



Open NFC Linux Edition - Porting Guide

Document Type:	Manual
Reference:	MAN_NFC_0711-028 Version 1.12 (11236)
Release Date:	Sept. 15, 2011
File Name:	MAN_NFC_0711-028 Open NFC Linux Edition - Porting Guide v1.12.pdf
Security Level:	General Business Use

Disclaimer

This document is licensed under the Creative Commons Attribution 3.0 license (<http://creativecommons.org/licenses/by/3.0/>). (You may use the content of this document in any way that is consistent with this license and if you give proper attribution (<http://www.open-nfc.org/license.html#attribution>)).

Copyright © 2007-2011 Inside Secure

Open NFC and the Open NFC logo are trademarks or registered trademarks of Inside Secure.

Other brand, product and company names mentioned herein may be trademarks, registered trademarks or trade names of their respective owners.

History

Version	Date	Comments
0.1	Nov. 3, 2007	First Draft
1.0	March 17, 2008	Release for MSF 1.4
1.1	March 18, 2008	Corrections on the source tree
1.2	April 10, 2008	Precisions about the license "Proprietary" of the Linux reference porting
1.3	Oct. 24, 2008	Release for MSF 3.0 Main changes concerning capability to connect to NCI Simulator or to a remote NFC Controller Server.
1.4	April 29, 2009	Precisions about GPL license, and about possible "tainting the kernel" messages in "troubleshooting" section
1.5	May 18, 2009	Include the support for the connection center.
1.6	Aug. 24, 2009	Add an advice about CHM viewer installation.
1.7	Oct. 26, 2009	Added list of supported Linux kernels
1.8	Feb. 4, 2010	Cosmetic changes
1.9	Dec. 8, 2010	Updated for the new client/server model.
1.10	Dec. 24, 2010	<ul style="list-style-type: none">• Corrected directory names.• Corrected library name (libuser.so)• Precision about "core" plus "nfc_hal_microread" tree requirement.• Add the "nfc_hal_aardvark" implementation for direct connection to the AARDVARK USB/I2C Adapter.
1.11	May 20, 2011	New document template
1.12	Sept. 15, 2011	<ul style="list-style-type: none">• Updating the document reference.• Removed references to the TTY interface• Added references to the NFC HAL for Simulator.• The Unicode wrapper has been removed.• Precision about validation with 2.6.28 kernel version.• Cosmetics and typos.

Summary of Contents

1	Introduction.....	6
2	Software description	7
2.1	Static description	7
2.2	Dynamic description of the server process	8
2.3	Dynamic description of the client library	9
2.4	Custom driver development.....	9
3	Delivery Description	11
3.1	File Tree	11
3.2	Contents	12
3.2.1	“core” directory	12
3.2.2	“nfc_hal_microread” directory	13
3.3	Reading CHM under Linux.....	13
3.4	Unicode Functions under Linux	13
3.5	Files Description	14
4	Generation.....	21
4.1	Configuration of the project.....	21
4.1.1	Setting the compilation environment	21
4.1.2	Setting the Open NFC Configuration	21
4.1.3	Setting the NFC HAL for MicroRead Configuration	22
4.2	Compiler command line	22
4.2.1	Generating the driver	22
4.2.2	Driver compatibility	22
4.3	Compiling.....	22
4.3.1	Generating the client library	22
4.3.2	Generating the server application using AARDVARK to access the NFCC	23
4.3.3	Generating the server application using CC client to access the NFCC	23
4.3.4	Generating the server application using driver to access the NFCC	23
4.3.5	Generating the driver	23
4.3.6	Generating the server application to access the NFC Simulator	23
4.4	Verifying the compilation process	23
4.5	Development Model.....	24
4.6	Development phase.....	24
4.6.1	Starting the server (AARDVARK variant)	24
4.6.2	Starting the server (cc client variant).....	24
4.6.3	Starting the server (custom driver variant)	25
4.6.4	Starting the User Application	26
4.6.5	Automating the startup.....	26
4.7	Using the driver in stand alone phase.....	26
5	License	28

Reference Documents

[1] SIS_NFC_0709-014 **Open NFC Core Edition - Porting Guide**

[2] SIS_NFC_0707-003 **Open NFC Core Edition – API Specification**

1 Introduction

This document is the Linux porting guide for the NFC software stack “Open NFC v4.3.1 r0”.

This document describes the Linux implementation of the “Linux porting” described in document reference [1].

The Linux porting of the Open NFC stack is based on the client-server porting of Open NFC. It relies on the generation of the following executables:

- A NFC Server, named **server**.

This server includes the server part of the hardware-independent Open NFC stack, and also the hardware-specific part (NFC HAL module).

We provide three variants of the server for accessing the MicroRead hardware:

- A server that uses a direct connection to the Aardvark, that gives a USB interface to I2C, for communication with the EVB. This variant uses the Totalphase Aardvark driver for Linux.
- A server that uses a TCP/IP connection to with the connection center application: it is the preferred way to communicate to the EVB.
- A server that uses a custom driver to communicate with the NFC controller. This is likely to be the configuration to be used on a custom board, for instance when the NFC controller is configured to communicate using I2C interface. We provide a sample driver that must be adapted to the hardware, named ***open_nfc_driver.ko***

We also provide a server variant for accessing the NFC Simulator. This variant uses a TCP link to the Connection Center (running on a Windows machine), which is the only available way to interface with the NFC Simulator.

- A dynamic library, for Client access in User-Mode, named ***userlib.so***,
- A TCP Server, used for running the test bundles, named ***test_agent***

This document describes:

- An overview of the Linux Porting software,
- How to install the Linux Porting delivery,
- The way to generate the executables named above
- The files requiring user customization, hopefully, limited as much as possible; the main customization consists in defining the cross-compilation.
- The way to set up and run the system for the first time
- The way to diagnose problems that could occur when starting the system.

2 Software description

2.1 Static description

The four binaries, that are part of the Linux porting software, are shown on the diagram below in dark blue. The Open NFC server implementation relies on the AARDVARK USB/I2C adapter, or TCP/IP to a NFC Controller Server (via the Connection Center), or to a custom driver implementation to address the NFC controller.

The test agent is delivered as an example of User Application. This component is optional, and will not be in the customer's final product.

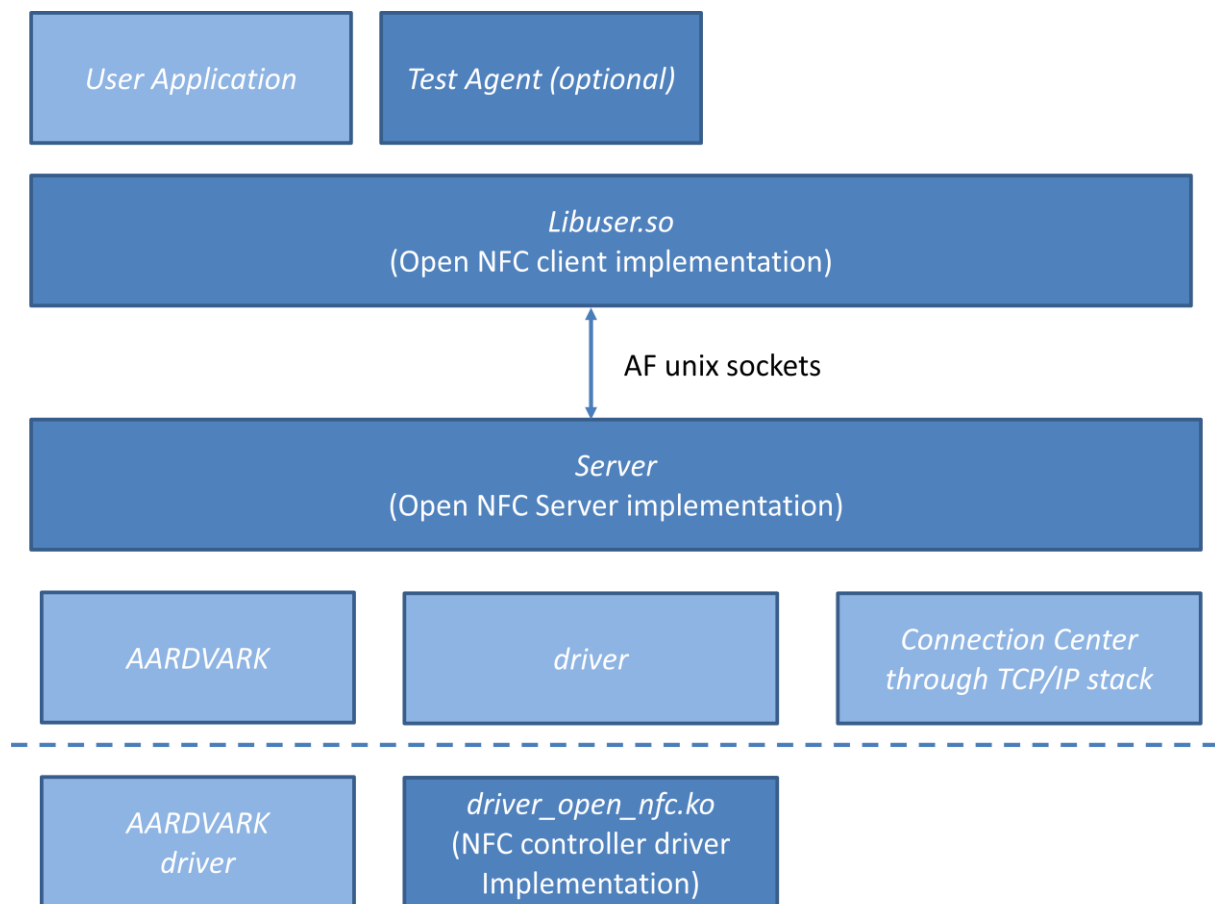


Figure 1: Static description of the Linux porting

2.2 Dynamic description of the server process

The figure below shows the different execution units created by the server:

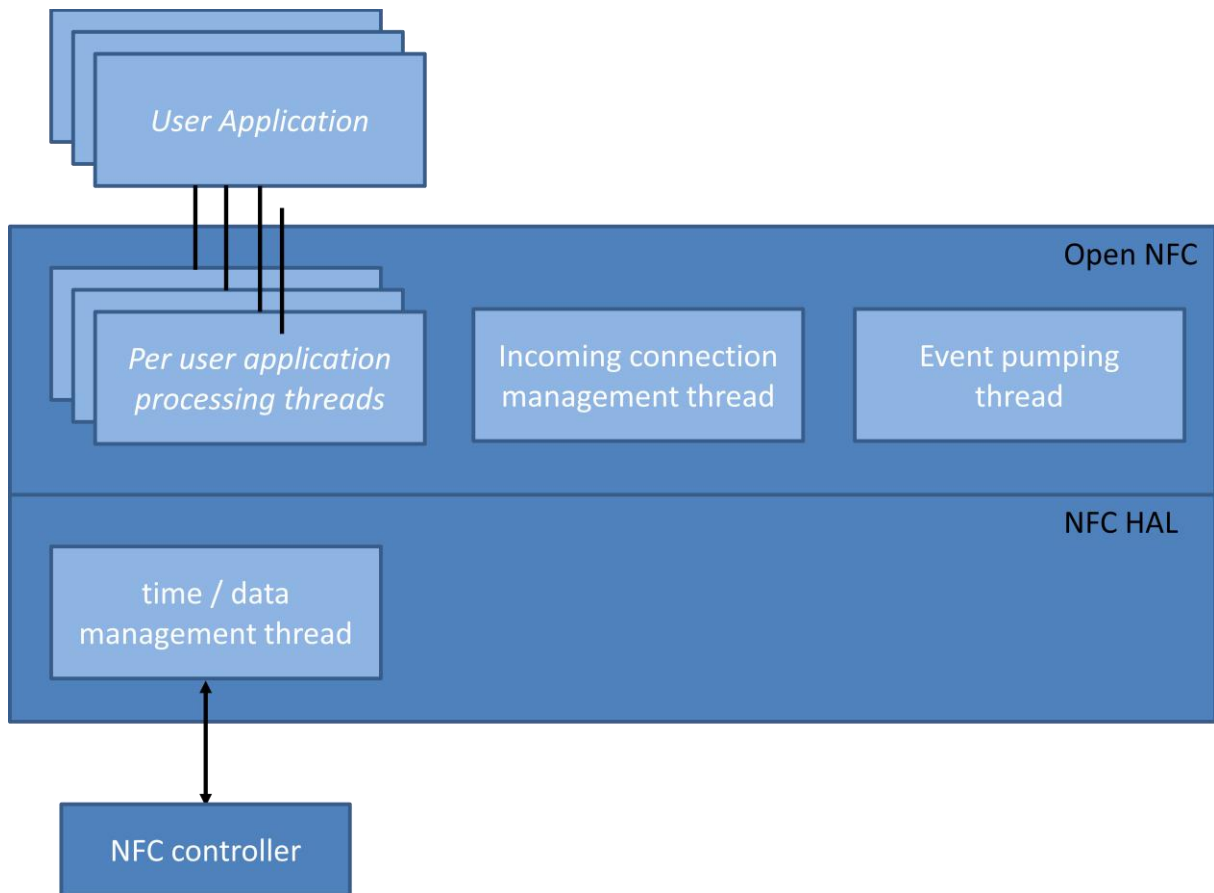


Figure 2: Dynamic description of the server application

The server first establishes a communication with the NFC controller.

Depending on the configuration of the server, the establishment of the communication consists in:

- Opening Aardvark driver using Linux USB hotplug, and the USB drivers for the Aardvark I2C/SPI Host Adapter provided by TotalPhase.
- Using the Connection Center library to establish a connection with a remote connection center application that provides a NFC controller access.
- Opening `/dev/nfcc` and performs some `ioctl`s to configure and establish the communication with the NFC controller. The driver interface will be described later in the document.

Once the connection with the NFC controller established, the server calls `PDriverCreate()` to boot the NFC controller, launches a thread used to pump events (event pumping thread) from the NFC HAL and waits the completion of the NFC controller boot sequence.

During PDriverCreate processing, a new thread dealing with the timer and data exchanges with the NFC controller is created in the NFC HAL implementation (time /data management thread).

After a successful NFC controller boot, the server creates a server UNIX socket and starts waiting for client connections (Incoming connection management thread).

For each incoming client connection, the server creates a new thread dealing with the processing of commands (per user application processing thread) sent by the client and restart waiting for new client connections.

Once a client connection is closed, the related resources (including processing threads) are released.

2.3 Dynamic description of the client library

The upper layers of Open NFC are compiled as a client library that must be linked towards Open NFC client applications.

The client library does not create execution units (threads) by itself. It is up to the application to decide how it wants to manage the tasks of pumping events (e.g. perform calls to WBasicPumpEvent()...)

As described in the Open NFC client API, a client application must call WBasicInit() to register itself in the server. The communication between the client application and the server application is established by the client library during the processing of this command.

A simple application model is to call WBasicInit() to connect to the server and then create a thread that calls WBasicEventLoop() to start processing of the events.

2.4 Custom driver development

The Open NFC server variant has been designed to use a driver conforming to the following requirements:

- The driver is implemented using the standard Linux character device model
- The transmission of data to the NFC controller is handled by the write method of the driver, e.g. correspond to a user write() operation.
- The reception of data from the NFC controller is handler by the read method of the driver, e.g. corresponds to by a user read() operation.
- The driver provides three IOCTL to reset the NFC controller, configure the driver and establish the connection with the NFC controller.
- The driver implements a poll method to inform the server of the availability of data sent by the NFC controller (read) and to inform the server of the end of the reset procedure (write)

The establishment of the communication channel with the NFC controller is performed during PComCreate() execution.

Once the communication established, the standard NFC HAL boot procedure is started.

Below, the main steps of the server application startup:

- The server opens the device associated with the driver, typically /dev/nfcc.
- Then the server performs the OPEN_NFC_IOC_CONFIGURE ioctl to provide configuration information to the driver. This step is always done, even if the driver does not require any configuration. In this case, the driver should simply return 0, indicating the ioctl success.
- Then, the server performs the OPEN_NFC_IOC_CONNECT ioctl to request establishment of the communication link with the NFC controller. This step is always be done, even if the driver does not require a connect operation. In this case, the driver should simply return 0, indicating the ioctl success.

The standard NFC HAL boot procedure starts here:

- Execution of OPEN_NFC_IOC_RESET ioctl
- The server waits for /dev/nfcc to become writable (indicating the end of the reset operation) using select().
- The server waits for /dev/nfcc to become readable (indicating the presence of data sent by the NFC controller), using select(), retrieves data sent by the NFC controller using read() and sends data to the NFC controller using write().

The description of the different ioctls can be found in the file open_nfc_int.h located in the driver example directory.

3 Delivery Description

This delivery contains the following elements:

- the implementation of Open NFC in portable C source code,
- a reference porting on Linux,
- the NFC Simulator tool for PC
- some examples of applications and
- the product documentation including the C API specification.

3.1 File Tree

The Linux Porting directory contains source files, written in C language, and makefiles for generation of the binaries.

Two packages need to be extracted into the same directory, indicated below “<Open NFC root directory>” :

- Open NFC - Linux Edition vx.y.z (nnnn).zip
It contains the “**core**” directory
- Open NFC - NFC HAL for Microread vx.y.z (nnnn).zip
It contains the “**nfc_hal_microread**” directory.

<Open NFC root directory>

```
|-- core
|  |-- connection_center
|  |-- documents
|  |-- interfaces
|  |-- nfcc_simulator
|  |-- porting
|  |  |-- linux
|  |  |  |-- client_server
|  |  |  |  |-- client
|  |  |  |  |-- common
|  |  |  |  \-- server
|  |  |      \-- microread
|  |  |         |-- hal_aardvark
|  |  |         |-- hal_cc_client
|  |  |         |-- hal_driver
|  |  |         |  \-- driver
|  |  |         |  \-- driver_i2c
|  |  |         \-- simulator
|  |  |            |-- nfc_hal_simulator
|  |  |            |  \-- interfaces
|  |  |            \-- sources
|  |  |-- examples
|  |  \-- test_agent
|  \-- sources
\-- nfc_hal_microread
    |-- bin
    |-- config
    |-- documents
    |-- interfaces
```

\-- sources

3.2 Contents

3.2.1 “core” directory

The delivery contains the following files and folders in “core” directory:

./

The Release Notes of this product.

./sources:

- The C source code of the implementation.

./interfaces:

- The C header files containing the API for the porting HAL and the C API.

./porting/linux:

- The reference porting on Linux. See section below.

./porting/linux/ examples:

- Some examples of applications to test Open NFC..

./nfcc_simulator:

- *MAN_NFC_0905-114 Open NFC - NFC Controller Simulator - User's Manual v1.1.pdf*
The manual describing the usage of the NFC Simulator.
- *NFC Simulator.exe*
The executable of the simulator of NFC Controller on PC.

./nfcc_simulator/virtualcards:

The files used for the simulation of the cards.

./connection_center

- *Connection Center.exe*
The connection center executable and the corresponding DLLs.

./documents:

- *MAN_NFC_0711-028 Open NFC Linux Edition - Porting Guide.chm*
The Porting Guide document describes the reference porting.
- *SIS_NFC_0707-003 Open NFC - API Specification.chm*
The C API documentation for the applications.
- *SIS_NFC_1005-194 Open NFC - NFC HAL API Specification.chm*
This document describes the specification of the interface between the NFC software stack and the NFC Controller.
- *SIS_NFC_0806-058_V3.1 Open NFC - NFC HAL Protocol Specification.pdf*
This document describes the protocol used for the NFC HAL.
- *STS_NFC_0707-001 NFC Tag Type 5 Specification - v1.6.pdf*
Specification for the Type 5 Tags
- *STS_NFC_0707-002 NFC Tag Type 6 Specification - v1.2.pdf*
Specification for the Type 6 Tags
- *MAN_NFC_0904-106 Open NFC - Connection Center - User's Manual v1.2.pdf*
The user's manual for the Connection Center tool.

3.2.2 “nfc_hal_microread” directory

The delivery contains the following file in “**nfc_hal_microread**” directory:

“REN_NFC_****-*** Open NFC - NFC HAL v*. * for Microread - Release Notes.pdf”

It contains the Release Notes of this product, and details the contents of this package. Read this document for details about the content of this package.

3.3 Reading CHM under Linux

The Open NFC API documentation is delivered in CHM files (Microsoft Compressed HTML Help file). This format is very convenient to describe APIs.

We strongly advise you to install a chm viewer on your Linux development platform, to be able to easily navigate through the Open NFC documentation.

Depending on your environment, you should install **gnochm** if you are under Gnome, **kchmviewer** if you are under KDE or some outdated tools but still working like **xchm**.

3.4 Unicode Functions under Linux

Most systems define the “wide-char” as a 16-bits unsigned integer. The Open NFC Library uses a ‘tchar’ type for accessing wide characters, which is defined with this 16-bits unsigned integer.

However, this type in Linux systems defaults to 32-bits value.

So, care must be taken that UTF-8 string and WCHAR_T strings must be converted to UTF-16 strings before calling Open NFC functions including “tchar” parameters.

An example of the simple conversion needed is demonstrated in the Linux examples provided with the Linux Edition.

3.5 Files Description

The following files are contained in the delivery; the following array also includes a short description of each file:

In Directory "core/porting/linux"

File Name	Description
LICENSE-2.0.txt	Apache v2.0 license
makefile.rules	Dependence rules for building the targets
makefile.settings	Environment variables common to the targets (cross_compiler definition, ...)
NOTICE	Notice file associated to Apache License

In Directory "core/porting/linux/client_server/client"

makefile	This is the makefile used for the client library generation.
porting_client.c	Contains the implementation of functions specific to the client porting of Open NFC.
porting_config.h	This header files contains the compilation configuration of Open NFC, customized for the client porting of Open NFC.
porting_inline.h	This header file contains the inline definitions for the porting of Open NFC, customized for the client porting of Open NFC.
porting_sync.c	Contains the implementation of synchronizations functions specific to the client porting of Open NFC.

In Directory "core/porting/linux/client_server/common"

File Name	Description
porting_mem.c	Contains the implementation of memory management functions common to client and server ports of Open NFC.
porting_traces.c	Contains the implementation of memory management functions common to client and server ports of Open NFC.
porting_types.h	This header files contains the basic types used by Open NFC, common to client and server ports of Open NFC.
porting_secure_element.c	Contains the implementation of the secure element HAL

In Directory "core/porting/linux/client_server/server"

File Name	Description
porting_config.h	This header files contains the compilation configuration of Open NFC, customized for the server porting of Open NFC.
porting_inline.h	This header file contains the inline definitions for the porting of Open NFC, customized for the server porting of Open NFC.
porting_server.c	Contains the implementation of functions specific to the server porting of Open NFC.
porting_startup.c	Contains the implementation of functions dealing with the start of the Open NFC server application.
porting_startup.h	This header file contains the prototypes of functions defined in the porting_startup.c file.
porting_sync.c	Contains the implementation of the synchronization functions
porting_types.h	This header file defines the types used in Open NFC

In Directory "core/porting/linux/client_server/server/microread"

File Name	Description
nal_porting_config.h	This header files contains the compilation configuration of NFC HAL for MicroRead.
nal_porting_inline.h	This header file contains the inline definitions for the porting of NFC HAL for MicroRead.
nal_porting_traces.c	Contains the implementation of trace functions specific to the porting of NFC HAL for MicroRead.
nal_porting_types.h	This header files contains the basic types used by NFC HAL for MicroRead, common to all server implementations of Open NFC.

In Directory "core/porting/linux/client_server/server/microread/hal_aardvark"

File Name	Description
aardvark.c	Aardvark Interface Library, provided by TotalPhase
aardvark.h	Header associated with the Aardvark Interface Library
aardvark.so	Aardvark Linux shared object, provided in binary by TotalPhase
I2CAardvark.c	Wrapper to the Aardvark library
I2CAardvark.h	Header of the wrapper to the Aardvark library
linux_porting_hal.c	Implementation of the HAL functions required by Open NFC, using Aardvark library to access to the NFC controller.
linux_porting_hal.h	Header associated with the file linux_porting_hal.c.
main.c	The main file of the server application
makefile	The makefile used to compile the server application

In Directory "core/porting/linux/client_server/server/microread/hal_cc_client"

File Name	Description
ccclient.c	Connection Center client library source code, used to communication with a remote connection center.
ccclient.h	Header containing the prototypes of public functions of ccclient.c
linux_porting_hal.c	Implementation of the HAL functions required by Open NFC, using CC client library to access to the NFC controller.
linux_porting_hal.h	Header associated with the file linux_porting_hal.c.
main.c	The main file of the server application
makefile	The makefile used to compile the server application

In Directory "core/porting/linux/client_server/server/microread/hal_driver"

File Name	Description
linux_porting_hal.c	Implementation of the HAL functions required by Open NFC, using a driver to access to the NFC controller.
linux_porting_hal.h	Header associated with the file linux_porting_hal.c.
main.c	The main file of the server application
makefile	The makefile used to compile the server application

In Directory "core/porting/linux/client_server/server/microread/hal_driver/driver"

File Name	Description
Makefile	The makefile used to compile the linux driver.
open_nfc_ccclient.c	Kernel port of the CC client library
open_nfc_ccclient.h	Header associated with the file open_nfc_ccclient.c
open_nfc_int.h	The interface file that describes the ioctl interface of the driver
open_nfc_main.c	The main file of the driver
open_nfc_main.h	The header associated with the file open_nfc_main.c

In Directory "core/porting/linux/client_server/server/microread/hal_driver/driver_i2c"

File Name	Description
Makefile	The makefile used to compile the linux driver.
open_nfc_custom.c	Skeleton of a real I2C implementation

NOTE: while almost complete, the file open_nfc_custom.c needs to be slightly adapted to reflect the hardware configuration (IRQOUT pin and RST/WakeUP pin configuration)

In Directory "core/porting/linux/client_server/server/simulator"

File Name	Description
ccclient.h, ccclient.c	Lightweight implementation of the Connection Center client library..
linux_nal_porting_hal.h, .c	Implementation of the NFC HAL interface.
linux_porting_hal.h	Header with the configuration for the NAL module.
main.c	The main file of the server application
makefile	The makefile used to compile the server application
nal_porting*	See MicroRead HAL for more information.

In Directory "core/documents"

File Name	Description
MAN_NFC_0711-028 Open NFC Core Edition - Porting Guide for Linux.pdf	This file.
Other files	Read the Release notes for details

In Directory "core/porting/linux/examples"

File Name	Description
linux_test_core.c	The common part of all examples
linux_test_core.h	The corresponding header file.
Makefile	The makefile used to compile all the examples

In Directory "core/porting/linux/examples/test firmware update"

File Name	Description
linux_test_firmware_update.c	The specific code of this example.
Makefile	The makefile to build the executable

In Directory "core/porting/linux/examples/ test_ndef_url"

File Name	Description
linux_test_ndef_url.c	The specific code of this example.
Makefile	The makefile to build the executable

In Directory "core/porting/linux/examples/ test_ndef_vcard"

File Name	Description
linux_test_ndef_vcard.c	The specific code of this example.
Makefile	The makefile to build the executable

In Directory "core/porting/linux/examples/ test_properties"

File Name	Description
-----------	-------------

linux_test_properties.c	The specific code of this example.
Makefile	The makefile to build the executable

In Directory "core/porting/linux/examples/ test_reader"

File Name	Description
linux_test_reader.c	The specific code of this example.
Makefile	The makefile to build the executable

In Directory "core/porting/linux/examples/ test_template"

File Name	Description
linux_test_reader.c	The specific code of this example.
Makefile	The makefile to build the executable

In Directory "core/porting/linux/examples/ test_agent"

File Name	Description
linux_ccclient.c	Connection Center Client for TCP/IP socket on Linux.
linux_ccclient.h	Header of the Connection Center Client.
linux_test_connector.c	Implementation of the Test Connector.
linux_test_connector.h	Header of the Test Connector.
Makefile	Makefile for generating the executable
test_agent.c	The TCP Server entry point

4 Generation

4.1 Configuration of the project

4.1.1 Setting the compilation environment

The file `<Open NFC root directory>/core/porting/linux/makefile.settings` is a template containing all the adaptation concerning the cross-compilation, post-generation actions (such as copying to an NFS directory)

The following environment variable requires adaptation to the customer environment:

variable name	description	Example setting
KDIR	The location of the Linux Kernel root directory	/lib/modules/\$(shell uname -r)/build
CROSS_PREFIX	The prefix used in the toolchain (gcc, ld, strip, ...)	arm-linux-
CROSS_COMPILER	The GCC cross-compiler full name	arm-linux-gcc
POST_COPY_DIR	When non empty, requests that all four executables produced during the build process be copied to a directory (typically a NFS directory, during development)	{ empty }
CFLAGS	User specific flags required for compilation	-g -O2 -D_DEBUG

4.1.2 Setting the Open NFC Configuration

The behavior of the Open NFC porting is configured at compilation time by a set of variables.

The following files, and only these files, contain some parameters to allow user customization of the Open NFC Porting; an explanation of the parameters contained in these files can be found in document reference [1]

- client_server/client/porting_config.h
- client_server/client/porting_inline.h
- client_server/server/porting_types.h

4.1.3 Setting the NFC HAL for MicroRead Configuration

The behavior of the NFC HAL for MicroRead porting is configured at compilation time by a set of variables.

The following files, and only these files, contain some parameters to allow user customization of the NFC HAL for MicroRead Porting; an explanation of the parameters contained in these files can be found in document reference [1]

- client_server/server/microread/nal_porting_config.h
- client_server/server/microread/nal_porting_inline.h
- client_server/server/microread/nal_porting_types.h
- client_server/server/microread/nal_porting_traces.c

4.2 Compiler command line

4.2.1 Generating the driver

When generating the driver open_nfc_driver.ko, the kernel build system is invoked with the following command line:

```
$(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules
```

The KDIR environment variable must be set accordingly to the root directory of kernel source tree (by default it is */lib/modules/\$(shell uname -r)/build* for a PC environment).

4.2.2 Driver compatibility

The compilation of the kernel module has been validated on the following kernel versions:

- 2.6.38.8 “vanilla” kernel from www.kernel.org with default configuration

Note: to obtain “default configuration” of a kernel, we use the following shell command:

```
make clean && make defconfig && make prepare && make modules_prepare
```

Compilation with other kernel versions should be quite straightforward, but may require some API adaptation due to the frequent changes of internal API of the Linux kernel in the 2.6.x series.

4.3 Compiling

4.3.1 Generating the client library

To generate the client library, issue a “make” command in the directory

```
<Open NFC root directory>/core/porting/linux/client_server/client
```

4.3.2 Generating the server application using AARDVARK to access the NFCC

To generate the server application, issue a “make” command in the directory

core/porting/linux/client_server/server/microread/hal_aardvark

4.3.3 Generating the server application using CC client to access the NFCC

To generate the server application, issue a “make” command in the directory

core/porting/linux/client_server/server/microread/hal_cc_client

4.3.4 Generating the server application using driver to access the NFCC

To generate the server application, issue a “make” command in the directory

core/porting/linux/client_server/server/microread/hal_driver

4.3.5 Generating the driver

To generate the server application, issue a “make” command in the directory

core/porting/linux/client_server/server/microread/hal_driver/driver

4.3.6 Generating the server application to access the NFC Simulator

To generate the server application, issue a “make” command in the directory

core/porting/linux/client_server/server/simulator

4.4 Verifying the compilation process

After generating the entire solution, the following files should exist in the directories as shown below :

Executable	Location
userlib.so	client_server/client
open_nfc_driver.ko	client_server/server/microread/hal_driver/driver
server	client_server/server/microread/hal_aardvark/
server	client_server/server/microread/hal_cc_client/

server	client_server/server/microread/hal_driver/
server	client_server/server/simulator

4.5 Development Model

In the development phase of an embedded Linux product, you need much more flexibility in the data exchange between target and host, than the actual product requires when it's ready.

A typical example is the use of an NFS server on the host computer, avoiding flashing the target every time a binary is generated. No downloading/uploading with FTP or similar tools is required, as the target accesses its file system directly on the host computer.

Later on, when your application and the target system configuration are ready, you want to build flashable file system images and boot the system entirely from flash (transition from development into stand alone).

4.6 Development phase

The following steps must be performed in order to use Open NFC in a development phase, where the file system is read-write.

4.6.1 Starting the server (AARDVARK variant)

Prerequisites:

The Linux driver is based on libusb and there is no need to install any other drivers on the operating system.

For information on access permissions, refer to the datasheets and the hotplug or udev infrastructure under Linux.

TotalPhase provides guidelines to set correctly the access permissions on various platforms in the file "tp-usb-drivers-v2.00" that can be downloaded on <http://www.totalphase.com>.

Launching the server:

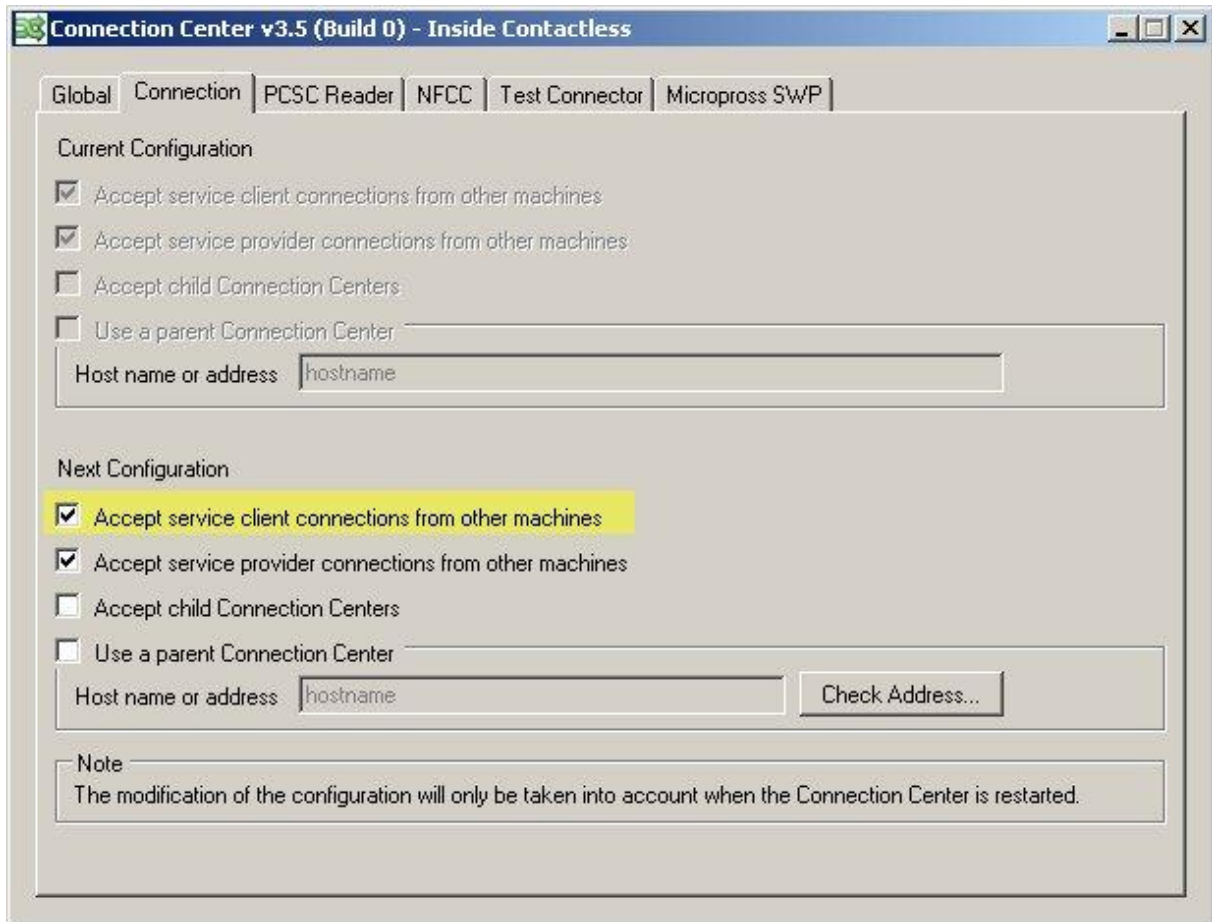
server

No argument is needed. The first available AARDVARK adapter will be used.

4.6.2 Starting the server (cc client variant)

Prerequisites:

The connection center must be running and configured to accept remote connections on the windows machine (be sure the firewall is not filtering this application).



Launching the server:

The CC client variant of the server application must be launched using the following syntax:

```
server 192.168.101.1
```

Where **192.168.101.1** corresponds to the IP address of the computer on which the connection server is running (the actual value must be adapted to your configuration).

4.6.3 Starting the server (custom driver variant)

Prerequisites:

The driver must be loaded prior launching the server. This can be done with an 'insmod' or 'modprobe' command.

```
insmod open_nfc_driver.ko
```

The example of driver sample uses devfs to create dynamically an entry in the /dev directory. According to your platform configuration, you may need to create a device entry manually using 'mknod' command.

If you do not make this character special file, the user applications will not be able to communicate with the driver.

```
mknod /dev/nfcc c 248 0
```

In this example, 248 corresponds to the value assigned when inserting the module in the upper insmod example command.

If you chose a dynamic major number, the major number has been assigned by the system. In this case, the user can retrieve the major number value by inspecting the `/proc/devices` entry.

```
cat /proc/devices
```

Then issue the appropriate mknod command.

Launching the server:

```
server /dev/nfcc
```

Where `/dev/nfcc` corresponds to the device node associated to the driver (the actual value must be adapted to your configuration).

4.6.4 Starting the User Application

At this stage, the user application can be started.

For example, if using the Test Server for starting test bundles onto the target, the following command must be issued:

```
test_agent -i 192.168.0.46
```

where **192.168.0.46** corresponds to the IP address of the computer on which the connection center is running.

4.6.5 Automating the startup

These tasks can also be automatically done during the system startup, by adding the corresponding commands to the start-up scripts of the target platform.

4.7 Using the driver in stand alone phase

When kernel and filesystem work as expected on the development platform, it is time to move to a stand alone product (where the target board is used without any host computer).

Several mechanisms are available for this :

- Initial RamDisk (initrd),
- cramfs,
- ramfs,
- jffs2

To build a root file system image, please consult the documentation of your target board, that will explain how to build and flash the images.

5 License

The source code of the driver sample "Open NFC Linux Reference Porting" is distributed using the "GPL v2.0" license. They can be freely adapted or modified and used to create a dynamic driver module or to be statically linked with the kernel without any license issue.

The remaining of the Open NFC core source code and the source code of the "Open NFC Linux Reference Porting" are distributed under Apache v2.0 license.