooth standards and the licensing of the Bluetooth technologies and trademarks to manufacturers. Founded in 1998, it is a privately held trade association headquartered in Bellevue, Washington, USA with Michael Foley presently its executive director. Additional offices are located in Malmö, Sweden and Hong Kong. It is comprised of more than 9,000 member companies and a small staff.

The IEEE Project 802.15.1 has derived a Wireless Personal Area Network standard based on the Bluetooth v1.1 Foundation Specifications.

2.1.10 Wibree

Wibree [32] is a new interoperable radio technology for small devices. It can be built into products such as watches, wireless keyboards, gaming and sports sensors, which can then connect to host devices such as mobile phones and personal computers. It is essential the missing link between small devices and mobile devices/personal computers.

[1]In writing and typography a *ligature* occurs where two or more letter-forms are joined as a single glyph.

The name for the new standard is an amalgam of “Wi” for wireless, and “bree” from an Old English word for crossroads, or a place where two things come together.

Wibree is the first open technology offering connectivity between mobile devices or personal computers, and small, button cell battery power devices. By extending the role mobile devices can play in consumers’ lives, this technology increases the growth potential in these market segments.

Wibree and Bluetooth technology are complementary technologies. Bluetooth technology is well-suited for streaming and data-intensive applications such as file transfer and Wibree is designed for applications where ultra low power consumption, small size and low cost are the critical requirements.

Wibree has two important implementation alternatives: Stand-alone, which uses only Wibree technology and dual-mode, which uses a combination of Bluetooth technology and Wibree technology.

Developed by Nokia Research Center [12] since 2001, adapted from the Bluetooth standard, which would provide lower power usage and price, while minimizing difference between Bluetooth and the new technology. The results were published in 2004 using the name Bluetooth Low End Extension. After further development with partners, the technology was released to public in October 2006 with brand name Wibree. After negotiations with Bluetooth SIG members, in June 2007, an agreement was reached to include Wibree in future Bluetooth specification as an ultra low power Bluetooth technology.

On 12th June 2007 Bluetooth SIG announced, that the Wibree specification will become part of the Bluetooth specification as an ultra low power Bluetooth technology. April 2009 the new protocol architecture “Bluetooth low energy” [64] was presented in Tokyo.

2.1.11 ZigBee

The ZigBee Alliance [5] is developing a very low-cost, very low power consumption, two-way, wireless communications standard. The Physical (PHY) and the Medium Access Control (MAC) Layer is based on IEEE 802.15.4.

IEEE 802.15 TG4 [44] features:

- Data rates of 250 kbps, 40 kbps, and 20 kbps
- Two addressing modes; 16 bit short and 64 bit IEEE addressing
- Support for critical latency devices, such as joysticks
- CSMA-CA channel access
- Automatic network establishment by the coordinator
- Fully handshaked protocol for transfer reliability
- Power management to ensure low power consumption
- 16 channels in the 2.45 GHz ISM band, 10 channels in the 915 MHz and one channel in the 868 MHz band

ZigBee uses mesh networking. In a mesh network, each device can find the nearest available path along which to send data. The zigzagging data path is similar to the way bees communicate the location of the latest flower buffet to the other bees in the hive, and it's where ZigBee gets its name.

2.1.12 Wireless USB

Certified Wireless USB from the USB-IF [30] is the wireless extension to USB that combines the speed and security of wired technology with the advantages of wireless technology. Certified Wireless USB will support robust high-speed wireless connectivity by utilizing the common WiMedia MB-OFDM Ultra-wideband (UWB) radio platform as developed by the WiMedia Alliance.

2.1.13 Wireless FireWire

On 10th May 2004 the "1394 Trade Association" [6] approved the new Protocol Adaptation Layer (PAL) for IEEE 1394 over IEEE 802.15.3. The PAL lies in-between the MAC layer of IEEE 802.15.3 and the application layer of FireWire.The connection management schemes, data formats and time synchronization procedures can be used as usual.

Wireless FireWire is being integrated into the WiMedia Alliance's WiMedia Ultra-Wideband (UWB) standard.

FireWire is Apple Inc.'s brand name for the IEEE 1394 interface. It is also known as i.LINK (Sony's name) and OHCI-Lynx (Texas Instruments).

About the 1394 Wireless Working Group: The Wireless Working Group was formed in 2001 under the original leadership of Steve Bard of Intel Corporation. Along with Peter Johansson, key members of the group include Bob Heile, CTO of Appairent, Inc. and Allen Heberling of Xtreme Spectrum, who chaired the PAL task group.

About the 1394 TA: The 1394 Trade Association is a worldwide organization dedicated to the enhancement and proliferation of the IEEE 1394 multimedia standard, known commercially as FireWire and i.LINK.

2.1.14 Ultra-wideband

Ultra-wideband (UWB, ultra-wide band, ultraband, etc.) is a radio technology that can be used for short-range high-bandwidth communications by using a large portion of the radio spectrum in a way that doesn't interfere with other more traditional 'narrow band' uses. It also has applications in radar imaging, precision positioning and tracking technology.

The UWB Forum [31] was an industry organization dedicated to ensuring that Ultra-Wideband (UWB) products from multiple vendors are truly interoperable. The majority members Motorola [45] and Freescale [63] left the group in 2006 and the UWB Forum is now defunct.

2.1.15 WiMedia Alliance

The WiMedia Alliance [4] is a global non-profit organisation which defines, certifies and supports enabling wireless technology for multimedia applications. WiMedia's UWB technology represents the next evolution of Wireless Personal Area Networks (WPANs), offering end users wireless freedom and convenience in a broad range of PC and consumer electronics products. Current WiMedia products are offered by major OEM's including Imation, Dell and Toshiba, as well as smaller OEMs such as Atlona and Warpia. These products include Wireless USB docking stations, hard drives, projectors and Laptop to HDTV audio/video extenders. Wimedia Alliance is also focused on providing specifications for streaming video applications.

WiMedia's technology is an ISO-published radio standard for high-speed, ultra-wideband (UWB) wireless connectivity that offers a combination of high data throughput rates and low energy consumption. With regulatory approval in major markets worldwide, this technology was selected for Wireless USB and high-speed Bluetooth.

2.1.16 HomeRF

Launched in March 1998, the Home Radio Frequency Working Group (HomeRF WG) has developed an open industry specification, the Shared Wireless Access Protocol (SWAP), operating in the 2.4 GHz ISM band, for a broad range of interoperable consumer devices.

HomeRF initially had a huge cost advantage over 802.11 and was therefore ideal for the SoHo market segment. The standard 1.2 firstly introduced to the market had a data rate of 1.6 Mbps with a wireless operating range of about 50 m. Version 2.0 reaches data rates up to 10 Mbps and the enhancements in version 2.1 should allow data rates of maximum 20 Mbps.

Unlike other wireless LAN standards, besides asynchronous data transfer the HomeRF protocol provides high quality, multi-user voice capabilities. This is enabled by isochronous data transfer, using a subset of the European digital enhanced cordless telecommunications (DECT) standard.

Through falling prices and a better performance of Wi-Fi 802.11b networks and Microsoft began to integrate the competing standard Bluetooth into Windows, HomeRF has fallen into obsolescence. As a result some members left the group and it was disbanded in January 2003.

There is currently no group developing the standard further. The archive of the HomeRF Working Group is maintained by Palo Wireless [72].

2.1.17 Infrared Data Association (IrDA)

The Infrared Data Association [47], often referred to as IrDA, is a non-profit organisation whose goal is to develop globally adopted specifications for infrared wireless communication and was formed in 1994.

The information is transmitted via infrared light. So intervisibility between the appliances is needed to perform a data transfer.

IrDA 1.0 specifies 9.6–115.2 kbit/s (SIR: Serial Infrared), IrDA 1.1 up to 16 Mbit/s (MIR: Mid-Infrared 1.152 Mbit/s, FIR: Fast Infrared 4 MBit/s and VFIR: Very Fast Infrared 16 Mbit/s). UFIR supports up to 96 Mbps, Giga-IR 512 Mbps–1 Gbps. Development is currently going on to increase data rates to 5 and 10 Gbps.

2.2 Cellular radio networks

Although these systems are related, a licence needs to be acquired for using it in the designated frequency band. Usually these systems only can be used as a subscriber, so only a short overview will be given.

2.2.1 History

At the beginning, analogue cellular phone services were offered, with distributed base stations. Later, modern services provided a dynamic change of the base station ("handover") while moving with the mobile handset. Because of enhancements in digital technology, this systems got ousted by digital systems, which had a lot of advantages compared to the old system: Better privacy (eavesdropping requires much more complex hardware), better channel utilisation, easier data transmission and much more.

To enable travelling in other countries with the telephone ("roaming"), uniform systems are required. To achieve this, the European Conference of Postal and Telecommunications Administrations (CEPT) [13] created the Groupe Spécial Mobile (GSM) in 1982 with the objective of developing a standard for a mobile telephone system that could be used across Europe. In 1989, GSM responsibility was transferred to the European Telecommunications Standards Institute (ETSI) [25] and phase I of the GSM specifications were published in 1990. 1991 the Groupe Spécial Mobile is renamed to Standard Mobile Group (SMG). GSM is preserved as the denomination of the standard itself and is now the abbreviation for Global System for Mobile Communications.

In 2007 GSM served more than $2.5 \cdot 10^9$ people across 218 countries and territories (for the GSM family of technologies – GSM/GPRS/EDGE/UMTS and HSDPA) and is the most popular standard for mobile phones in the world.

Founded in 1987, GSMA (GSM Association) [37] is a global trade association representing over 700 GSM mobile phone operators across 218 countries of the world. In addition, more than 200 manufacturers and suppliers support the Association's initiatives as associate members. The primary goals of the GSMA are to ensure mobile phones and wireless services work globally and are easily accessible, enhancing their value to individual customers and national economies, while creating new business opportunities for operators and their suppliers.

The Global mobile Suppliers Association (GSA) [55], representing the leading GSM/3G suppliers worldwide. GSA strengthens the promotion of GSM/EDGE/WCDMA worldwide in new and existing markets and promotes the evolution of GSM as the platform for delivery of 3G multimedia services.

2G (or 2-G) is short for second-generation wireless telephone technology. The main differentiator to previous mobile telephone systems, retrospectively dubbed 1G, is that the radio signals that 1G networks use are analogue, while 2G networks are digital.

The 3$^{\text{rd}}$ Generation Partnership Project (3GPP) [1] is a collaboration agreement that was established in December 1998. The collaboration agreement brings together a number of telecommunications standards bodies which are known as "Organizational Partners". The current Organizational Partners are ARIB [59], CCSA [11], ETSI [25], ATIS [8], TTA [50], and TTC [49]. The establishment of 3GPP was formalised in December 1998 by the signing of the "The 3$^{\text{rd}}$ Generation Partnership Project Agreement".

2.2.2 Important currently used technologies

GSM: The Global System for Mobile Communications is a TDMA-based technology standard for telephony. It is currently the most popular standard for mobile phones in the world.

CDMA: This is the a cellular technology originally known as IS-95, receiving its name from the used code division multiple access technology. This technology is in competition with GSM.

CSD: When a GSM channel is used for data transfer, the usable data rate is 9.6 kbit/s. Advanced channel coding technologies reach 14.4 kbit/s, but a higher probability for block errors. This transmission is called "Circuit Switched Data" (CSD).

HSCSD: High Speed Circuit Switched Data is an enhancement to CSD, offering data rates up to 115.2 kbit/s. It uses multiple time slots (up to eight time slots) of GSM time division multiple access structure. HSCSD is optimal for file transfer and applications that require constant high bit rate and constant transmission delay.

GPRS: General Packet Radio Service is a data service which uses a packet radio principle to carry end user's packet data protocol from mobiles to external packet data networks and visa versa. GPRS radio channel reservation and allocation is done flexible from 1 to 8 radio interface timeslots per TDMA frame and timeslots are shared by all the active users. Up and downlink are allocated separately. The radio interface resources are shared dynamically between data and speech services according to operator's preference and base station load.

Sometimes it is referred as 2.5G, a stepping stone between 2G and 3G cellular wireless technologies. The term "second and a half generation" is used to describe 2G-systems that have implemented a packet switched domain in addition to the circuit switched domain. 2.5G provides some of the benefits of 3G (e.g. it is packet-switched) and can use some of the existing 2G infrastructure in GSM and CDMA networks.

Several radio channel coding schemes are specified to allow data rates from 9.05 kbit/s up to 171.2 kbit/s per user. The available bandwidth per channel depends upon which coding scheme is used.

EDGE: Enhanced Data rates for GSM Evolution is a digital mobile phone technology that allows it to increase data transmission rate and improve data transmission reliability. EDGE can carry data speeds up to 236.8 kbit/s for 4 timeslots (theoretical maximum is 473.6 kbit/s for 8 timeslots) in packet mode.

EGPRS: Enhanced GPRS, another name for EDGE.

UMTS: Universal Mobile Telecommunications System (UMTS) is one of the third-generation (3G) cell phone technologies. Currently, the most common form uses W-CDMA as the underlying air interface. Data rates are 384 kbit/s for the downlink and 128 kbit/s for the uplink.

HSDPA: High-Speed Downlink Packet Access (also known as High-Speed Downlink Protocol Access) is a 3G mobile telephony communications protocol in the High-Speed

Packet Access (HSPA) family, which allows networks based on Universal Mobile Telecommunications System to have higher data transfer speeds and capacity. Current HSDPA deployments support down-link speeds of 1.8, 3.6, 7.2 and 14.4 Mbit/s.

HSUPA: High-Speed Uplink Packet Access is a 3G mobile telephony protocol in the HSPA family with up-link speeds up to 5.76 Mbit/s (category 6).

Next Generation Mobile Networks (NGMN): This is a project for the development of the next generation of mobile communication. Because of that it is often called "4G" (forth generation).

UMTS Terrestrial Radio Access Network (UTRAN), also called "Radio Network System (RNS)". Is one of the hierarchically structured mobile networks according to the UMTS standard.

3GPP Long Term Evolution (LTE): It is also called 3.9G, High Speed OFDM Packet Access (HSOPA), E-UTRAN (Evolved UTRAN) and Super 3G. This is a mobile communications standard, which is specified by the 3rd Generation Partnership Project (3GPP) as UMTS successor.

2.3 Smart Card basics

Smart Cards are often called chip cards or proximity integrated circuit(s) cards (PICCs). The integrated circuit incorporated in the credit card-sized plastic substrate contains elements used for data transmission, storage and processing. A Smart Card may also have an embossed area on one face and a magnetic stripe on the other. Figure 2.3 illustrates the physical appearance of a Smart Card. The physical appearance and the properties of a Smart Card are defined in *ISO 7816, Part 1* [48]. ISO 7816 is the document that sets a standard for the Smart Card industry. *ISO 14443, Part 1 to Part 4* set a standard for contactless Smart Cards.

Usually, a Smart Card does not contain a power supply, a display, or a keyboard. To communicate with the outside world, a Smart Card is placed in (contact Smart Card) a card acceptance device (CAD) or near (contactless Smart Card) a proximity coupling device (PCD); sometimes simply called "reader", which is connected to a host or back end system.

2.3.1 Basic card types

Smart Cards fall into several groups. They can be divided into *memory cards* and *microprocessor cards*. Smart Cards can also be categorised into *contact cards* and *contactless cards*, based on the difference of the physical interface.

Memory Cards versus Microprocessor Cards

The earliest Smart Cards produced in large quantities were *memory cards*. Memory cards are not really "smart", because they do not contain a microprocessor. They are embedded with a memory chip or a chip with memory and non-programmable logic. They are used

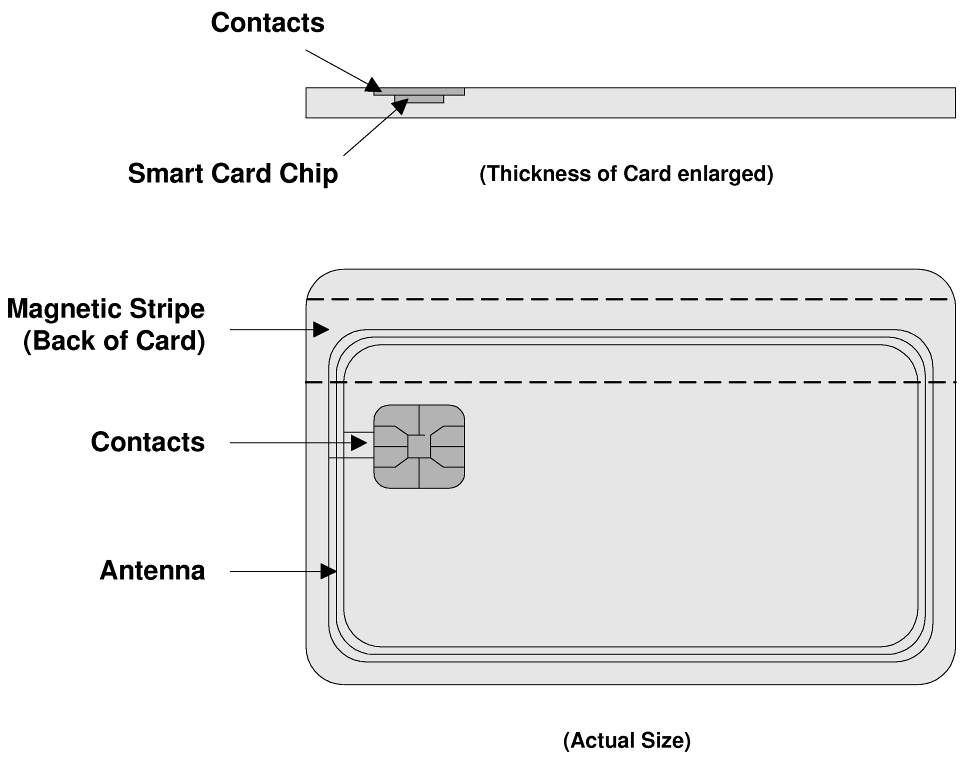


Figure 2.3: Smart card physical appearance

primarily as prepaid cards for public phones or other goods and services that are sold against prepayment.

Since memory cards do not have a CPU to process data, its data processing is performed by a simple circuit capable of executing a few pre-programmed instructions. Such a circuit has limited functions and cannot be reprogrammed. The advantage of memory cards lies in the simple technology. Therefore, they are favoured where low cost is particularly taken into consideration and there is no need for customer specific commands.

By contrast, *microprocessor cards*, as the name implies, contain a processor. They offer greatly increased security and multifunctional capability. With a microprocessor card, data are never directly available to the external applications. The microprocessor controls data handling and memory access according to a given set of conditions (passwords, encryption, and so on) and instructions from the external applications.

Microprocessor cards are widely used for access control, such as banking applications, retail loyalty applications, wireless telecommunications, where data security and privacy are major concerns.

Contact Cards versus Contactless Cards

Contact cards must be inserted in a card acceptance device, and they communicate with the outside world by using the serial communication interface via eight contact pads, as shown in Figure 2.3.

Because a contact card must be inserted into a mechanical card acceptance device in the correct way and in an exact orientation, contactless Smart Cards are popular in situations requiring fast transactions. Public transport systems and access control for buildings are excellent applications for contactless cards.

Contactless cards do not need to be placed in a card acceptance device. They communicate with the outside world through an antenna wound into the card. Power can be provided by an internal battery (active) or can be collected by the antenna (passive). Contactless cards transmit data to a proximity coupling device through electromagnetic fields.

Because the microcircuit of contactless cards is fully sealed inside the card, contactless cards overcome limitations of contact cards: there are no contacts to become worn from excessive use, cards do not need to be carefully inserted into CADs, and cards do not have to be of a standard thickness because they do not have to fit in CADs.

Contactless cards, however, have their own drawbacks. Contactless cards must be within a certain required distance to exchange data with the proximity coupling device. As the card may move out of range very quickly, only limited data can be transmitted due to the short duration of a transaction. It is possible that transactions take place on a card or transmitted data is intercepted, without the card holder recognising it.

2.4 Near Field Communication (NFC)

Near Field Communication (NFC) is closely related to the RFID technology. It is a short-range wireless connectivity technology that evolved from a combination of existing contactless identification and interconnection technologies, jointly developed by Royal Philips Electronics (now NXP Semiconductors) and Sony Corporation. Underlying layers of NFC technology are standardised in ISO/IEC (18092 and 21481) [48, 42], ECMA (340 and 352) [22], and ETSI TS 102 190 [25]. To further the distribution and to become an open platform, the NFC-Forum [56] was founded by Philips and Sony in 2004. At present it comprises a number of international companies as members.

Operating at 13.56 MHz with data rates of 106 kbit/s, 212 kbit/s and 424 kbit/s, with a typical range of a few centimetres.

NFC is also compatible with the broadly established contactless smart card markets based on ISO 14443, NXP Mifare technology and Sony's FeliCa technology. So NFC microchips can be easily integrated in mobile devices, set into card emulator mode and used for existing payment systems.

There is a wide area of application: Starting from downloading data from a passive tag – which can be supplied with energy by the NFC device – up to an easy setup of other longer-range technologies like Bluetooth or WLAN. One property of NFC is also a key advantage of the technology: Because the data transfer is performed via the near field (inductive coupling), it is hard to intercept the communication with a hidden receiver like in other technologies. When a user wants to start a communication or a payment transaction, the NFC device has to be put in proximity to a reader device intentionally. This means that physical proximity of the device to the reader also gives users the reassurance of being in control of the process.

Another important intrinsic advantage of NFC: Due to the fact it operates in the near field with very low power, the interference with other systems is small. This is quite important since the penetration of radio devices continues to rise.

2.4.1 NFC standards and affiliate standards

ISO/IEC 18092 Information technology — Telecommunications and information exchange between systems — Near Field Communication — Interface and Protocol (NFCIP-1)

ISO/IEC 21481 Information technology — Telecommunications and information exchange between systems — Near Field Communication Interface and Protocol -2 (NFCIP-2)

ISO/IEC 22536 Information technology — Telecommunications and information exchange between systems — Near Field Communication Interface and Protocol (NFCIP-1) — RF interface test methods

ISO/IEC 23917 Information technology — Telecommunications and information exchange between systems — NFCIP-1 — Protocol Test Methods

For better interoperability, the NFC Forum defines further specifications, available on their web page [29]:

- **NFC Data Exchange Format (NDEF) Technical Specification**
  Specifies a common data format for NFC Forum-compliant devices and NFC Forum-compliant tags
- **NFC Record Type Definition Technical Specifications**
  Technical specifications for Record Type Definitions (RTDs) and four specific RTDs: text, URI, Smart Poster, and Generic Control.
  - **NFC Record Type Definition (RTD) Technical Specification**
    Specifies the format and rules for building standard record types used by NFC Forum application definitions and third parties that are based on the NDEF data format. The RTD specification provides a way to efficiently define record formats for new applications and gives users the opportunity to create their own applications based on NFC Forum specifications.
  - **NFC Text RTD Technical Specification**
    Provides an efficient way to store text strings in multiple languages by using the RTD mechanism and NDEF format. An example of using this specification is included in the Smart Poster RTD.
  - **NFC URI RTD Technical Specification**
    Provides an efficient way to store Uniform Resource Identifiers (URI) by using the RTD mechanism and NDEF format. An example of using this specification is included in the Smart Poster RTD.
  - **NFC Smart Poster RTD Technical Specification**
    Defines an NFC Forum Well Known Type to put URLs, SMSs or phone numbers on an NFC tag, or to transport them between devices. The Smart Poster RTD

builds on the RTD mechanism and NDEF format and uses the URI RTD and Text RTD as building blocks.

  - **NFC Generic Control RTD Technical Specification**
    Provides a simple way to request a specific action (such as starting an application or setting a mode) to an NFC Forum device (destination device) from another NFC Forum device, tag or card (source device) through NFC communication.

- **Reference Application Technical Specifications**

  - **NFC Forum Connection Handover Candidate Technical Specification**
    Defines the structure and sequence of interactions that enable two NFC-enabled devices to establish a connection using other wireless communication technologies. Connection Handover combines the simple, one-touch set-up of NFC with high-speed communication technologies, such as WiFi or Bluetooth. The specification enables developers to choose the carrier for the information to be exchanged. If matching wireless capabilities are revealed during the negotiation process between two NFC-enabled devices, the connection can switch to the selected carrier. With this specification, other communication standards bodies can define information required for the connection setup to be carried in NFC Data Exchange Format (NDEF) messages. The specification also covers static handover, in which the connection handover information is stored on a simple NFC Forum Tag that can be read by NFC-enabled devices. Static mode is used in applications in which the negotiation mechanism or on-demand carrier activation is not required.

- **NFC Forum Tag Type Technical Specifications**
  The NFC Forum has mandated four tag types to be operable with NFC devices. This is the backbone of interoperability between different NFC tag providers and NFC device manufacturers to ensure a consistent user experience.

  The operation specifications for the NFC Forum Type 1/2/3/4 Tags provide the technical information needed to implement the reader/writer and associated control functionality of the NFC device to interact with the tags. Type 1/2/3/4 Tags are all based on existing contactless products and are commercially available.

  - **NFC Forum Type 1 Tag Operation Specification**
    Type 1 Tag is based on ISO 14443 A. Tags are read and re-write capable; users can configure the tag to become read-only. Memory availability is 96 bytes and expandable to 2 kB; communication speed is 106 kbit/s.
  - **NFC Forum Type 2 Tag Operation Specification**
    Type 2 Tag is based on ISO 14443 A. Tags are read and re-write capable; users can configure the tag to become read-only. Memory availability is 48 bytes and expandable to 2 kB; communication speed is 106 kbit/s.
  - **NFC Forum Type 3 Tag Operation Specification**
    Type 3 Tag is based on the Japanese Industrial Standard (JIS) X 6319-4, also known as FeliCa. Tags are pre-configured at manufacture to be either read and

re-writeable, or read-only. Memory availability is variable, theoretical memory limit is 1 MB per service; communication speed is 212 kbit/s or 424 kbit/s.

– **NFC Forum Type 4 Tag Operation Specification**
Type 4 Tag is fully compatible with ISO 14443 Type A and B standards. Tags are pre-configured at manufacture to be either read and re-writable, or read-only. Memory availability is variable, up to 32 kB per service; communication speed is up to 424 kbit/s.

2.5 Comparison of NFC with similar technologies

	NFC	**IrDA**	**Bluetooth**	**WLAN**	**ZigBee**
Data rate	424 kbps	16 Mbps	2.1 Mbps	248 Mbps	250 kbps
Range	0.1 m	1 m	100 m	250 m	100 m
Complexity	Low	Low	High	High	Medium
Power consumption	Low	Low	Medium	High	Low
Notes		Free line of sight necessary			

Table 2.1: Some key characteristics of wireless technologies

To establish a wired connection, a cable has to be connected to the devices which should communicate. No confusion is possible, the data transfer is performed by the two physically connected devices. In contrast to most other wireless connections, like Bluetooth, it is not clear which physical devices are the communicating parties, any one in the radio range is possible.

NFC and IrDA are wireless communication technologies with some characteristics of wired connections. NFC operates in the near field and requires proximity of the antennae and IrDA additionally needs a free line of sight, with the interfaces facing to each other. Hence these technologies merge the advantages of wireless communication, like convenience, no abrasion or corrosion of contacts with the security of easily selecting the desired communication partner with very low risk of confusion of the communicating parties. These properties open a broad range of applications for these technologies in public transport, secure payment and much more.

Table 2.1 contrasts some key attributes of different wireless communication standards.

2.6 Related topics

2.6.1 Security and privacy of NFC devices

The increasing penetration of electronic payment systems, ticketing and much more makes counterfeiting a more complex problem on the one hand and also allows tracking and data collection of individuals on the other hand. When the communication is performed wirelessly, fraudulent usage is easier. To avoid this, measures for security and privacy have to be taken into account.

Threats

Eavesdropping: In all wireless communications, this is an important issue, because it is much easier than for wired communications. The only countermeasure here is a secure channel.

Data Corruption: This is also easily possible. It is basically a denial of service attack, but also can be easily detected when the device checks the RF field.

Data Modification: The feasibility here depends on the modulation scheme. This can be also avoided when the NFC devices checks the RF field while sending. This means the sending device could continuously check for such an attack and could stop the data transmission when an attack is detected. A better solution would be also a secure channel.

Data Insertion: This is possible when the attacker can send the data before the original device can answer. The countermeasures here are avoiding a delay for the answer, listening by the answering device during the communication channel is open or – once again – a secure channel between the two devices.

Man-in-the-Middle-Attack: For NFC this is very unlikely in a real-world-scenario, because intercepting the communication within the short distance is very conspicuous. Although the probability is very low, listening to the channel by the answering device also can improve the security here.

Relay-Attack: Placing a transceiver close to a legitimate owners device and relaying the signal to a terminal can be preventing by requesting a PIN. Another possibility would be to deal with round trip times, e.g. a distance bounding protocol [39], but this is neither implemented in the smart card ISO 14443 nor in the NFC ISO 18092 standard.

Denial of Service: This is, of course, also possible and cannot be avoided, but can be handled in different ways, depending on the application. For example displaying an error message on the terminal display to prompt a user action.

Skimming: Smart cards often provide an index of applications and an unique serial number. This information can be easily retrieved without notice of the card holder. This enables 3rd party players to sniff habits and track the movement profile.

To avoid skimming, the device can be kept in a shielding case or be equipped with a switch, which prevents an unintentional data transfer. Other countermeasures can be a random UID and PIN protection of the content.

For further details on this topic see [40, 54].

Although NFC is only specified for a short distance, this does not mean that the communication is not possible over a longer distance. The fact that an attacker is not bound by the same transmission limits [14] adhered to by industry designers makes this attack practical. Practical values for possible distances can be found in [38].

When using simple low-cost tags another problem arises: those tags have only limited computing power, so advanced cryptographic operations cannot be performed within an adequate period of time. So special protocols have to be applied to overcome this limitations. For a proposal of an authentication protocol for anti-counterfeiting and privacy protection for a reader-tag-infrastructure see [15].

NFC Specific Key Agreement

Besides the standard key agreement mechanism, it is also possible to implement an NFC specific key agreement. This one does not require any asymmetric cryptography and therefore reduces the computational requirements significantly. Theoretically, it also provides perfect security. The scheme works with 100 % ASK only and it is not part of the ISO standard on NFC. The idea is that both devices, send random data at the same time and derive the key from the resulting field. The security here relies on that an eavesdropper cannot distinguish the data sent from both devices.

For further details on this topic see [40].

2.6.2 Applications for Near Field Communication

Smart cards (chip cards/IC cards) are widespread in a lot of applications: debit, credit and prepaid cards, access control systems and much more. A big advantage of NFC is the compatibility to the existing ISO 14443 contactless smart card infrastructure, also NXP (Philips) Mifare and Sony FeliCa, but can also profit from the higher computing power of current devices. So large existing networks can already be used and extended with new application areas.

NFC technology is also predestined for integration in mobile phones. Nowadays most persons have their own mobile and carry it with them most of the time. Consequently it can be used for many applications like access systems or payment without the need for another device. The display can also be used to output information and the keyboard to input PINs or passwords, furthermore it can provide an online connection to a network for data exchange. Another advantage is that the SIM card also can be used as secure element for the NFC applications. This is described in [53], also showing its advantages and disadvantages.

NXP Semiconductors splits contactless applications into four categories based on how consumers will use the applications. Those categories are

- *Touch and go*, where applications such as micro payment allow consumers to just wave the device over a POS-terminal without having to confirm the transaction

- *Touch and confirm*, such as mobile payment, where the user has to confirm the interaction by entering a password or just accepting the transaction
- *Touch and connect*, linking two devices to enable a peer to peer transfer of data or money
- *Touch and explore*, where devices may offer more than one possible function. The consumer will be able to explore the device's capabilities to find out which functionalities he or she wants to use

Some examples for new applications of NFC are given in [20].

Password container: The NFC technology could be used to replace typing a password or other personal data. Passwords are stored in an secure IC, which could be the SIM card of a mobile phone. The user now places the mobile on a NFC reader interface of a PC and starts surfing. When the password container application detects a login form, it tries to get the appropriate login data from the NFC enabled device. For enhanced security, the mobile can request a PIN before performing the action. When a password is found which fits to the website, it is returned to the target service to be accessed.

Copy protection and secure licensing: Another idea is to use NFC as solution to help addressing software and multimedia piracy by providing software and media publishers with strong copy protection and secure licensing. The NFC device acts as easy-to-use wireless hardware key, the licenses can be attached to natural persons and the applications can be accessed globally without implementing complex hardware schemes.

Virtual NFC tag embedded in to applications: The solution allows content and application providers to put virtual NFC tag information into the application or service and transfer data via NFC to the mobile device and back to any other application. Virtual tags also provide the opportunity to be integrated into web sites. Application areas: ticketing, customer loyalty, discount, gambling and so on.

GPS coordinates in NFC tag: The application helps the users to easily pick up and transfer destination information. It supports services where the customer and/or merchant geographical position is relevant. This use case can also be implemented as virtual tag.

Tourist information: Tourist information points and sights can also be equipped with tags, providing information about the location.

The idea of [21, 2] is to use NFC for coupons. Currently most coupons are paper-based or codes, which can be distributed over the web. In most of the cases, the value of each coupon is relatively low, but misuse in larger scale can result in a major damage for a company. For this cases, NFC-based coupons have many advantages: they can be secured against unauthorised copying, multiple use or altering.

To pick up such a coupon, a mobile device can be held to a advertisement poster, which is equipped with a passive NFC tag, to download the information. It can be honoured then via the Internet or at the cashier in a supermarket. Another advantage of this electronic coupons is that they can be processed automatically. Additional data, like the owner, date and time when it was spent, can be easily captured.

NFC in public transport

The infrastructure for public transport in huge cities is very big and their life time very long. Reliability and clearance speed is also very important in this business. Due to this facts, it would be very hard to establish a new technology. This is also another advantage of NFC – it is compatible with the widely used NXP Mifare and Sony FeliCa cards. So there is no need for big changes of the infra structure on operator side.

The integration of NFC in a mobile or smart phone is very beneficial for the customers, since nowadays most persons always have their mobile phone with them. Using the phone's online connection, tickets can be also bought via the Internet – a benefit for the public transport operator, who saves costs, and the customer because of the higher comfort than going to a ticket counter.

For some thoughts about NIC in public transport, see [70].

NFC for payment

The more value depends on the technology, the more security measures have to be taken, as already discussed in the preceding applications. Payment is the most rewarding application for illicit use, because it can be applied universally. Because of this reason, special attention has to be paid to security measures. For further proceedings about this topic see [61] and [60].

Medical applications for NFC

Also the medical sector has great potential for using NFC technology. The special challenges here are the high requirements for reliability and security. For some examples see [67, 51, 10].

2.7 User interfaces

A *User Interface* (or *Human Computer Interface*) is necessary for *Human-Computer Interaction* (HCI). It has essentially two components: input and output. *Input* is how a person communicates his or her needs or desires to the computer. Some common input components are the keyboard, mouse, trackball, touch-sensitive pads or screens and one's voice (for spoken instructions). *Output* is how the computer conveys the results of its computations and requirements for the user. The most common computer output mechanism is the display screen, followed by voice and sound.

Designing a good (graphical) user interface is a very complex task. There are various things which have to be taken into account. The intended target group, handicapped persons who operate the software, intuitiveness and a lot more. This is important for stressful situations and an effective, economic work flow.

2.7.1 Interaction styles

An interaction style is simply the method, or methods, by which the user and a computer system communicate with one another.

Command line

The command-line interface requires the user to press a function key or type a command into a designated entry area on the screen. It is flexible and able to incorporate options or parameters to vary its behaviour. One problem with command lines is that the syntax, commands, options and parameters must be remembered. Another problem is that command lines can be cryptic and obscure with complex syntax. They are also very prone to, and intolerant of, typing errors.

Menu selection

A menu is a set of options or choices from which a user must choose. On screens, the user selects a choice with a pointing device or keystrokes. Typically some kind of visual feedback is then provided to indicate the option selected. Menu selections can also be provided by voice as exemplified by the "Press 1 to . . . ", encountered at big telephone systems.

Screen menus are advantageous because they utilise a person's much stronger power of recognition, not recall. But the menu choice labels must be meaningful and understandable for the menu to be truly effective, otherwise speed of use will be degraded and errors increase. Menus can break a complex interaction into small steps, which structure the decision-making process. This is especially helpful for infrequent users who are unfamiliar with the system. On the other hand, many small steps may slow the knowledgeable user.

Form fill-in

The form fill-in style is very useful for collecting information. Today's typical form-structured screen contains a series of controls or fields into which the user either types information or selects an option, or options, from a listing of choices (technically, a listing of choices presented to the users is a menu). Screen fill-in forms are derived from their antecedents, paper forms. An advantage of a form is its familiarity. If it is designed well, a form will aid the user in understanding its purpose and allow fast and easy entry of information. Conversely, a poorly designed screen form can be inefficient and aggravating to complete.

Some popular examples where form fill-in's are used is software for generating invoices, for accounting and other database-related assignments.

Direct manipulation

A direct manipulation interface, as found in graphical systems, enables the user to directly interact with elements presented on the screen. These elements (called objects) replace the keyed entry of commands and menus. Users typically select screen objects and actions by using pointing mechanisms, such as the mouse, trackball, touchpad, trackpoint, graphics tablet, touchscreen or joystick, instead of the traditional keyboard. They navigate the screen and execute commands by using menu bars and pull-down menus. The direct manipulation style will be discussed in more detail in the "Graphical user interface" section later on.

Anthropomorphic

An anthropomorphic interface tries to interact with people the same way people interact with each other. Anthropomorphic interfaces include spoken natural language dialogues, gestures and eye movements.

With the exception of an anthropomorphic interface, most current systems contain a blend of these interaction styles. The proper mix can be created only after an understanding of the user, the tasks to be performed and the goals of the system are obtained. Within a CAD program for instance, the user often can utilise direct manipulation, menu selection and the command line. So the advantages of the command line can be exploited for frequently used instructions and searching in the menu supplements the seldom one's. Table 2.2 compares different user interaction styles and their advantages and disadvantages.

Style	Advantages	Disadvantages
Command Line	Powerful	Commands must be memorised
	Flexible	Requires learning
	Appeals to expert users	Intolerant of typing errors
	Conserves screen space	Difficult for casual users
Menu Selection	Utilises recognition memory	May slow knowledgeable users
	Reduces interaction complexity	Consumes screen space
	Aids decision-making process	May create complex menu hierarchies
	Minimises typing	
	Aids casual users	
Form Fill-in	Familiar format	Consumes screen space
	Simplifies information entry	Requires careful and efficient design
	Requires minimal training	Does not prevent typing errors
Direct Manipulation	Faster learning	Greater design complexity
	Easier remembering	Window manipulation requirements
	Exploits visual/spatial cues	Requires icon recognition
	Easy error recovery	Inefficient for touch typists
	Provides context	Increased chance for screen clutter
	Immediate feedback	
Anthropomorphic	Natural	Difficult to implement
		Strong hardware requirements

Table 2.2: Some advantages and disadvantages of interaction styles

2.7.2 Graphical user interfaces

A *user interface*, as recently described, is a collection of techniques and mechanisms to interact with something. In a *graphical* interface, the primary operating mechanism is usually a pointing device of some kind; this device is the electronic equivalent to the human hand. But also proprietary peripherals, especially designed for the software, can be used as control units – for example a remote control. What the user interacts with is a collection of elements referred to as *objects*, displayed on a screen. The objects for instance display pictures, information and values, report a state, represent controls the user can perform operations, called actions, on.

Design principles and techniques of graphical user interfaces

For enabling intuitive operation, some rules have to be obeyed. In everyday life we find numerous quasi-standards which are in our subconscious mind. For instance a turning knob: turning it clockwise means more volume when standing in front the hi-fi unit and hotter when adjusting a radiator thermostat – but usually an increasing value; counter-clockwise has the contrary effect. But there are also well-known exceptions, for example the water tap. It has a right-handed thread and to open it and let the water flow, it has to be turned left.

This everyday life experience should be also taken into account for a graphical user interface, which can also depict turning knobs, switches and other controls, and they should behave like real controls in the "real" world. Additionally the standard naming, shortcuts, icons and other "common" functions of their respective operating system should be used (e.g. 'Strg-S' for the *save* function in Windows); this means: follow the style guide. Another example is the behaviour of a button: usually an action is triggered on release of the button, which means a mouse click can be cancelled when clicking on the button by moving the mouse away from the button before releasing the mouse key. For a stop watch a different behaviour will be desired, a button should react on the "button down" event to minimise the delay.

Aims of graphical user interfaces

There are several design approaches for a graphical user interface, depending on the desired purpose of use. Inexperienced users may prefer an "assistant" which guides through the necessary steps with explanations, advanced users will prefer an ordinary interface. Some programs also provide both options, to offer help to an inexperienced user but also avoid slowing down a user who is familiar with the system.

A graphical front-end can be a graphical user interface which operates a command-line tool. Its aim is to facilitate the use of the command-line tool by offering the comfortable characteristics of a GUI.

It can be designed to perform particular actions, therefore no transparency of background operations is needed, only the result is of interest.

Another aim can be to demonstrate the functions of a certain library.

Selection of colours

For legibility it is important to use proper colours with enough contrast of background and font, also depending on the cultural background. Furthermore achromatopsy (colour blindness) or a defective colour vision should not handicap users. This can be solved by using the colour as aid but not as the only information of a result.

Some colours are associated with particular topics and can be experienced positive or negative. For instance a traffic light: red signifies "Stop!", amber (yellow) signalises 'Attention!" and green means "Go.". In software development errors often are marked red, to draw the user's attention to the problem occurred.

Layout of the graphical user interface

The structured and well arranged graphical user interface enables efficient and intuitive operation. All frequently used objects should be accessible immediately, other ones can be put to a separated “advanced options” area. All objects belonging together, like buttons, input fields, check boxes and so on should be arranged within a group. To avoid confusing the user with lots of inputs and controls, it is a good idea to place the most used objects visible on the page and move advanced ones to another place.

For further details on designing intuitive user interfaces with good usability see [35, 66].

2.8 Other Evaluation Kits

This section introduces other similar kits.

2.8.1 Texas Instruments reader evaluation kit

Figure 2.4: S4100 multi-function reader evaluation kit

Texas Instruments RFid Systems’ S4100 Multi-Function Reader (MFR) Evaluation Kit [46] (see Figure 2.4) takes RFID integration to a new level. The kit enables users to experience the MFR and its functions in a PC-based environment. The Base Application allows users to execute standard read/write/lock commands, as well as experiment with anti-collision and multiple-technology environments. With standard RS-232 and RS-485 serial I/O ports, the S4100 MFR is a plug-and-play solution, ideal for logical access applications, as well as financial, ticketing, access control, and others.

Modular firmware architecture combined with serial firmware programmability provides for system scalability, meaning that the reader can be programmed to operate as a stand alone unit, or as part of a host controlled, multi-point system.

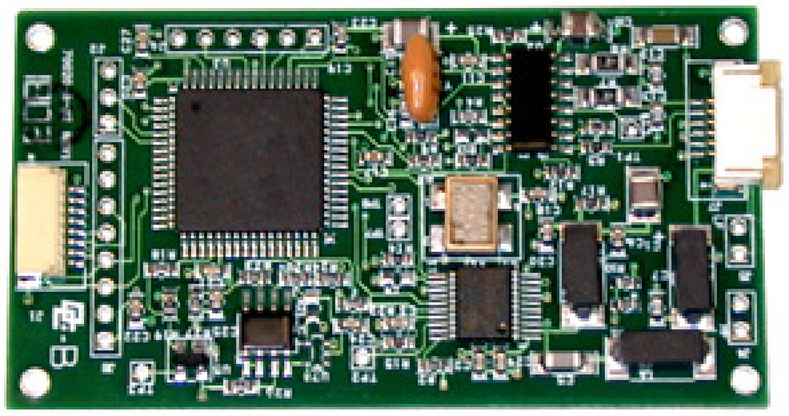


Figure 2.5: S4100 multi-function reader module

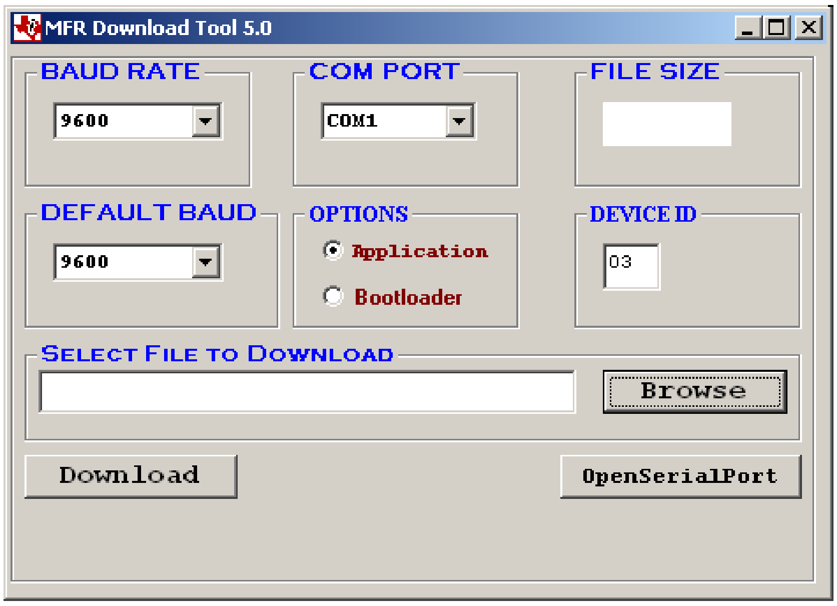
Contents of the kit:

- 1 MFR-based housed S4100 reader
- Universal power supply
- Global power adapters
- RS-232 communications cable
- Sample HF and LF transponders
- Spare MFR module RF-MGR-MNMN
- CD with demo software and reference manuals
- Getting started guide

The PC program `MFRApplicationDownload.exe` (see Figure 2.6) provides an easy way for the user to download HEX files to the evaluation board, for instance the Base Application. The host can then talk to the board via the serial interface by the Application Layer Protocol. The kit also includes a platform independent software API (cross-compatible with other HF readers in the TI portfolio), which facilitates development of PC based software.

Usability

The kit is very useful for designers building prototypes. It is also possible to get a first impression of the technology very quickly, using the demo software. Furthermore, tests in the field are feasible. Because of the universal design it is also applicable for small series production of custom-designed products with a very short time to market. The software is quite simple, but its functionality is limited to provided firmware files, which means functionality is limited. In order to use the full range of functionality, a separate firmware has to be developed, which requires too much time for a rapid test.



BAUD RATE: 9600
COM PORT: COM1
FILE SIZE:
DEFAULT BAUD: 9600
OPTIONS: [✓ Application] Bootloader
DEVICE ID: 03
SELECT FILE TO DOWNLOAD:

Figure 2.6: TI download utility (PC software)

2.8.2 Atmel RF design tool kit

Atmel [17] Smart RF - Tools & Software is a collection of software and boards to evaluate Atmel's RF products.

The design kits have a modular structure. They consist of a motherboard (microcontroller board), a RF receiver/transmitter/transceiver (design board), an interface board and various other components. For switching to another operating frequency, only the RF modules needs to be changed and the other modules can be reconfigured.

For a picture of the Quick-Start RF Transceiver Evaluation Kit see Figure 2.7.

The RF Design Kit software forms the PC-based Graphical User Interface for the RF Design Kit as well as for the Flamingo RF Design Kit. The RF Design Kit software is used to configure the RF transmitter and receiver. Parameters such as baud rate, modulation scheme, etc. can easily and quickly be changed. Software tools to evaluate the received data are also provided.

The software (for an example screenshot see Figure 2.8) includes a library of various transmitter and receiver hardware modules. They can be selected by a combo-box[2], then the configurable properties of the modules are shown by the software and can be changed. To facilitate comprehension, e.g. the position of push buttons, a labelled photo of the hardware is shown in some cases.

The provided program `AT86RF401 SPI Controller.exe` (see Figure 2.9 for a screenshot) is a software tool that allows evaluation of AT86RF401 RF performance without requiring the user to write code. It also allows the program and data memory of the AT86RF401 to be written/read with the same utility. It eases the setup and configura-

[2]This term refers to the Microsoft Visual Studio name "Combo-box Control" for pull-down selectors

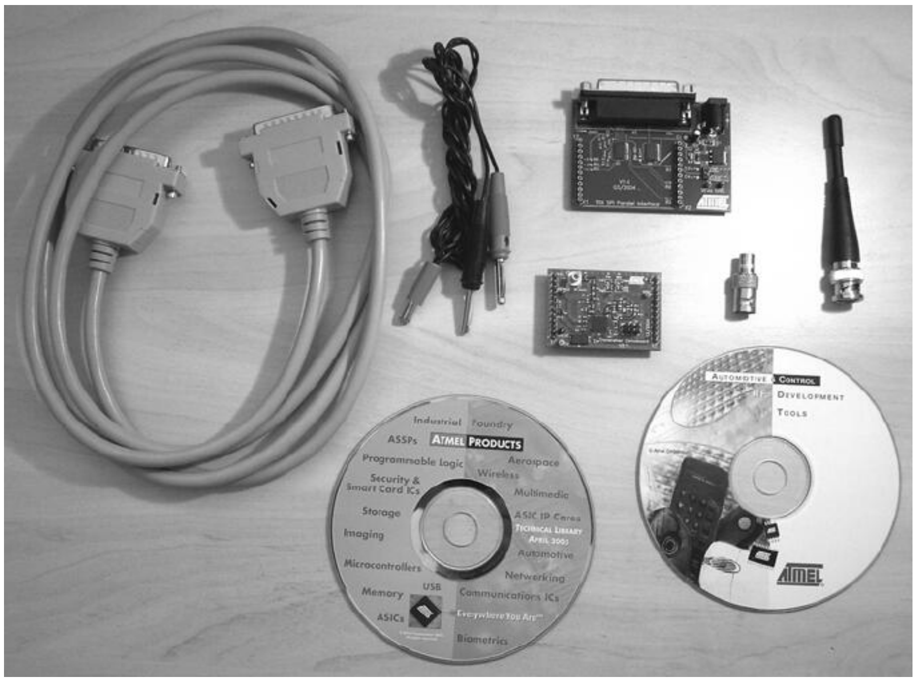


Figure 2.7: Atmel ATR2406 RF design kit

tion of the SPI controller. Single configuration bits can be altered by check-boxes[3] in a very uncomplicated way, without calculating hex values. Read-only or inaccessible bits are pictured as static squares and those which are not currently configurable are disabled. Furthermore tools, like a proprietary bit timer calculator, are integrated.

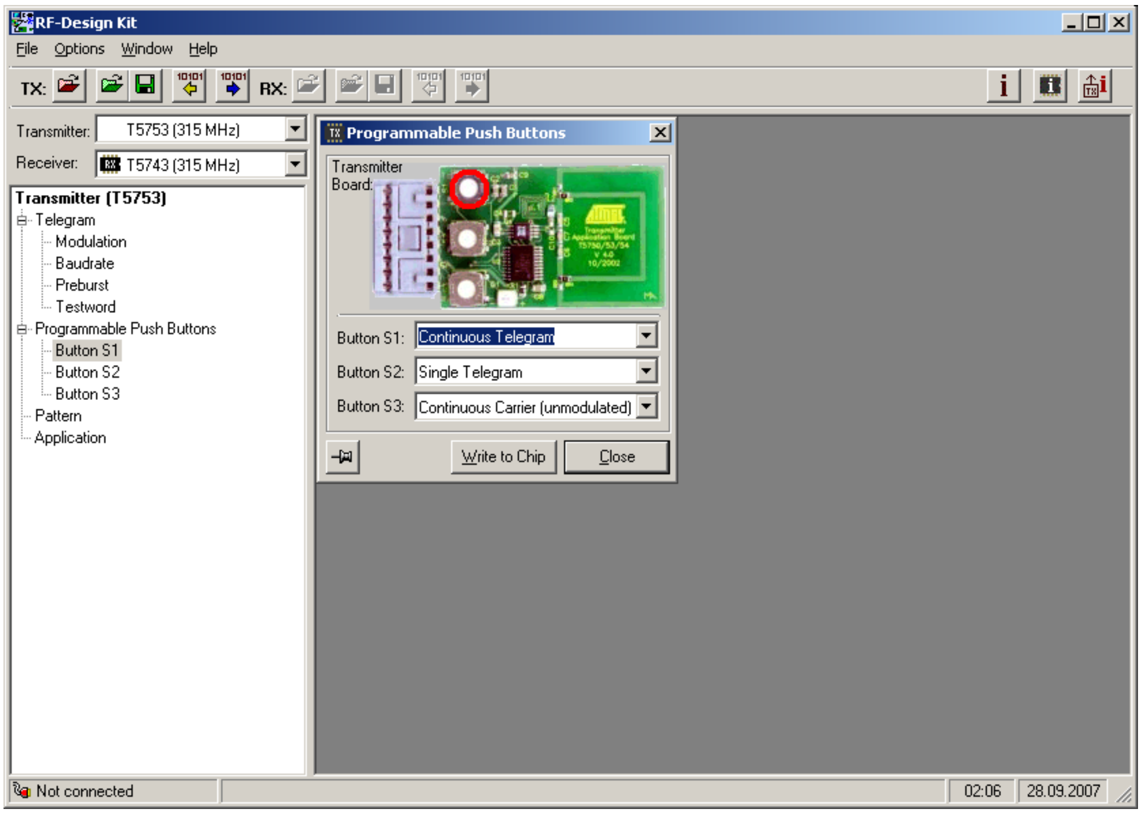
Usability

This kit also enables a quick evaluation of various products with a short setup time. The modular design enables the main and interface modules to be reused.

RF design kit PC software: Photos assist the user in locating buttons, which is much more comfortable than looking for numbers on the PCB or a description like "the first button" (since the module can be turned around, it is not clear where to start counting). Photos can be also used to identify different products, locate jumpers, interfaces and so on. Options can be selected using combo-boxes and its respective meaning, which is much more convenient than calculating register values. The disadvantage is that the corresponding register values are not shown, so they cannot be transferred to a self-written application – the values tested already have to be calculated from the data sheet.

SPI controller PC software: The register configuration is stored in several bytes. For more comfortable input, the single bits can be changed by corresponding check-boxes. Not used or *reserved for future use* bits are illustrated as black squares, the check-boxes for read-only bits are disabled (and greyed out). This is not completely consistent (not all

[3]This term refers to the Microsoft Visual Studio name "Check-box Control" for squares, which can be ticked and unticked



Transmitter: T5753 (315 MHz)
Receiver: T5743 (315 MHz)

Transmitter (T5753)
- Telegram
  - Modulation
  - Baudrate
  - Preburst
  - Testword
- Programmable Push Buttons
  - Button S1
  - Button S2
  - Button S3
- Pattern
- Application

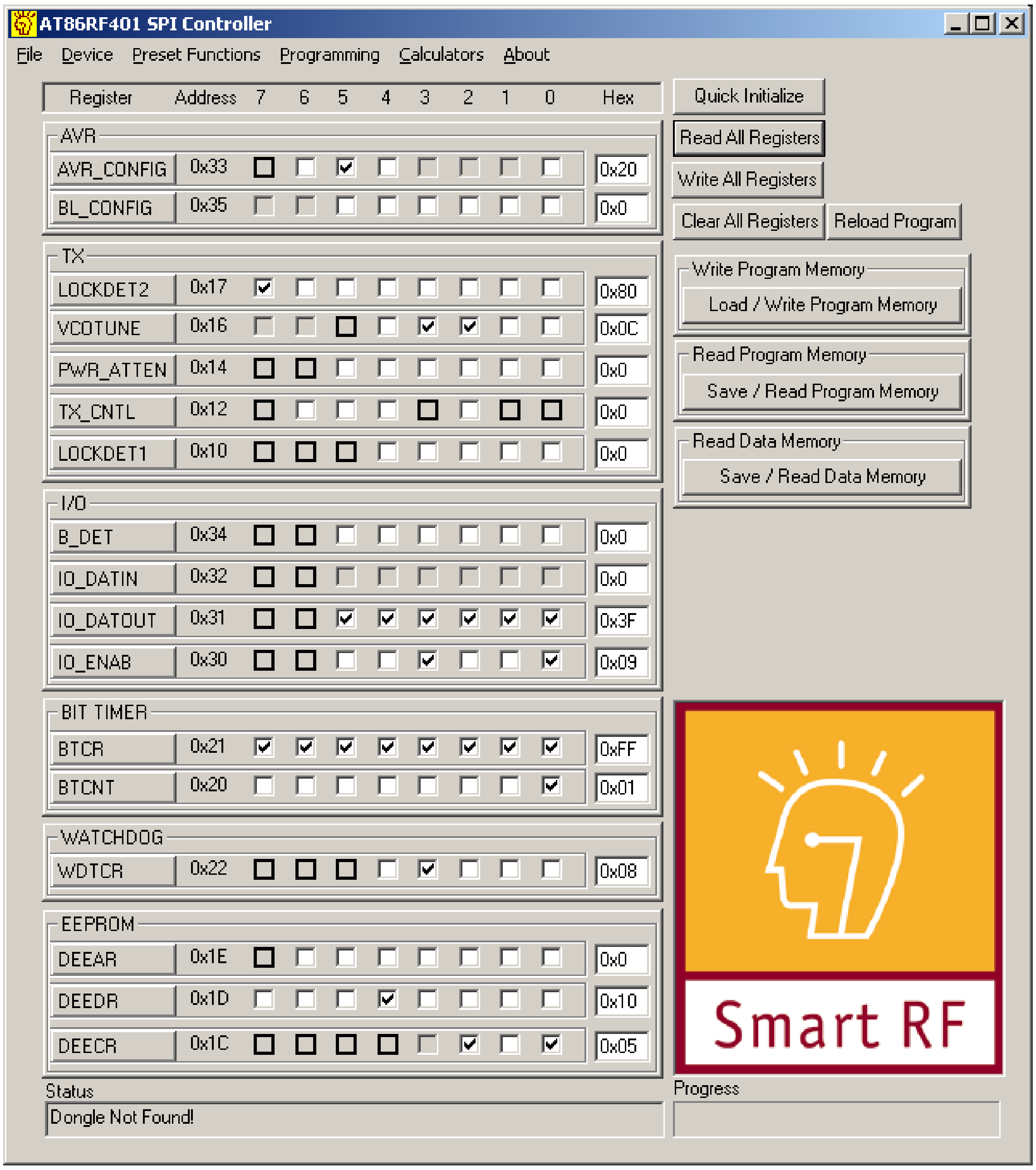
Programmable Push Buttons

Transmitter Board:

Button S1: Continuous Telegram
Button S2: Single Telegram
Button S3: Continuous Carrier (unmodulated)

Not connected | 02:06 | 28.09.2007

Figure 2.8: Atmel RF design kit application software V 1.0.5



Register	Address	7	6	5	4	3	2	1	0	Hex
AVR										
AVR_CONFIG	0x33			✓						0x20
BL_CONFIG	0x35									0x0
TX										
LOCKDET2	0x17	✓								0x80
VCOTUNE	0x16					✓	✓			0x0C
PWR_ATTEN	0x14									0x0
TX_CNTL	0x12									0x0
LOCKDET1	0x10									0x0
I/O										
B_DET	0x34									0x0
IO_DATIN	0x32									0x0
IO_DATOUT	0x31			✓	✓	✓	✓	✓	✓	0x3F
IO_ENAB	0x30					✓			✓	0x09
BIT TIMER										
BTCR	0x21	✓	✓	✓	✓	✓	✓	✓	✓	0xFF
BTCNT	0x20								✓	0x01
WATCHDOG										
WDTCR	0x22					✓				0x08
EEPROM										
DEEAR	0x1E									0x0
DEEDR	0x1D				✓					0x10
DEECR	0x1C						✓		✓	0x05

Status

Dongle Not Found!

Figure 2.9: Atmel AT86RF401 SPI controller version 1.3

read-only bits are disabled, the only write-only bit with a special function is also disabled) and the functions of the bits also have to be looked up in an external documentation. The function of the buttons in the first column “Register” is also not clear (read/write/detailed information about the register?). Also some calculators are included for battery low level, VCO switch capacitor, bit timer interval time and Intel Hex record checksum. This is much more comfortable than looking up the particular bits in the data sheet.

Chapter 3

Design

The aim is to create a modularised demonstration program with re-usable source code to allow a customer to check out the functionality and provide the possibility for easily re-configuring the advertised chip – which is evaluated – on a demonstration board.

3.1 Requirements for the software

- Easy to use
  It should be possible to operate the program intuitively.
- Flexibility
  Hence, it should have all parameters configurable in a clearly arranged way.
- Reusability
  The source code should be reusable as far as possible for future projects, ICs with a similar functionality or as a basic concept for customers.
- Portability
  A transfer to another operating system ought to require less effort.
- Embedded library
  The software should be built on an existing library.
- Maintainability
  The implementation should be easy to understand and self-explaining.

3.2 System architecture

The application is a PC based software, which functional modules should be easily portable to other operating systems. For a basic overview of the system architecture, see Figure 3.1.

3.3 Software architecture

The intention is to design a graphical front-end to an existing library.

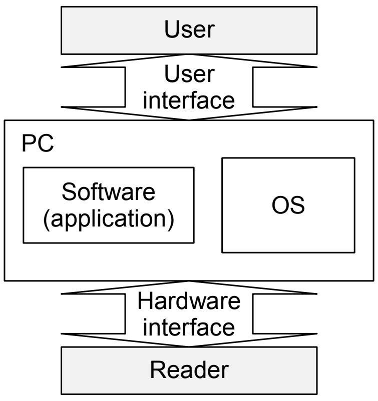


Figure 3.1: Basic system architecture

To meet the miscellaneous requirements and for simple porting, a modular approach is essential. To achieve this, the software is divided into two main parts: The operating system dependent graphical user interface and the functional "abstraction layer" in the background, which standard C++ code can be easily ported. To port the program to another OS, only the GUI has to be rewritten and the functional implementation can be reused.

The GUI has to accomplish several tasks: It should guide the user through the necessary basic steps to perform a data transfer, shall offer the option to modify parameters and accomplish several plausibility and parameter limit checks. It should also be aware of the state of the system and disable functions which are not allowed and/or not possible. For instance, it makes no sense to execute commands, as long as the interface is closed and no connection to the hardware is established. Those values are then passed to the next layer: the functional part.

This functional part is written ISO/IEC C++. The communication with the GUI is done by passing parameters calling encapsulated functions. It provides buffers for the communication, performs error handling and implements the functionality for the command buttons on the GUI.

The architecture is illustrated in Figure 3.2.

3.3.1 Use cases

The use case (see Figure 3.3) represents novice and advanced users.

The novice user, who wants to evaluate the product, wants to see the basic functionality and should not be overcharged with advanced functions and configurations.

The professional user or developer, on the other hand, wants to have the full flexibility and may also want to execute commands which are not allowed in a particular state to check the behaviour. This user does not want any of the kind of restrictions which might be helpful for a novice user who needs guidance.

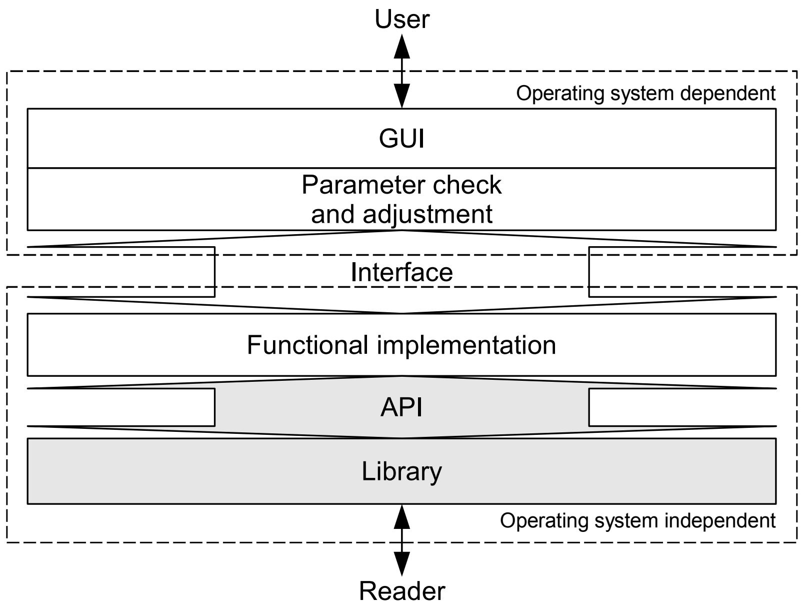

User
Operating system dependent
GUI
Parameter check
and adjustment
Interface
Functional implementation
API
Library
Operating system independent
Reader

Figure 3.2: Software architecture

3.3.2 State machines

A state machine represents the internal state of the IC. The state transitions are derived from the data sent to the IC and its response. Each state transition triggers an update of the GUI. These state machines are described in the particular modules.

3.4 Graphical user interface

The NExT GUI contains client frames for operating the Joiner in Initiator, Target and Mifare mode. Each client frame consists of several subwindows. This enables reusability of the particular modules for several applications. For instance the interface module is needed for the Initiator, the Target as well as for the Mifare frame. The communication between the particular subwindows is done by a separate message thread.

To guide the inexperienced user through the single steps for a successful communication, commands which are not possible or those which make no sense for the current state of the communication, are disabled.

3.5 Modules

3.5.1 Terminology

Main window: The main application window, containing the menu and the area for the client frames.

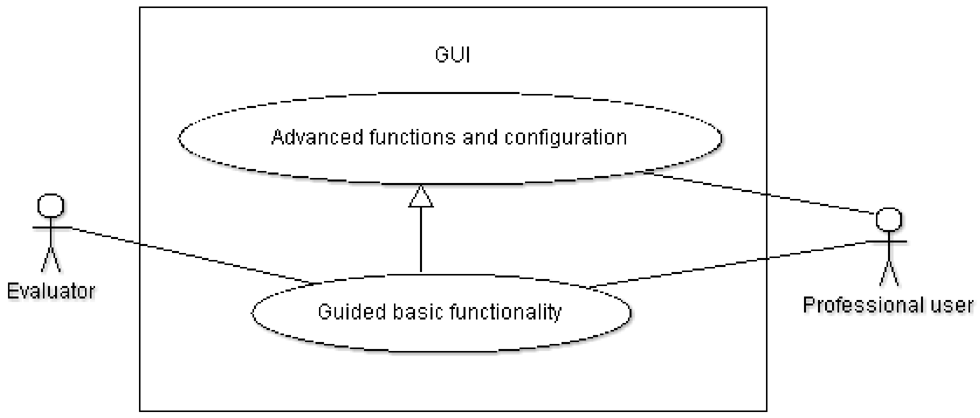

GUI
Advanced functions and configuration
Guided basic functionality
Evaluator
Professional user

Figure 3.3: Use case diagram

Client frame: Each client frame holds several subwindows and build a functional group, which can operate one evaluation board in a certain way:

- BFL Initiator
- BFL Target
- HAL Initiator
- HAL Target
- Mifare

Subwindow: Each subwindow is a functional block with commands, options and parameter configuration possibilities. Each subwindow consists of a GUI and a functional module pair. The following subwindows are intended:

- Serial interface
- NFC 106 kbps passive activation
- NFC 212/424 kbps passive activation
- Mifare
- BFL Initiator
- BFL Target
- Common HAL interface
- HAL Initiator
- HAL Target
- General preferences
- Data transfer
- History (log output)

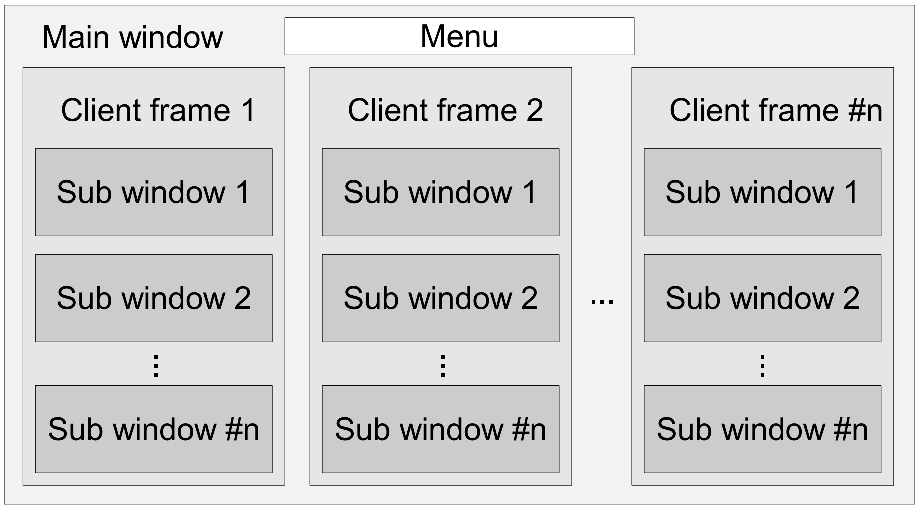


Figure 3.4: GUI architecture

3.5.2 Windows modules

The module structure of the GUI is shown in Figure 3.4.

Main application

This module, pictured in Figure C.1, is the main entry point for the Windows application. It initialises and manages the client frames.

Menu structure

1. File
    - 1.1 Exit
2. Mode
    - 2.1 BFL
        - 2.1.1 NFC Initiator
        - 2.1.2 NFC Target
        - 2.1.3 NFC MiFare
    - 2.2 HAL
        - 2.2.1 HAL Initiator
        - 2.2.2 HAL Target
3. Options
    - 3.1 BFL Interface

4. Help
   4.1 About

Client Frame

A client frame is composed of several subwindow modules to obtain the desired functionality. It combines the functionality of the subwindows and embeds their GUIs in the client frame.

Interface dialogue

The interface dialogue box is started by selecting the entry in the Options menu. Within this dialogue profiles for the serial RS-232 interface can be created respectively changed. Those settings are stored in an initialisation file.

Subwindows

Each *"subwindow"* consists of a GUI and a functional abstraction module pair.

The subwindows are described in more detail in the following sections.

3.5.3 BFL based subwindows

This functional blocks implement functions based on the BFL library. For a detailed description of the BFL API see Appendix A.

BFL interface subwindow

The interface subwindow allows the selection of a predefined profile. The profiles can be configured with the Interface Dialogue. For the RS-232 serial interface a set of parameters consists of the COM port and the baud rate. For the layout of the GUI see Figure C.3.

Required C++ API functions:
phcs_BFL::phcsBflNfcInitator_Wrapper::Initialise
phcs_BFL::phcsBflMfRd_Wrapper::Initialise

BFL based NFC 106 kbps passive subwindow

This subwindow implements Initialisation and Single Device Detection at 106 kbps. The design of the GUI is shown in Figure C.4.

This initialisation and anticollision can be also performed with Mifare cards, implementing the ISO 14443-3 standard.

Required C++ API functions:
phcs_BFL::phcsBflI3P3A_Wrapper::Initialise
phcs_BFL::phcsBflI3P3A_Wrapper::RequestA
phcs_BFL::phcsBflI3P3A_Wrapper::AnticollSelect
phcs_BFL::phcsBflI3P3A_Wrapper::Select
phcs_BFL::phcsBflI3P3A_Wrapper::HaltA

BFL based NFC Initiator subwindow

The design of the GUI is shown in Figure C.5. With this module the NFC Initiator functions are realised. It can operate NFC Targets, Mifare PICCs and FeliCa cards. Table 3.1 shows possible configuration schemes for different applications. Some cards do not support all possible baud rates, so further restrictions may appear.

Mode	**Speed**	**Mifare**	**FeliCa**	**NFC**
Passive	106 kbps	✓	✗	✓
Passive	212 kbps	✗	✓	✓
Passive	424 kbps	✗	✓	✓
Active		✗	✗	✓

Table 3.1: Possible operating modes for certain applications

Opening the interface enables further buttons for the activation sequence. Depending on the configuration, the appropriate buttons are activated. The different activation schemes can be gathered from Table 3.2.

Mode	**Speed**	**Activation**
Active	106, 212 and 424 kbps	Attribute Request
Passive	106 kbps	NFC 106 kbps passive activation
	212 or 424 kbps	NFC 212/424 kbps passive activation

Table 3.2: Matrix for possible activation of the NFC Target

After a successful Attribute Request command, all buttons are available and the operations must be manually selected, depending on the setup and available NFC Targets.

An example for a meaningful sequence in Initiator mode with one Target:
Open Interface → ATR_REQ → DSL_REQ → RSL_REQ → Close Interface

This sequence is implemented in a kind of state machine. On startup, all buttons except the one for opening the interface, are disabled.

Required C++ API functions:

phcs_BFL::phcsBflNfcInitator_Wrapper::AtrRequest
phcs_BFL::phcsBflNfcInitator_Wrapper::PslRequest
phcs_BFL::phcsBflNfcInitator_Wrapper::SetTRxProp
phcs_BFL::phcsBflNfcInitator_Wrapper::DepRequest
phcs_BFL::phcsBflNfcInitator_Wrapper::AttRequest
phcs_BFL::phcsBflNfcInitator_Wrapper::DslRequest
phcs_BFL::phcsBflNfcInitator_Wrapper::RlsRequest
phcs_BFL::phcsBflNfcInitator_Wrapper::ResetProt
phcs_BFL::phcsBflNfcInitator_Wrapper::WupRequest
phcs_BFL::phcsBflNfcInitator_Wrapper::RequestedToxValue

BFL based NFC Target subwindow

In this subwindow the NFC Target functions are implemented. The design of the GUI is shown in Figure C.6.

Required C++ API functions:
phcs_BFL::phcsBflNfc_TargetWrapper::Initialise
phcs_BFL::phcsBflNfc_TargetWrapper::ResetProt
phcs_BFL::phcsBflNfc_TargetWrapper::SetTRxBufProp
phcs_BFL::phcsBflNfc_TargetWrapper::SetUserBuf
phcs_BFL::phcsBflNfc_TargetWrapper::SetPUser
phcs_BFL::phcsBflNfc_TargetWrapper::Dispatch

BFL based NFC 212/424 kbps passive subwindow

This subwindow implements Initialisation and Single Device Detection at 212 and 424 kbps.

Sony's FeliCa cards use this type of activation. For the layout of the GUI see Figure C.7.

Required C++ API functions:
phcs_BFL::phcsBflPolAct_Wrapper::Initialise
phcsBflPolAct_Hw1Init
phcs_BFL::phcsBflPolAct_Wrapper::Polling
phcs_BFL::phcsBflPolAct_Wrapper::SetWaitEventCb

BFL based Mifare subwindow

Here Mifare specific functions are implemented (see Figure C.8).

Required C++ API functions:
phcs_BFL::phcsBflMfRd_Wrapper::Transaction

3.5.4 HAL based subwindows

For a detailed description of the HAL API see Appendix B.

Common HAL subwindow

This subwindow comprises common HAL functions which are used in NFC Initiator as well as Target mode. The design of the GUI is shown in Figure C.9.

Required C++ API functions:
phHalNfc_Enumerate
phHalNfc_Open
phHalNfc_GetDeviceCapabilities
phHalNfc_Close

HAL based NFC Initiator subwindow

This subwindow implements NFC Initiator specific functions, based on the HAL library. The design of the GUI is shown in Figure C.10.

Required C++ API functions:
phHalNfc_Connect
phHalNfc_Poll
phHalNfc_Disconnect
phHalNfc_Transceive

HAL based NFC Target subwindow

This subwindow implements NFC Target specific functions, based on the HAL library. The design of the GUI is shown in Figure C.11.

Required C++ API functions:
phHalNfc_StartTargetMode
phHalNfc_Receive
phHalNfc_Send

3.5.5 Description of common subwindow modules

This section describes the shared subwindow modules, used by the clients.

General preferences subwindow

Within this dialogue some generic preferences can be set, the timeout value, and a RF reset can be performed. The design of the GUI is shown in Figure C.12.

Data/file transfer subwindow

In this section data can be prepared for transfer. It offers a text box for data input, a selector for choosing hexadecimal or ASCII as input format and the option to select a file as data source. The design of the GUI is shown in Figure C.13.

History subwindow

The History window provides logging. Executed commands with used parameters are displayed, moreover responses and error messages are visualised in a human readable way. Some encoded information shall be decoded and translated for the user. For instance particular bits in a response message, coding device capabilities, can be presented as ‘function supported’ or ‘function not supported’. Also error codes should be translated into a text message. The design of the GUI is shown in Figure C.14.

3.5.6 Module group description

For operating a evaluation board in a particular mode (NFC Initiator/Target, Mifare), several subwindows are combined and their GUIs are embedded in a *client frame*.

BFL based NFC Initiator client

This frame should contain all necessary functions needed for Initiator operating mode. Used subwindows:

- Interface
- NFC 106 kbps passive activation
- NFC 212/424 kbps passive activation
- NFC Initiator
- History
- Data transfer
- General preferences

The client frame is shown in Figure C.15.

BFL based NFC Target client

This frame uses the same structure as the NFC Initiator, but the Target subwindow is used instead of the Initiator one. In the NFC Target frame, the interface enables and disables the buttons. When the interface is opened successfully, the BFL NFC Target frame is enabled. The client frame is shown in Figure C.16.

BFL based Mifare client

This frame should implements the Mifare specific functionality. The used subwindows are:

- Interface
- NFC 106 kbps passive activation
- Mifare
- General preferences
- History

The client frame is shown in Figure C.17.

NFC HAL Initiator client

The used subwindows are:

- HAL common
- HAL Initiator
- Data transfer
- History

The client frame is shown in Figure C.18.

NFC HAL Target client

The used subwindows are:

- HAL common
- HAL Target
- Data transfer
- History

This frame has the same structure as the NFC HAL Initiator, but integrates the Target subwindow instead of the Initiator one. The client frame is shown in Figure C.19.

3.6 Class structure

3.6.1 Subfunction class structure

See Figure 3.5, 3.6, 3.7 and 3.8 for the corresponding class and dependency diagrams. The structure of the modules is similar, so only the Mifare functions are depicted.

3.6.2 Error message generation

The error code returned by the library consists of a word. The MSB of this code corresponds with a component code, the LSB specifies the exact error which was reported by the component. To increase user friendliness, the codes are translated into human readable strings by a lookup table (see `error_strings.h` for details). See Figure 3.9, 3.10 and 3.11 for the corresponding diagrams.

Component (category/group) identifier (codes): Each of the components has its own ID, primarily to insert it into the return values (status/error messages) of the individual functions. This way the caller can determine the origin of a malfunction more easily. See Table 3.3 for a list.

The error specifier may have a different meaning in different components. See the particular module for details.

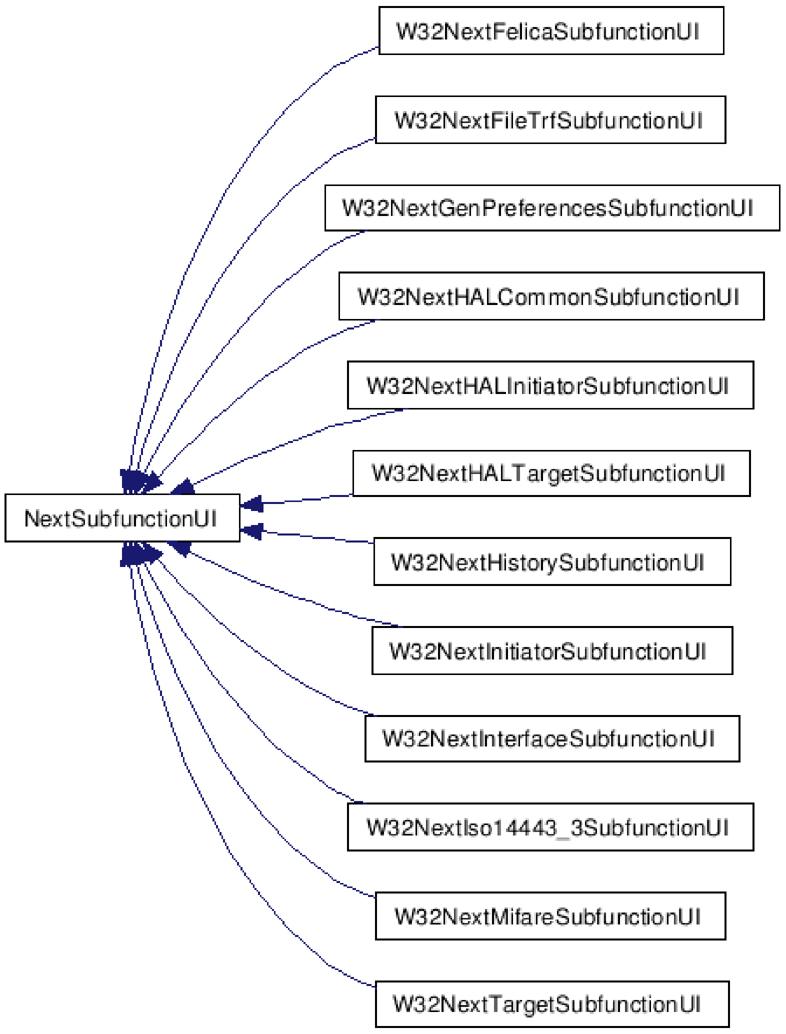

W32NextFelicaSubfunctionUI
W32NextFileTrfSubfunctionUI
W32NextGenPreferencesSubfunctionUI
W32NextHALCommonSubfunctionUI
W32NextHALInitiatorSubfunctionUI
W32NextHALTargetSubfunctionUI
NextSubfunctionUI
W32NextHistorySubfunctionUI
W32NextInitiatorSubfunctionUI
W32NextInterfaceSubfunctionUI
W32NextIso14443_3SubfunctionUI
W32NextMifareSubfunctionUI
W32NextTargetSubfunctionUI

Figure 3.5: Subfunction UI inherit graph

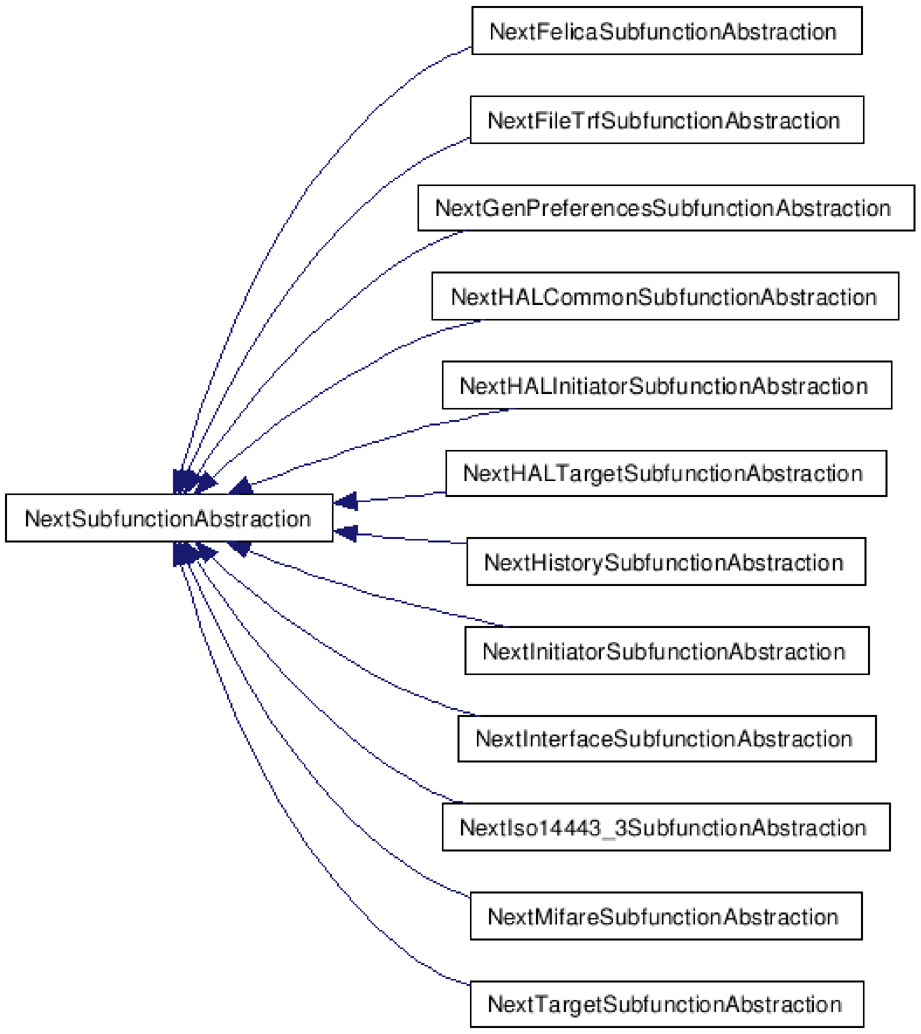

NextFelicaSubfunctionAbstraction
NextFileTrfSubfunctionAbstraction
NextGenPreferencesSubfunctionAbstraction
NextHALCommonSubfunctionAbstraction
NextHALInitiatorSubfunctionAbstraction
NextHALTargetSubfunctionAbstraction
NextSubfunctionAbstraction
NextHistorySubfunctionAbstraction
NextInitiatorSubfunctionAbstraction
NextInterfaceSubfunctionAbstraction
NextIso14443_3SubfunctionAbstraction
NextMifareSubfunctionAbstraction
NextTargetSubfunctionAbstraction

Figure 3.6: Subfunction abstraction inherit graph

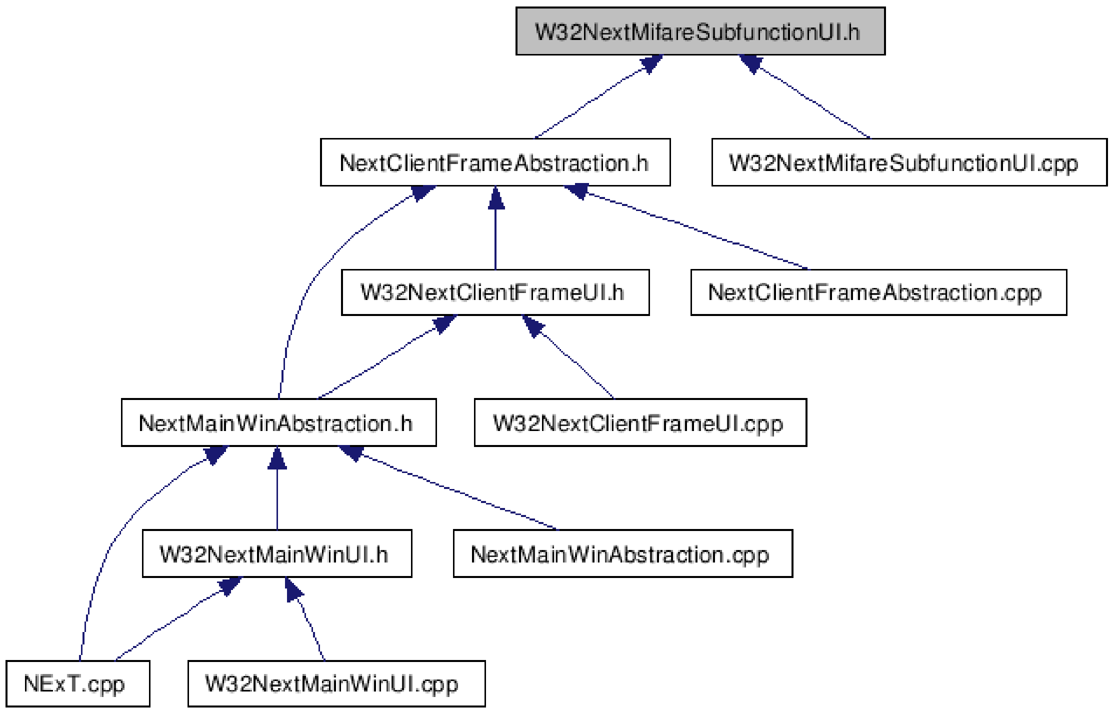

W32NextMifareSubfunctionUI.h
NextClientFrameAbstraction.h
W32NextMifareSubfunctionUI.cpp
W32NextClientFrameUI.h
NextClientFrameAbstraction.cpp
NextMainWinAbstraction.h
W32NextClientFrameUI.cpp
W32NextMainWinUI.h
NextMainWinAbstraction.cpp
NExT.cpp
W32NextMainWinUI.cpp

Figure 3.7: Mifare subfunction UI dependent includes

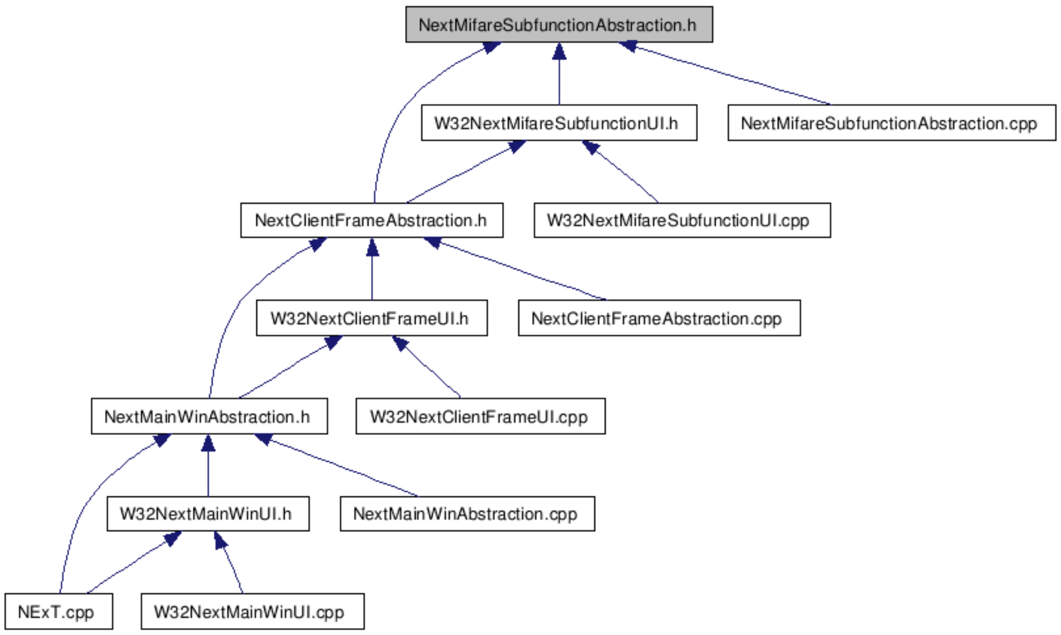

NextMifareSubfunctionAbstraction.h
W32NextMifareSubfunctionUI.h
NextMifareSubfunctionAbstraction.cpp
NextClientFrameAbstraction.h
W32NextMifareSubfunctionUI.cpp
W32NextClientFrameUI.h
NextClientFrameAbstraction.cpp
NextMainWinAbstraction.h
W32NextClientFrameUI.cpp
W32NextMainWinUI.h
NextMainWinAbstraction.cpp
NExT.cpp
W32NextMainWinUI.cpp

Figure 3.8: Mifare subfunction abstraction dependent includes

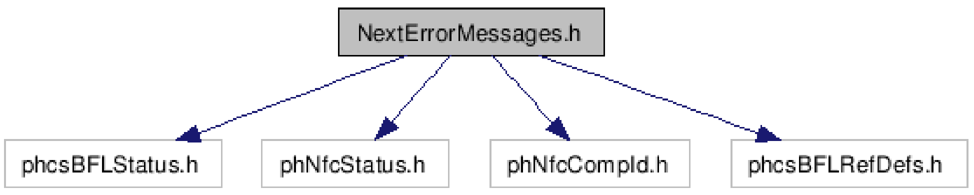


Figure 3.9: Error message generation includes

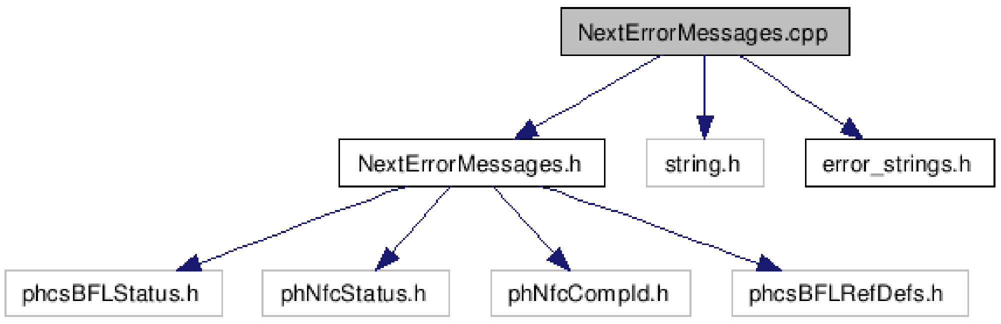


Figure 3.10: Error message generation dependencies

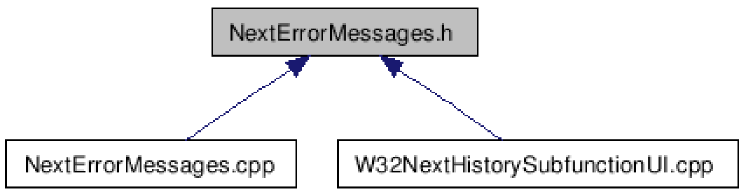


Figure 3.11: Error message generation references

Define	**Description**
PH_ERR_BFL_COMP_MASK	Mask to filter the component identifier and hide the error specifier
PH_ERR_BFL_BAL	BAL component
PH_ERR_BFL_REGCTL	Register Control component
PH_ERR_BFL_IO	IO component
PH_ERR_BFL_OPCLT	Operation Control component
PH_ERR_BFL_POLACT	Passive Polling Activation component
PH_ERR_BFL_I3P3A	ISO14443-3A component
PH_ERR_BFL_AUX	Auxiliary component
PH_ERR_BFL_NFC	NFC component
PH_ERR_BFL_MFRD	Mifare Reader component
PH_ERR_BFL_I3P4AACT	ISO14443-4A Activation component
PH_ERR_BFL_I3P4	ISO14443-4 component
PH_ERR_BFL_IDMAN	ID Manager component

Table 3.3: List of component codes

Chapter 4

Implementation

The program "NExT" (NFC Exploration Tool) is an application with a GUI for operating the Joiner[1] evaluation boards on a Windows PC, based on the BFL and HAL libraries.

4.1 Hardware

4.1.1 NFC Transmission Modules

Fully integrated ICs allow easy integration of NFC functionality into electronic devices. Equipped with a variety of interfaces, NFC transmission module ICs can easily be linked to other microcontrollers within the system.

In the following sections NFC ICs of NXP Semiconductors are introduced [57].

PN511 transmission module

The PN511 ("Joiner") is a highly integrated transmission module for contactless communication at 13.56 MHz. This transmission module utilises an outstanding modulation and demodulation concept completely integrated for a variety of passive contactless communication methods and protocols at 13.56 MHz. It also supports ISO 18092, MIFARE and FeliCa reader/writer modes and can act as a contactless smart card in combination with a security controller IC.

Key features:

- Supports ECMA 340 and ISO/IEC 18092 NFCIP-1 Standard with transfer speeds up to 424 kbit/s
- Supports ISO 14443A, MIFARE and FeliCa reader/writer mode can act as a smart card in combination with a security controller IC
- Hard reset with low power function, software power down mode
- Supported host controller interfaces:
    - SPI interface up to 10 Mbit/s

[1]Joiner is the internal code name for the PN511 transmission module

  - I^{2}C interface up to 400 kbit/s in Fast mode, up to 3,400 kbit/s in High-speed mode
  - Serial UART interface to 1.2 Mbit/s, framing according to the RS-232 interface with voltage levels according pad voltage supply
  - 8 bit parallel interface with Address Latch Enable (only available in HVQFN40 package)

- 2.5–3.3 V power supply

PN512 transmission module

In addition to the features of the PN511 this IC provides ISO 14443 type B reader/writer support.

PN531 smart transmission module

The PN531 (code name "TAMA") IC uses a 80C51 processor with 32 kB ROM and 1 kB RAM. It fully supports ISO 18092, MIFARE and FeliCa read/write modes. It can also act as a smart card in combination with a security controller IC. Furthermore the embedded firmware and the internal hardware support the handling and the host protocols for USB 2.0, I2C, SPI and serial UART interfaces.

Key features:

- 80C51 microcontroller core with 32 KB ROM and 1 KB RAM
- Highly integrated analogue circuitry to demodulate and decode card response
- Buffered output drivers to connect an antenna with minimum number of external components
- Integrated RF level detector
- Integrated card mode detector
- Integrated hardware and embedded firmware support for:
  - ISO 14443A reader/writer mode
  - MIFARE Classic encryption and MIFARE higher baud rate communication up to 424 kbit/s
  - Contactless communication according to the FeliCa scheme at 212 kbit/s and 424 kbit/s
  - NFC standard ECMA 340: NFCIP-1 interface and protocol
- Supported host interfaces:
  - USB 2.0 full speed device
  - SPI interface
  - I^{2}C interface
- 2.5–3.3 V power supply

4.1.2 Evaluation kit

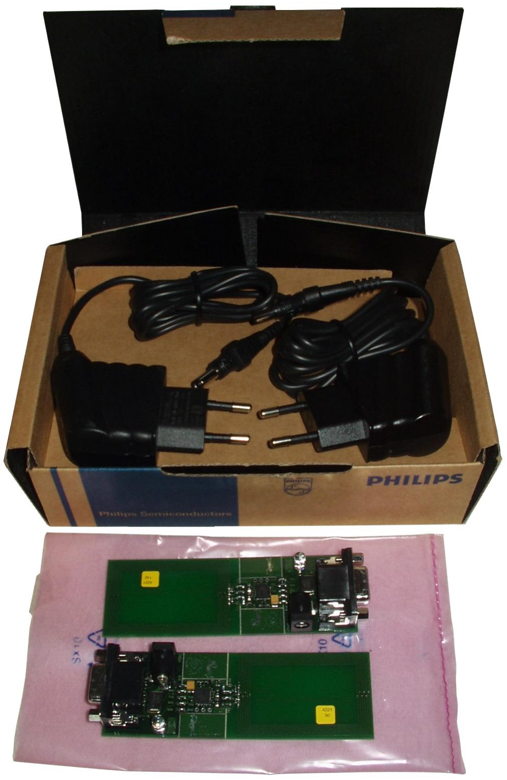


Figure 4.1: Joiner evaluation kit

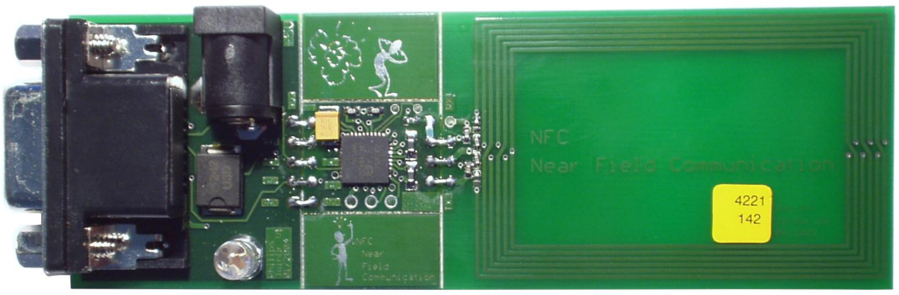


Figure 4.2: Joiner serial evaluation board

The evaluation kit consists of two Joiner evaluation boards and two mains adaptors.

The evaluation board integrates the PN511 and all necessary peripherals on a three-module PCB.

Interface

The first module provides a constant voltage power supply with a protection diode against polarity reversal and a serial interface converter (Analog Devices ADM3202: high speed, 2-channel RS-232/V.28 interface device), to enable connection between the PN511 and a standard RS-232 (ANSI EIA/TIA-232-F-1997) serial interface, including the charge pump capacitors.

Main module

This module carries the PN511 IC and some external components (quartz, blocking capacitor).

Antenna

On this part a printed antenna with six turns and its matching circuit is located.

4.2 Software

4.2.1 Basic Function Library (BFL)

The *Basic Function Library* or *RF Component Library (RCL)* is a unitised library, written in C. This decision was made to be able to use the library on a microcontroller platform with limited resources. For a convenient implementation in a high level language, a C++ wrapper is also available. Figure 4.3 shows the architecture of the BFL.

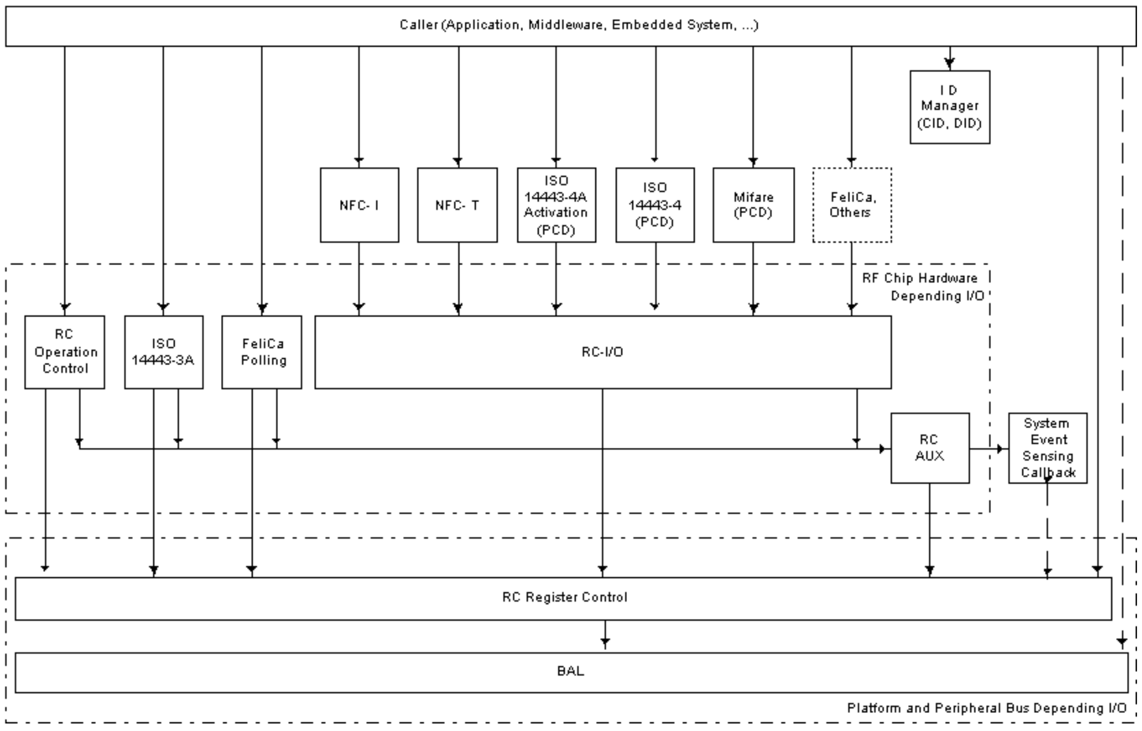

Caller (Application, Middleware, Embedded System, ...)
I D Manager (CID, DID)
NFC- I
NFC- T
ISO 14443-4A Activation (PCD)
ISO 14443-4 (PCD)
Mifare (PCD)
FeliCa, Others
RF Chip Hardware Depending I/O
RC Operation Control
ISO 14443-3A
FeliCa Polling
RC-I/O
RC AUX
System Event Sensing Callback
RC Register Control
BAL
Platform and Peripheral Bus Depending I/O

Figure 4.3: BFL Architecture

4.2.2 Hardware Abstraction Layer (HAL) Library

The Hardware Abstraction Layer is an additional library which provides some high-level functions.

The HAL has a layered design, the API is independent from different hardware. For a diagram of the HAL framework see Figure 4.4; a logical flow for using it is shown in Figure 4.5.

The implementation of the library was not finished at that time and the interface not completely defined, so the integration was a little bit experimental.

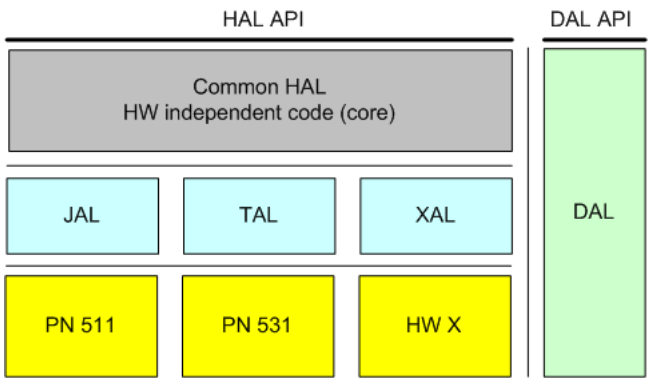


Figure 4.4: HAL structure

Used abbreviations:

DAL	Driver Abstraction Layer
HAL	Hardware Abstraction Layer
JAL	Joiner Adaptation Layer
TAL	Tama Adaptation Layer

4.2.3 Documentation with Doxygen

Doxygen [69] is a tool for creating documentation of various programming languages. At the beginning a configuration file has to be created. Doxygen then generates a cross-linked documentation, which contains an automatically generated file list, information about the data structures and the class hierarchy (textual and graphical). The functions, methods and members can be described inline and this information is then extracted for the documentation. The output can be in HTML and/or an offline reference manual in $\LaTeX$, RTF (Rich Text Format), PostScript, hyperlinked PDF, compressed HTML, and Unix man pages.

A big advantage of this system is that a deviation of the documentation from the implementation can be avoided, because an update can be performed simultaneously, since both parts are located in the same file.

4.3 NFC Exploration Tool (NExT)

The implementation was done with "Microsoft Visual Studio 2003 .NET". The documentation is inline source code and was extracted by Doxygen.

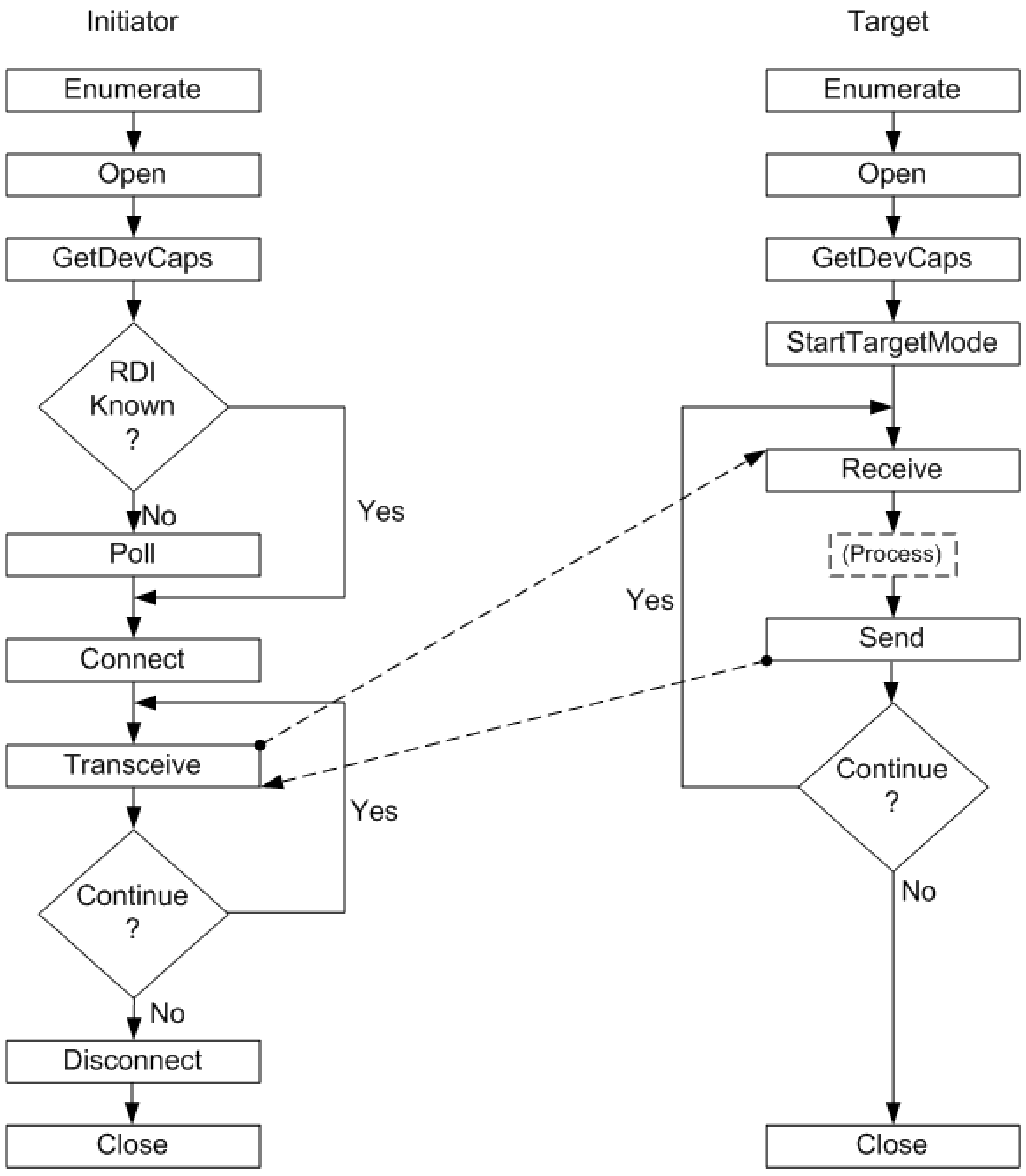


Figure 4.5: Logical flow for using the HAL

4.3.1 GUI parameter processing

Hexadecimal inputs

Each hexadecimal input checked for the correct length for the corresponding input. The Mifare authentication key, for example, requires 6 bytes (12 nibbles $\hat{=}$ 12 characters). If the user inputs or pastes less, the missing nibbles are filled up with zeros.

Bidirectional ASCII input

To enable comfortable text input, an ASCII input box is provided when applicable. The input in this control is automatically translated into the hexadecimal values. This works also the other way around; if the hexadecimal input control is edited, the values are automatically translated into ASCII text.

Mifare parameter

Mifare value block: The “Mifare value block” is a special format for ensuring data integrity for values. Therefore a value is saved in a special format. The value may consist of maximal 4 bytes, which leads to the maximum number of 4,294,967,295. The decimal value is converted into a hexadecimal number and stored LSB first. The value is then stored inverted and non-inverted once again. After the value, a block number can be specified, which is stored four times (non-inverted, inverted, non-inverted, inverted). See Table 4.1 for details. If a value is specified, the hexadecimal data is automatically prepared.

Byte number	Content
1	Value byte 1 (LSB)
2	Value byte 2
3	Value byte 3
4	Value byte 4 (MSB)
5	Value byte 1 inverted
6	Value byte 2 inverted
7	Value byte 3 inverted
8	Value byte 4 inverted
9	Value byte 1
10	Value byte 2
11	Value byte 3
12	Value byte 4
13	Block number
14	Block number inverted
15	Block number
16	Block number inverted

Table 4.1: Mifare value block format

This means that if hexadecimal data is read or input, it is checked for compliance with the Mifare value block format. If it does, the decimal value is automatically displayed.

4.3.2 Internal state machines

Interface state machine

This contains a very simple state machine, consisting of only two states, which only remembers whether the interface is closed or was successfully opened. A state change triggers an event, which enables or disables the corresponding subwindows.

With a closed interface only the interface configuration and the history window is enabled, and in any case, the user cannot use any other functions.

BFL Initiator state machine

This state machine guides the user through the activation process in passive and active operation mode.

Figure 4.6 shows the state diagram and its possible transitions. Table 4.2 lists the

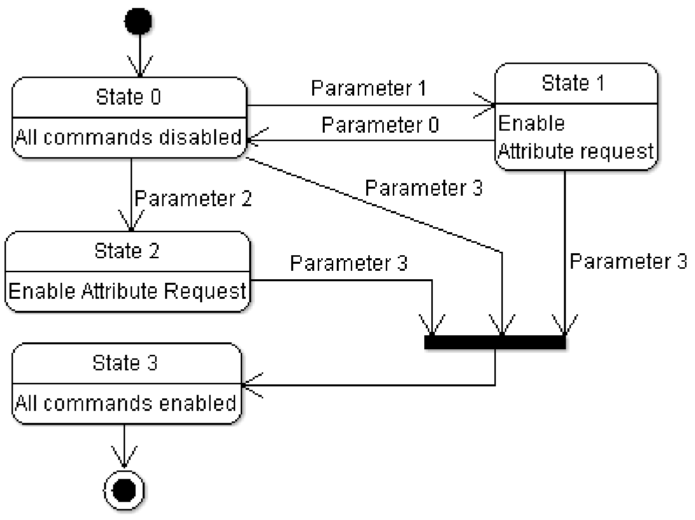


Figure 4.6: Initiator command button state diagram

Action	Parameter
Operation mode set to passive	0
Operation mode set to active	1
ISO 14443-3 activation successful	2
Attribute Request command successful	3

Table 4.2: State updates of BFL NFC Initiator, showing used parameter

state updates, depending on certain actions. Closing the interface resets the state machine.

4.3.3 Extension: Integrating the Hardware Abstraction Layer library

To integrate the HAL library which was still under development at the time, two additional frames were inserted. One for the HAL Initiator, another one for the HAL Target.

Chapter 5

Testing

Testing is also very important in software development, its effort is often underestimated. Tests can be performed automatically and/or manually. But the effort to cover all important checks automatically is too high and doesn't offer enough flexibility, because only a small change to the application can lead to the need for fundamental changes of the tests. Additionally some components and procedures can hardly be tested automatically, for instance the installation process may require reboots of the test machine, a clean installation of the operating system, additional components and frameworks. All this depends on the particular target system and it is hard to predict the required steps of the installation process. So the automatic tests are supplemented by manual tests. To avoid forgetting some tests, a check list for these checks is provided.

5.1 Test subjects

- The application itself
  - Functionality
  - Usability
  - Stability
  - Failure tolerance
- Deployment process
  - Installation
  - Uninstallation
  - Release notes
- Legal issues
  - Licensing
  - EULA
  - Export control

5.2 Development tools

5.2.1 GUI testing

To be able to test GUIs automatically, additional software is needed, which performs the clicks on the GUI and simulates key strokes. Such a tool is, for instance, AutoHotkey [9] (free, open-source, GNU GPL license). Additional challenges of GUI testing are that windows can be moved around on the screen, can be resized, closed, minimised or configured in another way. So it is not enough to just perform a click on a static position on the screen to push a button after starting a GUI. To be able to locate GUI resources on the screen and to evaluate outputs of the GUI, more complex software is required. Two commercial tools which can perform such tasks are, for instance, Ranorex [36] and Rational Robot [41].

5.2.2 Source inspection tools

Source code analysers can already identify some issues within the source code at an early stage. Some example of what can be identified on source code level:

- Coding standards
- Naming conventions
- Common coding mistakes (including thread synchronization problems, misuse of API methods)
- Unused local variables
- Empty catch blocks
- Unused parameters
- Empty 'if' statements
- Duplicate import statements
- Unused private methods
- Cyclic dependencies
- Evaluation of uninitialised variables
- Dead (unreachable) code
- Conditional statement with a condition which is always true or always false

Such tools can also perform statistics generation, syntax analysis and produce a visual component diagram showing the relationships, dependency graphs, inheritance diagrams and collaboration diagrams.

5.3 Common issues

Although the functionality may be completely different for various applications, the problems and bugs of applications with GUI are often the same. Some issues are listed below.

- Dialogues shall use clearly understandable button labelling which clearly states what happens. For a settings dialogue it is common to use “OK” (accept changes and exit the dialogue), “Apply” (accept the changes but don’t exit) and “Cancel” (exit and drop changes). For additional functionality like “Reset” it should be clear whether only the current changes are undone or everything is reset to default values. No choices with unclear behaviour should be used, e.g. “Save” or “Accept” in a settings dialogue (the changes are saved, but it is not clear whether the dialogue is closed or not), “Close” or “Exit” (it does not state what happens to the changes, whether they are saved or not).
- Multiple instances of the application can be allowed, denied or configurable; depending on that behaviour some issues have to be considered.
- If multiple instances are not allowed, opening another one shall be prevented by the application.
- If multiple instances are allowed, it must be able to handle possible problems with commonly used resources. The application should also ensure that session-dependant settings do not interfere with other instances of the application in a undesired way (e.g. by concurrent access to the setting files).
- When certain windows can be closed, there must be an option to open them again. Every element which can be hidden must have an option to show it again.
- In fully configurable GUIs there should be a possibility to reset all GUI elements to a default position and shown/hidden state.
- Damaged or deleted application setting should not result in a crash of the application or even prevent it from starting.
- If more than one version of the same application can be installed on the same machine, the version dependent preferences and settings should not interfere each other.
- If the application has a problem during startup, the error should be reported to the user even when the GUI was not loaded (e.g. with a popup or a notification that the problem was saved in a log file). Otherwise it’s very hard for the user to identify the problem when the application terminates right after startup without any response.
- When closing the program and saving of project settings or other data is available, the software should show a dialogue whether the changes should be saved if it was changed. This choices should be clearly understandable what happens to the changes, e.g. Save/Dismiss changes/Cancel rather than a twice negative question with the choice Yes/No.
- If the application has dependencies (e.g. Microsoft .NET framework), the application should detect and report a missing dependency instead of failing to start.

- Popups should be used very sparingly and only for important notifications where the user should be forced to acknowledge the information by clicking “OK” or for dialogues where a decision of the user is required. Other information can be written out into a log window rather then annoying the user by requiring when to click on a button when no interaction is necessary.

5.4 Special controls and GUI functionality

5.4.1 Inputs with a certain length

Some inputs require a certain length, e.g. encryption keys. For a good usability some issues have to be considered:

- The controls should reflect the required data (length) of the parameters to assist the user

- If the user inputs less data than required, it might be desired to fill up the data automatically. For numbers, inserting leading zeros could be a solution; for other data, e.g. ASCII text, additional spaces (0x20) or string terminators ('\0') could be appended at the end. The preferable concept depends on the intended purpose of the data and the application. This adjustments should be well recognisable by the user to avoid misunderstandings.

5.4.2 Controls with special input limitations

For instance hexadecimal input boxes: For decimal number inputs the framework provides usually a control or an option for a control to limit the inputs to decimal numbers, but hexadecimal inputs may be implemented by themselves. This requires certain issues to be considered:

Input odd number of characters: One ASCII character corresponds to a nibble, but a hexadecimal number consists of two nibbles. So, only even numbers of characters are allowed, this must be handled by the control and reported to the user, e.g. by highlighting the parameter and informing the user about the problem or filling in the missing character with '0'.

Upper and lower case characters: Apart from the decimal numbers, the letters 'A'..'F', respectively 'a'..'f', are used for hexadecimal numbers. These letters can be input in upper case, lower case or mixed. Despite the representation on the GUI, it has to be handled correctly internally.

Copying & pasting: Copy and paste can be performed by the menu, the mouse or by keyboard shortcuts. This is different for the control compared to the manual input and also has to be handled correctly. Also invalid characters may not be pasted into the control.

Interactivity: All these character restrictions, special limitations and plausibility checks can be realised in different ways. An automatic background correction might be comfortable but also can conceal typing errors, which is not intended. In contrast highlighting of the control or invalid character(s) forces the user to verify the input.

Background processing: Depending on the implementation, the data within a control may be invalid. It has to be ensured that no data is processed as long as it is invalid, regardless of whether it is adjusted in the background or not. The user should be informed that it is not possible to proceed without correcting the data before, should they try to do so.

Readability: For hexadecimal input, it is easier to read when the two nibbles of a byte are grouped together

5.4.3 Input data representation

For text inputs it might be an advantage to be able to input the text plain and have an additional hexadecimal representation. This can help to avoid problems with the different character sets and white spaces (for instance different line breaks CR LF '\r\n'/LF '\n'/CR '\r'; tabulator, which is also used to change to the next control, ...). This is very useful if the text is low level evaluated.

5.4.4 Library calls, processing time

Pressing a button may trigger a library call or initiate a time-consuming computation. To avoid the GUI being blocked during long-lasting operations, it is a good practice that the GUI runs in a separate thread to stay responsive. If the time is predictable, a progress bar is of advantage. Until the operation completes, some functions may not be available. This should be indicated for the user, for example by temporarily disabling particular buttons. If possible, it is beneficial if the running action can also be cancelled.

An example of this is a button to open an interface. Right after the click on the button, the button is disabled and the application tries to open the interface. If the process was successful, the button stays disabled and a log reports a successful operation; if the process failed, the problem is reported in the log and the button to open the interface is enabled again.

5.5 Automated tests

Automatic tests are very handy to quickly check a new build's functionality. Due to the automatism, tests can be repeated as often as necessary, with a constant quality and no need of an operator for the whole test. They should cover the most important functionality and ensure an easy verification after each new build before release. The advantages of automatic tests are that they can verify the application with very low manpower requirements, checks are always performed with the same quality and deviations in implemented checks cannot be overlooked. So the automatic tests provide a good possibility to get a quick impression of whether something obviously went wrong. For automatic tests, especially for the installation process, a virtual machine environment is a good approach.

For this project it would be unnecessary to implement these tests, because the effort for implementation is much higher than repetitive manual tests.

5.6 Manual tests

These tests are very important to supplement the automatic tests, because only the user can give impression of the design of the interface and evaluate the look and feel of the GUI – the interface between the human and the machine.

- Mouse clicking
  - Mouse clicks can be simulated by software, but some unintended behaviour may not be discovered automatically, because an automatic test can only check intended events meaning additional manual tests are beneficial.
- Installation process
  - Some optional components of the operating system may be needed to be installed for the application. This is depends on the operating system, its version, patch level and prior installed components. This requires a high effort to be performed automatically.
  - The computer often needs to be rebooted to complete the installation process. This is also hard to handle for an automatic test.
- Resizing windows
  - The look and feel of the application after resizing components cannot be evaluated automatically, because this is somehow a subjective impression of the user. If some elements are dynamically arranged, some labels may be not completely visible any more.
- Re-ordering elements and customisation
  - Some elements (windows, toolbar icons, menu entries, keyboard shortcuts) could be probably customised. The usability and appearance needs to be checked manually.
- Usability
  - Manual tests can check the intuitiveness and usability of the implementation. An advanced user should be able to operate the program without reading a lot of documentation.

5.6.1 Checklist for manual testing

For repetitive manual tests, a checklist is very useful to avoid forgetting to check any issue.

This checklist is held in keyword style; for a production environment execution instructions and defined pass/fail criteria should be clearly stated.

Software release

Final build

- Are conditional debug flags unset for the release build?
- Is the application build in *release* rather than in *debug* mode?
- Does the build have a version number and is it correct?

Deployment process

Standard installation (setup)

- Is a check performed by the installer whether compatible platform used (OS version)? Does it report incompatible operating systems?
- Is the installer able to resolve dependencies (required frameworks etc.)?
- Is meaningful default installation directory suggested?
- Does the installation also succeed without administration permissions? If not, are administrative permissions requested by the application or is a understandable error message shown by the installer?
- Are there problems when installing several (different) versions of the application? If yes, it should be prevented by the installer or by prior uninstallation of the old version.
- Does the installation succeed when the install package is located on a write protected media or network source?
- Does the installer create the desired links (shortcuts) to the application correctly?
- Is export controlled SW included in the distribution package (e.g. cryptopp.dll)?
- Does the installer include an EULA?
- Does the installer create uninstallation links?

Multiple installations

- Check whether the installer can detect an already installed version of the application.
- If an older version of the application is already present, check whether the installer can update the old version, remove the old version prior to installing the new version and/or whether both versions can be installed in parallel successfully.
- If another version of the application is already present, check if the installer can adopt compatible settings.

Uninstallation

- Check the removal process.
- No parts of the installation should remain (except user settings, if desired).

- For Windows OS: check "%ProgramFiles%\installation directory", "%CommonProgramFiles%", shortcuts on Desktop, Quick Launch bar and Start Menu; optionally registry settings and "%AppData%", "%UserProfile%\Local Settings\Application Data".
- If multiple versions of the program are present, check that the "Uninstall" only removes the parts which are used exclusively by the particular version. These should then be removed, leaving the shared components behind.
- Try to uninstall the software while the application is running. To succeed, the uninstall process will have to quit the application or perform the task at the next restart of the operating system. If possible, the user should have the choice of the procedure and get informed about the pending operations.

Application

General issues

- Are multiple program instances prevented?
- If multiple program instances are allowed, check that there are no side effects (especially on application settings and shared resources).
- Is a missing dependency (driver, framework, library) reported?
- Does the application work properly on x86 and x64 operating systems?
- Is a help system integrated (F1)?
- Localisation: If multiple languages are supported, does the different text fit into the controls?
- Is a update functionality provided?
- Is a licence management required and present?

Standard GUI functionality

- Check the ability of inserting and copying values by mouse and keyboard.
- Check the ability of keyboard navigation on the GUI.
- Is navigating through the application elements (user dialogues) with the "tab" key possible?

GUI related issues

- Are the changeable settings, disabled settings and displayed values easy to distinguish and recognise?
- Does the character restriction at inputs work properly, with keying in the data and copy & paste (e.g. if only hex characters are allowed)?
- Is a hexadecimal value input with upper/lower case characters possible?
- Check illegal commands and parameters for particular states?
- Check that resizing works properly (visibility of items; automatic rearrangement of items; (dis)appearing scrollbars etc.).
- Can results and logs be copied from the GUI as text?

Operation

- Verify the intended functionality.

- Trigger errors (e.g. try to execute functions which are not allowed in the current state or leave some required inputs empty) and check the behaviour.

Error handling: The normal flow can be disturbed by outer influences, e.g. the network cable can be disconnected during data transfer.

- File access
- Network resources
- Other interfaces (serial, USB)

Periphery: This includes standard peripherals like keyboard and mouse, but also special hardware for the software, like RFID readers.

- Trigger errors and check the behaviour.
- Check the behaviour when needed peripherals are missing.
- Check the behaviour when a driver for a required peripheral is missing.
- Check the behaviour when a required peripheral is not powered on.

Usability

- Inputs and outputs can be easily distinguished?
- Disabled inputs should be easily recognisable, e.g. greyed out rather than only write protected.

Stability

- Does the memory consumption increase when the application is running a long time (additional to reasonable requirements)?
- Does the application crash when performing actions very fast?
- Does the application stay responsive and stable when running a long time?

Miscellaneous

- Version number present and updated at every location where included (application, installer, release notes, documentation)?
- Release notes included and updated?
- No licenses of third-party software violated (e.g. GNU/GPL)?
- Any licences of third-party software required (e.g. libraries, frameworks)?

Chapter 6

Experimental results

6.1 Examples

This section describes how to set up the test installation and represents a kind of quick-start guide for the software.

6.1.1 Setup

Requirements:

- Joiner Evaluation Kit
- PC with MS Windows operating system and two free RS-232 serial interfaces
- Two RS-232 extension cables
- NExT application software

The setup is quite easy: Connect the two evaluation boards to the serial ports of the PC using the extension cables, attach the power supplies and arrange the two boards with overlapping antennae.

6.1.2 Sample communications

NFC Initiator and Target operation

Start the application `NExT.exe` and configure the COM settings in Options – BFL Interface. Then open the target window in the menu by selecting Mode – BFL – NFC Target, choose a COM port and open it. Then activate "auto respond" of the "Interactive operation" group box. Now the history window should inform about communication standby.

Then open the Initiator window by selecting "Mode – BFL – NFC Initiator" in the menu. In the Interface subwindow select a COM port and open it by pressing the button of the same name. Next the activation in the "NFC 106 kbps passive" group box follows: SENS_REQ – SDD/SEL_REQ.

Retrieve UID from a MIFARE card

Disable one evaluation board by software or unplug it and place a MIFARE card on its antenna. Interface: Open. NFC 106 kbps passive: ‘SENS_REQ’ – ‘SDD/SEL_REQ’. Then the 4 byte UID can be read in the NFCID1 text box.

Chapter 7

Conclusion

7.1 Summary

Thanks to the modular design and interface to the software library, a parallel development, update and extension of the particular parts it is easily achievable.

The stand-alone application with the integrated library has also shown a lot of advantages, because transferring the setup to another computer is very easy: simply copy the application and connect the hardware. There is no need to set up a compatible link library environment, no other requirements except a Windows operating system and at least one free serial interface.

7.2 Usability

The usability of a GUI based application is much better than those of a command line one. It is much easier to operate, because there is no need to search the manual for appropriate command line parameters. User guidance can also be offered on a GUI to suggest reasonable subsequent actions. With some basic knowledge about NFC principles it is possible to operate the evaluation board without prior knowledge of the software.

This could be driven even further by offering macros which already provide high-level functionality by combining several low-level functions.

Practical experience shows that it is very helpful when the software is tried out and evaluated by a person other than the programmer. Because the programmer obviously knows the program, whilst another person first tries to find elements intuitively. In this way the program can be optimised for other users.

7.3 Future work

This chapter summarises some possible improvements of the NFC Exploration Tool.

7.3.1 Possible improvements on the Graphical User Interface

Since the program should be easily portable to other operating systems, the focus was on dividing the software in a functional and a Windows depended part. Due to this, some standard Windows functions are not yet implemented and cannot be used.

Possible improvements to the graphical user interface:

- Scrollbars in the main window
- List of "opened windows" in the menu at entry "Window"
- Automatically align windows
- Support of keyboard shortcuts
  - "Tab" for selecting the next control on the graphical user interface (in a certain order)
  - "Strg-Tab" for changing sub-windows

7.3.2 Possible functional improvements

- Support of several interfaces instead of just the serial one
- Additional hardware support
- Support of standardised layer 4 (application layer) protocols
- Providing a script interpreter and recorder

7.3.3 Other possible improvements

- Cross-platform availability
- Improvements on testability

7.3.4 Details of improvements

Scripts

The support of scripts would be very useful and not only for demonstration purposes. The following functionality could be practical:

- Configuration commands (for selecting the device, configuring the serial interface)
- Standard ISO commands
- Arbitrary "raw" commands for low level functionality
- Control functions (loops, branches, functions for timing)
- Cryptographic functions (for online calculation of session keys and so on)
- User interaction (value input, option selection)

A common script language should be used, that the user is not forced to learn another proprietary one.

Cross-platform availability

To avoid the need of porting the application for another operating system, the implementation could also be realised in Java [28]. Alternatively the usage of the Qt class library [19], which is also available for multiple platforms, would have been possible. But because of an already existing software skeleton, the implementation was done in C++ with the Microsoft Win32 API.

Improvements on testability

For better testability, an additional test interface could be included in the application. This additional functionality can be integrated into the application as conditional code, which is used only for testing without disclose it to the user of the software. If it is done this way, it has to be kept in mind, that the software which is tested is actually different to that one which is actually used. Therefore it is important that the used tool chain is reliable and stable and to consider the possibility of side effects. Also at least a basic system test is required for the release version.

A test interface can be used as “back door” to be able to perform module and unit tests. This leads to easier test cases and the possibility to test error scenarios, which cannot be triggered easily from outside. After the basic module tests, integration tests can ensure a proper interaction of the modules. And the final system test can then test the application on user level. An additional advantage of such a test interface is that also GUI testing is easier. Usually the GUI framework is integrated in the development environment and trusted as stable. So instead of the need of additional software to simulate button clicks on the GUI for automatic testing and the effort of test development overhead to locate the GUI resources, for example a socket interface can be used to perform this task.

Bibliography

[1] 3rd Generation Partnership Project 3GPP. http://www.3gpp.org/.

[2] Manfred Aigner, Sandra Dominikus, and Martin Feldhofer. A System of Secure Virtual Coupons Using NFC Technology. *Pervasive Computing and Communications Workshops, 2007. PerCom Workshops '07. Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 362–366, March 2007.

[3] Wi-Fi Alliance. http://www.wi-fi.org/.

[4] WiMedia Alliance. http://www.wimedia.org/.

[5] ZigBee Alliance. http://www.zigbee.org/.

[6] The 1394 Trade Association. http://www.1394ta.org/.

[7] Wireless LAN Association. http://www.wlana.org/.

[8] Alliance for Telecommunications Industry Solutions (USA) ATIS (initially T1). http://www.atis.org/.

[9] AutoHotkey. Free, open-source automation utility for Windows. http://www.autohotkey.com/.

[10] J. Bravo, R. Hervas, C. Fuentes, G. Chavira, and S.W. Nava. Tagging for nursing care. *Pervasive Computing Technologies for Healthcare, 2008. PervasiveHealth 2008. Second International Conference on*, pages 305–307, 30 2008-Feb. 1 2008.

[11] China Communications Standards Association (CCSA). http://www.ccsa.org.cn/english/.

[12] Nokia Research Center. http://research.nokia.com/.

[13] Conférence Européenne des Administrations des Postes et des Télécommunications CEPT. http://www.cept.org/.

[14] CEPT/ERC REC 70-03 relating to the use of short range devices. Annex 9: Inductive applications.

[15] Yung-Chin Chen, Wei-Lin Wang, and Min-Shiang Hwang. RFID Authentication Protocol for Anti-Counterfeiting and Privacy Protection. *Advanced Communication Technology, The 9th International Conference on*, 1:255–259, Feb. 2007.

[16] Zhiqun Chen. *Java Card^{TM} Technology for Smart Cards*. The Java^{TM} Series. Addison-Wesley, June 2000.

[17] Atmel Corporation. http://www.atmel.com/.

[18] Sony Corporation. http://www.sony.net/.

[19] Qt cross-platform application and UI development framework. http://qt-project.org/.

[20] M. Csapodi and A. Nagy. New Applications for NFC Devices. *Mobile and Wireless Communications Summit, 2007. 16th IST*, pages 1–5, July 2007.

[21] Sandra Dominikus and Manfred Aigner. mCoupons: An Application for Near Field Communication (NFC). *Advanced Information Networking and Applications Workshops, 2007, AINAW '07. 21st International Conference on*, 2:421–428, May 2007.

[22] ECMA. European association for standardizing information and communication systems (former known as European Computer Manufacturers Association). http://www.ecma-international.org/.

[23] Ericsson. Telefonaktiebolaget LM Ericsson. http://www.ericsson.com/.

[24] ETSI. Broadband Radio Access Networks. http://www.etsi.org/bran or http://portal.etsi.org/bran.

[25] ETSI. European Telecommunications Standards Institute. http://www.etsi.org/.

[26] Klaus Finkenzeller. *RFID-Handbuch*, volume 3. aktual. und erw. Aufl. Hanser München, Wien, 2002. http://rfid-handbook.com/.

[27] NXP Semiconductors (former semiconductor division of Philips). http://www.nxp.com/.

[28] Oracle Corporation (formerly Sun Microsystems). http://www.oracle.com/technetwork/java/index.html, http://www.java.com/.

[29] NFC Forum. Specifications. http://www.nfc-forum.org/specs/spec_list/.

[30] Universal Serial Bus Implementers Forum. http://www.usb.org/.

[31] UWB Forum. Now defunct; the web address was http://www.uwbforum.org/.

[32] Wibree Forum. http://www.wibree.com/ (offline).

[33] WiMAX Forum. http://www.wimaxforum.org/.

[34] Free Software Foundation (FSF). http://www.fsf.org/.

[35] Wilbert O. Galitz. *The Essential Guide to User Interface Design*, volume Third Edition. Wiley, 2007.

[36] Ranorex GmbH. Test Automation Tools and Framework. http://www.ranorex.com/.

[37] GSM Association (GSMA). http://www.gsma.com/.

[38] Gerhard P. Hancke. Practical attacks on proximity identification systems. *Security and Privacy, 2006 IEEE Symposium on*, pages 6 pp.–, May 2006.

[39] Gerhard P. Hancke and Markus G. Kuhn. An RFID Distance Bounding Protocol. *Security and Privacy for Emerging Areas in Communications Networks, 2005. SecureComm 2005. First International Conference on*, pages 67–73, Sept. 2005.

[40] Ernst Haselsteiner and Klemens Breitfuß. Security in Near Field Communication (NFC)—Strengths and Weaknesses. IAIK http://events.iaik.tugraz.at/RFIDSec06/Program/papers/002%20-%20Security%20in%20NFC.pdf, 2006. Philips (now NXP) Semiconductors.

[41] IBM. Test automation tool for functional testing of client/server applications. http://www.ibm.com/software/awdtools/tester/robot/.

[42] IEC. International Engineering Consortium. http://www.iec.org/.

[43] IEEE. IEEE 802 LAN/MAN Standards Committee. http://ieee802.org/.

[44] IEEE. IEEE 802.15 Task Group 4. http://grouper.ieee.org/groups/802/15/pub/TG4.html.

[45] Motorola Inc. http://www.motorola.com/.

[46] Texas Instruments. RFid Systems' S4100 Multi-Function Reader (MFR) Evaluation Kit. http://www.ti.com/rfid/docs/manuals/pdfSpecs/RF-MFR-RNLK-00.pdf, http://www.ti.com/rfid/docs/manuals/refmanuals/rf-mgr-mnmn_ds.pdf.

[47] Infrared Data Association (IrDA). http://www.irda.org/.

[48] ISO. International Organization for Standardization. http://www.iso.org/.

[49] Telecommunications Technology Committee (TTC; Japan). http://www.ttc.or.jp/e/.

[50] Telecommunications Technology Association (TTA; Korea). http://www.tta.or.kr/English/.

[51] A. Lahtela, M. Hassinen, and V. Jylha. RFID and NFC in healthcare: Safety of hospitals medication care. *Pervasive Computing Technologies for Healthcare, 2008. PervasiveHealth 2008. Second International Conference on*, pages 241–244, 30 2008-Feb. 1 2008.

[52] Jerry Landt. *Shrouds of Time: The history of RFID*. AIM, Inc., 2001. http://www.aimglobal.org/, http://www.transcore.com/pdf/AIM%20shrouds_of_time.pdf.

[53] G. Madlmayr, O. Dillinger, J. Langer, C. Schaffer, C. Kantner, and J. Scharinger. The benefit of using SIM application toolkit in the context of near field communication applications. *Management of Mobile Business, 2007. ICMB 2007. International Conference on the*, pages 5–5, July 2007.

[54] G. Madlmayr, J. Langer, C. Kantner, and J. Scharinger. NFC Devices: Security and Privacy. *Availability, Reliability and Security, 2008. ARES 08. Third International Conference on*, pages 642–647, March 2008.

[55] Global mobile Suppliers Association (GSA). http://www.gsacom.com/.

[56] NFC Forum. http://www.nfc-forum.org/.

[57] NXP. NXP Semiconductors (former semiconductors division of Philips). http://www.nxp.com/products/interface_and_connectivity/nfc_devices/, http://www.nxp.com/products/identification_and_security/reader_ics/nfc_devices/.

[58] Institute of Electrical and Electronics Engineers. http://www.ieee.org/.

[59] Association of Radio Industries and Businesses (ARIB; Japan). http://www.arib.or.jp/english/.

[60] J. Ondrus and Y. Pigneur. An Assessment of NFC for Future Mobile Payment Systems. *Management of Mobile Business, 2007. ICMB 2007. International Conference on the*, pages 43–43, July 2007.

[61] M. Pasquet, J. Reynaud, and C. Rosenberger. Secure payment with NFC mobile phone in the SmartTouch project. *Collaborative Technologies and Systems, 2008. CTS 2008. International Symposium on*, pages 121–126, May 2008.

[62] Certified Wireless Network Professional. http://www.cwnp.com/.

[63] Freescale Semiconductor. Spin-off from Motorola in 2004. http://www.freescale.com/.

[64] Bluetooth SIG. Bluetooth Low Energy. http://www.bluetooth.com/Pages/Low-Energy.aspx.

[65] Bluetooth Special Interest Group (SIG). Official Bluetooth Technology Info Site: http://www.bluetooth.com/, Bluetooth SIG membership website: http://www.bluetooth.org/.

[66] Torsten Stapelkamp. *Screen- und Interfacedesign.* Springer Berlin, Heidelberg, New York, 2007.

[67] Esko Strömmer, Jouni Kaartinen, Juha Pärkkä, Arto Ylisaukko-oja, and Ilkka Korhonen. Application of Near Field Communication for Health Monitoring in Daily Life. *Engineering in Medicine and Biology Society, 2006. EMBS '06. 28th Annual International Conference of the IEEE*, pages 3246–3249, 30 2006-Sept. 3 2006.

[68] International Telecommunication Union. http://www.itu.int/.

[69] Dimitri van Heesch. Doxygen Documentation. http://www.doxygen.org/.

[70] J. Walko. A ticket to ride [Near field communications]. *Communications Engineer*, 3(1):11–14, Feb.-March 2005.

[71] WiBro Website. http://www.wibro.or.kr/.

[72] Palo Wireless. HomeRF. http://www.palowireless.com/homerf/about.asp, the original web site http://www.homerf.org/ is obsolete.

All web addresses checked in May 2013.

Appendix A

BFL API

This chapter describes the application programming interface of the BFL.

Library function: **phcs_BFL::phcsBflNfcInitator_Wrapper::Initialise**
Description: This function shall be called first to glue together the C-interface, its internal members parameter and to establish a link to the next lower layer. Moreover, the initialisation of operation parameters is done.
Parameters:

uint8_t tx_preamble_type [in]: Specifies the type of prefix the protocol shall add to the frame. Defined in `nfccommon.h`.

uint8_t rx_preamble_type [in]: Specifies the type of prefix the protocol shall expect at the beginning of each frame. Defined in `nfccommon.h`.

uint8_t *p_trxbuffer [in]: Pointers to pre-allocated TX/RX buffer. This is the internal frame-composition buffer with a size of 64..255 bytes.

uint16_t trxbuffersize [in]: Size of the TX/RX frame composition buffer.

uint8_t *p_nfcidt_buffer [in]: Pointer to a 10 byte pre-allocated buffer to receive the Target NFCID-3 upon ATR_RES.

class phcsBfl_Io *p_lower [in]: Pointer to the lower layer, the instance of a phcsBflIo_t structure.

Required GUI inputs: None
Return value: Void

Library function: **phcs_BFL::phcsBflMfRd_Wrapper::Initialise**
Description: The Initialise function does setup required by the component but not done by the default constructor.
Parameters:

phcsBfl_Io *p_lower [in]: Pointer to the C++ object of the underlying layer I/O.

uint8_t *p_trxbuffer [in]: Pointer to the system-wide TRx buffer, allocated and managed by the embedding software. The buffer serves as the source/destination buffer for the underlying I/O Transceive functionality.

Required GUI inputs: None
Return value: Void

Library function: **phcs_BFL::phcsBflI3P3A_Wrapper::Initialise**

Description: The Initialise method does setup required by the Component but not done by the default constructor.

Parameters:

pphcsBflI3P3A_Initialize_t c_iso_14443_3_initialise [in]: Pointer to the Initialise function of the type of C-kernel (for each hardware variant an own kernel must exist) to use with the current wrapper instance.

phcsBfl_RegCtl *p_lower [in]: Pointer to the C++ object of the underlying layer.

uint8_t initiator___not_target [in]: Specifier for operation mode (1 = Initiator, 0 = Target).

Required GUI inputs: None
Return value: Void

Library function: **phcs_BFL::phcsBflI3P3A_Wrapper::RequestA**

Description: This command handles the Request procedure of ISO14443-3. Depending on the Request Code (ISO14443_3_REQALL or ISO14443_3_REQIDL) and the state of the cards in the field all cards reply with their Tag-Type synchronously. The time between end of the Request command and start of reply of the card is exactly $8 \cdot 9.44\ \mu s$ long. The Tag-Type field is 16 bits long and only one bit out of 16 is set. Double and Triple serial numbers are possible.

Note: In case of an error, the appropriate error code is set. Nevertheless all received data during the RF-Communication is returned for debugging reasons.

Parameters:

phcsBflI3P3A_ReqAParam_t *request_a_param [in, out]: Pointer to the I/O parameter structure.

Required GUI inputs: None
Return value: phcsBfl_Status_t, unsigned 16 bit integer

PH_ERR_BFL_SUCCESS: Operation successful.

Others: Other error codes are propagated from lower layers.

Library function: **phcs_BFL::phcsBflI3P3A_Wrapper::AnticollSelect**

Description: This command combines the ISO14443-3 functions of anticollision and select. The functionality is split up into two independent internal procedures. One to perform the anticollision, the other one to perform the select. The cascade level is automatically increased if the Cascade Tag for a further level is received. The check-byte is verified, but not stored in the buffer. The sel_lv1_code contains the select code of cascade level 1. This is 0x93 for ISO14443-3 compatible devices. The select codes for cascade level 2 and 3 are chosen automatically according to the information in the SAK byte received. The UID references a buffer which may contain the known part of the UID when the function is called. When the function returns to the calling procedure, this buffer contains the received serial number of the target device. The length is according to the cascade level of the ID. The indicator of another cascade level (0x88) is not removed and also stored in the buffer. The uid_length parameter is the bit count (!) of the ID. As input it is the number of known bits, as output it shows the ID length, in bits, which is always a multiple of 32. At the end this function selects a card by the specified serial number. All other cards in the field fall back into the idle mode and they are not longer involved during the further communication.

Note: In case of an error, the appropriate error code is set. Nevertheless all received data during the RF-Communication is returned for debugging reasons.

Parameters:

phcsBflI3P3A_AnticollSelectParam_t *AnticollSelect_param [in, out]: Pointer to the I/O parameter structure.

Required GUI inputs:

Cascade level (sel_lv1_code)

Return value: phcsBfl_Status_t, unsigned 16 bit integer

PH_ERR_BFL_SUCCESS: Operation successful.

Others: Other error codes are propagated from lower layers.

Library function: **phcs_BFL::phcsBflI3P3A_Wrapper::Select**

Description: This command only performs the select of a known device. For detailed information on the parameters see the description of phcsBflI3P3A_SelectParam_t command. The sel_lv1_code is the first command to be sent to the passive device.
The UID contains the ID of the device. All 3 cascade levels can be transferred at once to the function. The uid_length parameter defines whether the ID is single, double or triple. At the end this function selects a card by the specified serial number. All other cards in the field fall back into the idle mode and they are not longer involved during the further communication.

Note: In case of an error, the appropriate error code is set. Nevertheless all received data during the RF-Communication is returned. This is done for debugging reasons.

Parameters:

phcsBflI3P3A_SelectParam_t *select_param [in, out]: Pointer to the I/O parameter structure.

Required GUI inputs:

Cascade level (sel_lv1_code)

Return value: phcsBfl_Status_t, unsigned 16 bit integer

PH_ERR_BFL_SUCCESS: Operation successful.

Others: Other error codes are propagated from lower layers.

Library function: **phcs_BFL::phcsBflI3P3A_Wrapper::HaltA**
Description: This function handles the halt command for ISO14443-3 in both the reader and the card mode.
In reader operating mode, this function sets a MIFARE (R) Classic compatible card into the halt state. Having send the command to the card, the function does not expect a cards response. Only in case of any error the card sends back a NACK. If the command was successful, the card does not return with an ACK. Thus, the function is successful, if a timeout is indicated. This is handled internally.
In card operating mode this functions evaluates the data received, sets the Halt flag of the PN511 and starts again the automatic anticolission procedure.
Parameters:

phcsBflI3P3A_HaltAParam_t *halt_a_param [in, out]: Pointer to the I/O parameter structure.

Required GUI inputs: None
Return value: phcsBfl_Status_t, unsigned 16 bit integer

PH_ERR_BFL_SUCCESS: Operation successful.

Others: Other error codes are propagated from lower layers.

Library function: **phcs_BFL::phcsBflNfcInitator_Wrapper::AtrRequest**
Description: This function issues an ATR_REQ and processed the Target's ATR_RES. The function is the entry point of the NFC protocol.
Parameters:

phcsBflNfc_InitiatorAtrReqParam_t *atr_req_param [in, out]: Pointer to the I/O parameter structure.

Required GUI inputs:

Initiator mode (passive/active)

NFCID3

DSi

DRi

LR (PPi, FSL)

General bytes

Use NAD switch

Return value: phcsBfl_Status_t, unsigned 16 bit integer. High byte (MSB): Category (group) Identifier; Low byte (LSB): Error Specifier.

PH_ERR_BFL_SUCCESS: Operation successful.

Others: Other error codes are propagated from lower layers.

Library function: **phcs_BFL::phcsBflNfcInitator_Wrapper::PslRequest**
Description: Immediately after ATR_REQ/ATR_RES, the protocol allows to change the operating parameters (bit rates and frame sizes). This operation is done using this command.
Parameters:

phcsBflNfc_InitiatorPslReqParam_t *psl_req_param [in, out]: Pointer to the I/O parameter structure.

Required GUI inputs:

DRI

DSI

Return value: phcsBfl_Status_t, unsigned 16 bit integer. High byte (MSB): Category (group) Identifier; Low byte (LSB): Error Specifier.

PH_ERR_BFL_SUCCESS: Operation successful.

Others: Other error codes are propagated from lower layers.

Library function: **phcs_BFL::phcsBflNfcInitator_Wrapper::SetTRxProp**
Description: The TRx buffer is internally used by the protocol to send and receive frames. This function is intended for use after ATR/PSL when new settings shall be applied and the caller wants to have full control over the memory available. The NFC protocol can insert a preamble before the composed PDU. Select the type of preamble with this function.
Parameters:

phcsBflNfc_InitiatorSetTRxPropParam_t *set_trx_properties_param [in, out]: Pointer to the I/O parameter structure.

Required GUI inputs: None
Return value: phcsBfl_Status_t, unsigned 16 bit integer. High byte (MSB): Category (group) Identifier; Low byte (LSB): Error Specifier.

PH_ERR_BFL_SUCCESS: Operation successful.

Others: Other error codes are propagated from lower layers.

Library function: **phcs_BFL::phcsBflNfcInitator_Wrapper::DepRequest**

Description: The DEP_REQ function exchanges user data (payload) with the Target. It includes all functionality according to the NFC specification (such as DID, NAD support, datalink functionality and error handling). When this function receives a request for timeout extension (RCLSTATUS_TARGET_SET_TOX) it exits. In this case the application must get the requested value (NfcIRequestedToxValue) apply it and restart the function. Then the function's behaviour is to exit, if everything goes well, a second time however with status RCLSTATUS_TARGET_RESET_TOX. This shall trigger the application to apply the old timeout values again with a subsequent second restart of the function. No modification of the parameter must be done in the meantime. The function also exits temporarily to indicate that the user buffer (the buffer the application specifies and hands over to this function as one parameter member) is not sufficiently large for all data to transfer. There are two cases:

- First, transfer form Initiator to Target: When the application sets the NFCI_CONTINUE_CHAINING bit in the flags member of dep_req_param the protocol is instructed to indicate "More Information" to the Target even if it is the last frame to be sent. Then the protocol stops with no data return, waiting to be called again to resume transfer with the next buffer. This mechanism can be called "Meta-Chaining" and enables the Initiator to send data of arbitrary length, exceeding 64K or the limit of available RAM which could be much less.
- Second, transfer from the Target to the Initiator: The application's responsibility is, among others, to provide RAM to the Initiator protocol for data return. The Target does not know how much the Initiator has of that resource. If the Target returns more data than the Initiator buffer (again the user buffer) can hold this function exits with RCLSTATUS_USERBUFFER_OVERFLOW, returning the data in the buffer. Additionally the NFCI_CONTINUE_CHAINING bit in the flags member of dep_req_param is set. The application can now save all data and resume receiving by simply calling the protocol again. Please note that the NFCI_CONTINUE_CHAINING bit must remain set for the second (or further) call(s), as long as the protocol signals this need. The transfer is finished when the bit is cleared and/or the status is different than RCLSTATUS_USERBUFFER_OVERFLOW.

The "Meta-Chaining" feature becomes especially useful in a peripheral with limited RAM resources.

Parameters:

phcsBflNfc_InitiatorDepReqParam_t *dep_req_param [in, out]: Pointer to the I/O parameter structure.

Required GUI inputs:

'NAD used' switch

If NAD is used: NAD

Return value: phcsBfl_Status_t, unsigned 16 bit integer. High byte (MSB): Category (group) Identifier; Low byte (LSB): Error Specifier.

PH_ERR_BFL_SUCCESS: Operation successful.

Others: Other error codes are propagated from lower layers.

Library function: **phcs_BFL::phcsBflNfcInitator_Wrapper::AttRequest**
Description: Under the hood, this function is a DEP_REQ. The Target confirms its presence with a DEP(Attention)_RES. Moreover, the function is used for error handling.
Parameters:

phcsBflNfc_InitiatorAttReqParam_t *att_req_param [in, out]: Pointer to the I/O parameter structure.

Required GUI inputs: None
Return value: phcsBfl_Status_t, unsigned 16 bit integer. High byte (MSB): Category (group) Identifier; Low byte (LSB): Error Specifier.

PH_ERR_BFL_SUCCESS: Operation successful.

Others: Other error codes are propagated from lower layers.

Library function: **phcs_BFL::phcsBflNfcInitator_Wrapper::DslRequest**
Description: The Deselect function moves the Target out of the protocol, before the SDD loop in case of passive communication mode. Targets in Active communication mode enter a special Deselect state. In order to exit this state a WUP_REQ is issued which makes it possible to apply a new DID.
All these steps allow a subsequent PSL_REQ, after SDD loop and ATR_REQ for passive or directly after WUP for active Target operation.
Parameters:

phcsBflNfc_InitiatorDslReqParam_t *dsl_req_param [in, out]: Pointer to the I/O parameter structure.

Required GUI inputs: None
Return value: phcsBfl_Status_t, unsigned 16 bit integer. High byte (MSB): Category (group) Identifier; Low byte (LSB): Error Specifier.

PH_ERR_BFL_SUCCESS: Operation successful.

Others: Other error codes are propagated from lower layers.

Library function: **phcs_BFL::phcsBflNfcInitator_Wrapper::RlsRequest**
Description: The RLS_REQ function exits the protocol completely (state before selecting the communication mode, RFCA, SDD).
Parameters:

phcsBflNfc_InitiatorRlsReqParam_t *rls_req_param [in, out]: Pointer to the I/O parameter structure.

Required GUI inputs: None
Return value: phcsBfl_Status_t, unsigned 16 bit integer. High byte (MSB): Category (group) Identifier; Low byte (LSB): Error Specifier.

PH_ERR_BFL_SUCCESS: Operation successful.

Others: Other error codes are propagated from lower layers.

Library function: **phcs_BFL::phcsBflNfcInitator_Wrapper::ResetProt**
Description: This function sets the protocol to initial state. The Initialise() function internally calls this method. Used for protocol-reinitialisation.
Parameters:

phcsBflNfc_InitiatorResetParam_t *rst_param [in, out]: Pointer to the I/O parameter structure.

Required GUI inputs: None
Return value: Void

Library function: **phcs_BFL::phcsBflNfcInitator_Wrapper::WupRequest**
Description: This function is only available in Active communication mode and after DESELECT. It allows to apply a new DID and to directly issue a further PSL_REQ.
Parameters:

phcsBflNfc_InitiatorWupReqParam_t *wup_req_param [in, out]: Pointer to the I/O parameter structure.

Required GUI inputs:

DIDi

Return value: phcsBfl_Status_t, unsigned 16 bit integer. High byte (MSB): Category (group) Identifier; Low byte (LSB): Error Specifier.

PH_ERR_BFL_SUCCESS: Operation successful.

Others: Other error codes are propagated from lower layers.

Library function: **phcs_BFL::phcsBflNfcInitator_Wrapper::RequestedToxValue**
Description: When the NfcDataExchangeProtocolRequest exits with RCLSTATUS_TARGET_SET_TOX or RCLSTATUS_TARGET_RESET_TOX (defined in `rclstatus.h`) it is a temporary exit, indicating that the Target has requested extended processing time. This function enables the application to query the requested value.
Parameters:

phcsBflNfc_InitiatorToxReqParam_t * req_tox_param [in, out]: Pointer to the I/O parameter structure.

Required GUI inputs: None
Return value: uint8_t, unsigned 8 bit integer.

PH_ERR_BFL_SUCCESS: Operation successful.

Others: Other error codes are propagated from lower layers.

Library function: **phcs_BFL::phcsBflNfc_TargetWrapper::Initialise**
Description:
Note: The small TX/RX buffer (TRxBuffer) used for building individual frames needs not to be specified here. It comes together with the Dispatch member function and must be managed by the embedding software. The size of the buffer (to indicate how many bytes the Target can return) is set to the default value (64). After ATR_REQ/PSL_REQ the final buffer size for the current session is fixed. In order to apply the corresponding settings, call the SetTRxBufProp function.
The large buffer is for easy protocol access when there are sufficient resources. However, in a system without abundant resources it may be desirable to control the Target protocol behaviour when sending and receiving data. In order to disable automatic chaining (handle individual frames with the embedding software) the user buffer pointer must be set to NULL and its size to zero.
Parameters:

uint8_t req_preamble_type [in]: Type of preamble to expect together with incoming requests, defined in `nfccommon.h`.

uint8_t res_preamble_type [in]: Type of preamble to return to the Initiator with the response, defined in `nfccommon.h`.

uint8_t *p_target_buffer [in]: Pointer to a large user buffer, provided by the embedding software. This buffer is used by the protocol as an information destination when receiving and as a source when returning user data. Set to NULL in order to disable automatic frame handling (chaining).

uint16_t target_buffer_size [in]: Size of the large data buffer for application (embedding software) use. Set to zero in order to disable automatic frame handling (chaining).

uint8_t *p_nfcidi_buffer [in]: Pointer to a pre-allocated 10 byte buffer receiving the NFCID-3 of the Initiator upon ATR_REQ.

uint8_t *p_nfcidt_buffer [in]: Pre-allocated and initialised 10 byte buffer for the Target protocol to provide the Target's NFCID-3.

uint8_t passive_mode [in]: Boolean specifier of the operation mode. If set (TRUE) the Target is told to work according to the specification of passive mode operation. If zero (FALSE) the protocol selects active mode.

class phcsBfl_NfcTargetEndpoint *p_upper [in]: Pointer to the protocol's endpoint. This is a component with a defined interface, however the implementation must be defined by the embedding software (since it is the point where requests arrive and responses are put together).

class phcsBfl_Io *p_lower [in]: Pointer to a component representing the lower instance, responsible for raw data exchange (RC-I/O component).

Required GUI inputs:

NFCID3

Return value: Void

Library function: **phcs_BFL::phcsBflNfc_TargetWrapper::ResetProt**
Description: This function resets the protocol to initial state before ATR_REQ. This function can also be called after a RLS_REQ/RLS_RES procedure to clear all internal states for the next activation.
Parameters:

phcsBflNfc_TargetResetParam_t *reset_param [in]: Pointer to the I/O parameter structure.

Required GUI inputs: None
Return value: phcsBfl_Status_t, unsigned 16 bit integer. High byte (MSB): Category (group) Identifier; Low byte (LSB): Error Specifier.

PH_ERR_BFL_SUCCESS: Operation successful.

Others: Other error codes are propagated from lower layers.

Library function: **phcs_BFL::phcsBflNfc_TargetWrapper::SetTRxBufProp**
Description: This function sets or modifies the TargetBuffer for internal chaining.
Note: The buffer specified with this function is a large user buffer for data to send or receive respectively. Chaining is done automatically, internally, by the protocol, if such a buffer is specified.
Parameters:

phcsBflNfc_TargetSetTRxBufPropParam_t *trx_buffer_param [in]: Pointer to the I/O parameter structure.

Return value: phcsBfl_Status_t, unsigned 16 bit integer. High byte (MSB): Category (group) Identifier; Low byte (LSB): Error Specifier.

PH_ERR_BFL_SUCCESS: Operation successful.

Others: Other error codes are propagated from lower layers.

Required GUI inputs:

Library function: **phcs_BFL::phcsBflNfc_TargetWrapper::SetUserBuf**
Description: This function sets or modifies the TargetBuffer for internal chaining.
Note: The buffer specified with this function is a large user buffer for data to send or receive respectively. Chaining is done automatically, internally, by the protocol, if such a buffer is specified.
Parameters:

phcsBflNfc_TargetSetUserBufParam_t *user_buffer_param [in]: Pointer to the I/O parameter structure.

Required GUI inputs:

Payload data, if sending data

Return value: Void

Library function: **phcs_BFL::phcsBflNfc_TargetWrapper::SetPUser**
Description: Sets a user pointer which might used for several interactions between main function and Target callback implementation. This is an general purpose pointer which may carry different data. Both, the main function and the callback have to know about the contents!
Parameters:

phcsBflNfc_TargetSetPUserParam_t * p_user_param [in]: Pointer to the I/O parameter structure.

Required GUI inputs: None
Return value: Void

Library function: **phcs_BFL::phcsBflNfc_TargetWrapper::Dispatch**
Description: Main entry point, contains state machine to handle request/response sequences.
Note: If the return value indicates unsuccessful completion the embedding software (which is essentially the receive loop) must get back into receiving mode. No transmission has been triggered by the protocol in this case and no transmission must be performed by the embedding software.
Parameters:

phcsBflNfc_TargetDispatchParam_t* dispatch_param [in]: Pointer to the I/O parameter structure.

Required GUI inputs: None
Return value: phcsBfl_Status_t, unsigned 16 bit integer. High byte (MSB): Category (group) Identifier; Low byte (LSB): Error Specifier.

PH_ERR_BFL_SUCCESS: Operation successful.

Others: Other error codes are propagated from lower layers.

Library function: **phcs_BFL::phcsBflMfRd_Wrapper::Transaction**
Description: The main MIFARE reader protocol entry point. All MIFARE functionality is concentrated in this place. According to the cmd parameter all possible Mifare commands are handled. For a detailed description of all commands have a look at any Mifare documentation.
Parameters:

phcsBflMfRd_CommandParam_t *cmd_param [in]: Pointer to the I/O parameter structure.

Required GUI inputs:

Used Mifare Key (A/B)

Mifare AuthentKey value

Mifare Block number to be authenticated

Return value: phcsBfl_Status_t, unsigned 16 bit integer. High byte (MSB): Category (group) Identifier; Low byte (LSB): Error Specifier.

PH_ERR_BFL_SUCCESS: Operation successful.

Others: Other error codes are propagated from lower layers.

Appendix B

HAL API

This chapter describes the application programming interface of the BFL.

Library function: **phHalNfc_Enumerate**

Description: This function allows the user to receive a list of devices connected to the system. The function itself does not open handles to the devices. It only returns information about them and their availability.

Notes:

- In fixed configurations (mostly embedded systems) the output of this function never changes.
- It is up to the system integrator to provide a correct implementation within the "Driver Abstraction Layer (DAL)".

Parameters:

phHal_sHwReference_t sHwReference[ ] [in, out]: Array of uninitialised Hardware Reference structures which are allocated by the caller (user application, upper layer). Each caller-provided structure (array element) represents one peripheral device. The function fills the array until no further device is detected or the array boundary is reached.
Those HW References which have not been initialised get the grp_comp_id "Component ID" CID_NFC_NONE in order to unambiguously mark it non-assigned.

uint8_t *pNbrOfDevDetected [in, out]: The caller is responsible to specify the maximum number of devices. When the function returns it yields the current number of devices found with this parameter.
Caution: When the caller provides less array spaces than devices found the function fills all array elements with device information and returns the *actual* number of devices found in this parameter (which in this case exceeds the number provided by the caller).
Additionally the status is set to indicate that more devices are available.

Required GUI inputs: None
Return value: NFCSTATUS, unsigned 16 bit integer

NFCSTATUS_SUCCESS: Enumeration successful.

NFCSTATUS_INVALID_PARAMETER: One or more of the supplied parameters could not be properly interpreted. In particular, this error is generated when the number of devices parameter is initialised to zero by the caller or the HW-Ref. array parameter points to NULL.

NFCSTATUS_MORE_INFORMATION: The user-provided array is too small for the number of devices detected. However, the available indices have been filled with information.

NFCSTATUS_CMD_ABORTED: The caller/driver has aborted the request.

Others: Other error codes are propagated from lower layers.

Library function: **phHalNfc_Open**

Description: This function establishes a link to the peripheral. It uses a Hardware Reference, pre-initialised during enumeration. The reference to the peripheral device (with the device handle embedded) is made valid by this command, if it completes successfully. Otherwise the structure and its members are not modified.

Notes:

- The device is in initial (reset) state after this command has completed successfully.
- It is up to the system integrator to provide a correct implementation within the "Driver Abstraction Layer" (DAL).

Parameters:

phHal_sHwReference_t *psHwReference [in, out]: Hardware Reference, pre-initialised during enumeration. Made valid by this function.

Required GUI inputs:

Selected device in the device list

Return value: NFCSTATUS, unsigned 16 bit integer

NFCSTATUS_SUCCESS: Open has been successful.

NFCSTATUS_INVALID_DEVICE: The device is not enumerated or the Hardware Reference points to a device which does not exist. Alternatively, also already open devices produce this error.

NFCSTATUS_INVALID_PARAMETER: The parameter could not be properly interpreted (structure uninitialised?).

NFCSTATUS_CMD_ABORTED: The caller/driver has aborted the request.

Others: Other error codes are propagated from lower layers.

Library function: **phHalNfc_GetDeviceCapabilities**

Description: Retrieves the capabilities of the device represented by the Hardware Reference parameter. The protocol supported, the MTU and other mandatory information are located inside the pDevCapabilities parameter.

Parameters:

phHal_sHwReference_t *psHwReference [in]: Hardware Device Reference, pre-initialised during enumeration, validated during opening.

phHal_sDeviceCapabilities_t *psDevCapabilities [out]: Pointer to the device capabilities structure where all relevant capabilities of the peripheral are stored.

Required GUI inputs: None
Return value: NFCSTATUS, unsigned 16 bit integer

NFCSTATUS_SUCCESS: Success.

NFCSTATUS_INVALID_PARAMETER: One or more of the supplied parameters could not be properly interpreted.

NFCSTATUS_INVALID_DEVICE: The device has not been opened or has been disconnected meanwhile.

NFCSTATUS_CMD_ABORTED: The command was aborted by the device.

Others: Other error codes are propagated from lower layers.

Library function: **phHalNfc_Close**

Description: Closes the link to the peripheral NFC device.

Note: It is up to the system integrator to provide a correct implementation within the "Driver Abstraction Layer (DAL)".

Parameters:

phHal_sHwReference_t *psHwReference [in, out]: Pointer to a Hardware Reference structure, pre-set during enumeration and finally initialised by opening the link to the NFC peripheral.

Required GUI inputs: None
Return value: NFCSTATUS, unsigned 16 bit integer

NFCSTATUS_SUCCESS: Closing successful.

NFCSTATUS_INVALID_DEVICE: The device has not been opened or has been disconnected meanwhile.

NFCSTATUS_INVALID_PARAMETER: One or more of the supplied parameters could not be properly interpreted.

NFCSTATUS_CMD_ABORTED: The caller/driver has aborted the request.

Others: Other error codes are propagated from lower layers.

Library function: **phHalNfc_Connect**

Description: Allows to connect to a single, specific, already known Remote Device. The Remote Device Information structure is already pre-initialised with data (e.g. from polling or an earlier session, after phHalNfc_Disconnect "disconnect"). A new session is started after the connect function returns successfully. The session ends with a successful disconnect (see phHalNfc_Disconnect function).

Note: If timeout functionality is required the system integrator has to install a mechanism to abort the operation at driver level. Alternatively, the caller, if running in a de-coupled thread, can implement this feature.

Parameters:

phHal_sHwReference_t *psHwReference [in]: Hardware Device Reference returned during enumeration.

phHal_eOpModes_t OpMode [in]: Enum value controlling the operation mode under which to activate the Remote Device. See the declaration of phHal_eOpModes_t.

phHal_sRemoteDevInformation_t *psRemoteDevInfo [in, out]: Points to the Remote Device Information structure. The members of it can be re-used from a previous session.

phHal_sDevInputParam_t *psDevInputParam [in]: Points to the input device parameter struct/union needed by the start up phase.

Required GUI inputs:

Selected remote device

Operation modes for activation

Return value: NFCSTATUS, unsigned 16 bit integer

NFCSTATUS_SUCCESS: Success.

NFCSTATUS_INVALID_PARAMETER: One or more of the supplied parameters could not be properly interpreted.

NFCSTATUS_ALREADY_CONNECTED: More than one phHalNfc_Connect is not allowed during a session on the same remote device. The session has to be closed before (see phHalNfc_Disconnect function).

NFCSTATUS_RF_TIMEOUT: Command received no response from the remote device and ended with a time out.

NFCSTATUS_INVALID_DEVICE: The device has not been opened before. The Remote Device Identifier is not valid.

NFCSTATUS_INVALID_DEVICE_REQUEST: The Initiator has requested a mode, not enabled in the mode selection bitfield.

NFCSTATUS_CMD_ABORTED: The caller/driver has aborted the request.

Others: Other error codes are propagated from lower layers.

Library function: **phHalNfc_Poll**

Description: This function allows the Initiator to search for Targets/cards ("Remote Devices") in the RF neighbourhood, following a defined sequence and allowing a configurable subset of modes. The number of Remote Devices detected is returned to the caller (upper layer) as well as information about the startup phase in the corresponding phHal_sRemoteDevInformation_t array element.
This function can, if the capabilities of it allow so (phHal_sDeviceCapabilities), be called several times phHal_sDeviceCapabilities) to add more Remote Devices to already existing/registered ones.

Note: If timeout functionality is required the system integrator has to install a mechanism to abort the operation at driver level. Alternatively, the caller, if running in a de-coupled thread, can implement this feature.

Parameters:

phHal_sHwReference_t *psHwReference [in]: Hardware Reference.

phHal_eOpModes_t OpModes[ ] [in]: Enumerated type array controlling operation modes under which to look for Remote Devices and the sequence order of the scan. Each member of the array represents a single mode to search within. When reading the array the function considers index 0 as most and index n as least priority. The stop-condition is the the Array Terminator. See the declaration of phHal_eOpModes_t.

phHal_sRemoteDevInformation_t psRemoteDevInfoList[ ] [in, out]: Points to the Remote Device Information structure list. After polling, the members of the list hold information about the remote device detected such as the state, the parameters of the start up phase including information about the protocol used. The caller has to provide the uninitialised structures within an already pre-allocated array. If there are already valid structures present in the array (e.g. from a previous call) the function can add, if supported, new devices to that list. If repeated polling, while RDIs are valid, is not supported (see phHalNfc_GetDeviceCapabilities), the function does nothing but return with the appropriate error. See phHalNfc_Connect and phHalNfc_Disconnect for information about functions for RDI (in)validation.

uint8_t *pNbrOfRemoteDev [in, out]: The caller has to provide the maximum number of remote devices (number of pre-allocated structures) to detect. The function returns the number of remote devices actually found in this parameter. It is a good idea that the caller remembers the original number of array fields (maximum number of devices) which is of importance when the function is called again.

phHal_sDevInputParam_t *psDevInputParam [in, out]: Points to the input device parameter union/structure needed by the start up phase.

Required GUI inputs:

Operation modes

Return value: NFCSTATUS, unsigned 16 bit integer

NFCSTATUS_SUCCESS: Success.

NFCSTATUS_INVALID_PARAMETER: One or more of the supplied parameters could not be properly interpreted.

NFCSTATUS_INVALID_DEVICE: The device has not been opened or has been disconnected meanwhile.

NFCSTATUS_MULTI_POLL_NOT_SUPPORTED: More than one phHalNfc_Poll is not allowed during a session. All Remote Devices have to be *closed* before (see phHalNfc_Disconnect function).

NFCSTATUS_CMD_ABORTED: The caller/driver has aborted the request.

NFCSTATUS_NO_DEVICES_FOUND: No remote device found in the RF.

Others: Other error codes are propagated from lower layers.

Library function: **phHalNfc_Disconnect**
Description: The function allows to disconnect from a specific Remote Device. This function closes the session opened with phHalNfc_Connect “Connect”.
Parameters:

phHal_sHwReference_t *psHwReference [in]: Device reference returned during enumeration.

phHal_sRemoteDevInformation_t *psRemoteDevInfo [in, out]: Points to the valid (connected) Remote Device Information structure.

Required GUI inputs: None
Return value: NFCSTATUS, unsigned 16 bit integer

NFCSTATUS_SUCCESS: Success.

NFCSTATUS_INVALID_PARAMETER: One or more of the supplied parameters could not be properly interpreted.

NFCSTATUS_INVALID_DEVICE: The device has not been opened before or has already been closed.

NFCSTATUS_NO_DEVICE_CONNECTED: The Remote Device is not connected.

NFCSTATUS_RF_TIMEOUT: The command received no response from the Remote Device and ended with a timeout.

NFCSTATUS_CMD_ABORTED: The caller/driver has aborted the request.

Others: Other error codes are propagated from lower layers.

Library function: **phHalNfc_Transceive**

Description: The phHalNfc_Transceive function allows the Initiator to send and receive data to and from the Remote Device selected by the caller. The caller has to provide the Remote Device Information structure and the command in order to communicate with the selected remote device.

Parameters:

phHal_sHwReference_t *psHwReference [in]: Device reference returned during enumeration.

phHal_sRemoteDevInformation_t *psRemoteDevInfo [in, out]: Points to the Remote Device Information structure which identifies the selected Remote Device.

phHal_uCmdList_t Cmd [in]: Command to perform. See the command list.

phHal_sDepAdditionalInfo_t *psDepAdditionalInfo [out]: Set/get the protocol info about NAD or Meta Chaining usage. The NAD value is specified by the Initiator.

uint8_t *pSendBuf [in]: Pointer to the data to be sent. See phHal_uCmdList_t "Command List" for additional information about the required format.

uint16_t SendLength [in]: Length, in byte, of the pSendBuf parameter.

uint8_t *pRecvBuf [out]: Pointer to the buffer that receives the data returned by the media.

uint16_t *pRecvLength [in, out]: Supplies the Length, in byte, of the pRecvLength parameter and receives the number of bytes actually received from the media.

Required GUI inputs:

Selected remote device

Command to send

Payload data to send

Return value: NFCSTATUS, unsigned 16 bit integer

NFCSTATUS_SUCCESS: Success.

NFCSTATUS_INVALID_PARAMETER: One or more of the supplied parameters could not be properly interpreted.

NFCSTATUS_INVALID_DEVICE: The device has not been opened or has been disconnected meanwhile.

NFCSTATUS_CMD_ABORTED: The caller/driver has aborted the request.

NFCSTATUS_BUFFER_TOO_SMALL: The buffer provided by the caller is too small.

NFCSTATUS_RF_TIMEOUT: No data has been received within the TIMEOUT period.

Others: Other error codes are propagated from lower layers.

Library function: **phHalNfc_StartTargetMode**

Description: Start Target mode. This is the counterpart of the polling or (actively) connecting functions (implicitly those functions select Initiator mode). The function waits until there is a connecting event such as an Attribute Request (ATR) or a Request Answer To Select (RATS).

Parameters:

phHal_sHwReference_t *psHwReference [in]: Hardware Reference prepared by the enumeration and open functions.

phHal_sTargetInfo_t *pTgInfo [in, out]: Pointer to the Target Information structure.

phHal_eOpModes_t OpModes[ ] [in, out]: The caller provides an array of enumerated values, initialised with the allowed modes. As an input, the parameter processing is the same as in the function phHalNfc_Poll: Stop-condition, Array Terminator. See also phHal_eOpModes_t. The function returns, in this parameter at index 0, the mode of operation the Initiator would like to select. See the declaration of phHal_eOpModes_t.

uint8_t *pConnectionReq [out]: Pointer to the buffer that receives the connection request coming from the Initiator. This parameter is only used if the Automatic response mechanism is activated. In this case, the request is only provided as information.

uint8_t *pConnectionReqBufLength [in, out]: Supplies the Length, in bytes, of the pConnectionReq parameter and receives the length, in bytes, of the request. If the Automatic response is deactivated, the pConnectionReqBufLength will be zero.

Required GUI inputs:

Selected device

Operation modes

Return value: NFCSTATUS, unsigned 16 bit integer

NFCSTATUS_SUCCESS: Successful.

NFCSTATUS_INVALID_DEVICE: The device has not been opened or has been disconnected meanwhile.

NFCSTATUS_INVALID_PARAMETER: One or more of the supplied parameters could not be properly interpreted.

NFCSTATUS_INVALID_DEVICE_REQUEST: The Initiator has requested a mode, not enabled in the mode selection bitfield.

NFCSTATUS_CMD_ABORTED: The caller/driver has aborted the request.

Others: Other error codes are propagated from lower layers.

Library function: **phHalNfc_Receive**
Description: Allows the Target to retrieve data/commands coming from the Initiator.
Parameters:

phHal_sHwReference_t *psHwReference [in]: Hardware Reference returned by phHal-Nfc_Enumerate function

phHal_sDepAdditionalInfo_t *psDepAdditionalInfo [out]: Retrieves the protocol info about NAD or Meta Chaining usage. The NAD value is specified by the Initiator.

uint8_t *pRecvBuf [out]: Pointer to the buffer that receives the data/command coming from the Initiator.

uint16_t *pRecvLength [in, out]: Supplies the Length, in byte, of the pRecvBuf parameter and receives the number of bytes received from the Initiator.

Required GUI inputs: None
Return value: NFCSTATUS, unsigned 16 bit integer

NFCSTATUS_SUCCESS: Successful.

NFCSTATUS_INVALID_DEVICE: The device has not been opened or has been disconnected meanwhile.

NFCSTATUS_INVALID_PARAMETER: One or more of the supplied parameters could not be properly interpreted.

NFCSTATUS_CMD_ABORTED: The caller/driver has aborted the request.

NFCSTATUS_BUFFER_TOO_SMALL: The buffer provided by the caller is too small.

NFCSTATUS_RF_TIMEOUT: No data has been received within the TIMEOUT period.

Others: Other error codes are propagated from lower layers.

Library function: **phHalNfc_Send**
Description: Allows the Target to send back data to the Initiator. It has to be the response to the previous command received via the phHalNfc_Receive function.
Parameters:

phHal_sHwReference_t *psHwReference [in]: Hardware Reference returned by the enumeration procedure and validated by opening the device.

phHal_sDepAdditionalInfo_t *psDepAdditionalInfo [in, out]: Tells the protocol about NAD or Meta Chaining usage. NAD must be the same as specified by the Initiator.

uint8_t *pSendBuf [in]: Pointer to the data to send back to the Initiator.

uint16_t SendLength [in]: Length, in byte, of the pSendBuf parameter.

Required GUI inputs:

NAD switch

If NAD present, NAD value

Meta chaining switch

Payload data to send

Return value: NFCSTATUS, unsigned 16 bit integer

NFCSTATUS_SUCCESS: Successful.

NFCSTATUS_INVALID_DEVICE: The device has not been opened or has been disconnected meanwhile.

NFCSTATUS_INVALID_PARAMETER: One or more of the supplied parameters could not be properly interpreted.

NFCSTATUS_CMD_ABORTED: The caller/driver has aborted the request.

Others: Other error codes are propagated from lower layers.

Appendix C

Screenshots

This chapter illustrates the graphical design of the application by screenshots. Shown are the individual subwindows, the client frames with a collection of subwindows, the main application and the configuration dialogue.

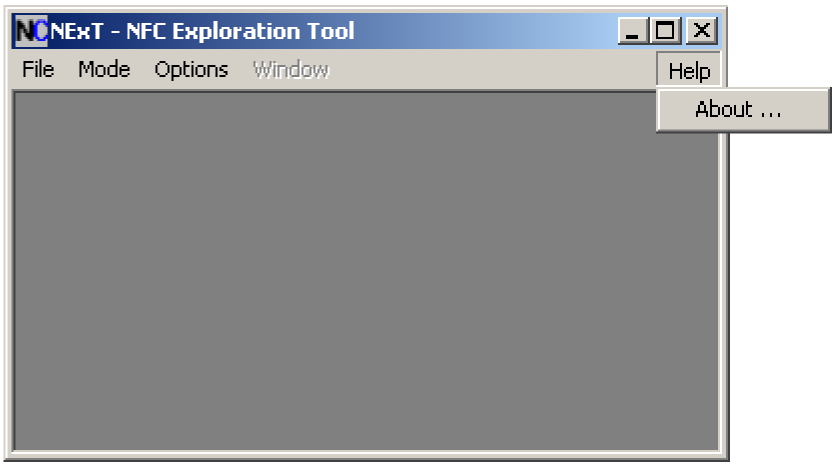
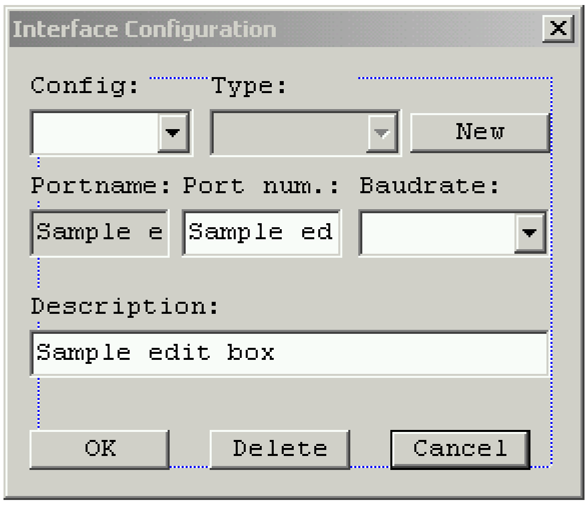


Figure C.1: NExT main window


Interface Configuration

Config: Type: New

Portname: Port num.: Baudrate:

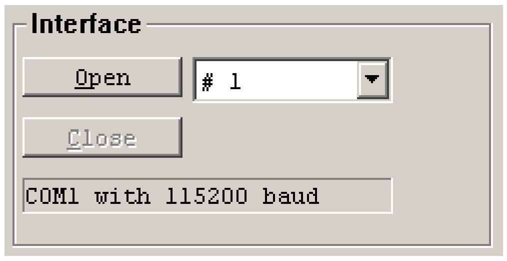
Sample e Sample ed

Description:

Sample edit box

OK Delete Cancel

Figure C.2: NExT interface configuration dialogue


Interface

Open # 1

Close

COM1 with 115200 baud

Figure C.3: NExT interface configuration subwindow

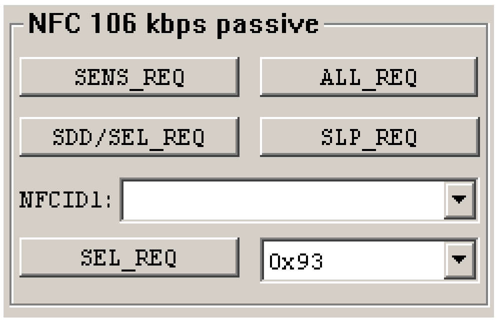


Figure C.4: NExT interface configuration subwindow

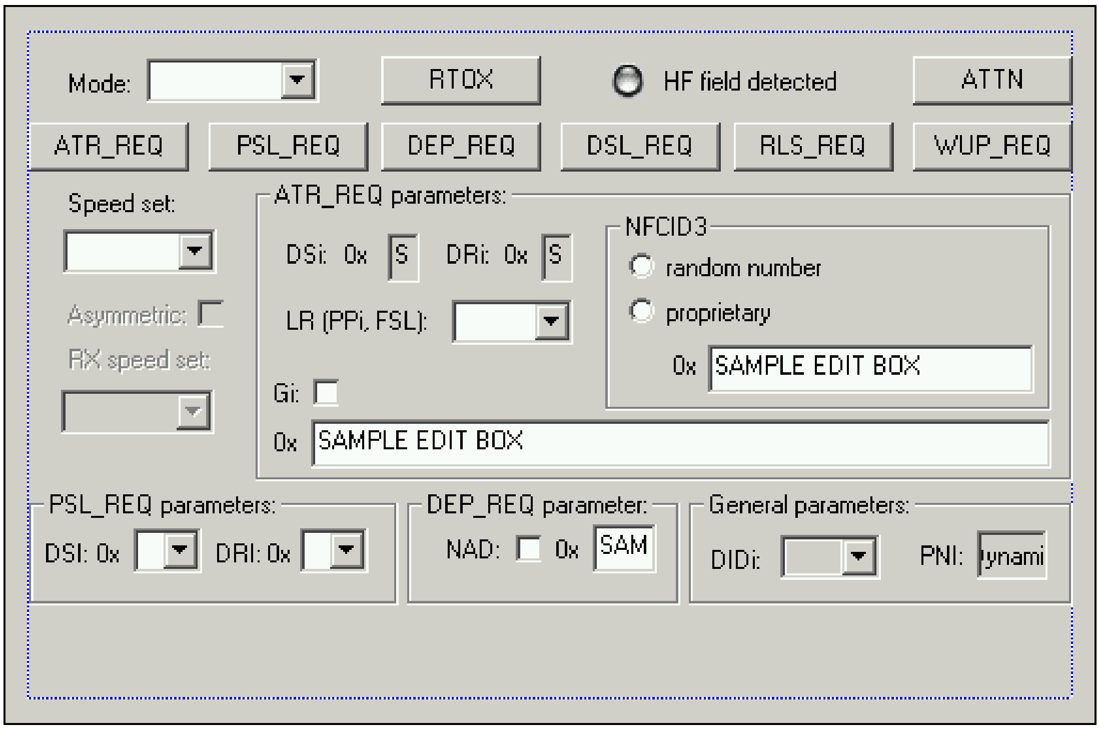


Figure C.5: BFL NFC Initiator subwindow

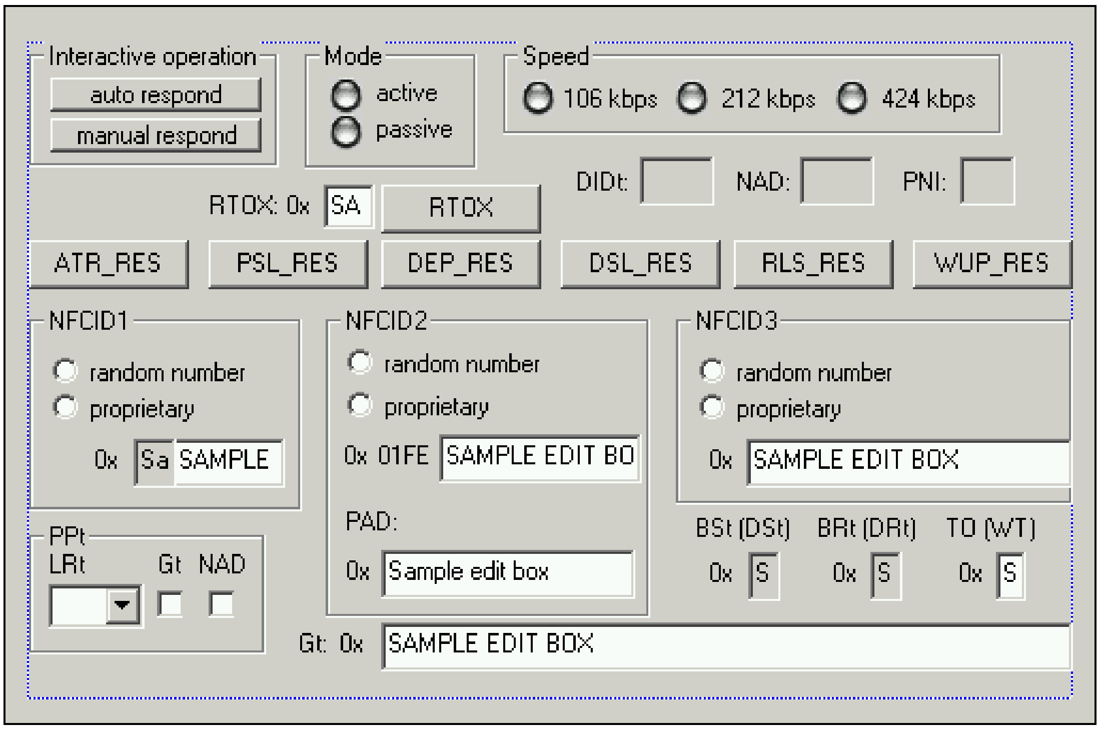


Figure C.6: BFL NFC Target subwindow

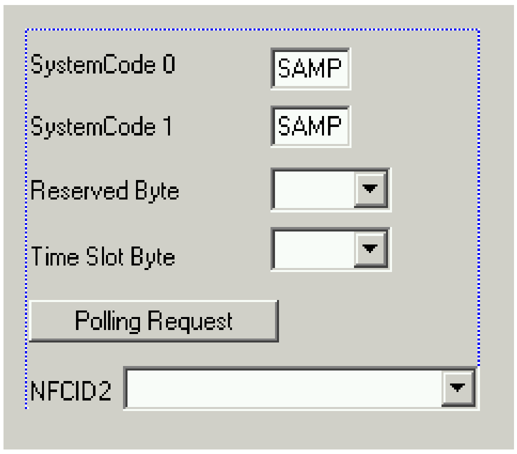


Figure C.7: BFL NFC 212/424 kbps passive subwindow

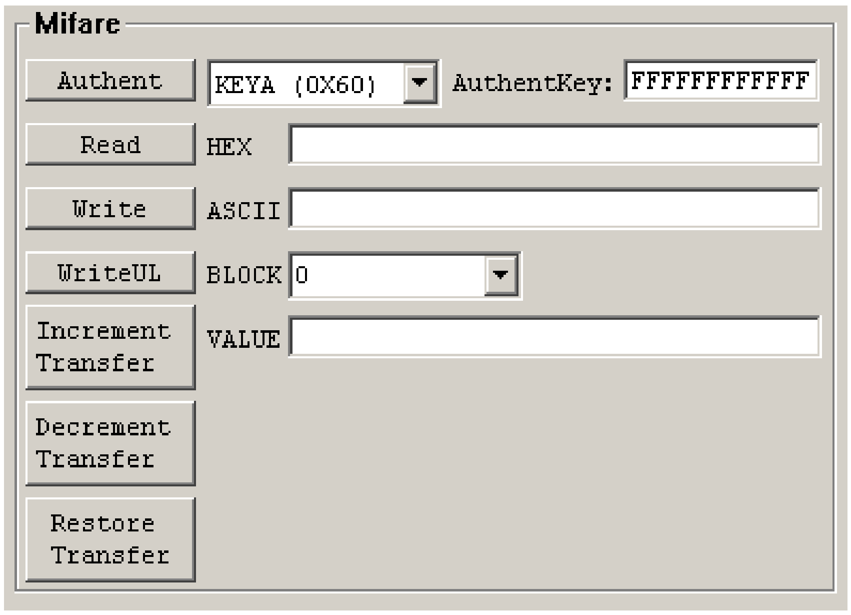


Figure C.8: BFL Mifare subwindow

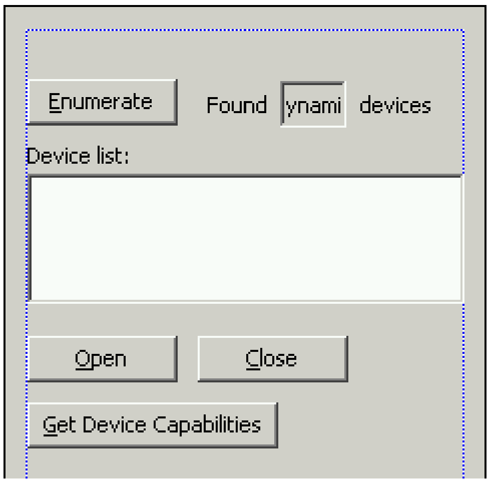
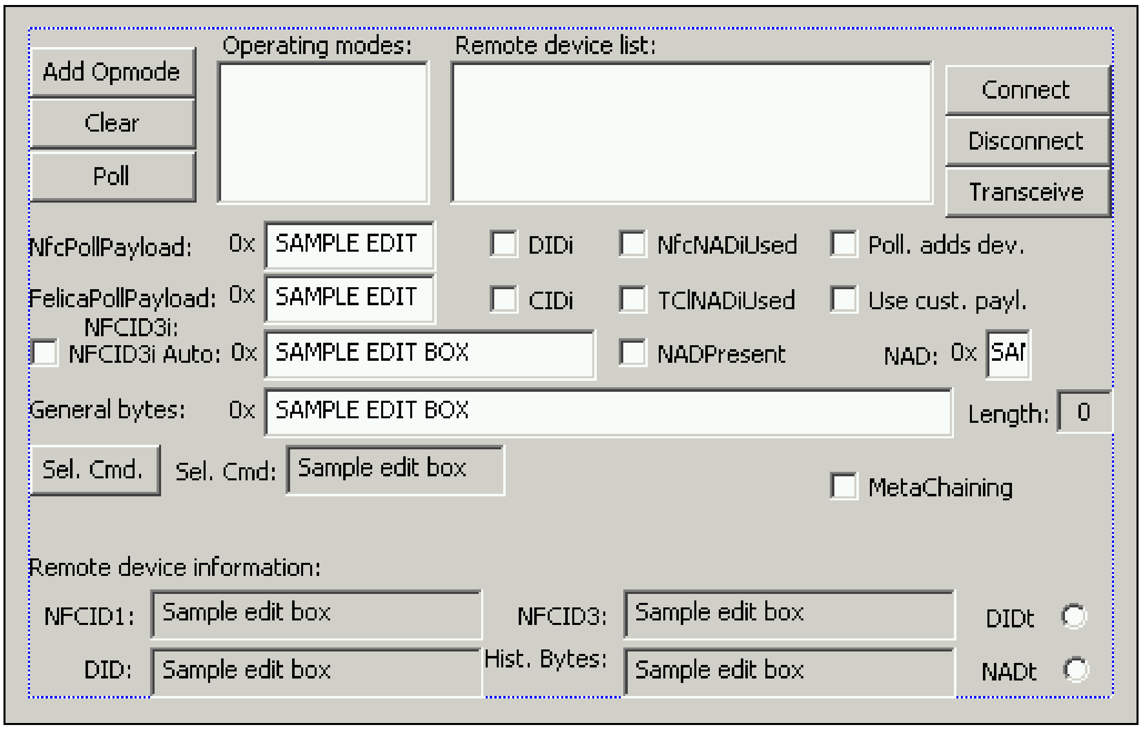


Figure C.9: HAL NFC Common subwindow



NfcPollPayload: 0x SAMPLE EDIT

FelicaPollPayload: 0x SAMPLE EDIT

NFCID3i:
NFCID3i Auto: 0x SAMPLE EDIT BOX

NAD: 0x SAI

General bytes: 0x SAMPLE EDIT BOX

Length: 0

Sel. Cmd: Sample edit box

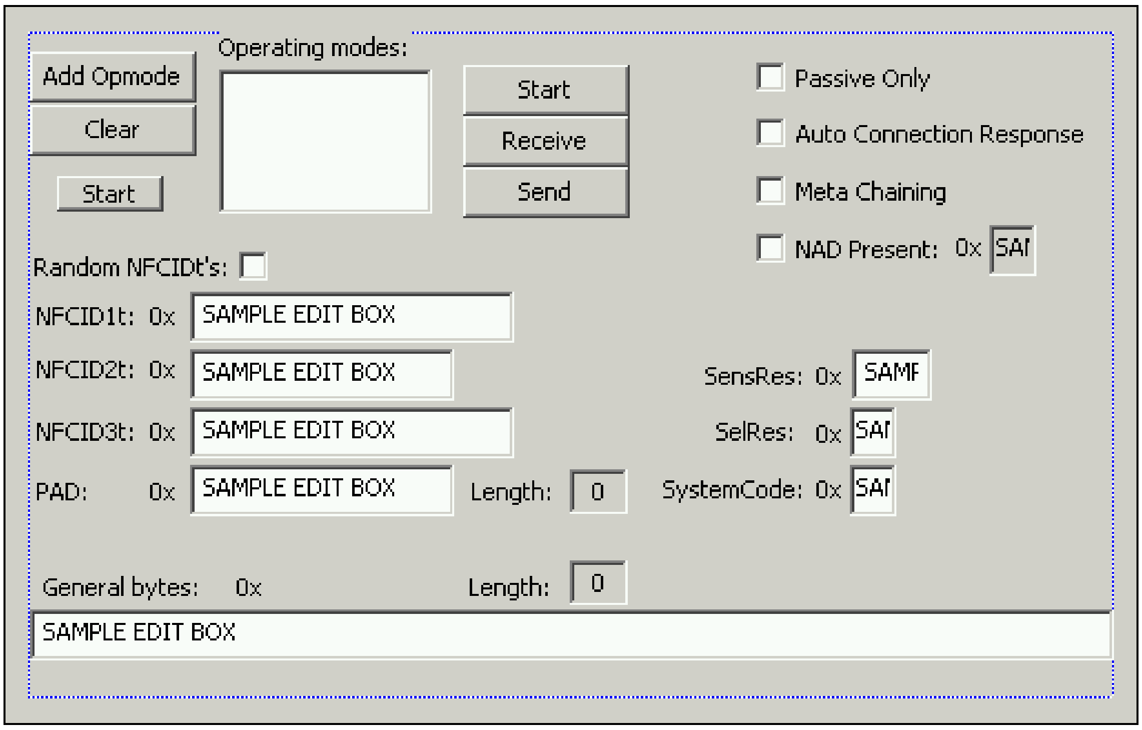
NFCID1: Sample edit box

NFCID3: Sample edit box

DID: Sample edit box

Hist. Bytes: Sample edit box

Figure C.10: HAL NFC Initiator subwindow



NAD Present: 0x SAI

Random NFCIDt's:

NFCID1t: 0x SAMPLE EDIT BOX

NFCID2t: 0x SAMPLE EDIT BOX

SensRes: 0x SAMF

NFCID3t: 0x SAMPLE EDIT BOX

SelRes: 0x SAI

PAD: 0x SAMPLE EDIT BOX

Length: 0

SystemCode: 0x SAI

General bytes: 0x

Length: 0

SAMPLE EDIT BOX

Figure C.11: HAL NFC Target subwindow

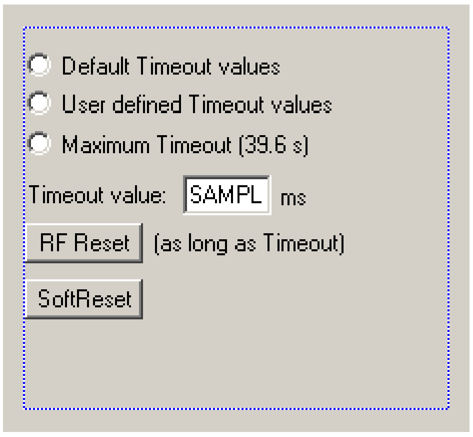


Figure C.12: General preferences subwindow

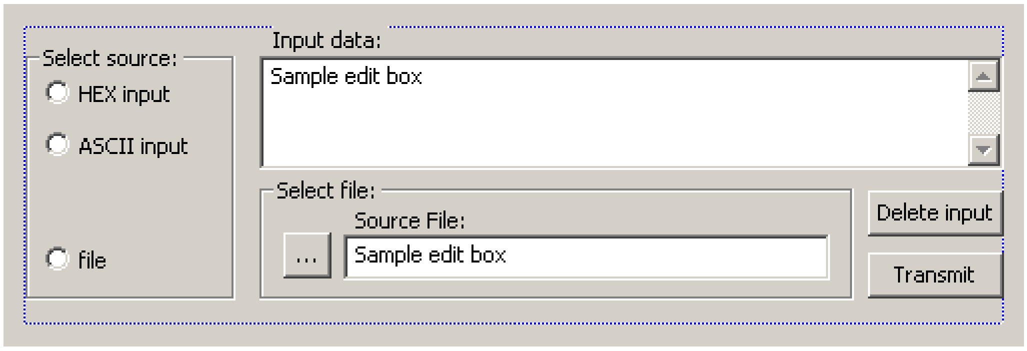


Figure C.13: Data/file transfer subwindow

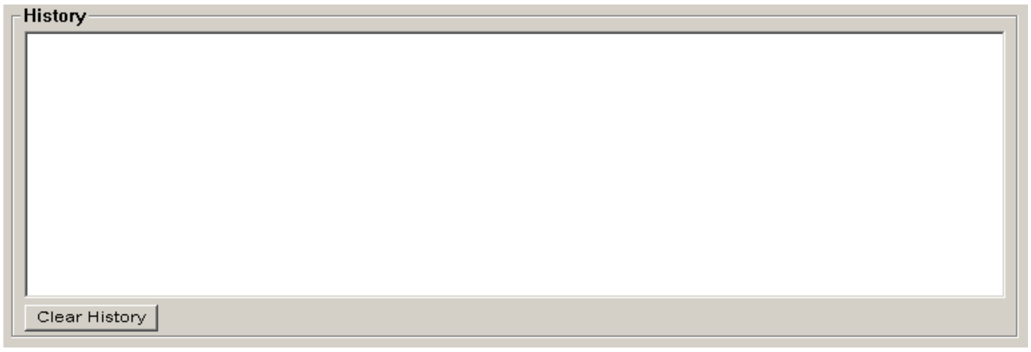
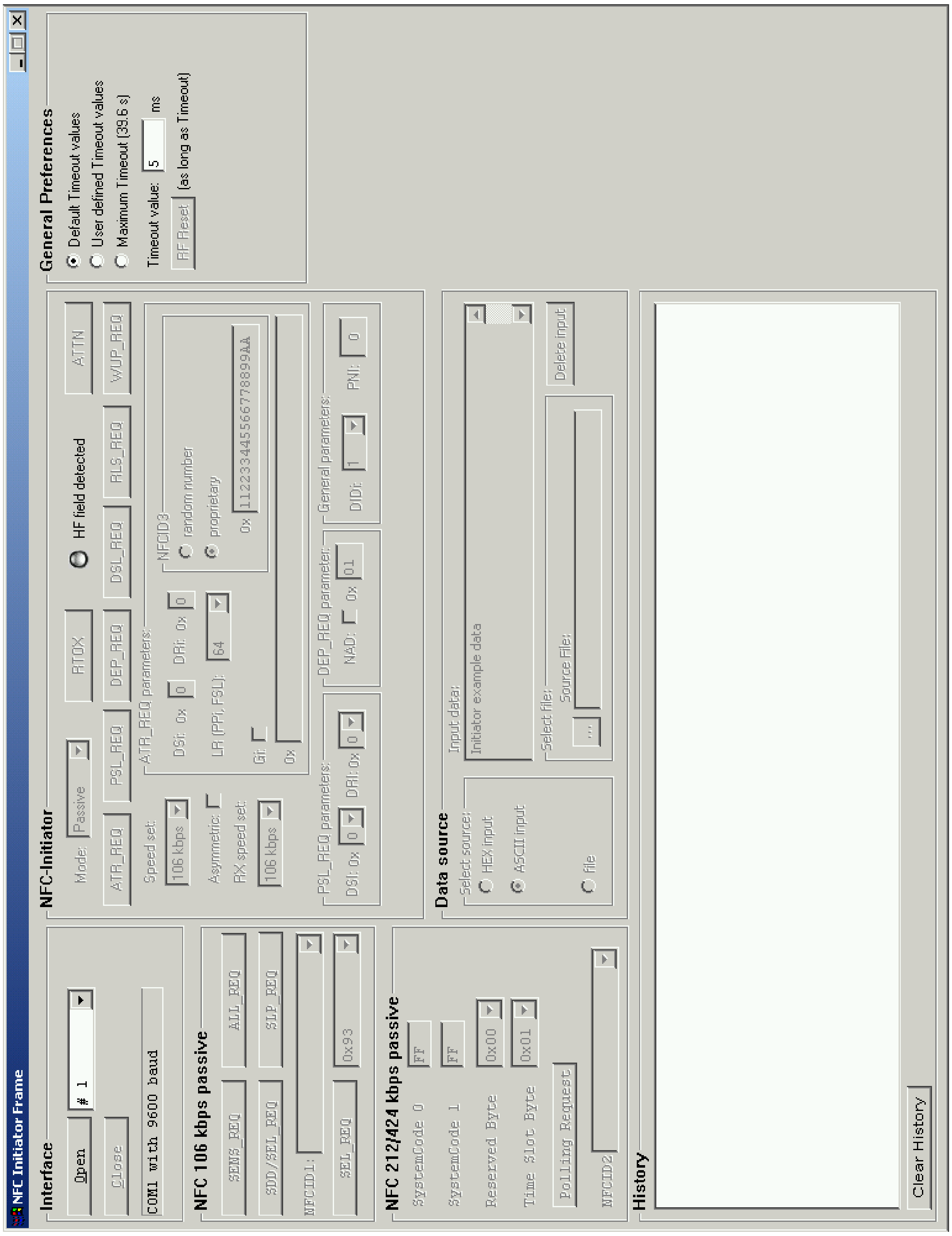


Figure C.14: History subwindow



NFC Initiator Frame

Interface

Open · Close · # 1 · COM1 with 9600 baud

NFC 106 kbps passive

SENS_REQ · ALL_REQ · SDD/SEL_REQ · SLP_REQ · NFCID1: · SEL_REQ · 0x93

NFC 212/424 kbps passive

SystemCode 0 FF
SystemCode 1 FF
Reserved Byte 0x00
Time Slot Byte 0x01
Polling Request
NFCID2

NFC-Initiator

Mode: Passive · RTOX · HF field detected · ATTN

ATR_REQ · PSL_REQ · DEP_REQ · DSL_REQ · RLS_REQ · WUP_REQ

Speed set: 106 kbps
Asymmetric:
RX speed set: 106 kbps

ATR_REQ parameters:
DSi: 0x 0 · DRi: 0x 0
LR (PPi, FSL): 64
Gi:
0x

NFCID3
random number
[✓ proprietary]
0x 112233445566778899AA

PSL_REQ parameters:
DSI: 0x 0 · DRI: 0x 0

DEP_REQ parameter:
NAD: 0x 01

General parameters:
DID: 1 · PNI: 0

Data source

Select source:
HEX input
[✓ ASCII input]
file

Input data:
Initiator example data

Select file:
Source File: ...

Delete input

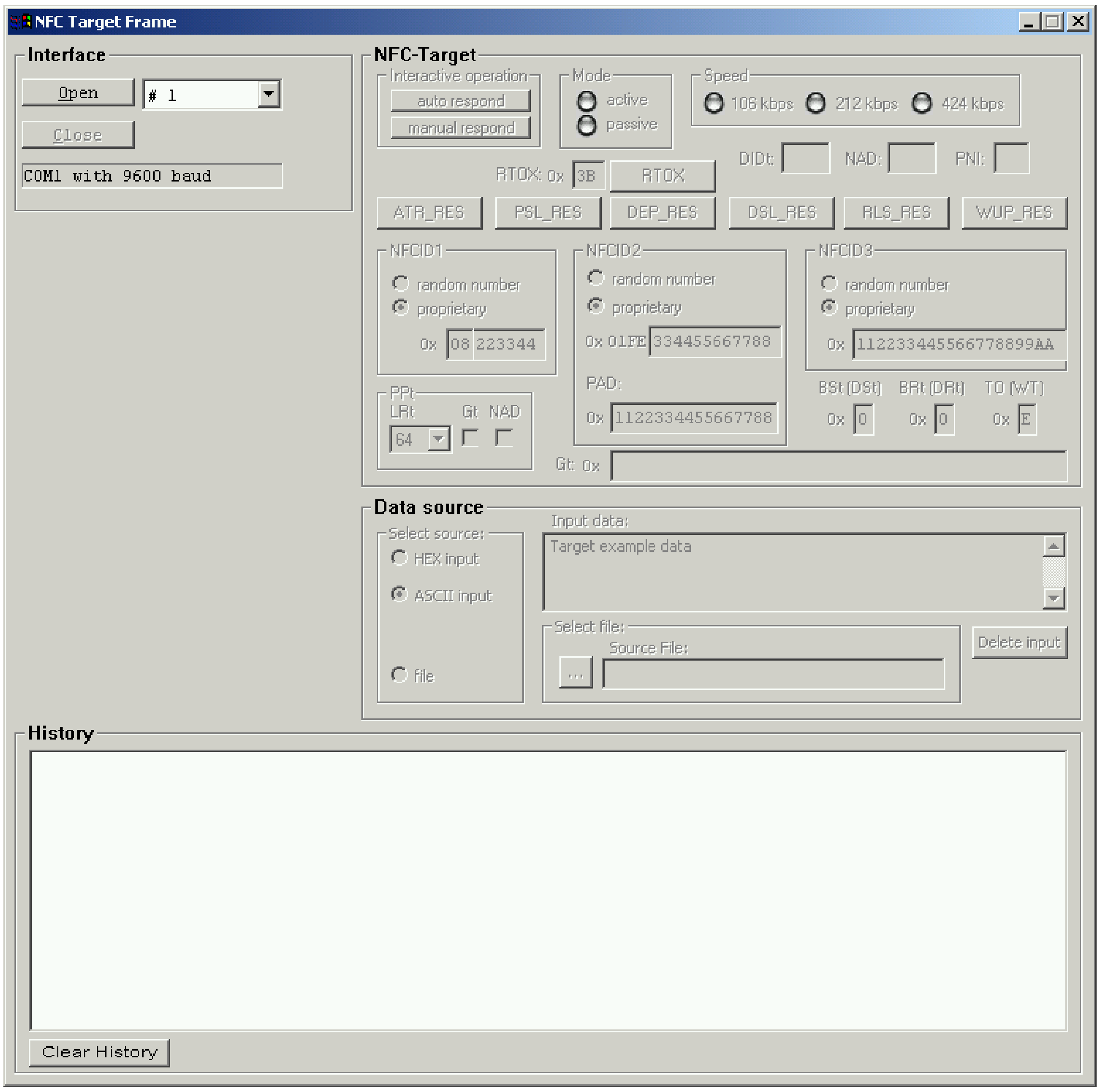
General Preferences

[✓ Default Timeout values]
User defined Timeout values
Maximum Timeout (39.6 s)
Timeout value: 5 ms
RF Reset (as long as Timeout)

History

Clear History

Figure C.15: BFL Initiator Window



Interface: # 1

COM1 with 9600 baud

RTOX: 0x 3B

DIDt: NAD: PNI:

NFCID1: [✓ proprietary] 0x 08 223344

NFCID2: [✓ proprietary] 0x 01FE 334455667788

PAD: 0x 1122334455667788

NFCID3: [✓ proprietary] 0x 112233445566778899AA

BSt (DSt) 0x 0 BRt (DRt) 0x 0 TO (WT) 0x E

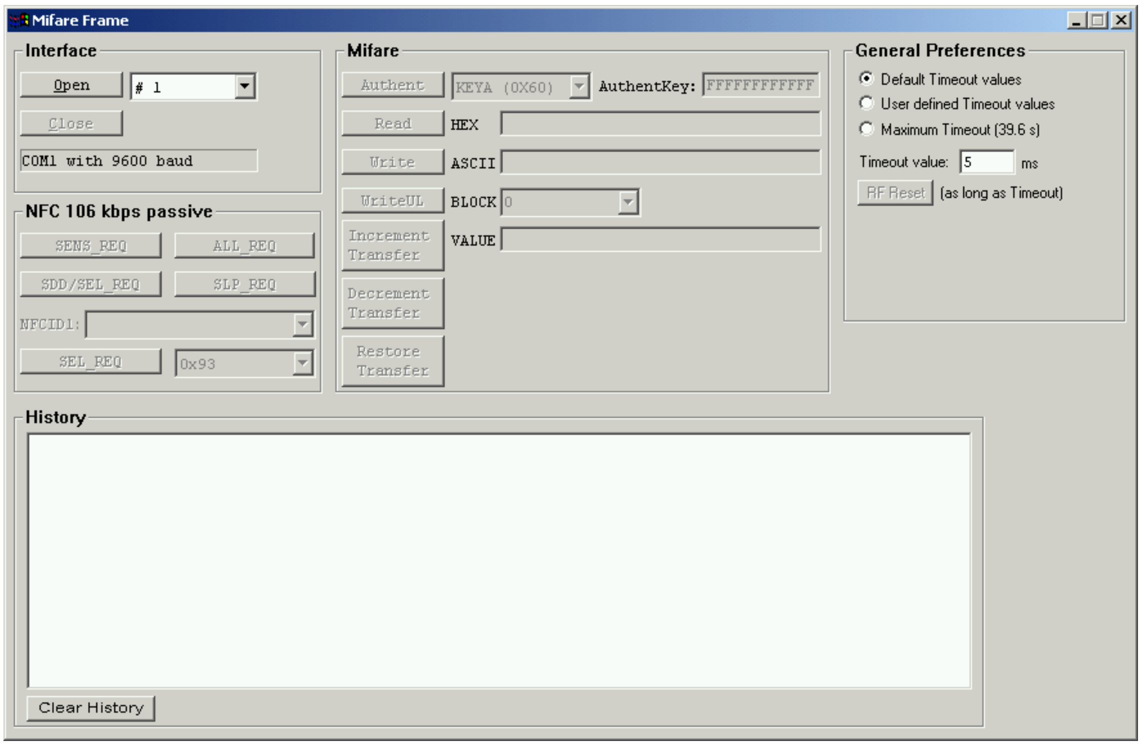
LRt: 64

Gt: 0x

Select source: [✓ ASCII input]

Input data: Target example data

Figure C.16: BFL Target Window



Interface: # 1

COM1 with 9600 baud

NFCID1:

0x93

KEYA (0X60) AuthentKey: FFFFFFFFFFFF

HEX

ASCII

BLOCK 0

VALUE

[✓ Default Timeout values]

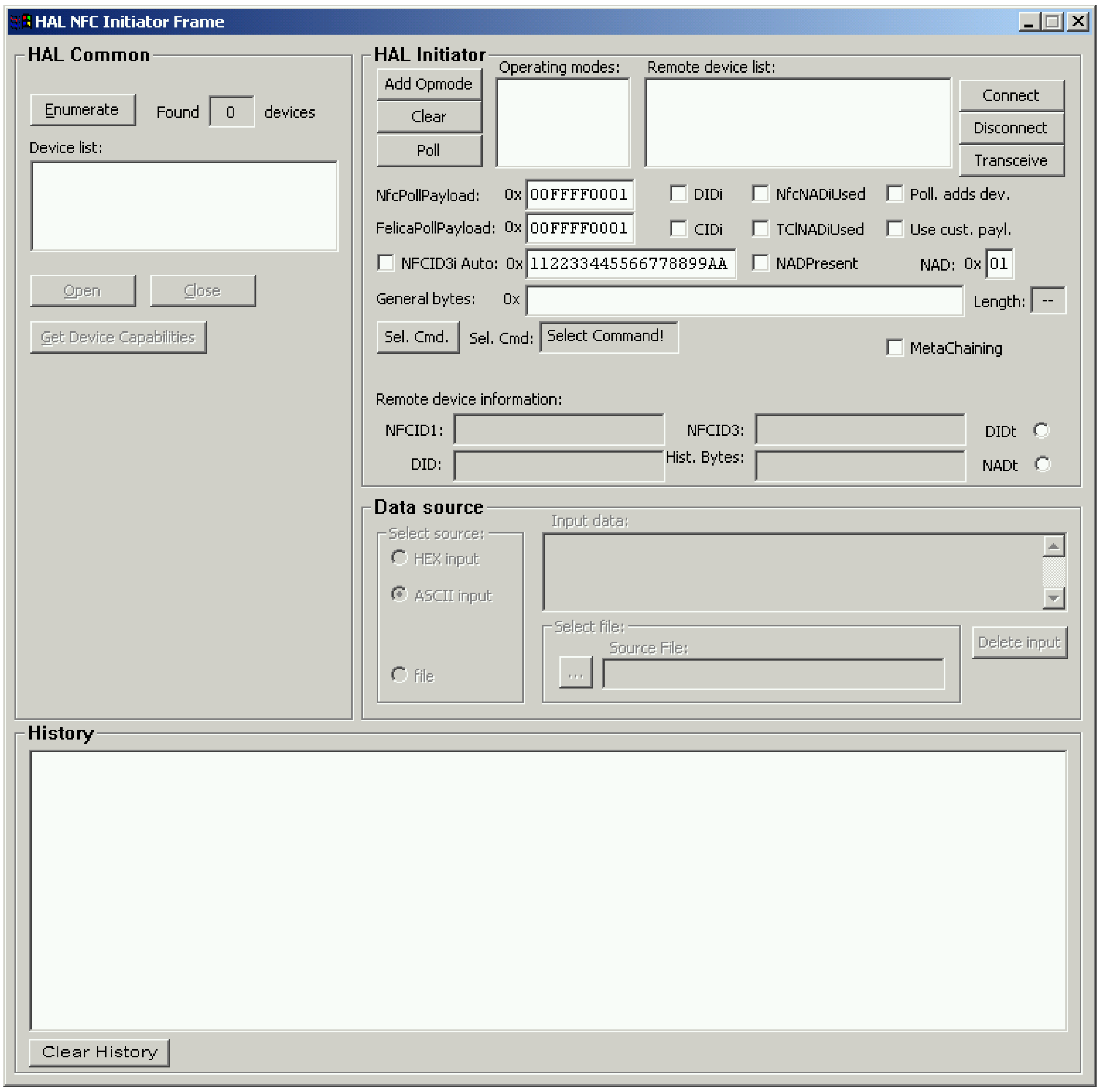
User defined Timeout values

Maximum Timeout (39.6 s)

Timeout value: 5 ms

(as long as Timeout)

Figure C.17: BFL Mifare Window



Found 0 devices

NfcPollPayload: 0x 00FFFF0001

FelicaPollPayload: 0x 00FFFF0001

NFCID3i Auto: 0x 112233445566778899AA

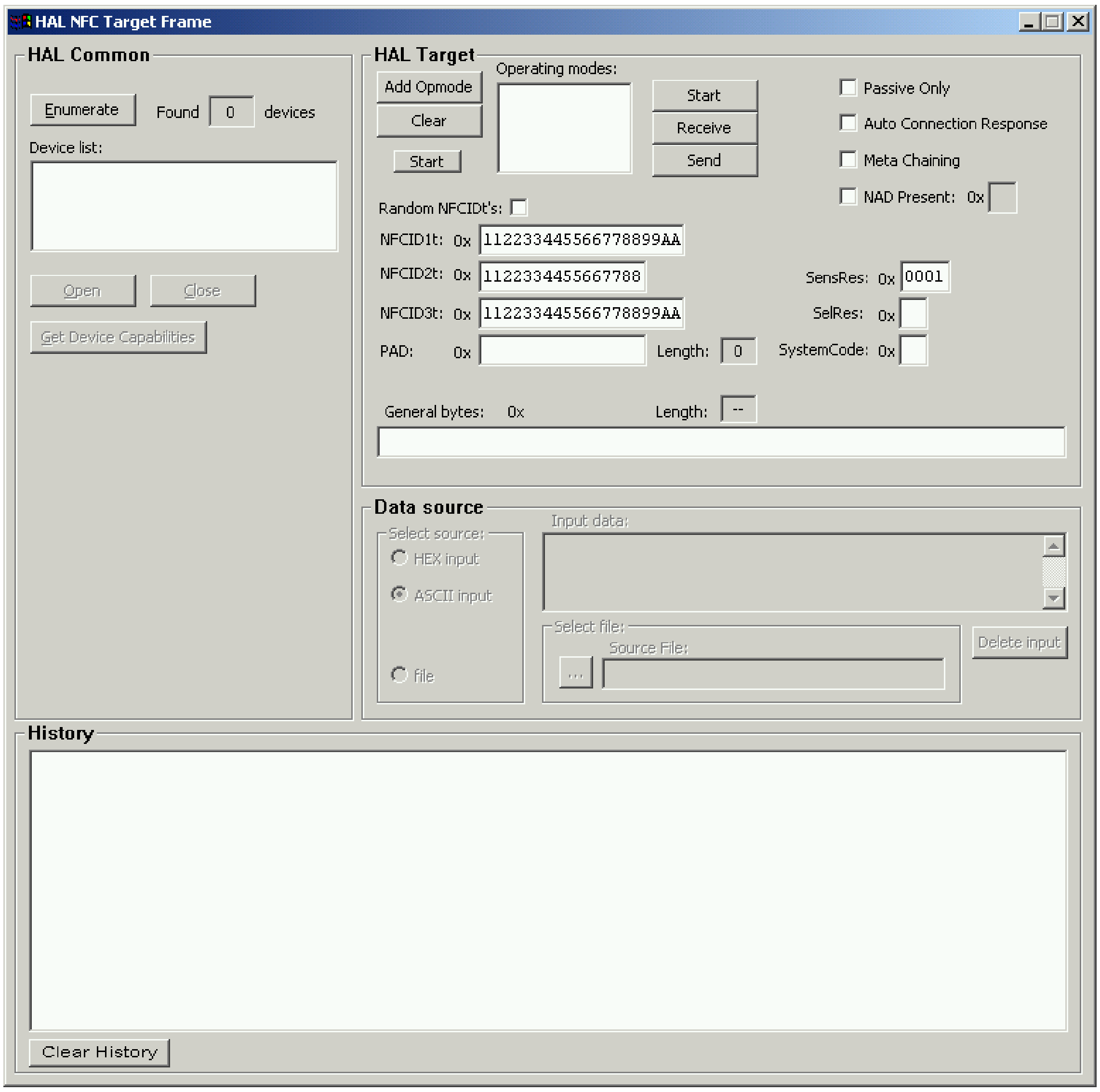
NAD: 0x 01

Length: --

Sel. Cmd: Select Command!

[✓ ASCII input]

Figure C.18: HAL Initiator Window



Found 0 devices

Random NFCIDt's: ☐

NFCID1t: 0x 112233445566778899AA

NFCID2t: 0x 1122334455667788

NFCID3t: 0x 112233445566778899AA

PAD: 0x Length: 0

SensRes: 0x 0001

SelRes: 0x

SystemCode: 0x

General bytes: 0x Length: --

[✓ ASCII input]

Figure C.19: HAL Target Window

Appendix D

File list

This chapter describes the files and their functions.

Windows modules: Those module's names have the prefix "`W32Next`".
Platform independent modules: Those module's names have the prefix "`Next`".

.\NExT\	
NExT.cpp	Main application
NExT.rc	Visual Studio resource file
NExT.sln	Visual Studio "Solution"[1]
NExT.vcproj	Visual Studio project file
resource.h	Resource definitions
.\NExT\Debug\	
BuildLog.htm	Visual Studio build log output
NExT.exe	Build application, including debug information
Next.ini	NExT preferences file
.\NExT\Documents\	Documentation
.\NExT\Documents\Doxygen\	Doxygen associated files
Clear.bat	Clean up HTML files
CommonProjectSettings.cfg	Project related configuration options
DoxyNExT.cfg	Doxygen settings
DoxyNExT.log	Doxygen log file
lastpage.sty	Automatically generated style definition
MainText.txt	This file is the basis for the structure of the documentation
my_doxygen.sty	
my_html_footer.html	Footer for HTML documentation generation
my_html_header.html	Header for HTML documentation generation
my_html_stylesheet.css	Cascading Style Sheets; formatting
my_latex_Frontpage.tex	LaTeX front page
my_latex_header.tex	LaTeX header design

[1]This file can combine multiple project files

Start.bat	Starts the documentation extraction and generation
Start_latex2pdf.bat	Generates a portable document format file
View.bat	Opens the HTML start page in the default browser
.\NExT\Documents\Doxygen\html\	HTML output of Doxygen
.\NExT\Documents\Doxygen\Images\	Images for Doxygen
InitiatorButtonSeqence.png	
philipslogo.eps	
philipslogo.png	
.\NExT\Documents\Doxygen\latex\	Doxygen LaTeX output files
.\NExT\inc\	Header files with declarations
error_strings.h	This files assigns error messages to tags
next_interfaces.h	Some global definitions for both GUI and functional part
next_platform.h	Definitions for GUI abstraction
next_strings.h	This files assigns text strings to tags
NextClientFrameAbstraction.h	Client frame abstraction header
NextErrorMessages.h	Header file to assign clear text messages to error codes
NextFelicaSubfunctionAbstraction.h	Felica functional header
NextFileTrfSubfunctionAbstraction.h	Data transfer functional header
NextGenPreferencesSubfunctionAbstraction.h	General preferences functional header
NextHALCommonSubfunctionAbstraction.h	Header for common HAL subfunctions
NextHALInitiatorSubfunctionAbstraction.h	Header for HAL Initiator subfunctions
NextHALTargetSubfunctionAbstraction.h	Header for HAL Target subfunctions
NextHistorySubfunctionAbstraction.h	Header for the history window subfunctions
NextInitiatorSubfunctionAbstraction.h	Header for BFL Initiator subfunctions
NextInterfaceSubfunctionAbstraction.h	Header for the interface subfunctions
NextISO14443_3SubfunctionAbstraction.h	Header for ISO 14443 subfunctions
NextMainWinAbstraction.h	Header for the Windows GUI abstraction layer
NextMifareSubfunctionAbstraction.h	Header for Mifare subfunctions
NextTargetSubfunctionAbstraction.h	Header for BFL Target subfunctions
NextUtility.h	Header for helper functions
RclEntity.h	Header for the BFL entity
W32NextClientFrameUI.h	Header for the graphical user interface
W32NextFelicaSubfunctionUI.h	Header for Felica GUI subfunctions
W32NextFileTrfSubfunctionUI.h	Header for the data transfer GUI subfunctions
W32NextGenPreferencesSubfunctionUI.h	Header for the general preferences GUI subfunctions
W32NextHALCommonSubfunctionUI.h	Header for common HAL GUI subfunctions
W32NextHALInitiatorSubfunctionUI.h	Header for HAL Initiator GUI subfunctions
W32NextHALTargetSubfunctionUI.h	Header for HAL Target GUI subfunctions
W32NextHistorySubfunctionUI.h	Header for the history GUI subfunctions

W32NextInitiatorSubfunctionUI.h	Header for BFL Initiator GUI subfunctions
W32NextInterfaceSubfunctionUI.h	Header for the interface subfunctions
W32NextIso14443_3SubfunctionUI.h	Header for ISO 14443 GUI subfunctions
W32NextMainWinUI.h	Header for the Windows Procedure
W32NextMifareSubfunctionUI.h	Header for Mifare GUI subfunctions
W32NextTargetSubfunctionUI.h	Header for BFL Target GUI subfunctions
.\NExT\pics\	Graphical files for the GUI
green.bmp	Image of a green lamp
grey.bmp	Image of a grey (off) lamp
red.bmp	Image of a red lamp
.\NExT\Release\	
BuildLog.htm	Visual Studio build log output
NExT.exe	Compiled and linked binary application file
.\NExT\src\	
NextClientFrameAbstraction.cpp	Builds abstraction classes for subwindows
NextErrorMessages.cpp	Classes for translating error codes into text
NextFelicaSubfunctionAbstraction.cpp	NFC 212/424 kbps passive activation functional implementation
NextFileTrfSubfunctionAbstraction.cpp	Data transfer interface GUI – functionality
NextGenPreferencesSubfunctionAbstraction.cpp	Options interface GUI – functionality
NextHALCommonSubfunctionAbstraction.cpp	Common NFC functionality (HAL based)
NextHALInitiatorSubfunctionAbstraction.cpp	Initiator functionality (HAL based)
NextHALTargetSubfunctionAbstraction.cpp	Target functionality (HAL based)
NextHistorySubfunctionAbstraction.cpp	
NextInitiatorSubfunctionAbstraction.cpp	Initiator functionality (BFL based)
NextInterfaceSubfunctionAbstraction.cpp	BFL Interface functional implementation
NextISO14443_3SubfunctionAbstraction.cpp	NFC 106 kbps passive activation functionality
NextMainWinAbstraction.cpp	Builds the client frames of subwindows
NextMifareSubfunctionAbstraction.cpp	Mifare functional implementation
NextTargetSubfunctionAbstraction.cpp	Target functionality (BFL based)
NextUtility.cpp	Helper functions (e.g. ASCII/hex conversion)
RclEntity.cpp	BFL based reconfiguration (e.g. mode change)
W32NextClientFrameUI.cpp	GUI implementation for client frames
W32NextFelicaSubfunctionUI.cpp	NFC 212/424 kbps passive activation GUI
W32NextFileTrfSubfunctionUI.cpp	Data transfer GUI implementation
W32NextGenPreferencesSubfunctionUI.cpp	Generic preferences GUI implementation
W32NextHALCommonSubfunctionUI.cpp	NFC common HAL GUI
W32NextHALInitiatorSubfunctionUI.cpp	NFC HAL Initiator specific GUI
W32NextHALTargetSubfunctionUI.cpp	NFC HAL Target specific GUI

W32NextHistorySubfunctionUI.cpp	History GUI implementation
W32NextInitiatorSubfunctionUI.cpp	NFC BFL Initiator specific GUI
W32NextInterfaceSubfunctionUI.cpp	NFC BFL Interface GUI
W32NextIso14443_3SubfunctionUI.cpp	NFC 106 kbps passive activation GUI
W32NextMainWinUI.cpp	Windows Procedure
W32NextMifareSubfunctionUI.cpp	NFC 106 kbps passive activation GUI
W32NextTargetSubfunctionUI.cpp	NFC BFL Target specific GUI