

NFC Handset Requirements
Version 7.0
23 March 2015

This is a Non-binding Permanent Reference Document of the GSMA

Security Classification: Non-confidential

Access to and distribution of this document is restricted to the persons permitted by the security classification. This document is confidential to the Association and is subject to copyright protection. This document is to be used only for the purposes for which it has been supplied and information contained in it must not be disclosed or in any other way made available, in whole or in part, to persons other than those permitted under the security classification without the prior written approval of the Association.

Copyright Notice

Copyright © 2015 GSM Association

Disclaimer

The GSM Association ("Association") makes no representation, warranty or undertaking (express or implied) with respect to and does not accept any responsibility for, and hereby disclaims liability for the accuracy or completeness or timeliness of the information contained in this document. The information contained in this document may be subject to change without prior notice.

Antitrust Notice

The information contain herein is in full compliance with the GSM Association's antitrust compliance policy.

Table of Contents

1 Introduction 5
1.1 Purpose 5
1.2 Scope and Objective 5
1.3 Use Cases/Services 6
1.4 Abbreviations 6
1.5 Definition of Terms 7
2 References 8
3 Terminology 10
4 Guidance for GSMA API Implementation 10
5 Generic Device Architecture 11
5.1 Dual Application architecture 11
5.2 Security 13
5.3 Mobile Wallet 14
6 Generic Device Requirements 14
6.1 NFC Device Architecture 14
6.2 Core Required NFC Features 15
6.2.1 NFC Controller Management 16
6.2.2 Card Emulation Mode Requirements 16
6.2.3 Reader/writer mode & TAG management requirements 17
6.3 Secure Element Access & Multiple Secure Elements Management 18
6.3.1 Mobile Device Modem Requirements 18
6.3.2 Secure Element Access API requirements 19
6.3.3 Multiple CEE support 19
6.4 UI Application triggering requirements 21
6.5 Remote Management of NFC services 22
6.5.1 Mobile Device APN Management Requirements 22
6.5.2 UICC Remote Management (Access to UICC in connected mode) requirements 22
6.6 Security 22
6.6.1 Access API & Secure Element Access Control Requirements 22
6.6.2 NFC Event & Access Control requirements 23
6.6.3 Protecting APIs with Operator Certificate 23
6.7 SCWS support 24
6.8 Card Application Toolkit Support 24
6.9 Platform Dependent Properties 25
7 Android Operating System 26
7.1 NFC Device Architecture 26
7.2 Core Required NFC Features 27
7.2.1 NFC Controller management API 28
7.2.2 Card Emulation mode requirements 28
7.2.3 Reader/writer & TAG management requirements 28
7.3 Secure Element Access & Multiple Secure Elements Management 28

7.3.1 Mobile Device Modem requirements 28
7.3.2 Secure Element Access API requirements 28
7.3.3 Multiple CEE support 28
7.4 UI Application triggering requirements 31
7.5 Remote Management of NFC Services 31
7.6 Security 31
7.6.1 Access API & Secure Element Access Control requirements 32
7.6.2 NFC Event & Access Control requirements 32
7.6.3 Protecting APIs with Operator Certificate 33
7.7 SCWS support 33
7.8 Card Application Toolkit Support 33
7.9 Platform Dependent Properties 33
8 JavaME 34
8.1 NFC Device Architecture 34
8.2 Core Required NFC Features 34
8.3 Secure Element Access & Multiple Secure Elements Management 34
8.3.1 Mobile Device Modem requirements 34
8.3.2 Secure Element Access API requirements 34
8.3.3 Multiple CEE support 35
8.4 UI Application triggering requirements 35
8.5 Remote Management of NFC Services 35
8.6 Security 35
8.6.1 Access API & Secure Element Access Control requirements 35
8.6.2 NFC Event & Access Control requirements 35
8.6.3 Protecting APIs with Operator Certificate 36
8.7 SCWS support 36
8.8 Card Application Toolkit Support 36
8.9 Platform Dependant Properties 36
9 Blackberry 10 OS 36
9.1 NFC Device Architecture 36
9.2 Core Required NFC Features 37
9.3 Secure Element Access & Multiple Secure Elements Management 38
9.4 UI Application triggering requirements 40
9.5 Remote Management of NFC Services 41
9.6 Security 41
9.7 SCWS support 41
9.8 Card Application Toolkit Support 41
9.9 Platform Dependant Properties 41
10 Windows Phone 8 42
10.1 NFC Device Architecture 42
10.2 Core Required NFC Features 42
10.3 Secure Element Access & Multiple Secure Elements Management 42
10.4 User Interface 42
10.5 Remote Management of NFC Services 42

10.6 Security 42
10.7 SCWS support 42
10.8 Card Application Toolkit Support 42
10.9 Platform Dependant Properties 42
Annex A Document Management 43
A.1 Document History 43
Other Information 44

1 Introduction

1.1 Purpose

With the increasing activity to deploy commercial Near Field Communication (NFC) services in a number of markets around the world, it is important to align implementation requirements and embrace common standards to support the global interoperability of services, while maintaining the momentum to meet time-to-market requirements in certain markets.

This document lists requirements for devices to support NFC services primarily focused on NFC services based on the UICC. It sets out a common framework of requirements, identifying and referencing relevant standards (or elements thereof), selecting options from among those allowed by existing standards to ensure interoperability. A list of relevant standards is captured in section 2 and further detailed by explicit requirements.

This document is delivered by the GSMA Terminal Steering Group (TSG), taking forward work driven by the GSMA NFC Fast Track project. It is an update to and replaces the NFC Handset APIs Requirement Specification Version 5 published by GSMA in March 2014.

Given the complexity of some of the underlying technology components and the variences across OS implementations, not all requirements could be finalised at this time. Where requirements are still work in progress, these are marked yellow. Work is ongoing to finalise these as soon as possible as well as to further enhance requirements and details/applicability for the various OS and to publish updates with the next document versions.

In case of any feedback or questions, you may notify us at prd@gsma.com.

1.2 Scope and Objective

The body of this document sets out requirements to be supported by mobile devices needed to support NFC services that are agreed globally, according to the GSMA's processes for consulting its members.

It should be noted that this document is expected to evolve by:

- Embracing new standards as and when they are published by the relevant industry organisations;
- Adding further requirements or further evolving current requirements as needed

The GSMA is defining the requirements' for NFC based services within Operating Systems (OS) and the device hardware which leverage the incumbent features of the OSs. Overall, the aim is to:

- Align members' terminal requirements for (primarily UICC) NFC based services
- Provide transferable solutions between different mobile device OSs and mobile devices;
- Provide the ecosystem with a quicker and simpler method for service deployment.

These ambitions will be fulfilled by adoption of the key NFC enablers, thereby facilitating a quicker time-to-market by providing clear and unambiguous device requirements.

This document defines at a high level the application architecture and lower layer enablers, required to fulfil NFC use cases. It further expands upon this, by detailing the particular mobile device Application Programming Interfaces (APIs) per OS (as applicable/ available) to enable a secured service use case and the requirements necessary to fulfil the NFC enabler software architecture.

Note, this Permanent Reference Document (PRD) does not exclude the possibility for support of additional NFC capabilities not mentioned in this document.

1.3 Use Cases/Services

The intended use cases for NFC can be grouped into *secured* and *non-secured* services. This document primarily targets the UICC based NFC *secured* service use cases, and can provide for the following propositions, but is not limited to:

- Plastic credit/debit card replacement
- Travel vouchers
- Business to Business transactions
- Secure access
- Mobile health
- IT system, e.g. RSA
- Touch and Pay
- Event ticketing.

It is required that the device and Universal Integrated Circuit Card (UICC) provide a *secured* environment, i.e. an environment which satisfies the security needs of Service Providers' (Mobile Network Operators' (MNOs)) and consumers.

1.4 Abbreviations

Term	Description
AC	Access Control
AID	Application ID
API	Application Programming Interface
APDU	Application Protocol Data Unit
APN	Access Point Name
BIP	Bearer Independent Protocol
CE	Card Emulation
CEE	Card Emulation Environment
CLF	Contactless Frontend
DUT	Device Under Test
JCP	Java Community Process
JVM	Java Virtual Machine
HCI	Host Controller Interface

Term	Description
JSR	Java Specification Request
MIDP	Mobile Information Device Profile
MNO	Mobile Network Operator
NFC	Near Field Communication
OS	Operating System
ODM	Original Device Manufacturer
OEM	Original Equipment Manufacturer
PoS	Point of sale
PRD	Permanent Reference Document
RIL	Radio Interface Layer
SCWS	Smart Card Web Server
SDO	Standards Organisations
SE	Secure Element
SIM	Subscriber Identity Module
SP	Service Provider
SWP	Single Wire Protocol
TSG	Terminal Steering Group
UI	User Interface
UICC	Universal Integrated Circuit Card
UID	User Identification

1.5 Definition of Terms

Term	Description
Active CEE	An Active CEE is a CE environment that can receive data based on routing mechanisms (e.g. AID, Protocol and Technology).
AID Conflict	When two or more applications register with the same Application Identifier
AID Conflict Detection	Conflict Detection is the procedure to check for AID Conflict. Conflict Detection can be done 1. During registration of Application Identifiers 2. When an Application selection is being received from a contactless reader.
APDU	An Application Protocol Data Unit (APDU) isthe communication unit between a smart card reader and a smart card.
Application Identifier	AID, as defined in ISO/IEC 7816-4, used to address a card emulation service or application
Battery Operational Mode	The battery of the DUT has sufficient power to support all functions in the mobile devices.
Battery Low Mode	The battery of DUT has reached “Battery Low Threshold” at which the display and most functionalities of the DUT are automatically switched off, except the clock and a few

Term	Description
	remaining functions. The battery of the DUT only has sufficient power to support NFC controller to function.
Battery Power-off Mode	The battery of the DUT has reached “Battery Power-off threshold” at which there is no residual power to support NFC controller to function. No functions are available in the DUT. The NFC controller can function if power is provided via the contactless interface (i.e. power by the field).

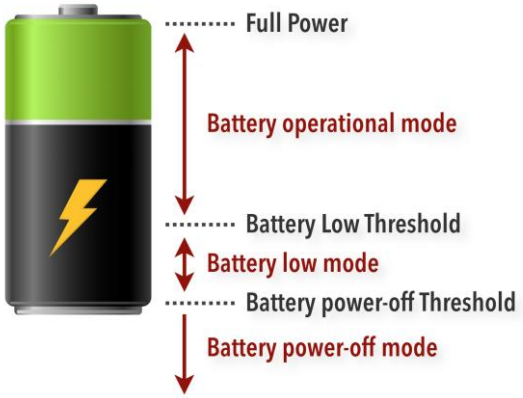

Figure 1: Battery power levels within the NFC mobile devices

Card Emulation Environment	A Card Emulation Environment is an execution environment used together with a NFC controller to manage a Card Emulation transaction. It can be a Secure Element (e.g. UICC, embedded Secure Element or micro-SD) or an application running in a device host.
Multiple Active CEEs model	A model where the device can activate several CEE at the same time. RF traffic can be provided to a CEE based on routing mechanisms. Note: an implementation may support Multiple Active CEEs model in Battery Operational Mode and Single Active CEE model in Battery Low or Power-Off Mode.
Screen Lock	The device functionality can only be accessed via a user intervention
Screen Off	When the phone screen is switched off e.g. the screen is black
Secure Element	A SE is a tamper-resistant component which is used to provide security, confidentiality, and multiple application environments required to support various business models.
Sensitive API	An API which shall be protected from malicious use
Single Active CEE model	A model where the device only activates one CEE at a time. Other CEEs, if available, are not active.

2 References

Note: Testing shall be based on the exact versions as indicated below. However if the manufacturers use a later release and/or version this should be indicated. TSG will take efforts to continually align with other SDOs for timely information about release plans.

3GPP Specifications	3GPP TS 31.111 (Release V9.8.0 or later) Universal Subscriber Identity Module (USIM) Application Toolkit (USAT) 3GPP TS 31.116 (Release V9.4.0 or later) Remote APDU (Application Protocol Data Unit) Structure for (Universal) Subscriber Identity Module (U)SIM Toolkit applications Later releases of 3GPP specifications shall be backward compatible The manufacturer can use Release 9 or a later release of the specifications
EMVCo Specifications	EMV Contactless Communication Protocol Specification, Book D, Version 2.4
ETSI SCP Specifications	ETSI TS 102 221 (Release V9.2.0 or later): Smart Cards; UICC- Terminal interface; Physical and logical characteristics ETSI TS 102 613 (Release V9.3.0 or later): UICC – Contactless Front End (Physical and data link layer characteristic) ETSI TS 102 622 (Release V9.4.0 or later): UICC – Contactless Front End, HCI (Host Controller Interface) ETSI TS 102 223 (Release V9.4.0 or later): Card Application Toolkit ETSI TS 102 226 (Release V9.6.0 or later): Remote APDU structure for UICC based applications Later releases of ETSI-SCP specifications shall be backward compatible The manufacturer can use Release 9 or a later release of the specifications
ISO Specifications	ISO/IEC 14443 Identification cards -- Contactless integrated circuit ISO/IEC 18092 Information technology -- Telecommunications and information exchange between systems -- Near Field Communication -- Interface and Protocol (NFCIP-1)
Japanese Industrial Standards	JIS X 6319-4 Specification of implementation for integrated circuit(s) cards -- Part 4: High speed proximity cards
Javadoc	Javadoc containing API’s requested within this PRD
JSR Specifications	JCP – JSR-000118: Mobile Information device Profile -Trusted MIDlet suites using X509 PKI JCP – JSR-000177: Security and Trust Services API for J2ME JCP – JSR-000257: Contactless Communication API JCP – JSR-000139 Connected Limited Device Configuration
NFC Forum Specifications	NFCForum-TS-Type-1-Tag_1.2 (or later) NFCForum-TS-Type-2-Tag_1.1 (or later) NFCForum-TS-Type-3-Tag_1.2 (or later) NFCForum-TS-Type-4-Tag_1.3 (or later) NFCForum-SmartPoster_RTD_1.0 (or later) NFCForum-TS-RTD_1.0 (or later) NFCForum-TS-RTD_Text_1.0 (or later) NFCForum DeviceRequirements_1.3 (or later) NFCForum-TS-Analog-1.0 (or later) NFCForum-TS-DigitalProtocol-1.1 (or later) NFCForum-TS-Activity-1.0 (or later) NFCForum-TS-NDEF_1.0 (or later) NFCForum Controller Interface 1.1 (or later)
SIMalliance	SIMalliance Open Mobile API Specification v2.05 or later (backwards compatible)

GSMA Requirements	NFC UICC Requirement Specification, Version 4.0
Global Platform	Secure Element Access Control Version 1.0
RSA Laboratory	PKCS #15 v 1.1: Cryptographic Token Information Syntax

3 Terminology

As per IETF Requirements terminology, reference RFC 2119, the following terms have the following meaning.

Term	Description
SHALL	Denotes a mandatory requirement
SHOULD	Denotes a recommendation
MAY	Denotes Optional

4 Guidance for GSMA API Implementation[1]

TS26_NFC_REQ_001	Where an OS API call does not return immediately, the OS implementation SHOULD be non-blocking.
TS26_NFC_REQ_002	Where an OS platform supports an exception mechanism, API design SHOULD use exceptions to indicate error conditions.
TS26_NFC_REQ_003	There SHALL be a single canonical API supporting the GSMA NFC Handset requirements for each OS.
TS26_NFC_REQ_004	Where an API is marked Deprecated, it SHALL be implemented by OS device platforms conforming to the API revision in which it was deprecated and by device platforms conforming to the next major revision. It MAY be implemented by device platforms conforming to later revisions.
TS26_NFC_REQ_005	Where support of the requirements in this document requires additions to the APIs provided by the vendor of an OS implementation, these additions SHALL be defined using a naming convention which reflects GSMA authorship of the API.

TS26_NFC_REQ_003 requires that there shall be a single canonical API supporting NFC Handset Requirements on each OS platform. This is intended to ensure that application developers for a given OS platform have a single, stable API to which they can develop on a given OS.

It is preferred that OS vendors provide support for all of the GSMA requirements in their NFC and Secure Element API definitions, since this provides the best assurance of a stable and well maintained developer API.

[1] Note: This section is considered relevant for any OS. It may however not equally and fully apply for every OS. Work will continue to make this section even more OS-agnostic, complemented by OS-specific guidance in the later chapters.

The GSMA may choose to define API extensions which provide the necessary features (for example, with the Android OS). Implementers of such extensions should take care to re-use existing OS mechanisms and practices where possible, to ensure the most consistent behaviour between different implementations of GSMA API extensions.

Where OS extensions are defined by the GSMA, the API for these extensions should be written clearly to identify GSMA authorship. The recommended way to implement this, for OS platforms supporting namespaces, is to use the namespace 'com.gsma.services.nfc' throughout the API definitions. For platforms not supporting namespaces – an example would be a C language API – a similar convention should be adopted, consistent with language limitations (e.g. com_gsma_services_nfc_ApiFunction() may be appropriate for a C language API). This is codified in TS26_NFC_REQ_005.

To assist platform vendors in implementing any GSMA platform specific requirements/APIs, the API definitions should include the following information:

Pre-conditions: These are assertions about platform state when an API is called. For example:

- NFC is enabled;
- The supplied instance of NfcAdapter is not null;

Processing: This describes the function performed by the API. It should include details of behaviour such as: blocking/non-blocking behaviour; threading behaviour (e.g. 'should only be called from the main UI thread'); timing constraints for operation, as appropriate.

Post-conditions: These are assertions about the platform state when the API returns. This section will typically indicate how the systems state should change after execution of the API.

Exception Handling: API definitions should include a clear description of expected API behaviour when an error occurs in processing, and/or when a precondition is not fulfilled. Experience shows that inadequately specified exception handling is among the most common causes of differing behaviour between implementations.

5 Generic Device Architecture

5.1 Dual Application architecture

GSMA Operators promote the following application architecture (below) to pragmatically support the key use case of *secured* NFC services.

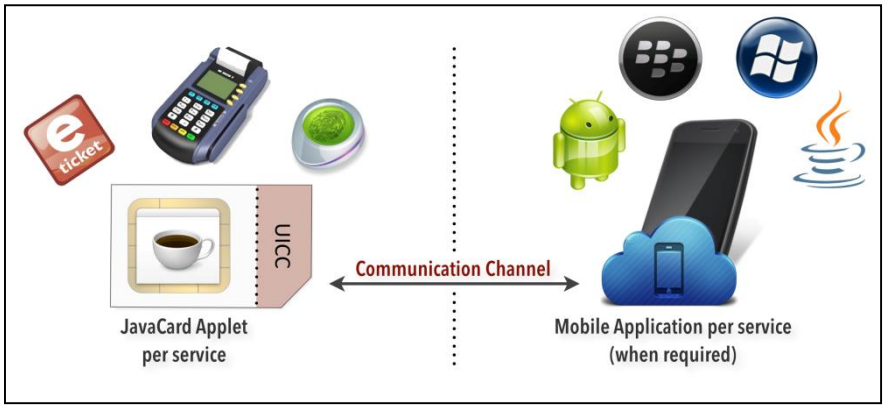

Figure 2: Dual application architecture

The mobile device User Interface (UI) application executing on the device OS is the consumer facing component. In this example, the UI application interacting with the UICC SE Application, communicating with the NFC reader, allows the customer to interact with the service functionalities, e.g. with a PoS (point of sale) for a financial service use case or a physical ticketing barrier in the case of an e-ticketing application. However the UI Application component is not seen as mandatory for all use cases, where the Service Provider (SP) could decide to have a UI-less service.

The *applet* component resides within the UICC, and works in tandem with the UI application when applicable. It holds the logic of the application and performs actions such as holding *secure authentication keys* or *time-stamped transaction data* for transaction resolution, history and fraud prevention etc.

Within this dual-application architecture for secured services, there is need for a consistent *communication channel* between these two applications. This communication channel could be used to transmit status information passed from the UICC to the UI for notifying the user on NFC events. It could also be used for more information exchanges between the UICC and the device UI like user authentication toward a UICC applet (e.g. PIN code verification).

As the communication channel accesses a *secured* storage space on the UICC, the communication channel itself must have attributes which allow it to be accessed only by authorised UI applications.

The following illustration gives an overview of the device software components required to satisfy the dual application architecture, which delivers key use cases for NFC.

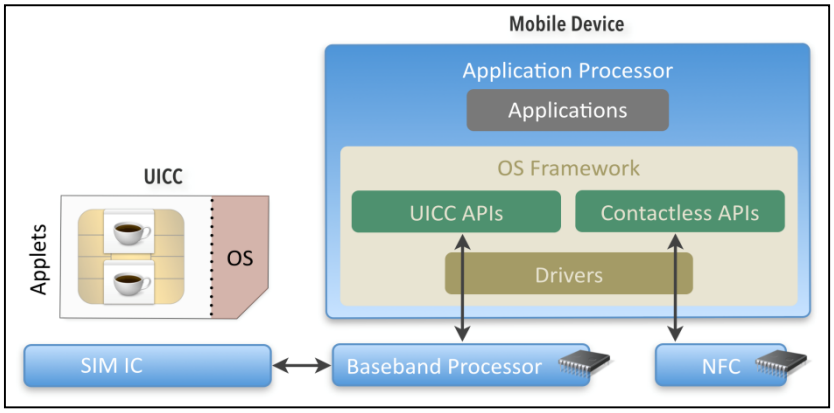

Figure 3: Mobile Device API generic software stack

The mandated method of communication between these two applications is ***APDU*** (Application Protocol Data Unit)**.**

The following figure depicts the typical data flow for an NFC transaction including the routing that the *event* will need to follow. The event is the trigger from the PoS to the user which indicates an activity in the NFC service. From this activity can be determined the nature of the *event* between the various components, for example where the *event* needs to be protected and has attributes which will allow for, or not allow for, any modification.

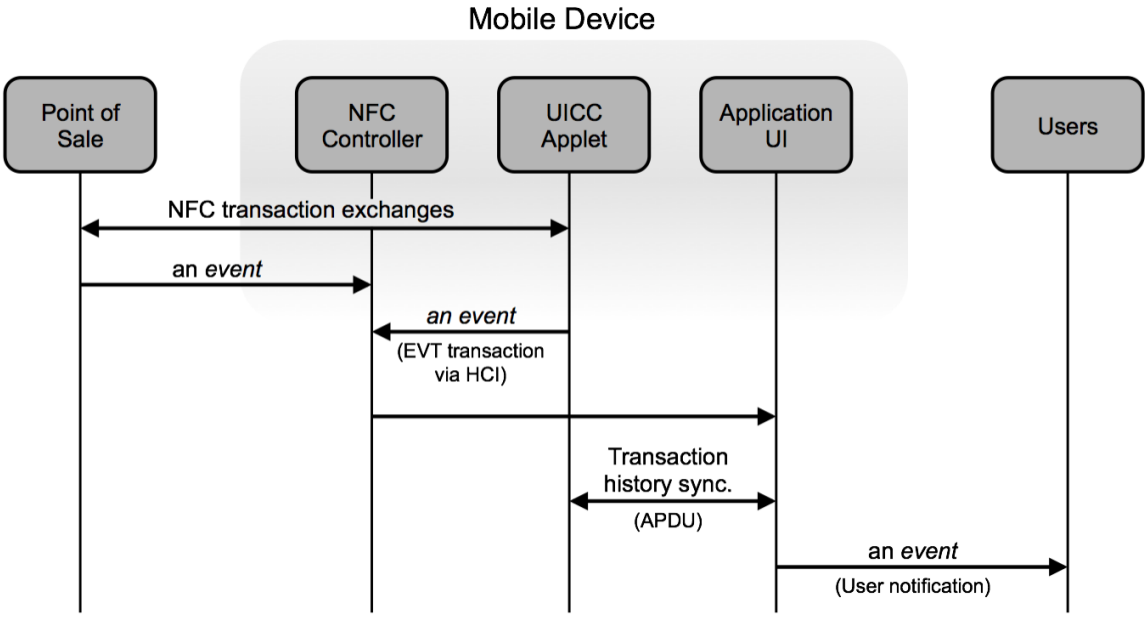

Figure 4 : Typical data flow for card-emulation mode

5.2 Security

For the *secured* services use case it is imperative for MNOs and SPs to continuously strive to provide best possible secured and trusted communication along the end-to-end chain of the various components necessary to provide for the *secured services* use cases.

- Two key areas where security is important are the UICC and the privileges available to communicate with the UICC NFC service applet. The UICC will securely hold

protected information (within the Secure Element), and provide a controlled access path to relevant parts of its internal memory.

- Access to services inside a Secure Element is requesting a specific care as a high level of security is required by some Service Providers. It is necessary to manage which device applications communicate with applets in the UICC. In addition to existing protection mechanisms provided by the mobile OS, a dedicated Access Control mechanism based on rules/rights provided by the UICC is needed. The main purpose of this *Access Control* is typically to prevent service attacks from malware applications.

5.3 Mobile Wallet

The *Mobile Wallet* is intended to facilitate the user experience, and allow the MNO or SP to optionally differentiate by providing targeted and convenient access to the NFC Services within the mobile device and UICC. The wallet application, for example, can typically list all SP services loaded into the mobile device or UICC and displays their current status. Additionally, this application may also allow the user to manage the NFC settings of their mobile device.

6 Generic Device Requirements

6.1 NFC Device Architecture

The following figure provides an overview of a typical Mobile NFC architecture:

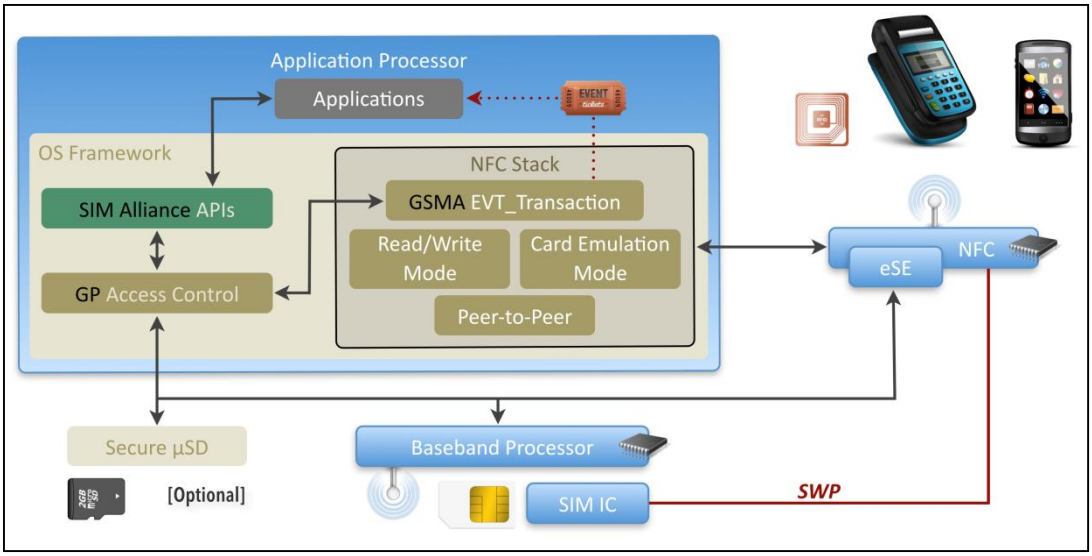

Figure 5: Mobile NFC Architecture

The device provides, as standard components, a NFC controller and one or more SEs.

The NFC Stack is driving the NFC Controller and is typically providing software APIs enabling:

- Management of Multiple Secure Element (activation, deactivation, routing, etc.)
- Management of the NFC events
- An external API available for 3rd party applications to manage reader/writer mode, Peer to Peer mode and Card Emulation mode from Device

- An internal API to provide a communication channel with an embedded Secure Element for APDU exchanges

The Secure Element Access API provides a communication channel (using APDU commands) allowing 3rd party applications running in the Mobile OS to exchange data with Secure Element Applets. This API provides an abstraction level common for all Secure Element and could rely on different low level APIs for the physical access:

- RIL extension for accessing the UICC
- Specific libraries for communicating with other embedded secure elements

In order to implement security mechanisms (e.g. Secure Element Access Control), the Secure Element Access API shall use Mobile OS mechanisms such as UIDs or application certificates to identify the calling application.

6.2 Core Required NFC Features

TS26_NFC_REQ_006	The NFC controller SHALL support SWP (Single Wire Protocol) interface with the UICC as per ETSI TS 102 613.
TS26_NFC_REQ_007	The NFC controller SHALL support HCI with the UICC as per ETSI TS 102 622.
TS26_NFC_REQ_006	The NFC controller SHALL support SWP (Single Wire Protocol) interface with the UICC as per ETSI TS 102 613.
TS26_NFC_REQ_007	The NFC controller SHALL support HCI with the UICC as per ETSI TS 102 622.
TS26_NFC_REQ_008	Contactless tunnelling (CLT=A) mode SHALL be supported for SWP (per ETSI TS 102 613).
TS26_NFC_REQ_009	Contactless tunnelling (CLT=F) mode SHOULD be supported for SWP (per ETSI TS 102 613). Note: This requirement is effective until 31st July 2015.
TS26_NFC_REQ_009.1	Contactless tunnelling (CLT=F) mode SHALL be supported for SWP (per ETSI TS 102 613). Note: This mandatory requirement will become effective from the 1st August 2015.
TS26_NFC_REQ_010	The device/ NFC controller interface with UICC SHOULD support Class B
TS26_NFC_REQ_011	The device/ NFC controller interface with UICC SHALL support Class C full power mode
TS26_NFC_REQ_012	The device/ NFC controller interface with UICC SHALL support Class C low power mode if the device supports battery power off mode
TS26_NFC_REQ_013	The device/ NFC controller interface with UICC SHOULD support Class C low power mode if the device supports battery power low mode
TS26_NFC_REQ_107	The device SHOULD indicate to the user the position of the NFC antenna reference point.
TS26_NFC_REQ_014	The device/NFC Controller interface with UICC SHALL support DEACTIVATED followed by subsequent SWP interface activation in full power mode.
TS26_NFC_REQ_015	The NFC controller SHOULD support both windows size set to 3 and set to

	4.
TS26_NFC_REQ_016	The NFC controller SHALL support SWP baud rate of at least 848 kb/s.
TS26_NFC_REQ_017	The NFC controller SHALL ensure that the UICC SWP/HCI initialization is finished before deactivating the SWP without full power down UICC. Note: The SWP/HCI specification is not integrating a recovery mechanism so in case of SWP line deactivation in the middle of the activation, it may lead to blocking situation, with SWP-UICC interface not usable until the next device boot.
TS26_NFC_REQ_018	VOID
TS26_NFC_REQ_019	The OS SHALL support configuration of the NFC Controller so that Card Emulation environment applications that are not accessed using ISO7816 AID can be configured to operate correctly. In particular, the OS SHALL support configuration of technologies & RF types.
TS26_NFC_REQ_020	When the mobile device is automatically switched off, and enters battery low mode, the mobile device SHALL be able to perform 15 transactions in card emulation within the following 24 hours.
TS26_NFC_REQ_021	NFC transactions SHALL be possible either in battery power off or battery low mode. Note: This is important for public transport services.

6.2.1 NFC Controller Management

The following features are needed:
Management of the NFC Controller state

- Check if the NFC Controller is enabled
- Activate the NFC Controller
- Check if Card Emulation mode is activated

TS26_NFC_REQ_022	There SHOULD be an API to ask the system to enable the NFC functionality. User input SHALL be required to enable NFC. This UI dialogue SHALL be generated by the OS and not by the calling application.
TS26_NFC_REQ_023	Devices SHALL provide APIs which allows the query of the NFC controller's state.
TS26_NFC_REQ_024	The OS implementation SHALL provide an API that allows an application to determine the state of the Card Emulation.
TS26_NFC_REQ_108	The device SHALL automatically turn off NFC, when the user activates Radio Off Mode, in line with regulation.
TS26_NFC_REQ_109	The OS SHALL allow the user to turn NFC on and off when the device is in Radio Off Mode.

6.2.2 Card Emulation Mode Requirements

TS26_NFC_REQ_025	The mobile device SHALL support Card-emulation as per ISO/IEC 14443 Type A and Type B PICC and SHOULD support Card Emulation as per ISO/IEC 18092 Type F. Note: For Type F implementation see JIS X 6319-4 which provides more details. (This is effective until the 31st July 2015)
TS26_NFC_REQ_025.1	The mobile device SHALL support Card-emulation as per ISO/IEC 14443

	Type A, Type B PICC and ISO/IEC 18092 Type F. Note: For Type F implementation see JIS X 6319-4 which provides more details. (Effective as of the 1st August 2015)
TS26_NFC_REQ_026	Card Emulation mode SHALL be enabled when the NFC is turned on.
TS26_NFC_REQ_027	For Card emulation mode the read distance SHALL be in the 0cm – 2cms range for battery operational mode, battery low or power off mode.
TS26_NFC_REQ_028	In Single Active CEE model the UICC shall be the default CEE, that is the active CEE at first start up or after a factory reset.
TS26_NFC_REQ_029	Manufacturers SHALL provide to operators the capability to customise settings for defining if the UICC card emulation is enabled / disabled when the device is powered off, screen is off or locked. Note: this will not be via a UI.
TS26_NFC_REQ_030	In the case of a factory reset, the operator customised settings (as per TSG26_NFC_REQ_029) SHALL remain.
TS26_NFC_REQ_031	Operator settings as stated in TS26_NFC_REQ_029 above SHALL only be valid if NFC is enabled.

6.2.3 Reader/writer mode & TAG management requirements

TS26_NFC_REQ_032	VOID
TS26_NFC_REQ_033	The mobile device SHALL support Reader/Writer Mode as per (ISO/IEC 14443 Type A and Type B PCD and ISO/IEC 18092 Type F Note: For Type F implementation see JIS X 6319-4 which provides more details
TS26_NFC_REQ_034	The mobile device SHALL support NFC Forum Tag Type 1, as specified in NFC Forum Type 1 Tag Operation Specification.
TS26_NFC_REQ_035	The mobile device SHALL support NFC Forum Tag Type 2, as specified in NFC Forum Type 2 Tag Operation Specification.
TS26_NFC_REQ_036	The mobile device SHALL support NFC Forum Tag Type 3, as specified in NFC Forum Type 3 Tag Operation Specification.
TS26_NFC_REQ_037	The mobile device SHALL support NFC Forum Tag Type 4, as specified in NFC Forum Type 4 Tag Operation Specification.
TS26_NFC_REQ_038	A reader mode events SHALL be routed exclusively to the UICC or the Application processor.
TS26_NFC_REQ_039	The default routing for the reader mode events SHALL be via the Application processor.
TS26_NFC_REQ_040	The NFC Controller SHOULD support Reader Mode as per ETSI TS 102 622
TS26_NFC_REQ_041	The device SHALL support automatic and continuous switching between card emulation and reader mode. Note: The impact on the standby time should be as minimal as possible. Further work will be progressed in TSG BLM to define power consumption test cases

Note: Default mode Card emulation mode, with a poll for Reader mode, the frequency for the Reader mode poll shall be such that the battery power consumption is kept to a minimum.

This implementation will require on-going optimisation; however, the aim is to provide good responsiveness to the consumer.

TS26_NFC_REQ_042	A transaction time SHALL take 500ms or less for TAG message length not exceeding 100 bytes. The transaction time is defined from the start of the frame of the first RF command receiving an answer, to the end of the frame of the response to the last received RF command by a device, where the RF command is used to read the content in a tag.
TS26_NFC_REQ_043	The mobile device SHALL be able to read/write the NFC Forum Smart Poster RTD.
TS26_NFC_REQ_044	The TAG SHALL be read at a distance of 1 cm and at distances between 0 to 1 cm. Note: This requirement will be tested with a TAG Test Reference system agreed in the Test Book group.
TS26_NFC_REQ_110	The TAG SHOULD be read at a distance from 1 cm to 4 cms. Note: This requirement will be tested with a TAG Test Reference system agreed in the Test Book group.

6.3 Secure Element Access & Multiple Secure Elements Management

This section details functionality which the GSMA requires to be implemented within the NFC Framework, in order to support requirements in this document related to handling of the NFC Controller, Card Emulation mode and multiple Secure Elements.

6.3.1 Mobile Device Modem Requirements

TS26_NFC_REQ_045	For handling logical channel Baseband SHALL provide interfaces based on either the following AT commands or equivalent functionality: AT+CCHO AT+CCHC AT+CGLA
TS26_NFC_REQ_111	Modem SHALL provide an interface based on AT+CRSM (Restricted SIM Access) or equivalent functionality for internal communication on basic channel
TS26_NFC_REQ_112	The Modem SHALL prevent the usage of the AT+CSIM or equivalent functionality
TS26_NFC_REQ_113	Modem SHALL support APDU transmission case 1, 2 3 & 4 as defined in ISO/IEC 7816-4 except Extended Length Field support.
TS26_NFC_REQ_114	For all APDU exchanges originating from the Secure Element Access API the Modem driver SHALL forward warning status codes (SW=62XX or 63XX) directly to the application level without any change and without automatically sending a GET RESPONSE command. In particular for case-4 APDU command, the Modem driver SHALL apply ISO 7816-3 Case 4S.4.
TS26_NFC_REQ_115	VOID
TS26_NFC_REQ_046	Access to the UICC (logical channel) SHALL be allowed even when the mobile device is in a Radio OFF state, i.e. flight mode, airplane mode etc.

6.3.2 Secure Element Access API requirements

The SIMalliance group has published the “Open Mobile API” specification. From this document, any mobile device manufacturer will be able to provide standardised APIs for access to the different Secure Elements such as the UICC SE.

TS26_NFC_REQ_047	OS implementations SHALL provide an API for communicating with the UICC
TS26_NFC_REQ_047.1	Communication with the UICC SHALL be done through the logical channels.
TS26_NFC_REQ_047.2	Communication with the UICC SHALL prevent access to basic channel (channel 0).
TS26_NFC_REQ_047.3	The API SHALL implement the SIMalliance Open Mobile API transport layer or provide an equivalent set of features.
TS26_NFC_REQ_048	VOID
TS26_NFC_REQ_049	VOID
TS26_NFC_REQ_050	VOID

6.3.3 Multiple CEE support

6.3.3.1 Single Active CEE model

The following requirements only apply where a device supports the Single Active CEE model.

6.3.3.1.1 NFC Controller Management API

TS26_NFC_REQ_051	The mobile device software platform SHOULD expose an API to select and switch the Active CEE. A change in the Active CEE selection SHALL not imply a power cycle of the device, modem or the UICC.
TS26_NFC_REQ_051.1	If available, the API for activating a CEE which is not the UICC SHALL be protected. This is considered a sensitive API.
TS26_NFC_REQ_051.2	If available, the API for activating the UICC as an active CEE SHALL NOT be protected
TS26_NFC_REQ_052	The OS implementation SHALL provide an API to query the status of the Active CEE for card emulation mode.

6.3.3.1.2 Card Emulation Mode Requirements

TS26_NFC_REQ_053	Only one CEE SHALL be active at any one time.
TS26_NFC_REQ_054	If there is more than one CEE, the default Active CEE SHALL be the UICC.
TS26_NFC_REQ_116	The OS implementation SHALL provide an API that allows an application to enable & disable Card Emulation Mode

6.3.3.2 Multiple Active CEE model

The following requirements only apply where a device supports the Multiple Active CEEs model.

TS26_NFC_REQ_117	In Multiple Active CEE model, the UICC SHALL be one of the active CEEs.

6.3.3.2.1 NFC Controller Management API

TS26_NFC_REQ_055	The device SHALL support an API that allows applications to dynamically register and un-register NFC application by list of AIDs
TS26_NFC_REQ_056	VOID
TS26_NFC_REQ_057	The device SHOULD support an API that allows applications to dynamically register and un-register NFC application by pattern (e.g. DESFIRE).
TS26_NFC_REQ_058	APIs SHALL provide a way for the applications to specify which card emulation environment their AIDs are to be routed to.
TS26_NFC_REQ_059	APIs SHALL provide a way for a mobile application to list all available secure elements in battery operational mode for NFC transactions.
TS26_NFC_REQ_060	APIs SHALL provide a way for a mobile application to list all configured secure elements in NFC controller for battery low and battery power off mode.
TS26_NFC_REQ_061	The device MAY support an OS mechanism that allows applications to statically register NFC application by list of AIDs.
TS26_NFC_REQ_062	The device SHOULD support an OS mechanism that allows applications to statically register NFC application by pattern (e.g. DESFIRE)

6.3.3.2.2 Card Emulation Mode Requirements

TS26_NFC_REQ_118.1	At first power up or factory reset the device SHALL set the route for NFCA and NFCB technologies (as defined in NFC Forum specification) to the UICC.
TS26_NFC_REQ_118.2	At first power up or factory reset the device SHALL set the route for ISO_DEP protocol (as defined in NFC Forum specification) to the UICC.
TS26_NFC_REQ_118.3	At first power up or factory reset the device SHOULD set the Default AID route (AID routing mechanism is defined in NFC Forum specification) to the UICC. Note: The Default AID Route" is the route used by the NFC Controller when a NFC reader is selecting explicitly a NFC Service by its AID but no NFC application registered this AID.
TS26_NFC_REQ_119	The device MAY provide a mechanism for changing the route for technology and protocol. Note: No API details provided at this stage. Further work will be progessed, to provide with the next release further details.
TS26_NFC_REQ_063	When an NFC application is uninstalled, the device SHALL remove all information related to this application from the routing table.
TS26_NFC_REQ_064	When an NFC application is updated, the device SHALL update the routing table according to the new registration information (removing/adding elements). Note: Static & dynamic elements from the previous version will be removed and static elements from the new version will be added.

TS26_NFC_REQ_065	The device SHALL provide a routing mechanism using the following priority: AID, then RF protocol and then RF technology defined in NCI specification [NFC Forum specifications]"[2].
TS26_NFC_REQ_066	VOID
TS26_NFC_REQ_067	When manual mechanism is used to register the new NFC service, the device SHOULD provide high level information to the user (i.e. NFC service name and not AID, etc.).

6.3.3.2.3 AID Conflict Resolution

TS26_NFC_REQ_068	The device SHALL provide a mechanism to handle AID Conflict.
TS26_NFC_REQ_068.01	AID Conflict resolution SHOULD follow the same mechanisms whether NFC services registration is dynamic or static.
TS26_NFC_REQ_068.02	When managing AIDconflict resolution, the device SHALL follow the end-user preferences.
TS26_NFC_REQ_068.03	When AID conflict resolution implies user choice at transaction time the user SHOULD be able to make their decision persistent, until the terms of the AID conflict change (another application is installed or removed, which claims the same AID), or the user goes into the menus to revisit their decision.

6.4 UI Application triggering requirements

When a transaction has been executed by an applet on a Secure Element, it may need to inform the application layer. To do this, an applet may trigger an event known as "EVT_TRANSACTION". This HCI event will be sent to the NFC Controller over SWP line. The NFC Controller will then forward this event to the device application processor where the event may trigger an authorized registered mobile application.

How to register a mobile application including the exact mechanism depends on the mobile OS used. This section intends to define the content of this event message and the main principles for its management.

The event message holds the following information:

- *SEName* (mandatory) reflecting the originating SE. It must be compliant with SIMalliance Open Mobile API naming convention and below complementary requirement in case of UICC.
- *AID* (mandatory) reflecting the originating SE (UICC) applet identifier if available
- *Parameters (mandatory)* holding the payload conveyed by the HCI event EVT_TRANSACTION if available
- When *AID* is omitted from the URI, application component are registered to any "EVT_TRANSACTION" events sent from the specified Secure Element.

TS26_NFC_REQ_069	For UICC, *Secure Element Name* SHALL follow the format hereafter: *"SIM" [smartcard slot]* (e.g. SIM/SIM1, SIM2, … SIMn).
TS26_NFC_REQ_070	For embedded SE, *Secure Element Name* SHALL follow the format

[2] Note: This requirement will be updated to also reference pattern, as soon as relevant specifications from NFC Forum or Global Platform are available.

	hereafter: "*eSE" [number]* (e.g. eSE/eSE1,eSE2, etc.).
TS26_NFC_REQ_071	The device SHALL support HCI event EVT_TRANSACTION as per ETSI TS 102 622.
TS26_NFC_REQ_072	The OS implementation SHALL provide a mechanism to inform authorised OS applications of Transaction Events and this SHALL include the Secure Element name and the AID of the applet which triggered the transaction and PARAMETERS holding the payload conveyed by the HCI EVT_TRANSACTION event.
TS26_NFC_REQ_073	The AID parameter SHALL be used during the process of triggering the UI application.
TS26_NFC_REQ_074	The NFC transaction mechanism SHALL be able to transfer to the targeted mobile application, the following information: Secure Element name, AID, PARAMETERS holding the payload conveyed by the HCI NFC transaction event.

6.5 Remote Management of NFC services

6.5.1 Mobile Device APN Management Requirements

TS26_NFC_REQ_075	For mobile devices supporting multiple APNs, the device SHALL be able to set-up a SIM OTA channel using the APN information that is provided in the OPEN CHANNEL command.
TS26_NFC_REQ_076	For devices which are configured as "Always-ON" and only support a single APN, the APN information provided in the OPEN CHANNEL command SHALL be ignored and the device SHALL use the device default APN.
TS26_NFC_REQ_077	If the APN information provided by the network in the OPEN CHANNEL command is empty the device SHALL always use the device default APN

6.5.2 UICC Remote Management (Access to UICC in connected mode) requirements

TS26_NFC_REQ_078	The mobile device SHALL support BIP in UICC client mode for UDP.
TS26_NFC_REQ_079	The mobile device SHALL support BIP in UICC client mode for TCP.
TS26_NFC_REQ_080	The mobile device SHALL support two concurrent channels, BIP in UICC client mode.
TS26_NFC_REQ_081	The mobile device SHALL support the SMS push (per ETSI TS 102 226 and 3GPP TS 31.116) to establish an open BIP channel as per ETSI TS 102 223 Open Channel Command
TS26_NFC_REQ_120	The device SHALL support the BIP session regardless of incoming or outgoing calls, incoming or outgoing MMS, SMS. Note: This is not applicable if the device is on a 2G network

6.6 Security

6.6.1 Access API & Secure Element Access Control Requirements

The main objective of the Access Control mechanism is to protect communication with the Secure elements.

To increase performance and to avoid physical access to the SE each time a control check is done, where most of the data which is *read* and *put* may be stored in a cached memory during boot time.

From this cache, the Access Control can determine if the relationship between the UI application and the SE applet (application signature/AID) is valid, and then authorise a communication or send an exception.

TS26_NFC_REQ_082	Open OS devices SHALL provide access control as per Global Platform, Secure Element Access Control specification.
TS26_NFC_REQ_083	When no access control data (files or applets) is found on the UICC the API SHALL deny access to the UICC.
TS26_NFC_REQ_121	Access Control Enforcer SHALL check if Access Rules has been updated only when a new logical channel is open. All rules SHALL stay valid until the channel is closed.
TS26_NFC_REQ_122	When the device is retreiving all rules from the Secure Element (ARF mechanism or ARA with GET DATA[ALL] mechanism), Access Control Enforcer SHALL cache the different data and only read rules from the UICC.

6.6.2 NFC Event & Access Control requirements

"EVT_TRANSACTION" messages are sensitive data. Intercepting these events might help a malicious application to lure a user into entering sensitive information into a fake UI.

The NFC stack shall therefore implement GlobalPlatform Secure Element Access Control specification to check that the recipient activity has been signed with an authorised certificate. This check is performed at the time the event is being forwarded from the lower layers to the target application. If no application is authorised as per "*Access Control"* check, then the event is discarded.

TS26_NFC_REQ_084	The OS implementation SHALL support the use of the GlobalPlatform Secure Element Access Control enforcer to manage Transaction Events originating from a Secure Element and SHALL ensure that this event is made available only to authorised OS applications.
TS26_NFC_REQ_085	The device SHALL prevent the case that an application UI is triggered from an applet when the access conditions would not allow the application UI to exchange APDUs with this applet and there is no rule explicitely granting the NFC event permission.

6.6.3 Protecting APIs with Operator Certificate

In order to protect the usage of *sensitive* APIs, the mechanism for granting access to them, shall be based on certificates stored within the UICC as illustrated below.

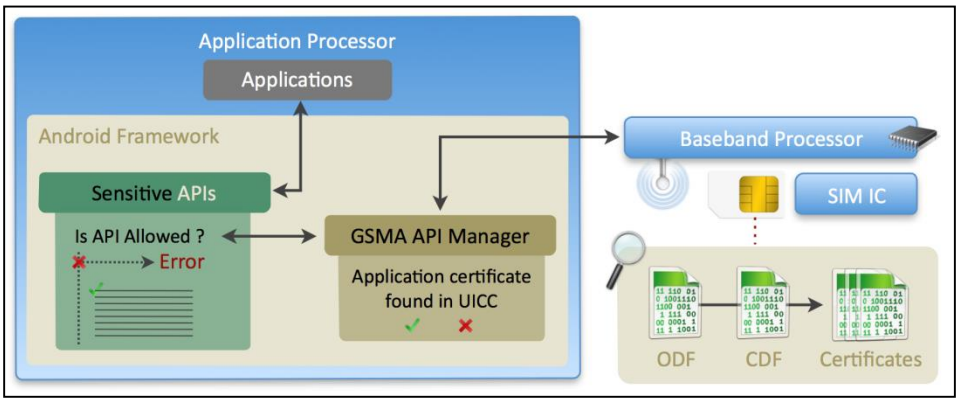

Figure 6: Protecting sensitive APIs with an Operator Certificate

As defined in the "PKCS #15 v 1.1: Cryptographic Token Information Syntax" specification from RSA, the certificates can be retrieved from the PKCS#15 structure where the EF_ODF points to elementary files that can be regarded as directories of certificates: "Certificate Directory Files" (CDFs).

Any sensitive API shall check if the application has been signed with one of the certificates stored in the UICC before executing the operation, and:

- Only authorised applications shall be granted access to this function.
- For any other non-authorised applications an exception needs to be raised.

6.7 SCWS support

TS26_NFC_REQ_086	VOID

6.8 Card Application Toolkit Support

The following requirements list the minimum letter classes' support for NFC device.

TS26_NFC_REQ_087	The device SHOULD support the letter class "c" with the following command and events: Proactive command: LAUNCH BROWSER Event download: Browser termination event Event download: Browsing status event
TS26_NFC_REQ_088	The device SHALL support the letter class "e" with the following commands and events[3]: Proactive command: OPEN CHANNEL ((UICC in client mode and with the support of UDP/TCP bearer)) Proactive command: CLOSE CHANNEL Proactive command: RECEIVE DATA Proactive command: SEND DATA

[3] Note: Letter class **e** (BIP commands and events set) is related to the requirement of section 6.5.2. (UICC Remote Management (Access to UICC in connected mode) requirements) Only if the OPEN CHANNEL command is in UICC server mode there is a relationship with SCWS.

	Proactive command: GET CHANNEL STATUS Event download: Data available Event download: Channel status
TS26_NFC_REQ_089	The device SHALL support the letter class “l” with the following command: • Proactive command: ACTIVATE
TS26_NFC_REQ_090	The device SHOULD support the letter class “m” with the following command and event: • Event download: HCI connectivity event
TS26_NFC_REQ_091	The device SHOULD support the letter class “r” with the following commands and events: • Proactive command: CONTACTLESS STATE CHANGED • Event download: Contactless state request

6.9 Platform Dependent Properties

This is intended to provide a mechanism to query for necessary platform dependent characteristics such as:

- number of secure elements supported
- use of battery low mode vs battery off mode
- support for NFC polling with backlight off
- support for NFC listening with backlight off

TS26_NFC_REQ_092	An API SHALL be provided to enable a calling application to query implementation specific choices and information. The information item made available provided by this API is referred to in subsequent clauses as a ‘property’[4]
TS26_NFC_REQ_092.01	The value returned by a property query is read-only. It SHALL not be possible for a caller to change the value returned in a property query. The value returned SHALL not change while the OS is running, but MAY change after a reboot or system update.
TS26_NFC_REQ_092.02	A property SHALL be provided that returns the version of device requirements supported. For implementations conforming to the requirements in this version of the document, the value 5000 SHALL be returned as a platform-appropriate integer value of not less than 32 bits precision
TS26_NFC_REQ_092.03	VOID
TS26_NFC_REQ_092.04	A property SHALL be provided that returns TRUE on devices supporting operation in Battery Low mode, otherwise it SHALL return FALSE
TS26_NFC_REQ_092.05	A property SHALL be provided that returns TRUE on devices supporting operation in Battery Power Off mode, otherwise it SHALL return FALSE.
TS26_NFC_REQ_092.06	A property SHALL be provided that returns TRUE on devices supporting NFC Forum Type 3 Tag card emulation on the device host, otherwise it SHALL return FALSE
TS26_NFC_REQ_092.07	A property SHALL be provided that returns TRUE on devices supporting

[4] Note: This requirement does not take precedence above related mandatory requirements. Rather it facilitates to provide confirmation to calling applications that the requested functions are indeed supported.

	Card Emulation for FeliCa on UICC, otherwise it SHALL return FALSE.
TS26_NFC_REQ_092.08	A property SHALL be provided that returns TRUE on devices supporting MIFARE Classic Reader/Writer, otherwise it SHALL return FALSE.
TS26_NFC_REQ_092.09	A property SHALL be provided that returns TRUE on devices supporting MIFARE DESFire Reader/Writer, otherwise it SHALL return FALSE.
TS26_NFC_REQ_092.10	A property SHALL be provided that returns TRUE on devices supporting OMAPI features, otherwise it SHALL return FALSE.
TS26_NFC_REQ_092.11	A property SHALL be provided that returns TRUE on devices supporting HCI/SWP functionality between the CLF & the UICC, otherwise it SHALL return FALSE.

Note: Version Numbering

Each revision of this document defines a version number for the Handset Requirements contained herein. Implementations are required, by TS26_NFC_REQ_092.02, to return the version number of the requirements supported.

The convention chosen is to return the version number as an integer value, with the precise type used being one which is natural to the OS platform (with the proviso of a minimum of 32 bits precision).

The overall version number is calculated based on a major version number and minor version number. The major version number is a value between 5 and 2147483 and the minor number is a value between 0 and 999. The calculation is:

version_number = (major_version_number * 1000) + minor_version_number

For this document, the major version number is ‘5’ and the minor version number is ‘0’.

Document versioning should be managed in such a way that an application can use the major version number alone to determine the supported requirements and APIs. Minor version number changes should be used for backward compatible changes and clarifications only.

7 Android Operating System

7.1 NFC Device Architecture

Android does not differ from other platforms regarding the NFC ecosystem. It is likely that Android will provide, as standard components, a NFC controller and one or more Secure Elements (SEs). Access to NFC has already been introduced with Gingerbread *(Android v2.3 SDK release)*; however one of the major issues remaining is the lack of access to the UICC SE.

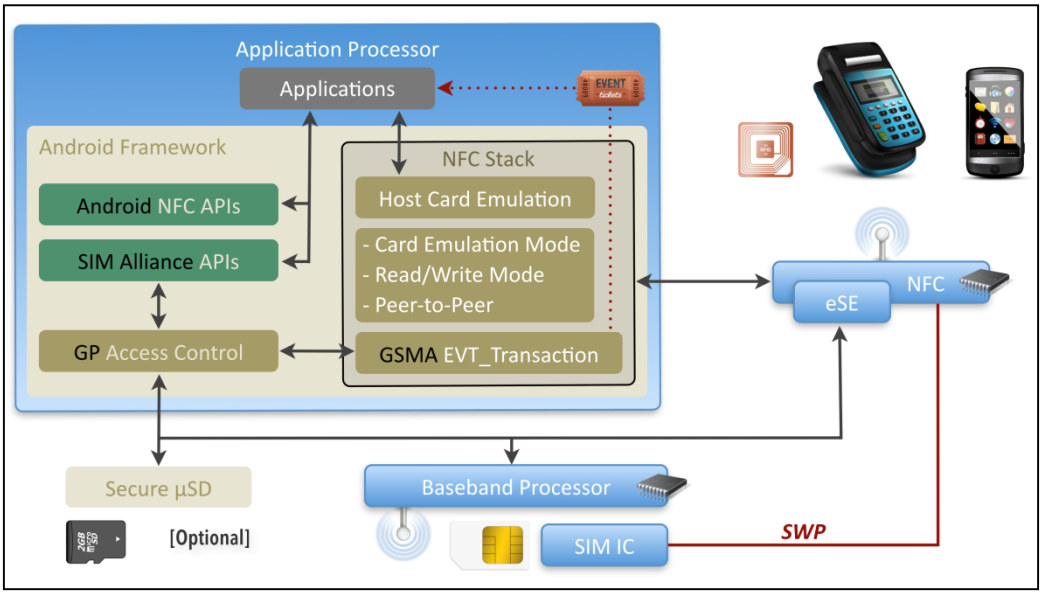

Figure 7 : Android NFC software stack

Previous figures give an overview of a possible Android implementation as an example showing how this requirement can be mapped to an OS

On Android the architecture could be encapsulated in an *Android Service*. Having a single service ensures that security checks (who is accessing the service) and resource management (freeing up a logical channel) can be guaranteed.

On Android, such a background component might rely on a RIL extension for accessing the UICC and on some specific libraries, for communicating with any embedded secure elements.

7.2 Core Required NFC Features

TS26_NFC_REQ_123	All GSMA APIs SHALL be included in the following jar file: “com.gsma.services.nfc.jar” Note: Library SHALL be built as a shared library in order to be used with the following tag: <uses-library android:name="com.gsma.services.nfc” android:required="false"/>

7.2.1 NFC Controller management API

TS26_NFC_REQ_093	There SHALL be APIs (as described below) to check if the NFC Controller is enabled and to ask the system to enable the NFC functionality. User input SHALL be required to enable NFC. This UI dialogue SHALL be generated by the OS and not by the calling application. Note: Refer to the Javadoc linked to this document for more details.

"com.gsma.services.nfc.NfcController"	
Classes	**Methods**
nested classes (NfcController.Callbacks)	All methods except "onCardEmulationMode"
NfcController	isEnabled, enableNfcController

7.2.2 Card Emulation mode requirements

No specific requirement, see Generic Device Requirements

7.2.3 Reader/writer & TAG management requirements

No specific requirement, see Generic Device Requirements

7.3 Secure Element Access & Multiple Secure Elements Management

7.3.1 Mobile Device Modem requirements

No specific requirement, see Generic Device Requirements

7.3.2 Secure Element Access API requirements

TS26_NFC_REQ_124	The device SHALL support SIMAlliance Open Mobile API, but only the transport layer.
TS26_NFC_REQ_125	SIMAlliance Open Mobile API and Global Platform Access Control SHALL be initialized and ready to use when the BOOT_COMPLETED intent is sent.

7.3.3 Multiple CEE support

All generic device requirements are applicable in addition to below specific requirements for Android.

7.3.3.1 Single Active CEE model

TS26_NFC_REQ_126	As per TS26_NFC_REQ_024, 051, 052 and 116 devices SHALL implement classes and methods as described below:

	"com.gsma.services.nfc.SEController"[5]
	Classes / **Methods**
	nested classes (SEController.Callbacks) — All methods
	SEController — All methods
	"com.gsma.services.nfc.NfcController"[5]
	Classes / **Methods**
	nested classes (NfcController.Callbacks) — onCardEmulationMode
	NfcController — disableCardEmulationMode, enableCardEmulationMode, isCardEmulationEnabled

7.3.3.2 Multiple Active CEE model

The following requirements only apply where a device supports Multiple Active CEEs model.

TS26_NFC_REQ_094	When a mobile application is registering an AID it SHALL be able to state the applicable CEE.
TS26_NFC_REQ_095	When multiple CEEs are available on the device, the device SHALL support routing between these.
TS26_NFC_REQ_133	The device SHALL support an OS mechanism that allows applications to statically register NFC application by list of AIDs.
TS26_NFC_REQ_127	As per TS26_NFC_REQ_055 devices SHALL implement classes and methods as described below. Note: Refer to the Javadoc linked to this document for more details.

"com.gsma.services.nfc.NfcController"

Classes	Methods
nested classes (NfcController.Callbacks)	All methods except "onCardEmulationMode"
NfcController	All methods except "disableCardEmulationMode, enableCardEmulationMode, isCardEmulationEnabled"

"com.gsma.services.nfc.AIDGroup"

Classes	Methods
AIDGroup	All methods

[5] Note: Refer to the Javadoc linked to this document for more details.

	"com.gsma.services.nfc.OffHostService" **Classes** / **Methods** OffHostService / All methods
TS26_NFC_REQ_127.1	APIs SHALL also manage "Off-Host" services registered statically through the Manifest
TS26_NFC_REQ_128	The device SHALL handle a "Long Press" on all "Tap&Pay" menu entries.
TS26_NFC_REQ_128.1	When a "Long Press" is done on a "Tap&Pay" menu entry, the device SHALL send the following intent to the application that has created this entry. intent: com.gsma.services.nfc.SELECT_DEFAULT_SERVICE
TS26_NFC_REQ_134	The device SHALL provide an additional menu entry in "Settings" in order to enable/disable group of AIDs (as defined by Android) belonging to the category "Other". A group of AIDs SHALL only be enabled/disabled as a single unit.
TS26_NFC_REQ_134.1	The menu entry SHOULD display: • A banner representing the groups of AIDs belonging to the category "Other" (and optionally the group description) with its current status (enabled/disabled) and a way for disabling/enabling it • A visual indication representing the NFC Controller capacity and showing - The space used by the selected group - If enablement of the selected group can fit with the remaining space of the NFC Controller routing table
TS26_NFC_REQ_134.2	The menu SHALL be hidden to the end user until the first time an NFC Service cannot be added in the NFC Controller routing table and thereafter the menu is visible.
TS26_NFC_REQ_135	When an application is trying to register new "Off-Host" NFC service(s) from the Manifest and the NFC Controller cannot add them in its routing table, the device SHALL • Inform the end user that some NFC Services proposed by the application cannot be used. A message SHALL provide the description of the group(s) of AIDs (android:description) which cannot be activated. • Propose the end user should disable some previously installed NFC services using the feature described in TS26_NFC_REQ_134 in order to free some NFC Controller routing table space When one AID from a group of AIDs cannot be added in the NFC Controller routing table, the entire group of AIDs SHALL not be enabled.
TS26_NFC_REQ_136	When a customer is selecting a service from the "Tap&Pay" menu and all related AIDs cannot be added in the NFC Controller routing table, the device SHALL • Inform the end user that activation of the selected NFC services cannot be performed • Propose the end user should disable some previously installed NFC services using the feature described in TS26_NFC_REQ_134 in order to free some NFC Controller routing table space

	If the end user doesn't disable enough NFC services to allow activation of the selected "Tap&Pay" menu, previous "Tap&Pay" entry SHALL stay active and the end users selection is cancelled.

7.4 UI Application triggering requirements

The same generic requirements are applicable to Android platform with the following requested implementation:

TS26_NFC_REQ_129	A Transaction Event (EVT_Transaction) SHALL be triggered based on the following information:

Action	*com.gsma.services.nfc.action.TRANSACTION_EVENT*
Mime type	-
URI	*nfc://secure:0/<SEName>/<AID>* - *SEName* reflects the originating SE It must be compliant with SIMAlliance Open Mobile APIs - *AID* reflects the originating UICC applet identifier

Table 1: Intent Details for TRANSACTION Event (EVT_TRANSACTION)

TS26_NFC_REQ_096	Transaction event data SHALL be set in the following extended field:

com.gsma.services .nfc.extra.AID `ByteArray`	Contains the card "Application Identifier" *[optional]*
com.gsma.services .nfc.extra.DATA `ByteArray`	Payload conveyed by the HCI event "EVT_TRANSACTION"

Table 2: Transaction Event (EVT_TRANSACTION) data

7.5 Remote Management of NFC Services

No specific requirement, see Generic Device Requirements

7.6 Security

TS26_NFC_REQ_130	The Android Framework SHALL define the following permissions for handling Open Mobile APIs and Transaction events: - Permission-group "**android.permission-group.NFC**" SHALL be defined in Android framework to host all NFC-related permissions as following: permission-group android:name = android.permission-group.NFC android:label = @string/permgrouplab_NFC = "NFC - Near Field Communication" android:icon = @drawable/perm_group_network android:description = @string/permgroupdesc_NFC = "Access various NFC features"

	android:priority = 270 - Permission "**org.simalliance.openmobileapi.SMARTCARD**" SHALL be defined in Android framework as following: android:name = org.simalliance.openmobileapi.SMARTCARD android:permissionGroup = android.permission-group.NFC android:protectionLevel = dangerous android:label = @string/permlab_SmartcardService = "Access to SIM Card" android:description = @string/permdesc_SmartcardService = "Allows application to communicate with the SIM card" - Permission "**com.gsma.service.nfc.permission.TRANSACTION_EVENT**" shall be defined in Android framework as following: android:name = **com.gsma.service.nfc.permission**.TRANSACTION_EVENT android:permissionGroup = android.permission-group.NFC android:protectionLevel = dangerous android:label = @string/permlab_nfcReceiveEvent = "NFC Transaction awareness" android:description = @string/permdesc_nfcReceiveEvent = "Allows application to receive information about NFC transaction performed"

7.6.1 Access API & Secure Element Access Control requirements

No specific requirement, see Generic Device Requirements

7.6.2 NFC Event & Access Control requirements

The same generic requirements are applicable to Android platform with the following requested implementation:

Handling transaction events
Usage of "EVT_TRANSACTION" events must request the following permissions. All of them must be set to "dangerous" protection level.

TS26_NFC_REQ_131	The device SHALL request the following permission for receiving a Transaction event:

NFC Controller	android.permission.NFC
Transaction Event	com.gsma.services.nfc.permission.TRANSACTION_EVENT

Table 3: EVT_TRANSACTION Permissions

Transaction intents link an Android application and an applet installed on a Secure Element. For this reason, securing them shall be done with the same rules that restrict applet access by the Android application through the SIMAlliance Open Mobile APIs.

If an application is registered to any "EVT_TRANSACTION", by omitting the AID in the Intent, it will receive the events of any applets to those accessible using the "*Access Control"*.

The NFC stack shall therefore use internal "*Access Control"* APIs to check that the recipient activity has been signed with an authorised certificate. This check is performed at the time the event is being forwarded from the lower layers to the target application. If no application is authorised as per "*Access Control"* check, then the event is discarded.

Receiving transaction events

- Unicast - Reception by only one activity

TS26_NFC_REQ_097	By default, each time an "EVT_TRANSACTION" event is sent; it SHALL be received by only one activity

For any "unicast" reception, framework shall use "`startActivity(…)`" methods to inform the selected activity of the event.

- Multicast - Reception by several application components (activity, services…)

TS26_NFC_REQ_098	The Device SHALL support for Multicast mode for "EVT_TRANSACTION"
TS26_NFC_REQ_099	The Device SHALL support an API to switch from Unicast mode to Multicast mode for "EVT_TRANSACTION" management as described below. "com.gsma.services.utils.Handset"[5] **Classes** / **Methods** Handset / enableMultiEvt_transactionReception This is considered a sensitive API

7.6.3 Protecting APIs with Operator Certificate

No specific requirement, see Generic Device Requirements

7.7 SCWS support

No specific requirement, see Generic Device Requirements

7.8 Card Application Toolkit Support

No specific requirement, see Generic Device Requirements

7.9 Platform Dependent Properties

TS26_NFC_REQ_132	As per TS26_NFC_REQ_092 , the device SHALL implement classes and methods as described below: "com.gsma.services.utils.Handset"[5] **Classes** / **Methods** Handset / getProperty, getVersion

8 JavaME

8.1 NFC Device Architecture

The following J2ME NFC architecture is used within mobile devices currently commercialised for NFC services. They host and run services based on the UICC card with a companion application to perform user-friendly display and interactions.

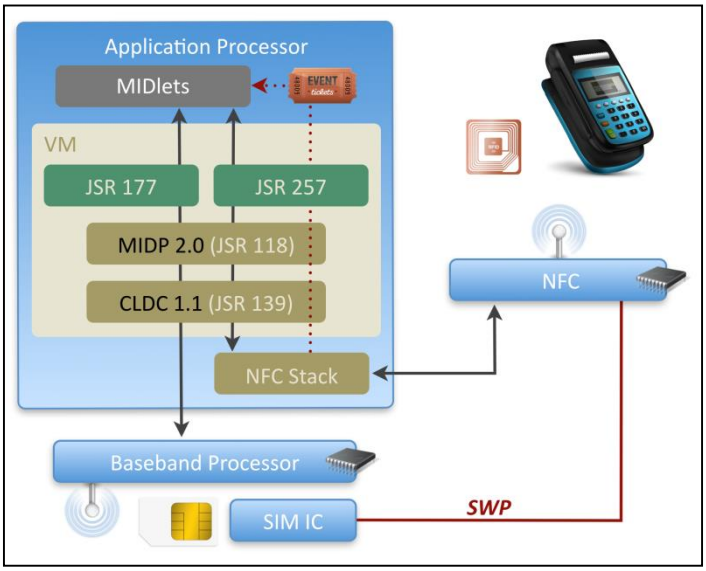

Figure 8: JavaME NFC software stack

The current implementation has been accepted by major suppliers within the ecosystem. Services such as payment and transport are already deployed with this architecture. Therefore no future work is planned regarding this platform. Recent efforts in this area aim to develop various mobile device models with a wider range of suppliers, in particular targeting the low-tier device market space.

8.2 Core Required NFC Features

No specific requirement, see Generic Device Requirements

8.3 Secure Element Access & Multiple Secure Elements Management

8.3.1 Mobile Device Modem requirements

No specific requirement, see Generic Device Requirements

8.3.2 Secure Element Access API requirements

Below requirements are replacing the generic requirements.

Service applications (MIDlets) communicate with their partner UICC application through JSR177.

TS26_NFC_REQ_100	The J2ME implementation SHALL support JSR 177: UICC secure application management (APDU Exchange) This capability enables to communicate with applications loaded into the UICC by exchanging ISO 7816-4 APDU.

TS26_NFC_REQ_101	The mobile device SHALL support SATSA-APDU package as per JSR 177 specifications.

8.3.3 Multiple CEE support

No specific requirement, see Generic Device Requirements.

8.4 UI Application triggering requirements

Below requirements are replacing the generic requirements.
Contactless transactions are registered for using JSR257 (push MIDlet mechanism). These events trigger the start of the MIDlet, which then exchanges data with the associated UICC application.

TS26_NFC_REQ_102	The mobile device SHALL support JSR 257:Contactless Communication API specifications.
TS26_NFC_REQ_103	For JSR 257 implementation, the mobile device SHALL NOT allow access to the UICC.

Note: Where JSR 177 should be used for communication with the UICC.

8.5 Remote Management of NFC Services

No specific requirement, see Generic Device Requirements.

8.6 Security

To protect against unauthorised applications, access to UICC is limited as follows:
An application's signature which does not link the application to a trusted domain has no access to the UICC:

- UICC defines the chain of certificates that the MIDlets need to be signed up with, in order to be attached to a trusted domain;
- Further restrictions can be specified using Access Control Files and Applets to limit some trusted applications to only access certain UICC applications and to only send certain APDUs (as per JSR177 Annex A).

Protection of card transaction events is not granted until the associated service is installed. Once installed, the MIDlet registered on the transaction events will block further registrations on these notifications.

8.6.1 Access API & Secure Element Access Control requirements

Below requirements are replacing the generic requirements.

TS26_NFC_REQ_104	The mobile device SHALL support Annex A of JSR 177 specifications.

8.6.2 NFC Event & Access Control requirements

Below requirements are replacing the generic requirements.

TS26_NFC_REQ_105	The mobile device SHALL check that the relationship between the UI application and its corresponding UICC applet exists before allowing the UI application registration to an event within the Push registry.

8.6.3 Protecting APIs with Operator Certificate

Below requirements are replacing the generic requirements.

TS26_NFC_REQ_106	The mobile device SHALL support Annex B of JSR 177 specifications

8.7 SCWS support

VOID.

8.8 Card Application Toolkit Support

No specific requirement, see Generic Device Requirements

8.9 Platform Dependant Properties

No specific requirement, see Generic Device Requirements (TBC)

9 Blackberry 10 OS

BlackBerry provides complete NFC documentation and API as part of Native SDK for BlackBerry 10 at BlackBerry Developer Web Page.

- NFC Documentation: http://developer.blackberry.com/native/documentation/core/com.qnx.doc.nfc/topic/c_stub_nfcdevguide_general_introduction.html
- NFC API: http://developer.blackberry.com/native/reference/core/com.qnx.doc.nfc.lib_ref/topic/manual/nfc_lib_nfc_conceptual_overview.html

The following sections explain how BlackBerry 10 OS supports GSMA certain NFC requirements.

9.1 NFC Device Architecture

The following diagram provides a high-level view of the NFC architecture in a BlackBerry 10 device.

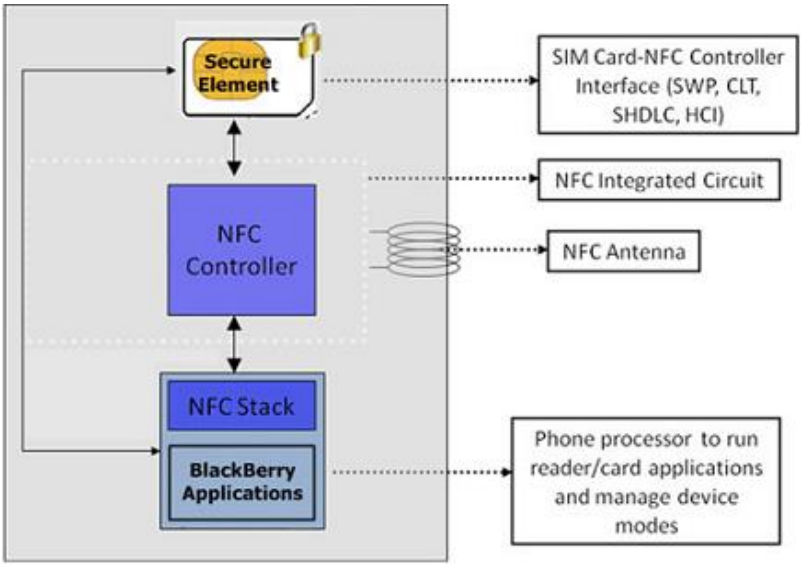

Figure 9: BlackBerry 10 NFC Device Architecture

The key components depicted here are:

- **NFC antenna**: NFC radio communications requires an antenna in the NFC device.
- **NFC controller**: Consists of a Hardware Controller Interface (HCI) and an NFC modem. This component interfaces with the NFC antenna and redirects the radio frequency (RF) communication to the currently selected Secure Element. It's the key piece of hardware that gives your BlackBerry smartphone its NFC capabilities. The NFC modem is sometimes termed the CLF or Contactless Front End. It's responsible for communication via radio with an external NFC reader.
- **Secure Element (SE)**: Consists of hardware and a smart card operating system, and is capable of running small applications known as applets. The SE provides a secure, tamperproof environment for running applets. Applications on the BlackBerry smartphone can interact with such applets using one of the NFC APIs. Applets can also interact with components such as NFC reader terminals (e.g., point-of-sale terminals) using one of the key NFC communication protocols plus a suitable application-layer protocol. The SE resides in the Universal Integrated Circuit Card (UICC), if your UICC is issued with an SE.
- **NFC stack**: Consists of a series of APIs that applications can use.
- **BlackBerry applications**: Apps you build use the NFC APIs to provide the desired functionality for NFC use cases. These applications can be written in a combination of C/C++, QML, and JavaScript.

9.2 Core Required NFC Features

Reference: TS26_NFC_REQ_022, 023 & 024

API used to change NFC settings

```
NFC_API nfc_result_t nfc_set_setting(nfc_settings_t setting, bool enable)

typedef enum {
      NFC_SETTING_ENABLED = 0
      NFC_SETTING_CE_BACKLIGHT_OFF = 1
      NFC_SETTING_CE_POWERED_OFF = 2
      NFC_SETTING_PROMPT_TO_SEND_FILES = 3
```

```
      NFC_SETTING_ENABLED_ASYNC = 4
} nfc_settings_t;
```

See also
http://developer.blackberry.com/native/reference/core/com.qnx.doc.nfc.lib_ref/topic/nfc_set_setting.html

API to retrieve NFC settings

```
NFC_API nfc_result_t nfc_get_setting(nfc_settings_t setting, bool *enabled)
```

See also
http://developer.blackberry.com/native/reference/core/com.qnx.doc.nfc.lib_ref/topic/nfc_get_setting.html

API to determine the state of Card Emulation

```
NFC_API nfc_result_t nfc_se_reader_get_technology_types(uint32_t
hSESession, uint32_t batteryMode, uint32_t *pTechnologyTypes)
```

See also
http://developer.blackberry.com/native/reference/core/com.qnx.doc.nfc.lib_ref/topic/nfc_se_reader_get_technology_types.html

9.3 Secure Element Access & Multiple Secure Elements Management

Reference: TS26_NFC_REQ_047

API to access UICC

```
NFC_API nfc_result_t nfc_se_channel_close_channel(uint32_t hSEchannel)

NFC_API nfc_result_t nfc_se_channel_get_session(uint32_t hSEChannel,
uint32_t *phSESession)


NFC_API nfc_result_t nfc_se_channel_get_transmit_data(uint32_t hSEChannel,
uint8_t *pReceiveAPDUBuffer, size_t *pnLengthOfReceiveAPDUBufferInBytes)


NFC_API nfc_result_t nfc_se_channel_is_basic_channel(uint32_t hSEchannel,
bool *pIsBasicChannel)


NFC_API nfc_result_t nfc_se_channel_is_closed(uint32_t hSEChannel, bool
*pIsClosed)


NFC_API nfc_result_t nfc_se_channel_transmit_apdu(uint32_t hSEChannel,
const uint8_t *pSendAPDUBuffer, size_t nLengthOfSendAPDUBufferInBytes,
size_t *pnLengthOfResponseInBytes)


nfc_se_open_logical_channel_direct(secure_element_id_type_t
secureElementType, const uint8_t *pAID, size_t nLengthOfAIDInBytes,
fcp_type_t fcpResponseType, uint32_t *phSESession, uint32_t *phSEChannel,
int32_t *pResponseLen)


NFC_API nfc_result_t nfc_se_reader_close_sessions(uint32_t hSEReader)
```

```
NFC_API nfc_result_t nfc_se_reader_get_name(uint32_t hSEReader, char
*pName, uint32_t *pLen)

NFC_API nfc_result_t nfc_se_reader_is_secure_element_present(uint32_t
hSEReader, bool *pIsPresent)

NFC_API nfc_result_t nfc_se_reader_open_session(uint32_t hSEReader,
uint32_t *pSESession)

NFC_API nfc_result_t nfc_se_service_get_num_readers(uint32_t
*pNumberOfReaders)

NFC_API nfc_result_t nfc_se_service_get_readers(uint32_t *phReaders,
uint32_t *pNumberOfReaders)

NFC_API nfc_result_t nfc_se_session_close_channels(uint32_t hSESession)

NFC_API nfc_result_t nfc_se_session_get_atr(uint32_t hSESession, uint8_t
*pAtr, size_t nLengthOfAtrInBytes, size_t *pLengthOfReturnedAtrInBytes)

NFC_API nfc_result_t nfc_se_session_get_reader(uint32_t hSESession,
uint32_t *phSEReader)

NFC_API nfc_result_t nfc_se_session_is_session_closed(uint32_t hSESession,
bool *pIsClosed)

NFC_API nfc_result_t nfc_se_session_open_basic_channel(uint32_t hSESession,
fcp_type_t fcpResponseType, uint32_t *phSEChannel, int32_t
*pnLengthOfResponseInBytes)

NFC_API nfc_result_t nfc_se_session_open_logical_channel(uint32_t
hSESession, const uint8_t *pAID, size_t nLengthOfAIDInBytes, fcp_type_t
fcpResponseType, uint32_t *phSEChannel, int32_t *pLengthOfResponseInBytes)
```

See also
http://developer.blackberry.com/native/reference/core/com.qnx.doc.nfc.lib_ref/topic/about_nfc__se__access_8h.html

Reference: TS26_NFC_REQ_051 & 052

API to set UICC as active SE:

```
NFC_API nfc_result_t nfc_se_set_uicc_active()
```

See also
http://developer.blackberry.com/native/reference/core/com.qnx.doc.nfc.lib_ref/topic/nfc_se_set_uicc_active.html

Reference: TS26_NFC_REQ_055

API to dynamically register and un-register by AID: FFS

Reference: TS26_NFC_REQ_119

API to dynamically register and un-register by protocol and technology:

```
NFC_API nfc_result_t nfc_se_reader_set_technology_types(uint32_t
hSESession, uint32_t technologyTypes)
```

See also
http://developer.blackberry.com/native/reference/core/com.qnx.doc.nfc.lib_ref/topic/nfc_se_reader_set_technology_types.html

Reference: TS26_NFC_REQ_057

API to dynamically register and un-register by pattern: FFS

Reference: TS26_NFC_REQ_058

API to support routing by AID: FFS

Reference: TS26_NFC_REQ_059 & 060

API to check if SE is configured in certain battery mode:

```
NFC_API nfc_result_t nfc_se_reader_get_technology_types(uint32_t
hSESession, uint32_t batteryMode, uint32_t *pTechnologyTypes)
```

See also
http://developer.blackberry.com/native/reference/core/com.qnx.doc.nfc.lib_ref/topic/nfc_se_reader_get_technology_types.html

Reference: TS26_NFC_REQ_061 - 067.

API to support static registration and conflict management: FFS

9.4 UI Application triggering requirements

To subscribe to transaction notifications, you need to specify a target in your bar-descriptor.xml file. Later Transaction notifications are delivered using the BB10 OS invocation framework.

API for handling received transaction notification:

```
NFC_API void nfc_se_free_transaction(nfc_se_transaction_t *transaction)

NFC_API nfc_result_t nfc_se_parse_transaction(nfc_se_transaction_t
**transaction, const char *transactionData)
```

```
NFC_API nfc_result_t nfc_se_transaction_get_aid(nfc_se_transaction_t
*transaction, unsigned int index, const uint8_t **aid, size_t *aidLength)


NFC_API const uint8_t*
nfc_se_transaction_get_event_data(nfc_se_transaction_t *transaction)


NFC_API size_t
nfc_se_transaction_get_event_data_length(nfc_se_transaction_t *transaction)


NFC_API unsigned int
nfc_se_transaction_get_number_of_aids(nfc_se_transaction_t *transaction)


NFC_API unsigned int nfc_se_transaction_get_protocol(nfc_se_transaction_t
*transaction)


NFC_API secure_element_id_type_t
nfc_se_transaction_get_se_type(nfc_se_transaction_t *transaction)


NFC_API nfc_result_t nfc_se_transaction_foreground_application()

NFC_API nfc_result_t nfc_se_parse_transactionV2(nfc_se_transaction_t
**transaction, constchar* transactionData, int64_t sourceGid)
```

See also

http://developer.blackberry.com/native/documentation/core/com.qnx.doc.nfc/topic/c_stub_nfcdevguide_transaction_listener.html
http://developer.blackberry.com/native/reference/core/com.qnx.doc.nfc.lib_ref/topic/about_nfc__se__transaction_8h.html

See also
http://developer.blackberry.com/native/reference/core/com.qnx.doc.nfc.lib_ref/topic/about_nfc__se__transaction_8h.html

9.5 Remote Management of NFC Services

No specific requirement, see Generic Device Requirements (TBC)

9.6 Security

No specific requirement, see Generic Device Requirements (TBC)

9.7 SCWS support

VOID

9.8 Card Application Toolkit Support

No specific requirement, see Generic Device Requirements (TBC)

9.9 Platform Dependant Properties

No specific requirement, see Generic Device Requirements (TBC)

10 Windows Phone 8

10.1 NFC Device Architecture

To be done

10.2 Core Required NFC Features

No specific requirement, see Generic Device Requirements (TBC)

10.3 Secure Element Access & Multiple Secure Elements Management

No specific requirement, see Generic Device Requirements (TBC)

10.4 User Interface

No specific requirement, see Generic Device Requirements (TBC)

10.5 Remote Management of NFC Services

No specific requirement, see Generic Device Requirements (TBC)

10.6 Security

No specific requirement, see Generic Device Requirements (TBC)

10.7 SCWS support

VOID

10.8 Card Application Toolkit Support

No specific requirement, see Generic Device Requirements (TBC)

10.9 Platform Dependant Properties

No specific requirement, see Generic Device Requirements (TBC)

Annex A Document Management

A.1 Document History

Version	Date	Brief Description of Change	Approval Authority	Editor / Company
1.0	July 2011	First draft submitted to DAG and PSMC for approval	PSMC and NFC (Rejected at PSMC level)	Sameer Tiku, Vodafone
2.0	November 2011	Second draft incorporating PSMC feedback submitted to DAG and PSMC for approval	PSMC and NFC (V2.0 Approved and published)	Sameer Tiku, Vodafone
3.0	03 October 2012	Submitted to DAG and PSMC for 7 day Committee Email approval	PSMC and NFC	Sameer Tiku, Vodafone
4.0	17 September 2013	Re-Submitted to DAG and PSMC for approval following comments received from PSMC	PSMC and NFC	Sameer Tiku, Vodafone
4.1	6 November 2013	Corrected and updated namespace to <**com.gsma.services.nfc**>. Impacted sections: 4.9, 4.10	Terminal Steering Group	Katrin Jordan, DT / TSG
5.0	12th March 2014	Implemented updates to: Title, Introduction, Abbreviations, Terms of Definitions, References, Doc. structure, Figures, Requirements across the document and API specifications. Added a new numbering scheme and provided editorial updates. Marked requirements still work in progress. Removed Annexes and requirements/references as applicable.	Terminal Steering Group	K.Jordan (DT)
6.0	21 July 2014	Implemented updates to: Introduction, Abbreviations, Definitions of Terms, References, Figures, Core Required NFC Features, Secure Element Access & Multiple Secure Elements Management, UI Application triggering requirements, UICC Remote Management, Security, SCWS support, Card Application Toolkit Support, Platform Dependent Properties, Android and Blackberry OS specific sections. Removed majority of yellow markings where req. could be	Terminal Steering Group	K.Jordan, G.Printemps, DT

		confirmed. Removed Android API details and provided updated API details as separate Javadoc with Readme file. Implemented further formatting and quality updates.		
7.0	23rd March 2015	Updated with TS.26 CR1007 Clarification of requirements on modem and Handling full CLF routing table	Terminal Steering Group	Paul Gosden GSMA

Other Information

Type	Description
Document Owner	GSMA TSG
Editor / Company	Paul Gosden (GSMA)

It is our intention to provide a quality product for your use. If you find any errors or omissions, please contact us with your comments. You may notify us at prd@gsma.com

Your comments or suggestions & questions are always welcome.