GlobalPlatform System

Messaging Specification for Management of Mobile-NFC Services

Version 1.1.2

October 2013

Document Reference: GPS_SPE_002

This page intentionally left blank.

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 Scope

This document describes the extension of the GlobalPlatform System Messaging Core Specification [2] allowing the exchange of messages for performing the delivery and the post-issuance management of mobile-NFC services, i.e. NFC Services deployed in a mobile phone environment.

The scope of this specification is defined by the "GlobalPlatform's Proposition for NFC Mobile: Secure Element Management and Messaging" white paper [9], the UICC Card Configuration document [7] and the GSMA/EPC white paper [25].

This version of the specification covers the following relations:

- Between the Service Provider and the Trusted Service Manager
- Between the Trusted Service Manager and the Mobile Network Operator
- Between the Trusted Service Manager and the Secure Element Provider
- Between the Trusted Service Manager and the Controlling Authority

Other functions and messages between additional actors will be added in future revisions of this specification.

Moreover, only technical functions and messages required for "live" OTA management of mobile-NFC services are described.

Future scope:

The following items are planned to be part of one of the next versions of this document

- Functions and messages targeting groups
- Service Provider – Controlling Authority interface and messages
- TSM – TSM interface and messages
- MNO swap
- Support for the GlobalPlatform Trusted Execution Environment

Not in scope:

All technical functions and messages that may be previously exchanged between the various actors to set up, configure or provision a system are out scope (e.g. the provisioning of a service definition or the upload of an Executable Load File in an application repository).

All non-technical functions and messages linked to business partnership are out of scope (e.g. service eligibility criteria exchange and partnerships management by the TSM).

Note also that the mandatory/optional aspect of the entry points of the interfaces is out of scope of this specification. Compliance packages will be defined in a separate document, through a dedicated compliance program.

1.2 Audience

The target audience for this specification are the system integrators, the system or component developers or architects, or the security experts of any entities involved in the ecosystem of mobile-NFC services; including service providers, mobile network operators, and Trusted Service Manager solution vendors participating in GlobalPlatform smart card implementations, and developing infrastructure components and support systems. Such systems include:

- Data preparation systems
- Secure Element personalization systems
- Mobile-NFC application management systems
- Smart Card Management Systems
- Personalization Collators/Decollators
- OTA communication systems
- Mobile Device management systems
- Mobile Device manufacturers

1.3 Organization

Chapter 1 provides an introduction to the mobile-NFC ecosystem.

Chapter 2 identifies the use cases and the deployment modes that are targeted in this specification.

Chapter 3 provides the list of all functions that are required for managing the relations identified between all the actors of the system.

Chapter 4 provides the mapping of those functions to the GlobalPlatform core messages defined in [2] and then chapter 5 defines the binding of those messages to the Web Services technology.

Chapter 6 defines exception codes.

1.4 Context

The combination of Secure Elements and Near Field Communication (NFC) technology presents significant business opportunities when used in mobile phones for applications such as payment, transport ticketing, loyalty, physical access control, and many other exciting new services. To support this fast evolving business environment, several actors, in addition to Mobile Network Operators (MNO), will become involved in the NFC mobile ecosystem, such as the service providers (e.g. banks, transit authority, etc.) and the Secure Element providers (e.g. micro-SD (µSD) issuers).

By nature of their individual roles, these business actors will need to communicate with each other and exchange messages in a reliable and interoperable way: for a bank (i.e. a service provider) to request the deployment of a payment application (i.e. a mobile-NFC service) to an actor having an Over-The-Air (OTA) capability; for a Mobile Network Operator to notify the various life cycle events of the end-user mobile environment (device lost, etc.) to the service providers, etc.

These actors come from different business environments with heterogeneous systems. The mobile-NFC ecosystem then definitely requires the interoperable interface between those entities, ensuring critical data exchange to support business functionality (a banking NFC application is, for example, provisioned with banking secrets that must be transported in a fully secured way end-to-end, without allowing the SE provider accessing it).

NFC Services are either composed of a running application that interacts with a NFC reader, or by a dedicated memory area that is accessed by the NFC reader. In both cases, it is required that these NFC service items are hosted in a secure environment such as a Secure Element and a Trusted Execution Environment.

In a mobile phone, the component providing such secure and confidential environment is referred to as a Secure Element (SE).

The three main types of Secure Elements which may host mobile NFC services are:

- Universal Integrated Circuit Card (UICC)
- Embedded Secure Element (ESE)
- Secure Memory Card (SMC)

The aim of the present specification is to define standard messages to be exchanged between the mobile-NFC ecosystem actors to jointly deploy and remotely manage mobile-NFC services.

1.5 Ecosystem

1.5.1 Actors

The main actors involved in mobile-NFC service deployment and management are:

- The **Secure Element Provider** (SE Provider): the owner of the SE
- The **Service Provider** (SP): an entity such as a bank, transport authority, retailer, etc., providing NFC services to consumers. The SP needs its services to be deployed within the end-users' mobile equipment.
- The **Mobile Network Operator** (MNO): provides the technical capability to access the mobile environment using an OTA communication channel. The MNO may also be the SE Provider in case the secure element is the UICC. In this case, the MNO may provide a UICC OTA Management System, which is also called the OTA Platform.
- The **Trusted Service Manager** (TSM): a trusted third party, which provides one or more technical roles and possibly business roles to the other actors. A TSM is trusted by the other actors to be in charge of service management and delivery. The TSM acts as an aggregator and may simultaneously support several SPs and MNOs while maintaining confidentiality between the actors.
- The **Controlling Authority** (CA): the entity responsible for confidentially enabling the initial key set in a GlobalPlatform Supplementary Security Domain.

1.5.2 Modes for Card Content Management and for Service Deployment

1.5.2.1 Card Content Management Modes

The respective responsibilities of all the actors are not the same for all the mobile-NFC deployments. For example, the responsibilities of the SE Provider and of the TSM will mainly depend on the business agreement that is established between them around the Secure Element Card Content Management (CCM).

Such one-to-one relation is indeed not limited to "TSM to SE Provider". It can be generalized to any relation between an actor requesting Card Content Management actions, and another actor providing such a Card Content Management capability. The three different Card Content Management relation modes are represented below:

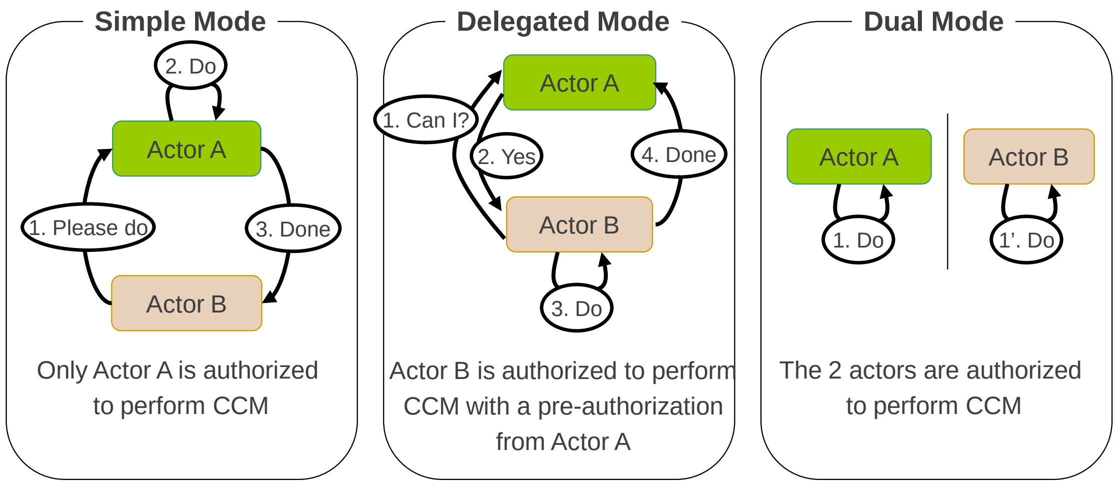

Simple Mode
2. Do
Actor A
1. Please do
3. Done
Actor B
Only Actor A is authorized to perform CCM
Delegated Mode
Actor A
1. Can I?
2. Yes
4. Done
Actor B
3. Do
Actor B is authorized to perform CCM with a pre-authorization from Actor A
Dual Mode
Actor A
Actor B
1. Do
1'. Do
The 2 actors are authorized to perform CCM

Figure 1-1: Card Content Management Modes

Such business relation is usually known as the following:

- **Simple Mode**: Actor B is requesting Actor A to perform CCM actions on its behalf.

  A well known possible usage of this mode is the Issuer-centric model, where a TSM would for example request the SE Provider to perform a Card Content Management operation, which then returns back the execution result to the TSM.

- **Delegated Mode**: Actor B is able to perform CCM actions, but it requires a previous authorization coming from Actor A.

  A well known possible usage of this mode is when Card Content Management is delegated to the TSM, but each operation requires a delegation token from the SE Provider. The execution result of the operation is optionally sent to the SE Provider.

- **Dual Mode**: Actor A and Actor B are fully independent to perform Card Content Management actions on their own respective area.

A well known possible usage of this mode is when Card Content Management is fully delegated to the TSM on a dedicated area of the SE. The SE Provider remains autonomous on its own area of the SE. Dual mode is characterized by the presence of at least two Security Domains with the Authorized Management privilege in the Secure Element.

Note that even if this CCM business relation can be defined between any actors of the system; this is most commonly a SE Provider centric relation (so Actor A is most commonly the SE Provider). In some other very specific use cases, Actor A may be the TSM, for example in case the SE Provider has fully devolved the SE management to the TSM ("TSM to SE Provider" relation being then not qualified as a CCM relation in Dual mode as the SE Provider is not making any CCM action on the SE).

Business agreements may also impact the ability of an actor to communicate OTA to the Secure Element. Depending upon the mode above that is implemented and/or the capabilities of the Secure Element:

- Actor B may have its own OTA capability to send the commands for the Secure Element by itself.
- Actor B may not have any OTA capability. In this case, Actor A is required to provide an OTA channel for enabling Actor B to send the data/scripts that have been previously prepared by Actor B.

This is not only the case for the SE Card Content Management actions, but also for other actions such as OTA personalization of applications, UI application management (such as Device UI applications), or any OTA dialog with the SE or a Device application for a post-personalization use case (mobile-NFC service update).

1.5.2.2 Service Deployment Modes

Service Providers and TSMs will also split the responsibilities between them for the management, and in particular the deployment, of the mobile-NFC service. Such a split is defined according to business agreements between parties.

While this could be done in many ways, this document covers three modes that will be expanded upon later in the document:

- **Service Deployment Mode #1**: the SP wishes to limit its technical involvement in service deployment, and thus delegates the eligibility and global service management (high level scheduling of SE Card Content Management and Device management to be performed to deploy the mobile-NFC service) as well as the service personalization to a TSM.
- **Service Deployment Mode #2**: the SP wishes to limit its technical involvement in service deployment but would like to retain the service personalization responsibility (and thus the confidentiality of its keys), and thus only delegates the eligibility and global service management to a TSM.
- **Service Deployment Mode #3**: the SP is inclined to retain the technical involvement in service deployment and personalization. However, it still delegates some lower level activities to a TSM, such as the global eligibility and the Secure Element Card Content Management operations.

Note that the Service Deployment Modes are different from the Card Content Management Modes because they are not addressing the same scope:

- The Service Deployment Mode is usually related to a "SP to TSM" relation. It concerns a high level delegation of the service management (including Secure Element and Device management), without taking care about who is really performing the low level Card Content Management actions on the Secure Element, or the Device management actions.
- The Card Content Management Modes can be used for any one-to-one relation of the ecosystem that implies single Card Content Management operations on the Secure Element. As mentioned in section 1.5.2.1, the CCM relation is most commonly (but not restricted to) a SE Provider centric relation.

Examples provided in section 2.2.1.2 and the following chapters highlight these differences between those two modes.

1.6 Assumptions

Prior to any technical integration, the actors in the NFC ecosystem have to set up business relationships. The document covers the main relationship between the various actors; TSM, MNO, Service Provider, SE Provider, and Controlling Authority. In order to increase the readability of the document, not all of the possible combinations of actors are described in detail. The main example used for explanation of business processes considers that the MNO is also acting as the Secure Element Provider.

In case the SE provider is not the MNO, the MNOs business role in the NFC ecosystem will be different. Technically, the MNO still will cover the role of mobile access communication provider and most likely there will be special business agreements with the SE Provider (e.g. Mobile Device manufactures or Service Providers). Nevertheless, when the Service Provider is also the SE Provider and a TSM fulfills the technical roles on behalf of the Service Provider, the interaction with the MNO might be limited (e.g. Secure Element Access Control rules deployment, or UI Device application signing and SE Access Control management).

Another assumption for the business processes described in this document is that the TSM might act on behalf of the Service Provider by providing the necessary technical and business functions to enable NFC services.

It must be noted that a TSM as an actor might also provide technical and business functions on behalf of the MNO.

1.6.1 Business Process Management

As part of the business relationships between the actors some crucial information has to be agreed and exchanged in order to configure and implement the relevant business processes.

The following contains a list of information that is considered necessary. The list is for information only and does not claim to be complete.

Commercial Relationship

- Commercial Model:

  The actors have to agree on their commercial model. For instance, how will the MNO charge for its service and the usage of its Secure Element?

  - Possible revenue sharing model (might be even SE profile dependent),
  - Pricing of "rented" SE space,
  - Pricing for providing technical Service Management functions (e.g. responding to eligibility checks, creating a SSD, etc.)
- Billing:
  - What kind of billing information needs to be exchanged (format, content)?
- SE Management mode:
  - Which privileges will be given to the TSM?
  - Which CCM will apply ? (Simple Mode, Delegated Mode or Dual Management Mode)
    - It might depend on the SE profile
- Usage of the CA:
  - Details on the CA and business process
  - Commercial details (who is charged)
- Customer care:
  - Work split between the actors
  - Commercial agreement regarding the services provided
- Use-cases to be supported

Technical Relationship

- Inter-service Communication:
  - Communication details (e.g. SOAP endpoint definitions) for accessing services and notification listeners of other GlobalPlatform infrastructure actors

- SE profile
  - SE Type:
    - Different form factors may require different processing rules. For example, Device Change Notification might have a low impact on the UICC SE, but when using an ESE this change is significant, and when using a SMC, it has yet a different behavior.
  - Communication:
    - Preferred communication channel (e.g. SIM OTA/proxy application)

  - SIM OTA configuration/setup (this information is especially important for setups where the TSM has its own connectivity to the SE. Some of the information is also needed whenever a TSM needs to build a secure channel to a SE's SD.):
    - Supported protocols (SMS, CAT-TP, Secure packet over TCP, RAM over HTTP), versions, configuration options, security options, buffer sizes, etc.
      - Note: It can be partly covered by eligibility check.
    - Initial counter values
  - SD setup:
    - Initial quota, quota management policy (can be partly covered by eligibility check)
    - SCP details
    - DAP verification scheme
    - Confidential loading setup

- Applications Management:

  The specification defines different modes to control the management of the card content. As different SE Providers working with one TSM might have selected different modes, the TSM must be capable of adapting to the scheme based on partner configuration.

  Some of the variables are:

  - Who owns the applications repository (does the TSM store the application's byte code, or does it refer to a byte code stored in the MNO application repository)?
  - How the applications are registered to a remote repository?
  - Which process for DAP signature management?

- UI Applications:
  - Who manages the download of the UI applications?
  - How is the mapping of different applications to handset types done?
  - Definition of Secure Element Access Control or ACF management process: What is the MNO policy to serve Secure Element Access Control rules or ACF change requests?

1.6.2 NFC Service Use-Cases

With respect to the NFC service use cases that are covered by this specification, the following assumptions are made.

In the mobile-NFC ecosystem, it can be considered that NFC services are composed of:

- One to several applications running in a Secure Element. The main role of such application(s) is to implement the NFC service business logic by being accessible by the service reader through the NFC field (payment application, ticketing application, etc.).

  As an assumption, it is considered that all the applications of a mobile-NFC service shall be deployed on the same Secure Element. Deployment of parts of a mobile-NFC service on several different Secure Elements is not considered as a possible use case (e.g.. a first application of the service in the UICC, and another application of the service in an Embedded Secure Element).

- And optionally, one to several User Interface applications, either running in the device that hosts the SE or in the SE itself if it has display capabilities (such as the UICC for example, through the Toolkit Framework or through the Smart Card Web Server functionalities).

  The User Interface of the mobile-NFC service may be split in several parts/applications that are hosted by different components of the mobile equipment, so that for example the mobile-NFC service UI can be partly deployed on the UICC and partly on the device itself.

  More generally, all device applications that take part of the service –including the ones that do not really provide a UI feature–, shall be taken into account in the global service management.

Whatever the used technology for the UI application (device application or SE application), these UI applications are fully considered as part of the NFC services to be delivered and managed: they shall be considered as the "visible part" of the NFC SE applications.

As a consequence, the mobile-NFC services management and the relevant interfaces between actors of the mobile-NFC ecosystem shall take care of both the SE applications and the "visible part" of these SE applications, i.e. the UI applications.

Note that the Amendment C of the GP 2.2 Card Specifications [5] clearly defines the necessary components within the Secure Element to globally manage a set of applications belonging to the same NFC service, and clearly defines the link between these applications and the UI application/Wallet applications, through the CRS notion. It also defines how to manage multiple NFC services within a single Secure Element.

Note also that end-users may subscribe to several mobile-NFC services. Due to multi-application support, **the hosting of several mobile-NFC services in the same SE is in the scope of this specification**.

However, despite the fact that an end-user may hold several Secure Elements, and that devices are technically able to host several NFC Secure Elements (e.g. a SWP UICC and an ESE and a SMC), **this version of document does not support management of a single service deployed over multiple Secure Elements**.

1.7 References, Abbreviations and Definitions

1.7.1 Normative References

Standard / Specification	Description	Ref
GlobalPlatform Card Specification v 2.2.1	GlobalPlatform – Card Specification v2.2.1	[1]
GlobalPlatform Messaging Specification	GlobalPlatform – Messaging Specification	[2]
GlobalPlatform Card Specification v2.2 Amendment A – Confidential Card Content Management	GlobalPlatform – Confidential Card Content Management	[3]
GlobalPlatform Card Specification v2.2 Amendment B – RAM over HTTP	GlobalPlatform – Remote Application Management over HTTP	[4]
GlobalPlatform Card Specification v2.2 Amendment C – Contactless Services	GlobalPlatform – Contactless Services	[5]
GlobalPlatform Card Specification v2.2 Amendment E – Security Upgrade	GlobalPlatform – Security Upgrade for Card Content Management	[6]
GlobalPlatform Card UICC Configuration 1.0.1	Configuration requirements for implementing GlobalPlatform Specifications on the UICC platform	[7]
GlobalPlatform Secure Element Configuration 1.0	Configuration requirements for implementing GlobalPlatform Specifications for Secure Elements	[8]
GlobalPlatform White Paper on NFC SE management	GlobalPlatform White Paper: GP's Proposition for NFC Mobile: Secure Element Management and Messaging	[9]
GlobalPlatform Secure Element Access Control v1.0	GlobalPlatform – Secure Element Access Control	[10]
ETSI TS 101 220 Rel. 9	ETSI – ETSI numbering system for telecommunication application providers	[11]
ETSI TS 102 127 Rel. 6	ETSI – Transport protocol for CAT applications	[12]
ETSI TS 102 225 Rel. 9	ETSI – Secured packet structure for UICC based applications	[13]
ETSI TS 102 226 Rel. 9	ETSI – Remote APDU structure for UICC based applications	[14]
ETSI TS 102 613 Rel. 9	ETSI – UICC - Contactless Front-end (CLF) Interface	[15]
ETSI TS 102 622 Rel. 9	ETSI – UICC - Contactless Front-end (CLF) Interface; Host Controller Interface (HCI)	[16]
OMA TS SCWS v1.2	OMA – Smart Card Web Server	[17]
RFC 2119	Keys words for use in RFC to indicate requirement levels	[18]
RFC 2986	PKCS #10: Certification Request Syntax Specification	[19]

Standard / Specification	Description	Ref
RFC 4646	Tags for Identifying Languages	[20]
W3C.REC-xmlschema-2	Malhotra, A. and P. Biron, "XML Schema Part 2: Datatypes Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-2-20041028, October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028>	[21]
Web Services Profile for GlobalPlatform Messaging	Describes GlobalPlatform selection of Web Services standards for the transmission and security of GlobalPlatform messages in a SOA environment	[22]
AFSCM TECH - LIVBL - Interface Specification – v2.0.1	AFSCM interface specification v2.0.1, issued September 2011	[23]
Payez Mobile Specifications v2.1	Payez Mobile Specification v3.0, issued April 2011	[24]
GSMA/EPC EPC 220-08	Mobile Contactless Payments Service Management Roles Requirements and Specifications v2.0	[25]
GSMA/EPC MNO-TSM	NFC MNO-SP interface Business Process implementation guidelines using GP protocols v1.0	[26]
ITU E.164	International Public Telecommunication Numbering Plan - E.164	[27]
ITU E.212	International Public Telecommunication Numbering Plan - E.212	[28]

Table 1-1: Normative references

1.7.2 Terminology and Definitions

Term	Definition
Application Provider	The Application Provider (AP) terminology defines a role of the ecosystem. The Application Provider role holds the responsibility for the global (SE and Device) mobile-NFC service management towards end-users as it has a direct business relationship with them. It defines the various elements of the mobile-NFC service (both SE and Device elements) and procures the necessary components to load into the Mobile Device (i.e. the SE and handset applications code and data, the SE applications keys and/or certificates, and the data belonging to a specific cardholder). However, the Application Provider does not perform the true service management operations: it rather delegates these operational tasks to the Security Domain Manager, but may retain responsibility for personalization of the application, e.g. the generation of personalization data.
Contactless Application	Applications that are configured to be usable on a contactless interface (e.g. ISO 14443 based).

Term	Definition
Controlling Authority	The Controlling Authority (CA) terminology defines a role of the ecosystem. The Controlling Authority role is a third party authority that enforces the security policy in a multi-actor environment accessing a Secure Element. It may be used in particular for secure Security Domain creation in a SE. The actor playing the Controlling Authority role can also be named Controlling Authority.
Device	In the context of this specification, the Device term is used to represent any electronic equipment into which a NFC Secure Element can be plugged, and that provides a capability for a server to reach the SE through an Over The Air (OTA) or Over The Internet (OTI) link. A mobile phone is a good example of such device.
Device and Mobile Subscription Registrar	The Device and Mobile Subscription Registrar (DMSR) terminology defines a role of the ecosystem. The Device and Mobile Subscription Registrar role holds the following responsibilities: • Perform the eligibility checking on the Mobile Subscription: it is able to provide a business (e.g. "NFC option" activated or not) and a technical (e.g. data bearer available or not) authorization regarding the Mobile Subscription to be used to reach the Secure Element and the Device. • Act as a Device registrar to track the association between the Secure Element, the Device, and the Mobile Subscription. • Provide the capabilities of the Device.
End-user	An end-user is a humain being that, as an actor of the ecosystem, is willing to get some mobile-NFC services. The end-user is (or will become) a customer of a particular Service Provider (the one that is proposing the mobile-NFC service). The end-user also has (or will have) a mobile subscription contracted to a Mobile Network Operator. Finally, he owns (will own) a mobile equipment that is NFC compatible, and that is able to host the mobile-NFC service.
Mobile Network Operator	The Mobile Network Operators (MNO) terminology defines an actor of the ecosystem. The Mobile Network Operator actor provides the technical capability to access to the mobile environment, using an OTA communication channel. The MNO may also be the SE Provider in case the secure element is the UICC, and thus may provide UICC OTA management system, also called OTA platform.

Term	Definition
Mobile-NFC Service	A mobile-NFC service is a NFC service that is deployed in a mobile equipment. A mobile-NFC service is composed of: • One to several applications running in a Secure Element. The main role of such application(s) is to implement the NFC service business logic by being accessible by the service reader through the NFC field (e.g. payment application, ticketing application). • And optionally, one or more User Interface applications, either running in the device that hosts the SE or in the SE itself if it has display capabilities (such as the UICC for example, through the Toolkit Framework or through the Smart Card Web Server functionalities). More generally, all device applications that take part of the service, including the ones that do not really provide a UI feature, shall be taken into account by the global service management.
Secure Element	A Secure Element (SE) is a component in a device providing the security and confidentiality required to support various business models. A SE can exist in any form factor such as UICC, Embedded SE, Secure Memory Card, etc
Secure Element Issuer	The Secure Element Issuer (SEI) terminology defines a role of the ecosystem. The Secure Element Issuer role holds the ultimate responsibility for the GlobalPlatform card. The SEI has the responsibility to develop the card product profile, to choose the platform and application technologies to design card layout. The Secure Element Issuer usually holds a particular Security Domain in the SE: the Issuer Security Domain (ISD). Card Content Management (CCM) operations that can be performed on this ISD are associated to the Security Domain Manager role that a SE provider may also play.
Secure Element Provider	The Secure Element Provider (SE Provider) terminology defines an actor of the ecosystem. The Secure Element Provider actor is the owner of a Secure Element.

Term	Definition
Security Domain Manager	The Security Domain Manager (SDM) terminology defines a role of the ecosystem. The Security Domain Manager role holds three possible facets, each of them implying some responsibilities: • *Eligibility and Service Management:* if implementing this facet, the SDM is responsible for the high level service management operations on behalf of the Application Provider. It handles the high level scheduling of Card Content Management and Device management to be performed on the SE and on the Device, in order to deploy and manage the mobile-NFC service. However, this facet does not include the specific personalization script building. • *SE Content Management:* if implementing this facet, the SDM is responsible for the execution of single actions related to the SE (such as loading an ELF, installing a SE application, generating a Delegated Management Token, performing SE content audit, etc.), that takes part of a mobile-NFC service management The SDM is then responsible for a (set of) Security Domain(s) in a SE. Depending on the privileges associated to its Security Domains, the SDM may have the capability to directly load, install, extradite or personalize applications on behalf of an Application Provider. In case it does not have the necessary Card Content Management privileges or Secure Channel keys, or it does not have an OTA capability, it may request the help of another SDM to perform Card Content Management or OTA dialog. On the other side, a SDM may perform CCM operations on behalf of another SDM which does not have those necessary privileges, or grant CCM Delegated Management tokens for other SDMs, or may serve as OTA provider for other SDMs or even AP for application personalization. • *Personalization Script Management:* if implementing this facet, the SDM is responsible for the generation and securization of the mobile-NFC service personalization script (SE and/or Device applications personalization scripts) The SDM is then responsible of the Security Domain that holds the keys necessary to secure the personalization script, but may rely on another SDM as OTA provider for the script OTA sending.
Service Provider	The Service Provider (SP) terminology defines an actor of the ecosystem. The Service Provider actor is an entity such as a bank, a transport company, a retailer, etc., that owns NFC services provided to consumers. These services need to be deployed in a SE within the end-users' mobile equipments.

Term	Definition
Trusted Service Manager	The Trusted Service Manager (TSM) terminology defines an actor of the ecosystem. The Trusted Service Manager actor is a trusted third party which provides one or more technical roles and possibly business roles to the other actors. The TSM is responsible for the service management and delivery and it provides a level of trust and confidentiality between these actors. The TSM is able to act as aggregator between Service Providers and MNOs.

Table 1-2: Terminology and definitions

1.7.3 Abbreviations and Notations

Abbreviation	Meaning
ACF	Access Control Files
AFSCM	Association Française du Sans Contact Mobile
AID	Application IDentifier
AP	Application Provider
API	Application Programming Interface
APDU	Application Protocol Data Unit
APSD	Application Provider Security Domain
ARA-C	Access Rule Application Client
ARA-M	Access Rule Application Master
ARF	Access Rule File
CA	Controlling Authority
CASD	Controlling Authority Security Domain
CAT-TP	Card Application Toolkit-Transport Protocol
CCM	Card Content Management
CIN	Card Identification Number
CL	Contactless
CRS	Contactless Registry Service
DAP	Data Authentication Pattern
DMSR	Device and Mobile Subscription Registrar
ETSI	European Telecommunications Standards Institute
ESE	Embedded Secure Element
HTTP	HyperText Transfer Protocol
ICCID	Integrated Circuit Card IDentifier
IIN	Issuer Identification Number
ISD	Issuer Security Domain
ISDN	Integrated Services Digital Network
KCV	Key Check Value

Abbreviation	Meaning
KPD	Key Provisioning Data
KPOD	Key Provisioning Output Data
MEP	Message Exchange Pattern
MIDP	Mobile Information Device Profile
MNO	Mobile Network Operator
MSISDN	Mobile Station ISDN (Integrated Services Digital Network) Number Mobile Subscriber Integrated Services Digital Network Number
NFC	Near Field Communication
OASIS	Organization for the Advancement of Structured Information Standards
OID	Object IDentifier
OMA	Open Mobile Alliance
OTA	Over The Air
OTI	Over The Internet
RAM	Remote Application Management
RGK	Randomly Generated Key
SCP	Secure Channel Protocol
SCWS	Smart Card Web Server
SDM	Security Domain Manager
SE	Secure Element
SEI	Secure Element Issuer
SIR	Service Instance Reference
SMC	Secure Memory Card
SMS	Short Message Service
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SSD	Supplementary Security Domain
SP	Service Provider
SWP	Single Wire Protocol
TAR	Toolkit Application Reference
TLV	Tag, Length, Value
TSM	Trusted Service Manager
UI	User Interface
UICC	Universal Integrated Circuit Card
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium

Table 1-3: Abbreviations and notations

1.8 Conventions

Throughout this document, normative requirements are highlighted by use of capitalized key words as described below.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [18]:

- MUST - This word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification.
- MUST NOT - This phrase, or the phrase "SHALL NOT", mean that the definition is an absolute prohibition of the specification.
- SHOULD - This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
- SHOULD NOT - This phrase, or the phrase "NOT RECOMMENDED" mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
- MAY - This word, or the adjective "OPTIONAL", mean that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option MUST be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein an implementation which does include a particular option MUST be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides.)

Tables may include a "MOC" column meaning "Mandatory/Optional/Conditional". This column specifies the obligation of presence of the data in the function or in the message. The following definitions apply to these terms:

- Mandatory (M): Means that an entry must be supplied
- Optional (O): Means that an entry can be supplied, but is not required to be supplied.
- Conditional (C): Means that the usage of an entry is dependent upon a particular condition.

1.9 Revision History

The following versions of the specification are considered:

v1.0 Initial Release - This release of document only describes functions and messages between the TSM and the MNO, between the TSM and the SE Provider, and between the TSM and the CA. Issued on February 2011.

v1.0.1 Errata and Precisions list v1.0 on the version 1.0 of the specification. Issued on September 2011.

- **v1.0.2** Errata and Precisions list v1.1 on the version 1.0 of the specification. Issued on May 2012.

v1.1 Adds the functions and messages related to the SP to TSM interface. Issued on February 2013. Also integrates the "Errata and Precisions list" document, version 1.1.

- **v1.1.1** Errata and Precisions list v0.1 on the version 1.1 of the specification. Issued on April 2012.
- **v1.1.2** Additional Errata and Precisions on the version 1.1 of the specification. Issued on October 2013.

Note that a full specification document has not been issued for versions 1.0.1, 1.0.2 and 1.1.1: an Errata and Precisions document, listing the updated text of the specification, has been issued instead.

A full specification document has been issued for versions 1.0, 1.1, and 1.1.2.

1.9.1 Updates from Version 1.0 to Version 1.1

The following changes have been made between version 1.0 and version 1.1.

[1.0.1] is mentioned when the change has been introduced through the "Errata and Precisions list – Messaging Specification for Management of Mobile-NFC Services v1.0" document in version 1.0.

[1.0.2] is mentioned when the change has been introduced through the "Errata and Precisions list – Messaging Specification for Management of Mobile-NFC Services v1.0" document in version 1.1.

- Support of the new SP to TSM interface:
  - *(Section 1)* Definition of Service Deployment modes
    - (Section 2) Detailed examples of interactions in the various modes
  - *(Section 1)* Definition of the SDM facets (Eligibility and Service Management facet, SE Content Management facet, and Personalization Script Management facet)
  - *(Sections 2, 3, 4 & 5)* Definition of the new functions for global eligibility and service management:
    - Definition of the service life cycle
    - Definition of the Global Eligibility function group (`CheckGlobalEligibility` function)
    - Definition of the Global Service Management function group (`LookupServiceInstanceReference`, `DeclareServiceInstanceReference`, `GetServiceInstanceReferenceDescriptor`, `GetServiceState`, `DeployService`, `UpgradeService`, `ExchangeServiceData`, `SuspendOrResumeService`, and `TerminateService` functions)
  - *(Sections 2, 3, 4 & 5)* Definition of the new notifications for service environment change (`HandleServiceEnvironmentChangeNotification` and `HandleActionDoneOnServiceNotification` functions)
- Update of the existing TSM to MNO API:

- *(Section 1)* Clarification on the Card Content Management modes: not restricted to a TSM to SE Provider relation
- *(Section 2)* Detailed examples of possible roles to actors allocation: depending on the SDM facets, depending on the Service Deployment modes
- *(Section 3)* Clarification on validity period start [1.0.2]
- *(Section 3)* Correction of the ICCID type format [1.0.1]
- *(Section 3 & 4)* Clarification of the Service Qualifier type [1.0.2]
- *(Section 3)* Clarification of the usage of the "start service state change" and the "end service state change" notifications (`HandleStartServiceStateChangeNotification` and `HandleEndServiceStateChangeNotification` functions)
    - Additional Operations and Reasons [1.0.2]
    - Extension of the usage fo SP to TSM interface
- *(Section 3)* Clarification of the usage of an alternate identifier computed with the Service identifier (`GetMobileSubscriptionAlternateIdentifier`, `GetSEMobileSubscriptionIdentifier`, `HandleSEDeviceChangedNotification`, and `HandleSEMobileSubscriptionChangedNotification` functions) [1.0.2]
- *(Sections 3, 4 & 5)* Definition of a function to retrieve the SE Identifiers that can be accessed by a given Mobile Subscription Identifier (`GetMobileSubscriptionSEIdentifiers` function)
- *(Sections 3, 4 & 5)* Definition of a notification to notify events related to the Device that is hosting a Secure Element (`HandleSEDeviceStatusChangeNotification` function)
- *(Section 3)* Additional reasons for Mobile Subscription status change (`HandleMobileSubscriptionStatusChangeNotification` function)
- *(Section 3)* Additional reasons for Secure Element status change (`HandleSEStatusChangeNotification` function)
- *(Sections 3, 4 & 5)* Definition of a new function similar to the `GetCAInformation` function, but with an asynchronous MEP (`AuditCAInformation` Function). Keep `GetCAInformation` for backward compatibility [1.0.2]
- *(Section 3)* Clarification on the of response APDU when executing SE Commands (`SECommandsGenerationAndRemoteExecution` function) [1.0.1]
- *(Sections 3, 4 & 5)* Clarification on the creation of the first keyset in a SSD (`CreateFirstSSDKeysetCommand` SE Command):
    - Clarification on the possible implementations of scenario #2B [1.0.2]
    - Additional support of scenario #3 for first keyset creation in a SSD (`CreateFirstSSDKeysetCommand` SE Command)
    - Clarification on the creation of a first keyset for each type of key
    - Clarification on the ownership of the created keys
    - Clarification on the behavior in case of pre-created SSD [1.0.1]
    - Allowance to create a keyset without giving the SSD AID

        - Additional support of a Basic Diversified Create mode for first keyset creation in a SSD (`CreateFirstSSDKeysetCommand` SE Command)
        - Definition of the KeySet Description profiles for specifying the type of keys to be created (secure channel, key type, key length) [1.0.2]
            - Creation of a new SCP81 KeySet profile
        - Mandate usage of Key Version Number [1.0.1]
        - Clarification of the usage of RGK for the related scenarios [1.0.1 & v1.0.2]
        - Additional support of KCV [1.0.2]
        - Clarification of the transport key usage
        - Clarification of the key generation modes and encrypted key values (for scenario #1 using PK, scenario #2A, and scenario #2B) [1.0.2]
    - *(Sections 3, 4 & 5)* Retrieval of the parent SD AID when instantiating an application (`InstantiateApplicationCommand` SE Command)
    - *(Sections 3, 4 & 5)* Clarification of the deletion modes when deleting related objects (`DeleteCommand` SE Command)
    - *(Sections 3, 4 & 5)* Clarification on the scripts sending (`BeginConversation` and `SendScript` functions):
        - Possibility to target several applications in a script sending conversation
        - Possibility to have not-limited APDU script size and not-limited number of APDU [1.0.1]
        - Clarification in case of non respect of `BeginConversation` returned values [1.0.1]
        - Clarification on the computation of the response script size [1.0.1]
        - Clarification on the format of the response script [1.0.1]
    - *(Sections 3)* Clarification on the error case when requesting information about components present on a Secure Element (`GetApplicationOrELFStatus` function)
    - *(Sections 3, 4 & 5)* Definition of a new function to get the remaining available memory space in a SE (`GetSDFreeMemory` function)
    - *(Sections 3)* Clarification on the responsibilities when loading or deleting device applications (`LoadDeviceApplication` and `DeleteDeviceApplication` functions)
    - *(Sections 3)* Mention GlobalPlatform SE Access Control as an example of security binding between a SE and a Device application (`BindDeviceApplicationToSEApplication` and `UnbindDeviceApplicationToSEApplication` functions)
    - *(Sections 2, 3, 4 & 5)* Definition of a new notifications for Device applications life cycle (`HandleActionDoneOnDeviceApplicationNotification` function)
    - *(Section 4)* Clarification of the mapping to the `CardAuditTrail` message (`SECommandsGenerationAndRemoteExecution` function) [1.0.2]
    - *(Section 4)* Clarification of the mapping of the KPD and KPOD for key creation (`CreateFirstSSDKeysetCommand` SE Command) [1.0.1]
- General:

- *(Section 2)* Removal of Use Case #16: Temporary Suspension of Mobile-NFC Services: already covered by Use Case #3
- *(Section 6)* Update of the GPHeader types according to new functions and to errata
- *(Section 5)* Clarification on the security for the SOAP binding [1.0.1]
- *(Section 6)* Update of the subject and reason codes according to new functions and to errata
- *(Section 7)* Update of the sequence diagrams according to new functions

1.9.2 Updates from Version 1.1 to Version 1.1.2

The following changes have been made between version 1.1 and version 1.1.2.

[1.1.1] is mentioned when the change has been introduced through the "Errata and Precisions list – Messaging Specification for Management of Mobile-NFC Services v1.1" document in version 0.1.

[1.1.2] is mentioned when the change has been introduced in v1.1.2 of the specification (as additional Errata and Precisions on Messaging Specification for Management of Mobile-NFC Services v1.1).

- Updates on functions:
  - *(Section 3)* Suggest a MNO Name format [1.1.2]
  - *(Section 3)* Clarification difference between `DeployService`, `UpgradeService` and `ExchangeServiceData` [1.1.2]
  - *(Section 3)* SIR is not invalidated when SE is renewed [1.1.2]
  - *(Section 3)* Clarification of Service Instance life cycle when upgrading a Service [1.1.2]
  - *(Section 3)* Clarification of Service Instance life cycle when terminating a Service [1.1.2]
  - *(Section 3)* Clarification of the `LookupServiceInstanceReference` and `DeclareServiceInstanceReference` function in case several Secure Elements can be accessed through a Mobile Subscription [1.1.2]
  - *(Section 3)* Clarification on the duplicate declaration of SIR [1.1.2]
  - *(Section 3)* Clarification of Service re-deployment in Start/End Service State Change notification [1.1.2]
  - *(Section 3)* Clarification of `ResponseAPDU` field of `SECommandGenerationAndRemoteExecution` function [1.1.2]
  - *(Section 3)* Clarification of the "Executed-With Warning" Command Execution Status code usage for `SECommandGenerationAndRemoteExecution` and `VerifyDMReceipt` [1.1.2]
- General:
  - *(WSDL files)* Correction of the `Transport` attribute of the `/soap12:binding` element [1.1.1]

- *(Section 2)* Clarification of the scope of use cases #1 (re-deployment is part of deployment use case), #14 (Mobile Subscription status change instead of simply Mobile Subscription termination) and #15 (MNO swap use case is not addressed in this version of the specification) [1.1.2]
- *(Section 4)* Clarification on Integer value range [1.1.2]
- *(Annex B)* New annex: creation of track changes tables [1.1.2]
- *(Annex F)* New annex: how to extend the GlobalPlatform System Messaging schema [1.1.2]
- *(Status Code Matrix)* Status codes removed for notification, as they are using the One-Way MEP [1.1.2]

2 Interactions (informative)

2.1 Use Cases

Use cases around mobile-NFC service management can be separated into two groups:

- **Mobile-NFC Service Life Cycle Management Use Cases**: These use cases are technical use cases that cover the pure OTA management of the mobile-NFC services: deployment, suspension, resumption, update, upgrade and deletion.

  These use cases are usually triggered by the Service Provider or by the SE Provider, following a dedicated enrollment process, a call to a customer care center, or a specific mass or single action decided by the SP or the SE Provider.

- **End-User Life Cycle Management Use Cases:** during the service life, the end-user may encounter events or situations that have impacts on the mobile-NFC service status: Secure Element changed, Device changed, mobile phone number changed, Device/SE lost or stolen, and maybe recovered later, etc.

  Some of these end-user use cases may trigger some of the technical use cases mentioned above, in order to ensure mobile-NFC service continuity.

The following sections give highlights of the technical (see section 2.1.1) and the end-user life cycle use cases (see section 2.1.2). Note that this list of use cases and the behavior detailed in each use case are not normative, i.e. they do not provide the exhaustive and only possible behavior of mobile-NFC service management. The goal of this section is rather to provide to the reader the essential use cases of the mobile-NFC ecosystem, for a better understanding and readability of the functions and their related messages defined in this document.

The following figure presents the overall set of use cases, mapped to the service and to the end-user life cycles:

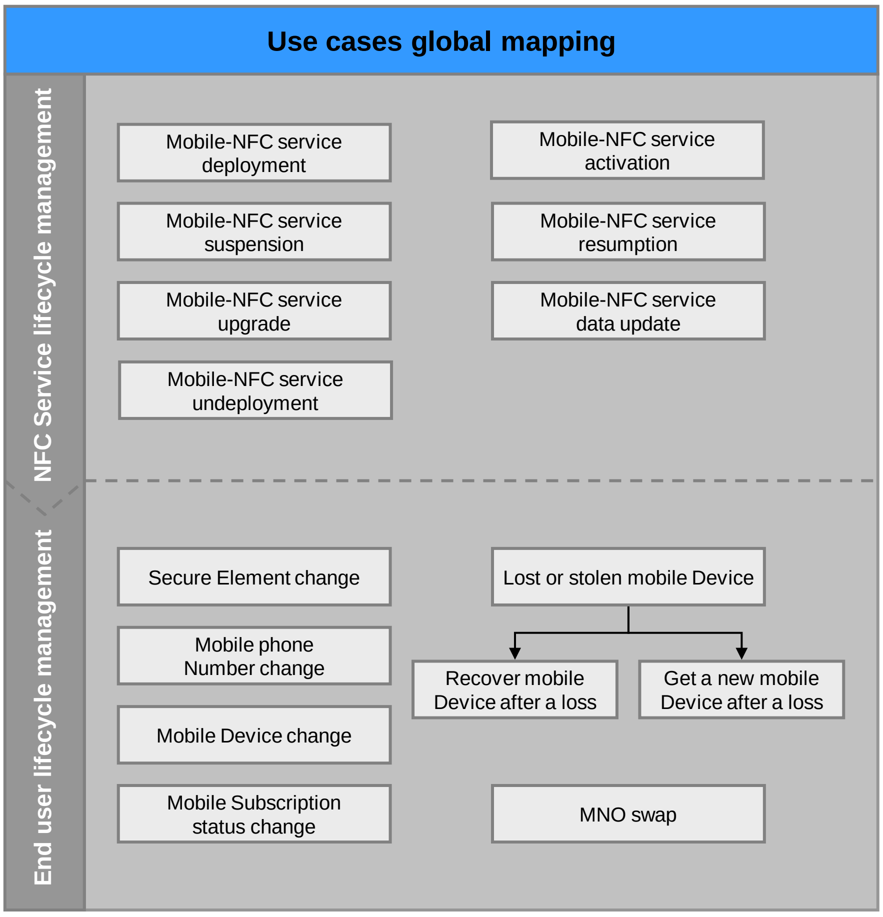

Use cases global mapping
NFC Service lifecycle management
Mobile-NFC service deployment
Mobile-NFC service activation
Mobile-NFC service suspension
Mobile-NFC service resumption
Mobile-NFC service upgrade
Mobile-NFC service data update
Mobile-NFC service undeployment
End user lifecycle management
Secure Element change
Lost or stolen mobile Device
Mobile phone Number change
Recover mobile Device after a loss
Get a new mobile Device after a loss
Mobile Device change
Mobile Subscription status change
MNO swap

Figure 2-1: Global use cases

General pre-conditions applicable to all use cases:

- The end-user is a MNO's subscriber (the means that have been used by the end-user to subscribe to the MNO network or to any particular "NFC" option to the MNO is out of scope of the use case)
- The end-user is a Service Provider's subscriber (the means that have been used by the end-user to subscribe to the SP service or to any particular "NFC" option of the SP is out of scope)

2.1.1 Use Cases Related to Mobile-NFC Service Life Cycle Management

2.1.1.1 Use Case #1: Mobile-NFC Service Deployment

Pre-conditions:

- The Secure Element and the Device may already host applications that are parts of the to-be-deployed NFC service. Such applications may have been pre-loaded in factory or during a previous NFC service deployment (e.g. when components are shared between several services)

Description:

The mobile-NFC service deployment can happen several times during the life cycle of the service:

- For initial deployment of the service on the mobile environment,
- For service renewal or full re-personalization.
- Following mobile environment change (e.g. SE renewed, Device changed) – refer to Use Case #8: Secure Element Change and Use Case #10: Mobile Device Change.

<u>As a first example: Initial service deployment</u>

The end-user subscribes to a mobile NFC service provided by the Service Provider. A service is composed of a set of applications to be deployed in the end-user Secure Element, and optionally of a set of UI applications to be deployed in the Device or in the Secure Element.

The SP delegates the global OTA deployment of the service to its TSM.

The TSM first verifies the end-user mobile environment compliancy (Device and SE) with NFC requirements (e.g. SE is NFC capable), OTA requirements (e.g. SE is reachable OTA), and specific mobile-NFC service deployment requirements (e.g. enough free memory in the SE, SE GlobalPlatform level compliancy). These checks are performed by the TSM itself, or through collaboration with the SE Provider and the MNO.

Then, depending on the current SE and Device content, that may be audited using the SE Provider, the TSM determines the set of single OTA operations to be performed for delivering the NFC service. Each single OTA operation is then performed by the TSM itself, or by the SE Provider or the MNO.

An example of the different operations that can be required for the deployment of a service is given below:

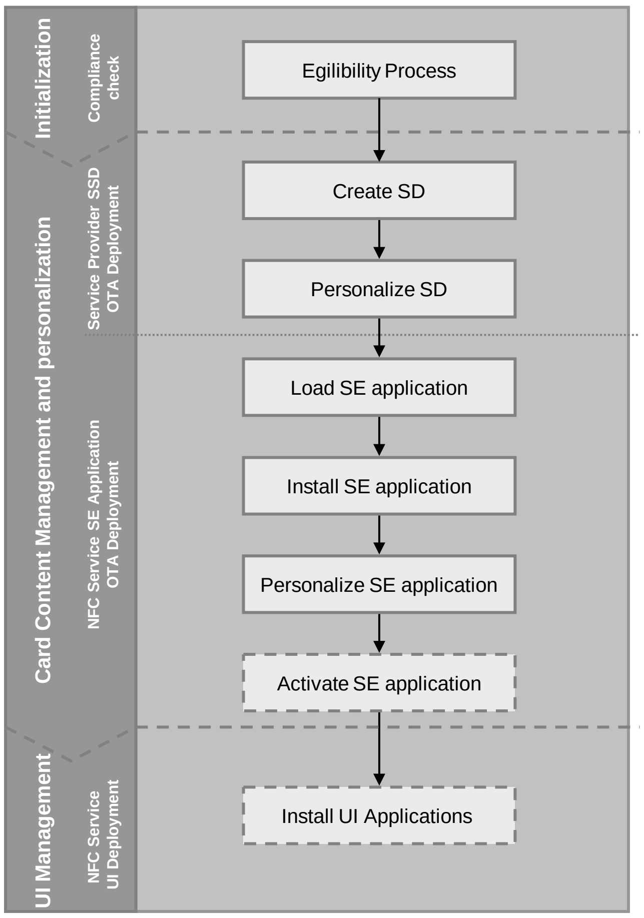


Figure 2-2: Operations of mobile-NFC Service deployment

Note however that this series of steps is purely informative: depending on the mobile environment and on the mobile-NFC service, actions may be executed in a different order, or even other actions may be required (including the locking/unlocking or the deletion of components).

Depending on the state of the mobile-NFC Service at Secure Element, some of these operations are not required.

With regards to Service Deployment Mode #2, note that the Service Provider may retain responsibility for personalization of the application, e.g. the generation and encryption of personalization data. If this is the case then the SP sends the personalization script to a TSM or MNO, which will manage the dialog with the Secure Element.

With regards to Service Deployment Mode #3, note that the Service Provider may also retain responsibility of the global service deployment (i.e. sequencing the set of single operations to be performed for delivering the NFC service), only delegating to the TSM these single operations on the Secure Element or on the Device.

As a second example: Service renewal/re-personalization

Once deployed, in order to comply with security policies, a mobile-NFC service may need to be renewed or fully re-personalized after a pre-defined duration.

The SP delegates the renewal and re-personalization of the service to its TSM.

The TSM first verifies that the end-user mobile environment (Device and SE) is still compliant with the service requirements.

Then the TSM determines the set of single OTA operations to be performed for renewing and re-personalizing the NFC service (for example, first deletion of the existing application instance, then re-instantiation and finaly re-personalization). Each single OTA operation is then performed by the TSM itself, or by the SE Provider or the MNO.

With regards to Service Deployment Mode #2, note that the Service Provider may retain responsibility for personalization of the application, e.g. the generation and encryption of personalization data. If this is the case then the SP sends the personalization script to a TSM or MNO, which will manage the dialog with the Secure Element.

With regards to Service Deployment Mode #3, note that the Service Provider may also retain responsibility of the global service deployment (i.e. sequencing the set of single operations to be performed for delivering the NFC service), only delegating to the TSM these single operations on the Secure Element or on the Device.

Post-conditions:

- The NFC service is ready to be activated, or ready to be used if no service activation is required.

2.1.1.2 Use Case #2: Mobile-NFC Service Activation

Pre-conditions:

- The end-user has an already deployed NFC service. This service may have been previously deployed OTA, or provisioned at manufacturing time.

Description:

Following its deployment, the NFC service may require an explicit service activation operation. This service activation phase is optional depending either on the service or on the Service Provider deployment policy.

This service activation is either:

- Automatically performed by the TSM, after the NFC service deployment
- Or explicitly requested by the Service Provider to the TSM

The OTA service activation of the application(s) of the NFC service is performed by the TSM itself, or thanks to collaboration with the SE Provider and the MNO.

Note: this use-case can rely on, but does not specifically refer to GP technical process to Make Selectable a NFC application hosted in a Secure Element.

Post-conditions:

- The end-user is able to use its new mobile NFC service.

2.1.1.3 Use Case #3: Mobile-NFC Service Suspension

Pre-conditions:

- The end-user has an operational mobile-NFC service deployed and activated.

Description:

For various reasons such as billing problems with the end-user, the Service Provider may require to the NFC service to be temporarily un-usable by the end-user.

To do so, the SP asks the TSM for suspending the service. Knowing the exact service status and the exact NFC service items configuration values (such as the instance AID of the applications in the SE), the TSM performs an OTA lock of the applications of the NFC service. This OTA lock may concern the SE applications as well as the UI applications. The TSM may also delegate this locking to the SE Provider or the MNO.

At the end of this suspension mechanism, the TSM shall notify the relevant actors of the ecosystem of the status of the service in order for them to perform the relevant OTA and backend actions on their information systems.

The SE Provider may also want the NFC services to be temporarily un-usable, for example when the Secure Element has been lost.

Thanks to its potential knowledge of the content of the Secure Element, the SE Provider may lock the Secure Element applications using its own capabilities. However, to perform locking of the mobile equipment service components (i.e. also non-SE applications) and of the backend service components, the SE Provider shall notify the TSM that it has suspended part of the service, providing the reason of this suspension. Following this notification, the TSM may perform its own complementary suspension actions, and/or notify the SP of this suspension status.

Finally, the end-user itself, for instance prior to a planned period of non-usage of the service (e.g. temporary loaning of device to party not authorized to use the service), may require the NFC service to be un-usable. The end-user will contact either the SP or the SE Provider to suspend the service, and one of the processes described above will be launched.

According to the specification GP 2.2 Amendment C [5], one possible OTA operation to lock a SE application may be to send the GlobalPlatform SET STATUS command to the Head application.

Post-conditions:

- The NFC service items are still present in the SE and in the Device, but the SE applications for the NFC service are in a locked state, preventing from the usage of the NFC service through a NFC reader.
- The UI applications of the service may also be locked.

2.1.1.4 Use Case #4: Mobile-NFC Service Resumption

Pre-conditions:

- The mobile-NFC service of an end-user has been previously suspended.

Description:

Following a previous service usage suspension, as described in section 2.1.1.3, and for various reasons the SE Provider (e.g. Secure Element recovered), the SP (e.g. end of billing problems), or the end-user itself (e.g. return of the device that was temporarily loaned) may request the NFC service to be usable again:

- The SP may ask the TSM for resuming the service. Knowing the exact service status and the exact NFC service items configuration values (such as the instance AID of the Applications in the SE), the TSM performs by itself an OTA unlock of the applications of the NFC service or may delegate this unlocking to the SE Provider or the MNO.

  The TSM shall then notify the relevant actors of the ecosystem of the status of the service in order for them to perform the relevant OTA and backend actions on their information systems.
- The SE Provider may potentially perform the resumption of the service by itself, thanks to its visibility on the content of the Secure Element.

  The SE Provider shall then notify the relevant actors of the ecosystem that it has resumed part of the service, and shall provide the reason for this resumption.
- The end-user may ask either the SP or the SE Provider to perform the resumption of the service. One of the processes described above will then be launched.

Particular business agreements between parties may require that the resumption process can only be initiated by the actor who previously suspended the mobile-NFC service.

According to GP 2.2 Amendment C [5], one possible OTA operation for unlocking a SE application may be to send the GlobalPlatform SET STATUS command to the Head Application.

Post-conditions:

- The end-user is able to reuse a mobile NFC service.

2.1.1.5 Use Case #5: Mobile-NFC Service Upgrade

Pre-conditions:

- The end-user has a NFC service deployed in the Secure Element of its Device. The service may be suspended or not.
- A new version of the same service is made available by the SP.

Description:

During the life of a service, several versions may be issued by the SP: to enhance the service features, to fix a security problem, to provide specific UI layout (e.g. Christmas-oriented UI, summer-oriented UI), etc.

The SP delegates the OTA upgrade operation of the service to its TSM.

Knowing the exact service items and the changes in the service version, the TSM determines the set of single OTA operations to be performed for upgrading the NFC service. Each single OTA operation is then performed by the TSM itself, or thanks to collaboration with the SE Provider and the MNO.

Such single OTA operations may be:

- Locking the Application not in scope of upgrade, to prevent usage of the service during the service upgrade
- Existing Application deletion,
- Existing Executable Load File deletion and new Executable Load File loading (and extradition),
- New Application instantiation (and extradition), new Application personalization and new Application activation,
- Application locking/unlocking, in order to keep the service status (suspended or not) that the service had before the upgrade,
- UI Application re-loading in the Device or in the SE, UI Application re-personalization and activation, if required.

Post-conditions:

- The new version of the NFC service is operational and can be used by the end-user.
- The service has the same status as prior to the OTA upgrade (i.e. suspended or not).

2.1.1.6 Use Case #6: Mobile-NFC Service Data Exchange/Update

Pre-conditions:

- The end-user has a NFC service deployed in the Secure Element of its Device. The service may be suspended or not.

Description:

During the life of a service, it may be required to update some particular data stored in the SE application of the NFC service (balance value, tickets, service's validity period, service's security counters and information, etc.) or more generally to exchange data from the application and the Service Provider.

The SP delegates the OTA operation of the service data exchange to its TSM.

Knowing the exact service items and the technical operations to update the data in the applications of the NFC service, the TSM determines the set of single OTA operations to be performed for updating the NFC service. Each single OTA operation is then performed by the TSM itself.

The data exchange action does not impact the service status (suspended or not). However, depending on the technology used for service suspension (for example: SE application lock), data exchange might not be possible in the suspended state.

In some cases, the SE application might not allow updating some particular data and requires a complete re-personalization. In these cases, the usual service personalization capability depicted in use case #1 in section 2.1.1.1 should be used (either for directly re-personalizing the existing service, or for deleting the existing service and then re-instantiating it and re-personalizing it).

Post-conditions:

- The internal data of the service are updated. Depending on these updated data, the behavior of the service may change.
- The service has the same status as prior to the data OTA update (suspended or not).

2.1.1.7 Use Case #7: Mobile-NFC Service Un-Deployment

Pre-conditions:

- The end-user has a NFC service deployed in the Secure Element of its Device. The service may be suspended or not.

Description:

Mobile-NFC service deletion may be trigger by:

- The end -user itself, that for example, decides to unsubscribe to the service
- The SP, that for example, terminates the end-user subscription to the NFC service (e.g. bills not paid).

The SP or the end-user (through the SP) asks the TSM for the deletion of the service.

Knowing the exact NFC service items (such as the instance AID of the Applications in the SE), the TSM determines the set of single OTA operations to be performed for deleting the NFC service. Each single OTA operation is then performed by the TSM itself, or thanks to collaboration with the SE Provider and the MNO.

Such single OTA operations may be:

- Application deletion from the SE
- Executable Load File deletion from the SE
- Security Domain deletion from the SE
- UI Application deletion from the Device or from the SE

Some Executable Load File or UI Applications may not be deleted, if they are also used by other mobile NFC services, or depending on the SP or SE Provider policy (which may want to keep some applications or data in the Device or in the SE, for later re-use or re-subscription of a service).

Post-conditions:

- The NFC service is no longer available for the end-user.
- The service items that are not used by other services, or that should not be kept in the mobile environment are totally removed from the SE and from the Device. Items that are shared with other services are kept in the SE and in the Device.

2.1.2 Use Cases Related to End-User Life Cycle Management

2.1.2.1 Use Case #8: Secure Element Change

Pre-conditions:

- The end-user has an operational mobile-NFC service deployed and activated (or suspended).

Description:

The end-user may request to replace a Secure Element to obtain larger memory space, additional services, or new SE capabilities. Similarly, the SE Provider may decide to replace the Secure Element in order to provide additional services to its subscribers or to renew its SE installed base.

In order to ensure mobile-NFC service continuity, the SE Provider will request to install on the new SE all the mobile-NFC services that were present in the old Secure Element.

Various technical services may be triggered to perform this service transfer:

- Audit of the content of a service (to get current balance, amount, points, etc.) on the old Secure Element
- Service un-deployment, or at least service suspension, on the old Secure Element
- Service deployment on the new Secure Element. A newer version of the service may be delivered.

Information of this SE renewal should be made available to the relevant actors of the ecosystem so that they can perform the relevant OTA and backend actions on their information system.

Post-conditions:

- The end-user has a new SE with the same mobile-NFC services (or an updated version of same mobile NFC services).
- The services are in the same state (suspended or not) as they were in the old SE.
- The actors of the ecosystem have updated their internal information system with the new SE knowledge.

2.1.2.2 Use Case #9: Mobile Phone Number Change

Pre-conditions:

- The end-user has an operational mobile-NFC service deployed and activated (or suspended).

Description:

The end-user wants to change the mobile phone number without changing the UICC, or the mobile phone, or the Mobile Network Operator.

This change results in a change of the end-user identifier and potentially the means to reach OTA the Device and the Secure Element.

The MNO provides a new mobile phone number to the end-user and notifies the relevant actors of the ecosystem of this change; specifically the TSM. All of the relevant actors can then update their information system with this change.

Post-conditions:

- The Device and the Secure Element are accessible through a new mobile phone number.
- Actors of the ecosystem have updated their internal information system with the new mobile phone number.

2.1.2.3 Use Case #10: Mobile Device Change

Pre-conditions:

- The end-user has an operational mobile-NFC service deployed and activated (or suspended).

Description:

The end-user requests a change (i.e. replacement) of the Mobile Device.

In case of removable Secure Element (UICC, SMC), this corresponds to a change of the Device only. In case of Embedded Secure Element, this corresponds also to a SE change (see section 2.1.2.1).

From the Secure Element point of view, the new Device may have different capabilities than the former Device (e.g. screen size, display capabilities, OTA capabilities, other technical features, etc). Such variation may impact the correct processing of the mobile-NFC services already installed in the Secure Element.

Moreover, if the UI Application of the mobile-NFC service was deployed in the Device, these UI Applications needs to be re-deployed on the new Device.

So, to maintain continuity of the mobile-NFC service, it may be required to partially reinstall the mobile-NFC services: for example, re-installation of the UI Application(s) on the new Device.

Information of this Device renewal should be made available to the relevant actors of the ecosystem so that they can perform the relevant OTA and backend actions on their information system.

Post-conditions:

- The end-user has a new Device with the same mobile-NFC services available.
- The services are in the same state (suspended or not) as they were before.
- The actors of the ecosystem have updated their internal information system(s) with the new Device knowledge.

2.1.2.4 Use Case #11: Lost or Stolen Mobile Device

Pre-conditions:

- The end-user has an operational mobile-NFC service deployed and activated (or suspended).

Description:

The Device holding the Secure Element has been lost or stolen.

The end-user either contacts the MNO, the SE Provider, or the various Service Providers. The contacted entity may perform internal updates of its information system to disable the service usage. It may also try to temporary render the service un-available (e.g., by triggering a service suspension).

Then, it will notify the relevant actors of the ecosystem in order for them to perform the relevant OTA and backend actions on their information system.

Note that when the MNO is notified, it might also suspend the mobile network line, preventing any OTA communication to the Device.

Post-conditions:

- The Mobile-NFC services may have been suspended.
- The actors of the ecosystem have updated their internal information system(s) to prevent any usage of the NFC services.

2.1.2.5 Use Case #12: Recover Mobile Device After a Loss

Pre-conditions:

- The end-user has an operational mobile-NFC service deployed and activated (or suspended).
- The Device has previously been identified as lost or stolen.

Description:

After reporting a loss or theft of the Device, the end-user recovers the Device with the Secure Element.

The end-user either contacts its MNO, the SE Provider, or the various Service Providers. The contacted entity may perform internal updates of its information system to re-enable service usage. It may also make the service available again (e.g., by triggering the resumption of the service)

Then, it will notify the relevant actors of the ecosystem so that they can perform the relevant OTA and backend actions on their information system.

Note that in case the mobile network line has previously been suspended by the MNO, the MNO must be notified first to re-activate the line so that OTA access to the Device can be restored.

Post-conditions:

- Mobile-NFC services status is restored.

2.1.2.6 Use Case #13: Get a New Mobile Device After a Loss

This use case is the same as the Mobile Device change use case (see section 2.1.2.3).

In the situation where the Secure Element has been lost together with the Mobile Device, the Secure Element change use case also applies (see section 2.1.2.1).

2.1.2.7 Use Case #14: Mobile Subscription Status Change

Pre-conditions:

- The end-user has an operational mobile-NFC service deployed and activated (or suspended).

Description:

During the life of the Mobile Subscription, several events may happen: Mobile Subscription suspension/re-activation, NFC option (attached to the Mobile Subscription) suspension or restoration, or even Mobile Subscription or NFC option termination.

<u>As a first example:</u>

The end-user asks for, or the MNO mandates the temporary suspension of the end-user Mobile Subscription. This is for example the case if the end user knows he will travel in a foreign country for a long duration and consequently asks for a temporary suspension of its Mobile Subscription.

Information of the Mobile Subscription suspension should be made available to the relevant actors of the ecosystem so that they can perform the relevant OTA and backend actions on their information system, or even suspend the mobile-NFC services.

<u>As a second example:</u>

The end-user asks for, or the MNO mandates the permanent termination of the end-user Mobile Subscription or the permanent termination of a mobile-NFC option attached to the Mobile Subscription.

Prior to the Mobile Subscription or NFC option termination, the MNO may notify the end-user that a limited period of time exists to backup its mobile-NFC services data. The end-user can then contact the mobile-NFC service providers to perform these backup actions.

Information of the Mobile Subscription termination should be made available to the relevant actors of the ecosystem so that they can perform the relevant OTA and backend actions on their information system, or even the suspension or the un-deployment of the services.

Post-conditions:

- The actors of the ecosystem have updated their internal information system with the new status of the Mobile Subscription.

2.1.2.8 Use Case #15: MNO Swap

***Note**: the MNO swap use case is not covered in this version of the specification.*

Pre-conditions:

- The end-user has an operational mobile-NFC service deployed and activated (or suspended).

Description:

The end-user changes the Mobile Subscription (MNO swap) and asks for the transfer of the mobile-NFC services from the previous Mobile Subscription to a new Mobile Subscription

Depending on who owns the SE, this could lead to different processing:

- If the SE is not owned by the MNO, only a simple update of the information systems of all the actors of the ecosystem (i.e. to reference the new Mobile Subscription to target the Secure Element) is required,
- If the SE is owned by the MNO, a full transfer of the services to a new Secure Element (i.e. suspending or even un-deploying the services, and then re-deploying the same services to another Secure Element and Device) is required.

Information of the MNO swap should be made available to the relevant actors of the ecosystem so that they can perform the relevant OTA and backend actions on their information systems.

Post-conditions:

- The actors of the ecosystem have updated their internal information systems with the new Device knowledge.
- The services are available for the new Mobile Subscription, in the same state (suspended or not) as they were before.

2.2 Interactions Between Roles

There are various actors involved in the mobile-NFC ecosystem. This section of the specification defines the discrete roles that may be played by those actors.

An actor is a physical entity in a specific business model; for example, a MNO or a Bank.

A role refers to a logical grouping of functions. For example, Security Domain Manager refers to Security Domain management functions.

Actors are entities assuming roles, meaning they are assuming the responsibility of the associated functionality.

An actor may play several roles; for example, a MNO playing Secure Element Issuer and Security Domain Manager roles.

Several actors may play the same role. For example, the TSM and MNO may play the Security Domain Manager role as highlighted in section 2.2.1.2.

2.2.1 Roles

2.2.1.1 Roles Description

The main properties of the roles taking part in the mobile NFC service deployment and management are described below:

- The **Application Provider** (AP) is responsible for the global (SE and Device) mobile-NFC service management towards the end-users as it has the direct business relationship with them.

  The Application Provider role defines the various elements of the mobile-NFC service (both SE and Device elements) and procures the necessary components to load into the mobile environment (i.e. the SE and Device applications code and data, the SE applications keys and/or certificates, and the data belonging to a specific cardholder). However, the Application Provider does not perform the service management operations. Instead, it delegates these operational tasks to the Security Domain Manager though it may retain responsibility for personalization of the application, e.g. the generation of personalization data.

- The **Secure Element Issuer** (SEI) holds ultimate responsibility for the GlobalPlatform card. The SEI has the responsibility to develop the card product profile, to choose the platform and application technologies to design card layout.

  The Secure Element Issuer usually holds a particular Security Domain in the SE: the Issuer Security Domain (ISD). Card Content Management (CCM) operations that can be performed on this ISD are associated to the Security Domain Manager role that a SE provider may also play.

- The **Security Domain Manager** (SDM) has three possible facets, each of them implying some responsibilities:
  - o *Eligibility and Service Management:* if implementing this facet, the SDM is responsible for the high level service management operations on behalf of the Application Provider.

It handles the high level scheduling of Card Content Management and Device management to be performed on the SE and on the Device, in order to deploy and manage the mobile-NFC service. However, this facet does not include the specific personalization script building.

  - *SE Content Management:* if implementing this facet, the SDM is responsible for the execution of single actions related to the SE (such as loading an ELF, installing a SE application, generating a Delegated Management Token, performing SE content audit, etc.), that takes part of a mobile-NFC service management

    The SDM is then responsible for a (set of) Security Domain(s) in a SE. Depending on the privileges associated to its Security Domains, the SDM may have the capability to directly load, install, extradite or personalize applications on behalf of an Application Provider.

    In case it does not have the necessary Card Content Management privileges or Secure Channel keys, or it does not have an OTA capability, it may request the help of another SDM to perform Card Content Management or OTA dialog. Otherwise, a SDM may perform CCM operations on behalf of another SDM which does not have those necessary privileges, or grant CCM Delegated Management tokens for other SDMs, or may serve as OTA provider for other SDMs or even AP for application personalization.

  - *Personalization Script Management:* if implementing this facet, the SDM is responsible for the generation and securization of the mobile-NFC service personalization script (SE and/or Device applications personalization scripts)

    The SDM is then responsible of the Security Domain that holds the keys necessary to secure the personalization script, but may rely on another SDM as OTA provider for the script OTA sending.

- The **Device and Mobile Subscription Registrar** (DMSR) has the following responsibilities:
  - Perform the eligibility checking on the Mobile Subscription: is able to provide a business (e.g. "NFC option" activated or not) and a technical (e.g. data bearer available or not) authorization regarding the Mobile Subscription to be used to reach the Secure Element and the Device.
  - Act as a Device registrar to track the association between the Secure Element, the Device, and the Mobile Subscription
  - Provide the capabilities of the Device
- The **Controlling Authority** (CA) is a third party authority that enforces the security policy in a multi-actor environment accessing a Secure Element. Specifically, it may be used for secure Security Domain creation in a SE.

The five roles described in this section interact with each other through standardized functional entry points that are specified later in this document. The use of the APIs is not limited to the examples provided in this document, but is dependent on the business agreements between the actors.

The APIs are composed of request-response functions and notification handler functions (no response) that are described in section 3. Functions are gathered into functions groups that correspond to a consistent functional behavior (e.g. the "Delegated Management" functions group contains all the functions related to the Delegated mode).

The following tables present the functions groups that are provided by each of the roles of the ecosystem: request-response functions (Table 2-1), and notification handler functions (Table 2-2).

Providing Role		Functions Group	Associated Functionality
SDM	Eligibility and Service Management facet	Global Eligibility Info	Provides information about the global eligibility of a mobile environment to a NFC service.
		Global Service Management	Management of the overall life cycle of the NFC service, driven by the Application Provider.
	Secure Element Content Management facet	Card Content Management	Card Content Management actions for Simple mode support.
		Delegated Management	Token and Receipt management for Delegated mode support
		SE Audit	Get the free memory of a Security Domain, or the status of Applications or Executable Load Files of a SE
		Script Sending	Pass thru script OTA transmission to the Secure Element
		Device Application Binding	Management of the security binding between Device applications and SE applications
		SCWS Management	Smart Card Web Server portal management
	Perso. script Mgt facet	*No function group*	-
SEI		SE Info	Provides information about a Secure Element
		CA Info	Provides information about the Controlling Authority of a SE
DMSR		Device Info	Provides information about a Device
		Mobile Subscription Info	Provides information about a Mobile Subscription
		Device Application Management	Management of Device applications
CA		CCCM Certificates Management	Management of certificates for Confidential Card Content Management
AP		*No function group*	-

Table 2-1: Functions groups for each role

Notification Handler Role	Functions Group	Associated Functionality
AP	Service Environment Change Notification	Notifications related to change of the mobile-NFC service environment
SDM	Service Life Cycle Notification	Notifications related to the life cycle of the mobile-NFC service
	SE Life Cycle Notification	Notifications related to the life cycle of the Secure Element
	Mobile Subscription Life Cycle Notification	Notifications related to the life cycle of the Mobile Subscription
	Device Application Life Cycle Notification	Notifications related to the life cycle of the Device applications
SEI	Service Life Cycle Notification (on SE)	Notifications related to the life cycle of the mobile-NFC service – Secure Element centric

Notification Handler Role	Functions Group	Associated Functionality
DMSR	Service Life Cycle Notification (on Mobile Subscription)	Notifications related to the life cycle of the mobile-NFC service – Mobile Subscription centric
	Device Application Life Cycle Notification	Notifications related to the life cycle of the Device applications
CA	*No function group*	-

Table 2-2: Notification handler functions group of each recipient role

2.2.1.2 Common Role-to-Actor Allocations

There are many possible allocations of roles to actors, as will be exemplified in the following sections.

2.2.1.2.1 A Typical Allocation

Below is presented a typical allocation of roles to actors, but should not be considered the only way of allocating roles to actors.

- The SE Provider actor would typically take the SEI and SDM (for the Issuer Security Domain) roles.
- The TSM actor would typically take the SDM role (for the TSM Security Domain, or for the Service Provider Security Domain on behalf of the Service Provider).
- The MNO actor usually plays the DMSR role, and may also play the SEI and SDM roles for the Issuer Security Domain (in case the NFC Secure Elements are UICCs).
- A Service Provider would typically take the AP role.

2.2.1.2.2 SDM Roles Allocation

Multiple roles of the same kind, such as the SDM role, may co-exist in the ecosystem, and be performed by different actors. This SDM roles allocation highly depends on the business relations that have been setup between actors for Card Content Management – defining the Card Content Management Modes (see section 1.5.2.1).

For example:

- The SE Provider may play the SDM role and thus provide the SE Content Management facet to the TSM. This defines the Card Content Management Mode between those two actors as being the Simple mode. On his side, the TSM may also play the SDM role but for the Eligibility and Service Management facet.
- As another example, the SE Provider may play the SDM role for Token generation and receipt verification whereas the TSM could play the SDM role for generating the SE Card Content Management commands for the Secure Element. This defines a Delegated mode CCM relation between those two actors.

- As a third example, the TSM may play the SDM role to autonomously perform SE Card Content Management: the Card Content Management relation between the SE Provider (or the MNO for UICC) and the TSM is Dual mode, as the TSM is fully autonomous in its SSD hierarchy. The Eligibility and Service Management facet of the SDM role may then be partially played by the Service Provider (for the Global Service Management purpose) and partially by the TSM SDM (for the Global Eligibility purpose).

  Note that in this example, several actors not only share the same role, but even have the same SDM facet: the Eligibility and Service Management facet of the SDM role is played by the Service Provider and by the TSM.

Note that as CCM modes are one-to-one relations around Card Content Management:

- There might be several of such relations within an ecosystem. For example, in the Secure Element, the SEI enables service management to Service Providers in Simple mode using the ISD, and in parallel a TSM uses another Security Domain for enabling service management to other Service Providers in Dual mode.
- In case a relation between two actors does not imply direct Card Content Management actions execution in the Secure Element, such a relation should not be qualified with a CCM mode.

In other words: **Card Content Management Modes only qualify a "SDM to SDM" relation (as a role to role relation)**. As mentioned in section 1.5.2.1, the CCM relation is most commonly (but not restricted to) a SE Provider centric relation; so a single CCM mode most commonly qualifies the relation between the Service Provider SDM and another actor's SDM. In some very specific use cases where this is not directly the SE Provider SDM that is involved in such relations, this is a SDM that is representative of the SE Provider, i.e. to which the SE Provider has devolved the SE management (e.g. a TSM SDM).

2.2.1.2.3 Allocation of Roles to the Service Provider and the TSM

Roles repartition between the Service Provider and the TSM depends on the delegation of the mobile-NFC service deployment to the TSM (defining the Service Deployment Modes depicted in section 1.5.2.2). Moreover, when playing the SDM role, SDM facet allocation can differ from one Service Deployment Mode to another:

- Service Deployment Mode #1: The SP only takes the AP role. It fully delegates the Eligibility and Service Management facet, SE Content Management facet and Personalization Script Management facet to the TSM SDM.

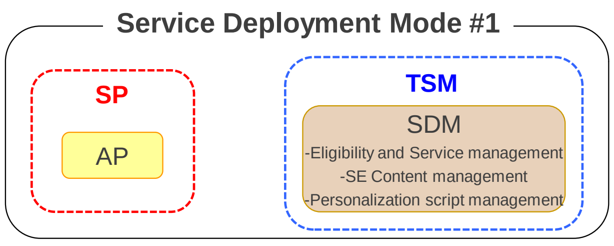

Service Deployment Mode #1
SP
AP
TSM
SDM
-Eligibility and Service management
-SE Content management
-Personalization script management

Figure 2-3: Service Deployment Mode #1: roles repartition

- Service Deployment Mode #2: The SP takes both the AP role and the Personalization Script Management facet of the SDM role. However, it delegates the Eligibility and Service Management facet and the SE Content Management facet to the TSM SDM.

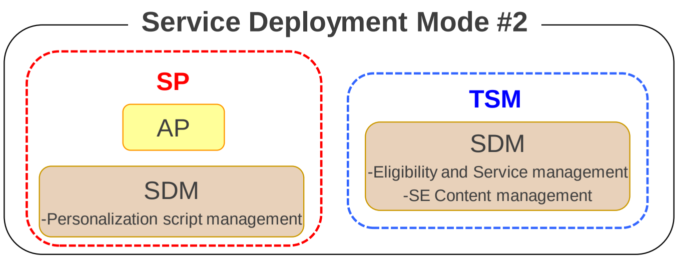

Service Deployment Mode #2
SP
AP
SDM
-Personalization script management
TSM
SDM
-Eligibility and Service management
-SE Content management

Figure 2-4: Service Deployment Mode #2: roles repartition

- Service Deployment Mode #3: The SP takes both the AP role, and the Eligibility and Service Management and the Personalization Script Management facets of the SDM role. However, the Eligibility and Service Management facet of the SP SDM excludes the Global Eligibility Info functions group which is provided by the TSM SDM. SE Content Management facet of SDM is also delegated to the TSM.

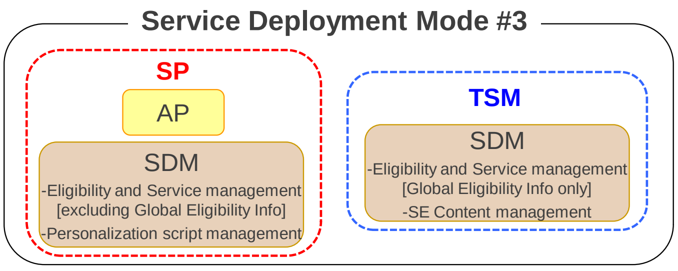

Service Deployment Mode #3
SP
AP
SDM
-Eligibility and Service management
[excluding Global Eligibility Info]
-Personalization script management
TSM
SDM
-Eligibility and Service management
[Global Eligibility Info only]
-SE Content management

Figure 2-5: Service Deployment Mode #3: roles repartition

In each of these relations, the TSM plays some facets of the SDM role, and **the Service Deployment Mode qualifies the "SP to TSM" relation (as an actor to actor relation, each actor potentially playing several roles)**. This could also be generalized to a "SP to another actor" relation, in case no TSM exists in the ecosystem.

2.2.2 Message Flow

Functions are mapped into messages that are exchanged between roles (see section 4 for specific function to message mapping).

This section presents the following generic message flows between roles:

- Figure 2-6: the request-response message flows
- Figure 2-7: the notification message flows

The purpose of having a generic flow is to show that it is possible to use the same roles and APIs independently of the mode (Simple Mode, Delegated Mode and Dual Mode). In the next sections we give examples of how the generic message flow can be realized in different modes and with different types of SE. This is not an exhaustive list of possible deployment scenarios.

Legend used in the following message flow diagrams:

- Functions groups that are in the scope of this specification are in black solid line,
- Functions groups that are in the scope of this specification but not specified in this release are in grey solid line,
- Functions groups out of the scope of this specification are in grey dashed line (with italic description).
- Functions groups and steps that are put in brackets are optionally processed within the global flow

The messaging between OTA server and Device is managed via network equipment that belongs to a MNO (SMSC, GGSN, etc.). As this equipment already has standardized interfaces (SMPP, IP, etc.), and in order not to add complexity to the diagram, a direct arrow is drawn from the SDM to the Device.

When an actor plays several roles, the interfaces between these internal roles may be implemented in a proprietary way.

2.2.2.1 Generic Message Flow

2.2.2.1.1 Request-Response Message Flow

The following figure presents the request and response message generic flow:

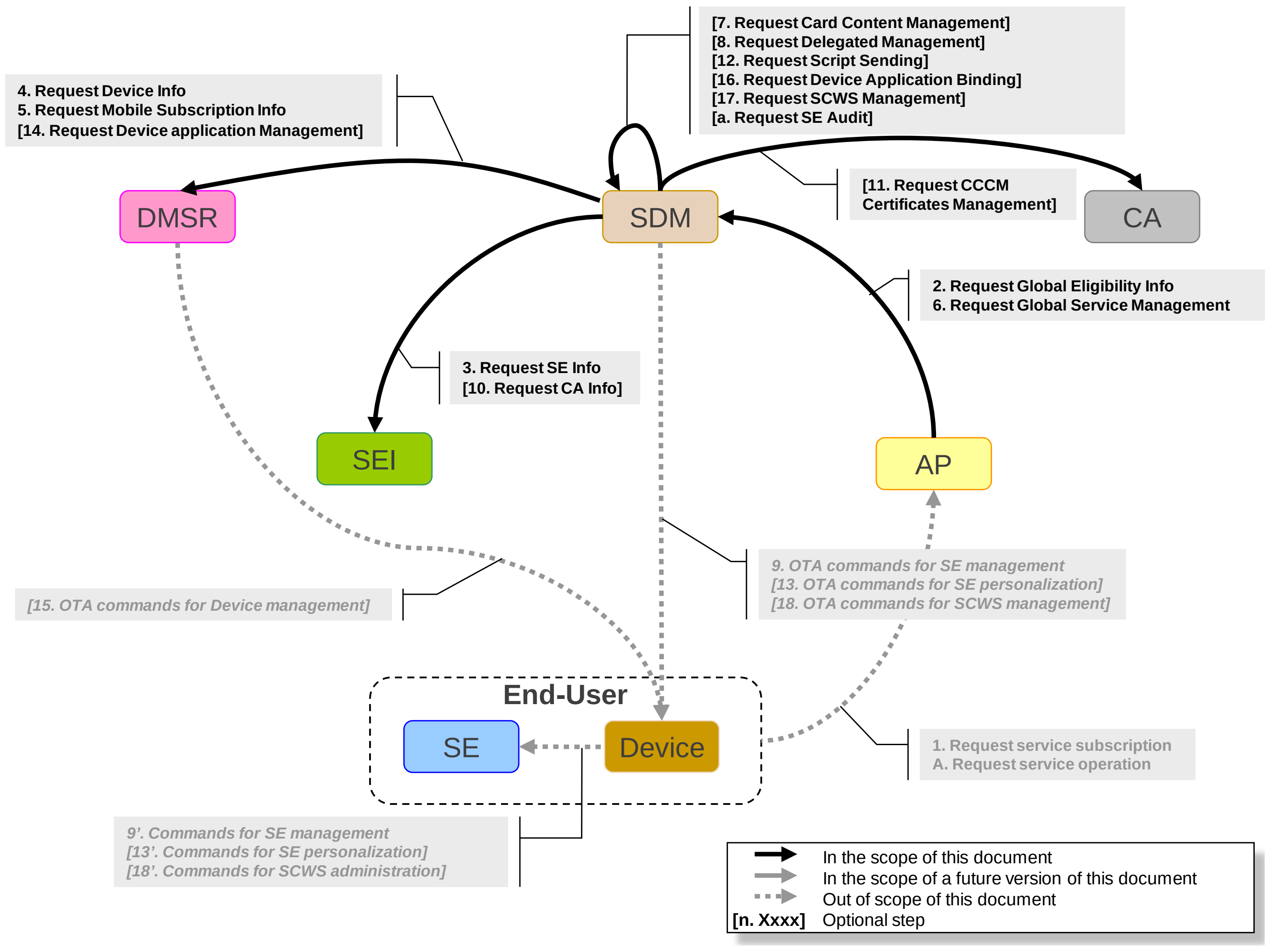

[7. Request Card Content Management]
[8. Request Delegated Management]
[12. Request Script Sending]
[16. Request Device Application Binding]
[17. Request SCWS Management]
[a. Request SE Audit]
4. Request Device Info
5. Request Mobile Subscription Info
[14. Request Device application Management]
[11. Request CCCM Certificates Management]
DMSR
SDM
CA
2. Request Global Eligibility Info
6. Request Global Service Management
3. Request SE Info
[10. Request CA Info]
SEI
AP
9. OTA commands for SE management
[13. OTA commands for SE personalization]
[18. OTA commands for SCWS management]
[15. OTA commands for Device management]
End-User
SE
Device
1. Request service subscription
A. Request service operation
9'. Commands for SE management
[13'. Commands for SE personalization]
[18'. Commands for SCWS administration]
In the scope of this document
In the scope of a future version of this document
Out of scope of this document
[n. Xxxx] Optional step

Figure 2-6: Generic request-response flow between roles

Note: numbers and their related steps are given to help understand the flow sequence. Depending on the business agreement, the order may be different. However, the order of the steps remains constrained by GlobalPlatform State transitions.

The flow of messages in the general case is as follows:

1. The end-user requests a subscription to a mobile-NFC service ("Service subscription" functions group)

Phase 1: Initialization

2. The AP should first request for the global eligibility of the end-user mobile environment to a SDM ("Global Eligibility Info" functions group).

To determine the global eligibility to the mobile-NFC service:

3. The SDM requests information from the SEI regarding the Secure Element capabilities ("SE Info" functions group).

   These capabilities are needed by the SDM for example to take business decisions on whether a customer SE fits for the requested service.

4. The SDM also requests information from the DMSR regarding the Device capabilities ("Device Info" functions group).

   These capabilities are needed by the SDM for example to take business decisions on whether a customer Device fits for the requested service.

5. Finally, the SDM requests authorization and information to the DMSR regarding the Mobile Subscription ("Mobile Subscription Info" functions group) to be used to reach the SE and the Device.

6. If global eligibility is confirmed, the AP now delegates the deployment of the mobile-NFC service to a SDM ("Global Service Management" functions group)

The SDM then starts the global deployment of the mobile-NFC service. The SDM may request SE and Device information again:

3. [On the Secure Element ("SE Info" functions group): for example to optimize the configuration of the service or the installation process (e.g. the structure of the script to be sent, or which UI application to use), or to get information on the Controlling Authority of the Secure Element]
4. [On the Device ("Device Info" functions group): for example to optimize the installation process of the service (e.g. the structure of the script to be sent to the device, which UI service to be used depending on screen characteristics, etc).]

Phase 2: Card Content Management

If the SDM does not have sufficient privileges on the SE, the SDM needs the help of another SDM having the required privileges:

7. [If the SDM does not have sufficient privileges on the SE, the SDM requests for CCM operations to another SDM ("Card Content Management" functions group)].

Otherwise, if the SDM has sufficient privileges on the SE, it generates the necessary Card Content Management commands.

8. [But, if Delegated Management tokens are required, the SDM asks the relevant SDM to generate these Tokens ("Delegate Management" functions group).]

Finally:

9. Commands are sent by the relevant SDM to the Secure Element, through the Device ("OTA commands for SE management" and then "Commands for SE management").

Phase 3: Personalization

It may be required to personalize Supplementary Security Domains (i.e. to put keys inside the SSD) and to personalize the SE applications of the mobile-NFC service.

Security Domain personalization may require the involvement of the Controlling Authority, for confidential SD personalization:

10. [For such confidential SD personalization, the SDM requests the SEI to know which CA is present in the SE ("CA Info" functions group)…]
11. [The SDM requests the Controlling Authority to deliver a certificate ("CCCM Certificates Management" functions group)]

The SDM generates the personalization commands for the SSD or for the NFC service application. Depending on the OTA capability, the SDM sends the script itself to the Secure Element or requests the OTA operation to another SDM.

12. [If the SDM does not have any OTA capability, it requests the OTA sending of the script to another SDM ("Script Sending" functions group).]
13. [The personalization commands are sent OTA to the Secure Element ("OTA commands for SE personalization"), –through the Device ("Commands for SE personalization").]

Note that the second and the third phases (from 6. to 12.) may be executed a first time for creating and personalization a Supplementary Security Domain in the Secure Element, and then several times for delivering the SE applications of a mobile-NFC service.

Phase 4: UI management

The SDM deploys the User Interface part of the mobile-NFC service. Depending on the service and on the Secure Element, this part can either be located in the Device (as a Device Application) or in the Secure Element (as a Smart Card Web Server [17] portal in the UICC – only possible if the SE is a UICC).

The SDM may either have the capability to perform the OTA management of the Device applications or of the SCWS portal by itself, or may need to request it to another SDM.

Note that in case of a Device application, security binding of the Device application to the Secure Element application may be requested by the actors of the ecosystem. This binding capability however highly depends on the Device technology.

14. [If a Device application is to be loaded, the SDM requests for the loading of the Device applications part of the service to the DMSR ("Device Application Management" functions group).]
15. [Commands are sent by the DMSR to the Device, for performing or just triggering the Device application loading ("OTA commands for Device management").]
16. [Then, if the Device application is to be bound to the SE application of the mobile-NFC service, the SDM may request another SDM to perform this binding ("Device Application Binding" functions group) if it does not have the capability and privileges to do it by itself.]
17. [If a SCWS Portal is to be loaded, and if the SDM does not have the capability and privileges to do it by itself, the SDM requests another SDM for the loading of the SCWS portal of the service ("SCWS Management" functions group).]

18. [SCWS administration commands are sent by the relevant SDM to the Secure Element, through the Device ("OTA commands for SCWS management" and then "Commands for SCWS management").]

Phase 5: Deployment Process Finalization

At the end of the global service delivery, the TSM SDM notifies of the global status of the service ("Service Life Cycle Notification" functions group).

Note that at any stage of the process, the SDM may request for an audit of the content of the Secure Element to get the available free memory in the SE or to get the status of the Applications, Security Domains or Executable Load Files (a. "SE Audit" functions group).

Later on, the end-user may require making some operations on its service (A. "Service operation" functions group). Following those requests, or without any involvement of the end-user, the AP may ask the SDM to perform some OTA service upgrade or some service data update ("Global Service Management" functions group).

For those OTA actions, similar behavior as the one depicted for the service deployment is performed: Device or SE OTA commands are generated and sent by the SDM to the Secure Element, through the Device.

Table 3-1 lists the functions defined in this specification, and that are involved in the links/functions groups presented in the here above diagram.

The following links are out of scope of the specification:

- Request service subscription
- Request service operation
- OTA commands for SE management
- OTA commands for SE personalization
- OTA commands for Device management
- OTA commands for SCWS management
- Commands for SE management
- Commands for SE personalization
- Commands for SCWS management

Sections 2.2.2.2.1, 2.2.2.2.2, and 2.2.2.2.3 describe the specific interactions in the different deployment modes (Simple, Delegated and Dual) and for different types of SE (UICC, ESE and SMC). It should be noted that the generic message flow described in this section applies to all cases. It is emphasized that the roles to actors allocations are just provided as example.

2.2.2.1.2 Notification Messages Flow

Each global operation performed on a mobile-NFC service requires several unitary tasks to be executed by different roles. It is consequently difficult for a particular role that is not initiating a global operation to be aware of the global mobile-NFC service status.

"Service Life Cycle Notification" (α) provides visibility on the global state of a mobile-NFC service to the other roles. They shall be generated before and at the end of each global operation performed for a mobile-NFC service.

"Service Environment Change Notification" (δ) notifies the Application Provider that something has changed in the environment of the mobile-NFC service.

Moreover, during the overall life of a mobile-NFC service, several events may impact the service status. Those events may be of three types:

- Secure Element centric events: they are related to a modification of the SE or of the environment around the SE, such as: the SE is renewed, the Device hosting the SE has changed, the SE is reachable through another Mobile Subscription, etc.
- Device centric events: they are related to actions on the Device that hosts the Secure Element. For example the loading of a Device application directly requested by the end-user using an application store.
- Mobile Subscription centric events: they are related to a modification of the Mobile Subscription that is used to reach the Secure Element/Device hosting the mobile-NFC service, for example identifier of the Mobile Subscription changed, status of the Mobile Subscription changed, etc.

These events may imply several administrative tasks related to the mobile-NFC service: for example re-deployment of the UI part, reinstallation of the service, etc. Those tasks are covered by the message flow presented in section 2.2.2.1.1. However, the role that detects the event and the one that is able to process the tasks on the mobile-NFC service are often different. Therefore, "SE Life Cycle Notification" (β), "Mobile Subscription Life Cycle Notification" (γ), and "Device Application Life Cycle notification" (ε) are defined and used for this role-to-role notification.

Note that in this version of the specification, some events related to a change of actors are not supported (e.g. SE renewal to a SE, issued by a different issuer). Only the Mobile Subscription change (with MNO change) is supported. The other events will be addressed in a future version of the specification.

Specific functions are required to manage the notifications. The following figure presents generic flow related to the notification functions.

Note that the notification functions may be generated by any role, so the following diagram and the description of the notification functions (provided in section 3) only highlight the roles that receives and processes those events. An example of a possible deployment or responsibilities in event generation is provided in Figure 2-8.

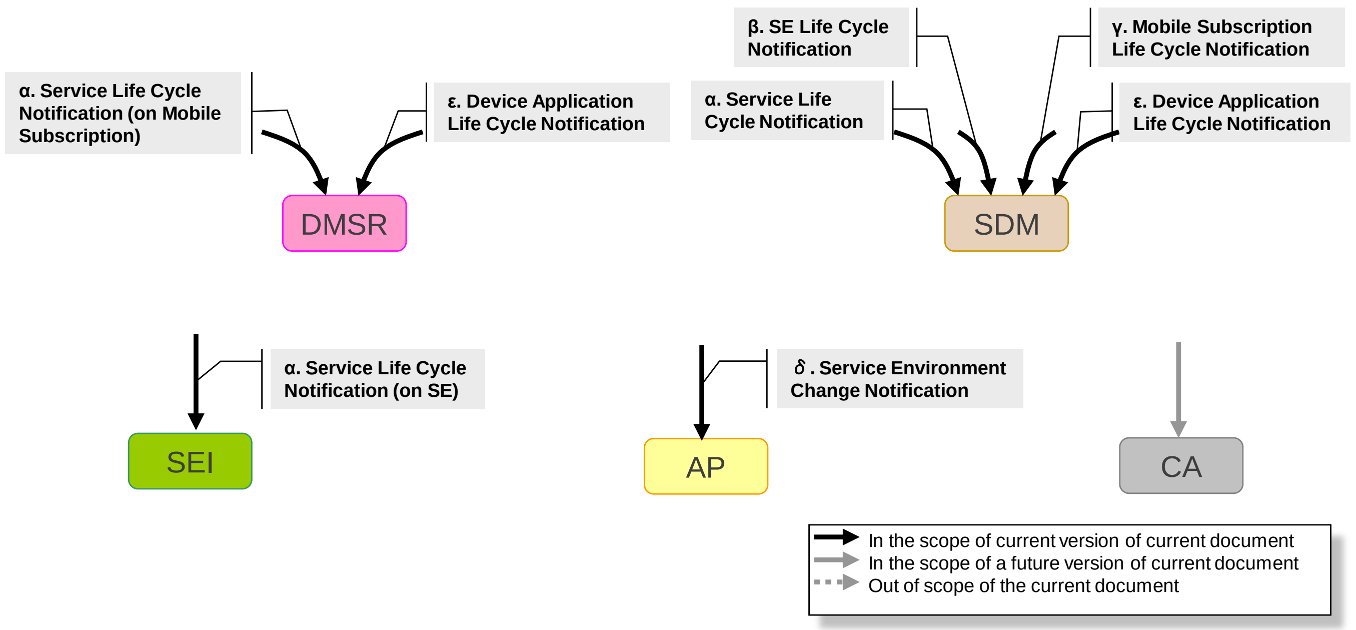

α. Service Life Cycle Notification (on Mobile Subscription)
ε. Device Application Life Cycle Notification
β. SE Life Cycle Notification
α. Service Life Cycle Notification
γ. Mobile Subscription Life Cycle Notification
ε. Device Application Life Cycle Notification
DMSR
SDM
α. Service Life Cycle Notification (on SE)
δ . Service Environment Change Notification
SEI
AP
CA
In the scope of current version of current document
In the scope of a future version of current document
Out of scope of the current document

Figure 2-7: Generic notification flow between roles

Table 3-2 lists the notification functions defined in this specification, and that are involved in the links presented in the here above diagram.

The following schema presents an example of a possible deployment. It highlights a possible assignment of the event generation to roles.

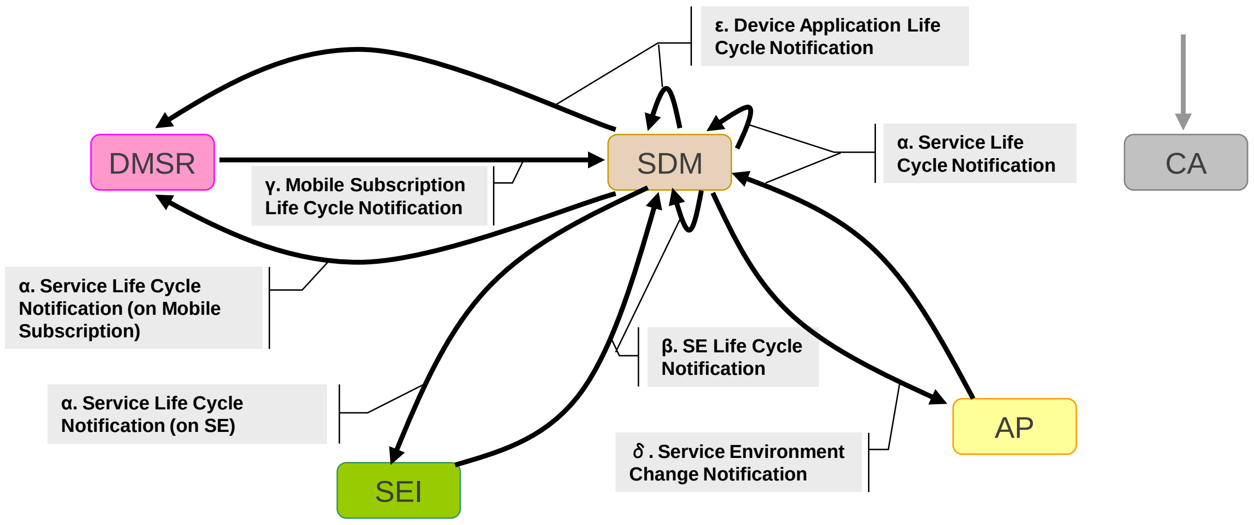

ε. Device Application Life Cycle Notification
DMSR
SDM
CA
γ. Mobile Subscription Life Cycle Notification
α. Service Life Cycle Notification
α. Service Life Cycle Notification (on Mobile Subscription)
β. SE Life Cycle Notification
α. Service Life Cycle Notification (on SE)
δ . Service Environment Change Notification
AP
SEI

Figure 2-8: Possible event generation responsibilities

2.2.2.2 Example of Interactions for Service Deployment Mode #1

The following sub-sections give examples of roles to actors allocation in the context of the Service Deployment Mode #1.

In this mode, the Service Provider only plays the AP role and fully delegates the Eligibility and Service Management, SE Content Management and Personalization Script Management to the TSM.

A Card Content Management relation exists between the TSM SDM and the SE Provider SDM:

- Section 2.2.2.2.1 presents a possible ecosystem with a Simple mode relation between those two actors, for a Secure Element being either a UICC or an Embedded Secure Element or a Secure Memory Card.
- Section 2.2.2.2.2 presents a possible ecosystem with a Delegated mode relation.
- Section 2.2.2.2.3 presents a possible ecosystem with a Dual mode relation.

Such a descriptive, but non exhaustive set of examples enables to compare the various ecosystems, using the same Service Deployment Mode as assumption. Section 2.2.2.3 provides other examples for other Service Deployment Modes.

2.2.2.2.1 Example of Interactions in Simple Mode Between TSM and SE Provider

In the "Simple mode" relation between the TSM SDM and the SE Provider SDM (or a representative SDM), only the SDM of the SE Provider (or a representative SDM) can perform Card Content Management.

When the Secure Element is a UICC, the MNO is the SE Provider. So in Simple mode, it is assumed that the MNO will have the SEI role and the SDM role (it will perform the SE Content Management facet using its own OTA capabilities). The other operations targeting the UICC (such as application personalization or lock/unlock) may be managed by the MNO itself or by the TSM (assuming it has the required OTA capability).

When the Secure Element is not a UICC, the SE Content Management is managed by the SE Provider; the other operations may be managed by the SE Provider itself or by the MNO or the TSM. The SE Provider itself can provide the OTA capability. But, the OTA capability can also be provided by the MNO or the TSM depending on their capabilities and on their business agreement.

<u>Legend used in the following message flow diagrams is detailed in the introduction part of section 2.2.2.</u>

2.2.2.2.1.1 UICC

The following diagram presents the various interactions between the roles of the actors of the mobile-NFC ecosystem, in the following example context:

- The SE is a UICC, so the MNO has the SEI role,
- The Card Content Management Mode between the TSM SDM and the SE Provider SDM is the "Simple mode" (see section 1.5.2.1), meaning that the MNO has the SDM role for SE Content Management through the ISD and the TSM has the SDM role for Eligibility and Service Management.

- The mobile-NFC service is being deployed according to Service Deployment Mode #1 (see section 1.5.2.2). The Service Provider takes the AP role and delegates full operational deployment of the service to the TSM SDM (see section 2.2.1.2.3).

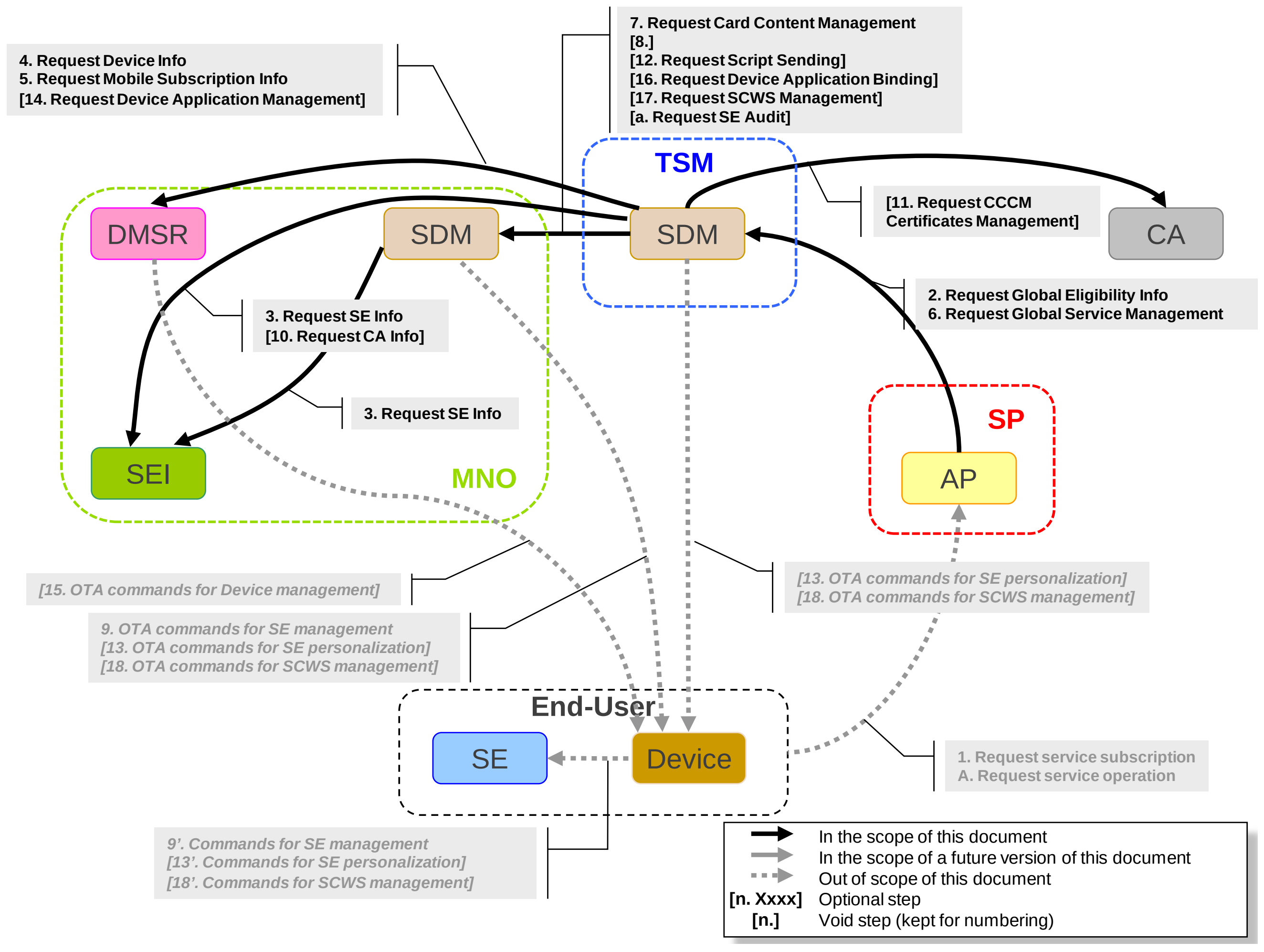

4. Request Device Info
5. Request Mobile Subscription Info
[14. Request Device Application Management]
7. Request Card Content Management
[8.]
[12. Request Script Sending]
[16. Request Device Application Binding]
[17. Request SCWS Management]
[a. Request SE Audit]
TSM
DMSR
SDM
SDM
CA
[11. Request CCCM Certificates Management]
2. Request Global Eligibility Info
6. Request Global Service Management
3. Request SE Info
[10. Request CA Info]
3. Request SE Info
SEI
MNO
SP
AP
[15. OTA commands for Device management]
[13. OTA commands for SE personalization]
[18. OTA commands for SCWS management]
9. OTA commands for SE management
[13. OTA commands for SE personalization]
[18. OTA commands for SCWS management]
End-User
SE
Device
1. Request service subscription
A. Request service operation
9'. Commands for SE management
[13'. Commands for SE personalization]
[18'. Commands for SCWS management]
In the scope of this document
In the scope of a future version of this document
Out of scope of this document
[n. Xxxx] Optional step
[n.] Void step (kept for numbering)

Figure 2-9: Simple mode with UICC. Example of roles to actors allocation.

1. The end-user requests for the subscription to a mobile-NFC service ("Service subscription" functions group)

Phase 1:

2. The AP should first request for the global eligibility of the end-user mobile environment to the TSM SDM ("Global Eligibility Info" functions group).

To determine the global eligibility to the mobile-NFC service:

3. The TSM SDM requests information from the SEI regarding the Secure Element capabilities ("SE Info" functions group).
4. The TSM SDM also requests information from the DMSR regarding the Device capabilities ("Device Info" functions group).

5. Finally, the TSM SDM requests authorization and information to the DMSR regarding the Mobile Subscription ("Mobile Subscription Info" functions group) to be used to reach the SE and the Device.

6. If global eligibility is confirmed, the AP now delegates the deployment of the mobile-NFC service to the TSM SDM ("Global Service Management" functions group)

The TSM SDM starts the global mobile-NFC service deployment by declaring the deployment operation (α. "Service Life Cycle Notification" functions group). Then the TSM SDM may request SE and Device information again:

3. [On the Secure Element capabilities ("SE Info" functions group)],
4. [On the Device capabilities ("Device Info" functions group)]

Phase 2:

7. As the TSM SDM does not have sufficient privileges on the SE, the TSM SDM requests for CCM operations to the MNO SDM ("Card Content Management" functions group)
8. *[Intentionally void step to harmonize enumeration of steps between the different cases: not relevant in Simple mode]*
9. The MNO SDM may request information from the SEI regarding the Secure Element capabilities (3. "SE Info" functions group), then generates the CCM commands, and finally sends them to the Secure Element, through the Device ("OTA commands for SE management" and then "Commands for SE management").

Phase 3:

Security Domain personalization may require the involvement of the Controlling Authority, for confidential SD personalization:

10. [For such confidential SD personalization, the SDM requests the SEI to know which CA is present in the SE ("CA Info" functions group)…]
11. [The SDM requests the Controlling Authority to deliver a certificate ("CCCM Certificates Management" functions group)]

The TSM SDM or the AP generates the personalization commands for the SSD or for the NFC service application. Then, it sends the script OTA by itself or delegates the OTA operation to another SDM:

12. [If the TSM SDM does not have any OTA capability, it requests the OTA sending of the script to the MNO SDM ("Script Sending" functions group).]
13. [The personalization commands are sent OTA to the Secure Element ("OTA commands for SE personalization"), through the Device ("Commands for SE personalization").]

Phase 4:

Note: to be able to communicate with the Secure Element, it might be required to download a Device application. If so, steps 14 to 16 might be required to be performed in a prior phase.

14. [If a Device application is to be loaded, the SDM requests for the loading of the Device applications part of the service to the DMSR ("Device Application Management" functions group).]
15. [Commands are sent by the DMSR to the Device, for performing or just triggering the Device application loading ("OTA commands for Device management").]
16. [Then, if the Device application is to be bound to the SE application of the mobile-NFC service, the TSM SDM may request the MNO SDM to perform this binding ("Device Application Binding" functions group) if it does not have the capability and privileges to do it by itself.]
17. [If a SCWS Portal is to be loaded, and if the TSM SDM does not have the capability and privileges to do it by itself, the TSM SDM requests the MNO TSM for the loading of the SCWS portal of the service ("SCWS Management" functions group).]
18. [SCWS administration commands are sent by the MNO SDM or by the TSM SDM to the Secure Element, through the Device ("OTA commands for SCWS management" and then "Commands for SCWS management").]

Phase 5:

At the end of the global service delivery, the TSM SDM notifies of the global status of the service (α. "Service Life Cycle Notification" functions group).

Note that at any stage of the process, the TSM SDM may request the MNO SDM for an audit of the content of the Secure Element to get the available free memory in the SE or to get the status of the Applications, Security Domains or Executable Load Files (a. "SE Audit" functions group).

The functions of the functions groups presented in the above diagram are listed in Table 3-1. The notification functions are listed in Table 3-2. These functions and notification functions are defined in section 3.

2.2.2.2.1.2 ESE or SMC

The following diagram presents the various interactions between roles of the actors of the mobile-NFC ecosystem, in the following example context:

- The SE is an ESE or a SMC. Contrary to the previous example with a UICC as Secure Element, the MNO is not the SE Provider, and is thus not playing the SEI role. The SE Provider is now involved as an additional and independent actor in the ecosystem.
- The Card Content Management Mode between the TSM SDM and the SE Provider SDM is the "Simple mode" (see section 1.5.2.1), meaning that the SE Provider has the SDM role for the CCM through the ISD and the TSM has the SDM role for Eligibility and Service Management.
- The mobile-NFC service is being deployed according to Service Deployment Mode #1 (see section 1.5.2.2). The Service Provider takes the AP role and delegates full operational deployment of the service to the TSM SDM (see section 2.2.1.2.3).

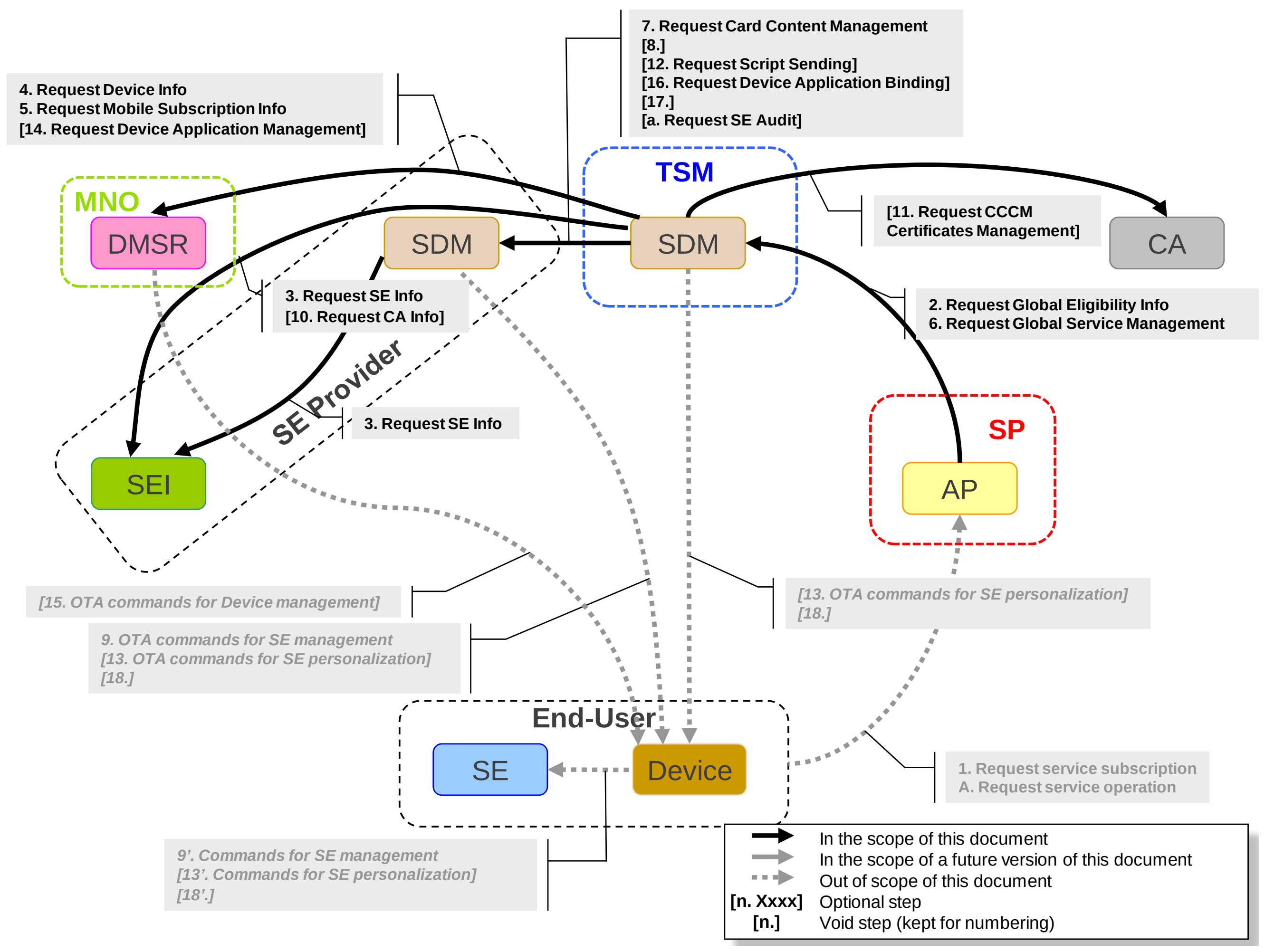

7. Request Card Content Management
[8.]
[12. Request Script Sending]
[16. Request Device Application Binding]
[17.]
[a. Request SE Audit]
4. Request Device Info
5. Request Mobile Subscription Info
[14. Request Device Application Management]
MNO
DMSR
SDM
TSM
SDM
[11. Request CCCM Certificates Management]
CA
3. Request SE Info
[10. Request CA Info]
2. Request Global Eligibility Info
6. Request Global Service Management
SE Provider
3. Request SE Info
SP
AP
SEI
[15. OTA commands for Device management]
[13. OTA commands for SE personalization]
[18.]
9. OTA commands for SE management
[13. OTA commands for SE personalization]
[18.]
End-User
SE
Device
1. Request service subscription
A. Request service operation
9'. Commands for SE management
[13'. Commands for SE personalization]
[18'.]
In the scope of this document
In the scope of a future version of this document
Out of scope of this document
[n. Xxxx] Optional step
[n.] Void step (kept for numbering)

Figure 2-10: Simple mode with ESE or SMC. Example of roles to actors allocation.

1. The end-user requests for the subscription to a mobile-NFC service ("Service subscription" functions group)

Phase 1:

2. The AP should first request for the global eligibility of the end-user mobile environment to the TSM SDM ("Global Eligibility Info" functions group).

To determine the global eligibility to the mobile-NFC service:

3. The TSM SDM requests information from the SEI regarding the Secure Element capabilities ("SE Info" functions group).
4. The TSM SDM also requests information from the DMSR regarding the Device capabilities ("Device Info" functions group).
5. Finally, the TSM SDM requests authorization and information to the DMSR regarding the Mobile Subscription ("Mobile Subscription Info" functions group) to be used to reach the SE and the Device.

6. If global eligibility is confirmed, the AP now delegates the deployment of the mobile-NFC service to the TSM SDM ("Global Service Management" functions group)

The TSM SDM starts the global mobile-NFC service deployment by declaring the deployment operation (α. "Service Life Cycle Notification" functions group). Then the TSM SDM may request SE and Device information again:

3. [On the Secure Element capabilities ("SE Info" functions group)],
4. [On the Device capabilities ("Device Info" functions group)]

Phase 2:

7. As the TSM SDM does not have sufficient privileges on the SE, the TSM SDM requests for CCM operations to the SE Provider SDM ("Card Content Management" functions group)
8. *[Intentionally void step to harmonize enumeration of steps between the different cases: not relevant in Simple mode]*
9. The SE Provider SDM may request information to the SEI regarding the Secure Element capabilities (3. "SE Info" functions group), then generates the CCM commands, and finally send them by itself to the Secure Element – through the Device ("OTA commands for SE management" and then "Commands for SE personalization").

Phase 3:

Security Domain personalization may require the involvement of the Controlling Authority, for confidential SD personalization:

10. [For such confidential SD personalization, the SDM requests the SEI to know which CA is present in the SE ("CA Info" functions group)…]
11. [The SDM requests the Controlling Authority to deliver a certificate ("CCCM Certificates Management" functions group)]

The TSM SDM or the AP generates the personalization commands for the SSD or for the NFC service application. Then, it sends the script OTA by itself or delegates the OTA operation to another SDM:

12. [If the TSM SDM does not have any OTA capability, it requests the OTA sending of the script to the SE Provider SDM ("Script Sending" functions group).]
13. [The personalization commands are sent OTA to the Secure Element ("OTA commands for SE personalization"), through the Device ("Commands for SE personalization").]

Phase 4:

Note: to be able to communicate with the Secure Element, it might be required to download a Device application. If so, steps 14 to 16 might be required to be performed in a prior phase.

14. [If a Device application is to be loaded, the SDM requests for the loading of the Device applications part of the service to the DMSR ("Device Application Management" functions group).]
15. [Commands are sent by the DMSR to the Device, for performing or just triggering the Device application loading ("OTA commands for Device management").]

16. [Then, if the Device application is to be bound to the SE application of the mobile-NFC service, the TSM SDM may request the SE Provider SDM to perform this binding ("Device Application Binding" functions group) if it does not have the capability and privileges to do it by itself.]
17. *[Intentionally void step to harmonize enumeration of steps between the different cases: not relevant for non UICC Secure Elements]*
18. *[Intentionally void step to harmonize enumeration of steps between the different cases: not relevant for non UICC Secure Elements]*

Phase 5:

At the end of the global service delivery, the TSM SDM notifies of the global status of the service (α. "Service Life Cycle Notification" functions group).

Note that at any stage of the process, the TSM SDM may request the SE Provider SDM for an audit of the content of the Secure Element to get the available free memory in the SE or to get the status of the Applications, Security Domains or Executable Load Files (a. "SE Audit" functions group).

The functions of the functions groups presented in the above diagram are listed in Table 3-1. The notification functions are listed in Table 3-2. These functions and notification functions are defined in section 3.

2.2.2.2.2 Example of Interactions in Delegated Mode Between TSM and SE Provider

In the "Delegated mode" relation between the TSM SDM and the SE Provider SDM, the TSM SDM generates the Card Content Management commands, under control of the SE Provider SDM that delivers Tokens required to grant the commands execution.

Either the TSM itself or the MNO can provide the OTA capability, depending on the availability of the OTA capability in the TSM SDM, and on the business agreement between TSM and MNO.

Legend used in the following message flow diagrams is detailed in the introduction part of section 2.2.2.

2.2.2.2.2.1 UICC

The following diagram presents the various interactions between the roles of the actors of the mobile-NFC ecosystem, in the following example context:

- The SE is a UICC, so the MNO has the SEI role,
- The Card Content Management Mode between the TSM SDM and the SE Provider SDM (see section 1.5.2.1) is the "Delegated mode", meaning that the MNO has the SDM role for Token generation, and the TSM has the SDM role for the CCM command generation and the Eligibility and Service Management.

- The mobile-NFC service is being deployed according to Service Deployment Mode #1 (see section 1.5.2.2). The Service Provider takes the AP role and delegates full operational deployment of the service to the TSM SDM (see section 2.2.1.2.3).

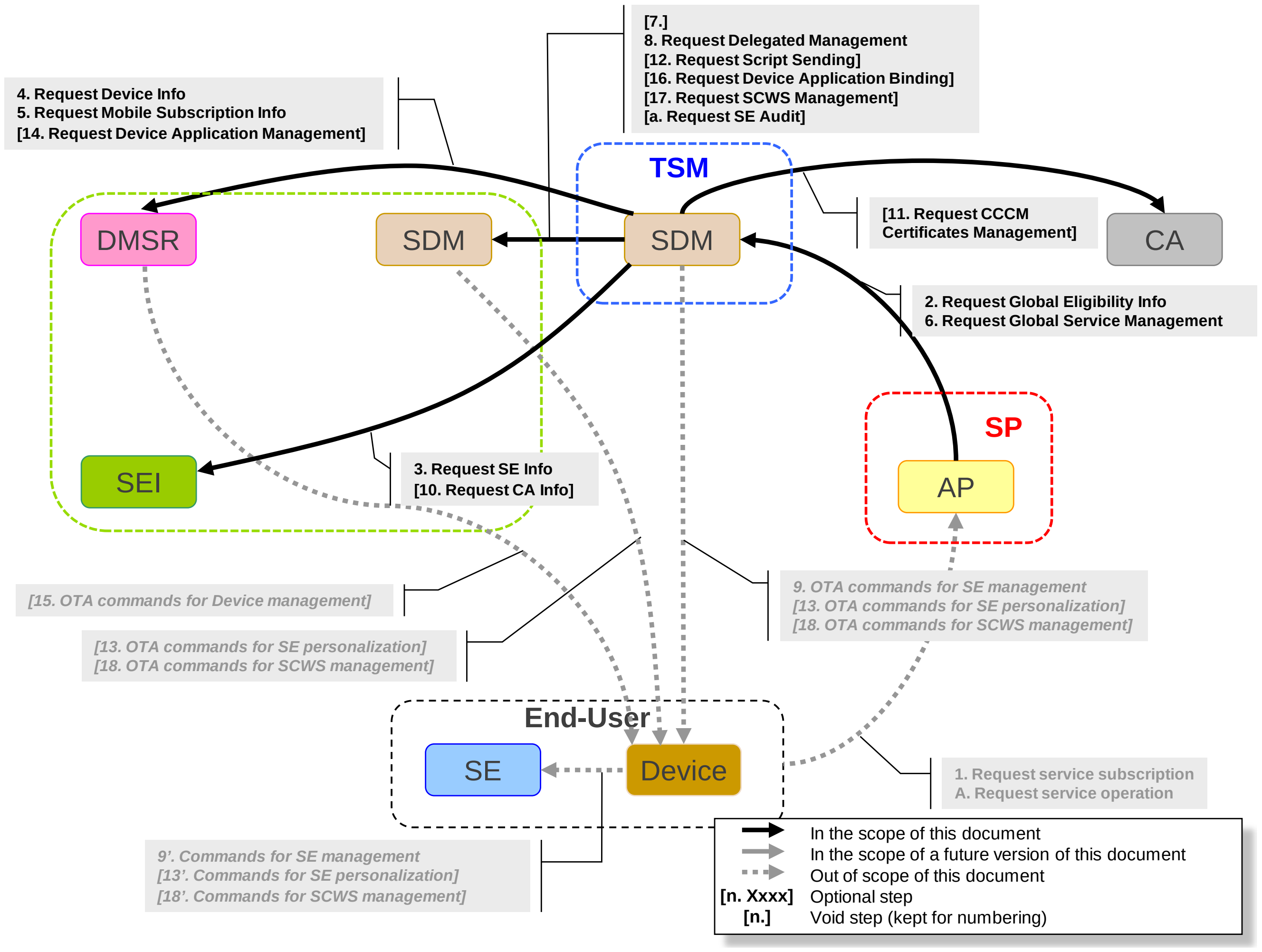

[7.]
8. Request Delegated Management
[12. Request Script Sending]
[16. Request Device Application Binding]
[17. Request SCWS Management]
[a. Request SE Audit]
4. Request Device Info
5. Request Mobile Subscription Info
[14. Request Device Application Management]
TSM
DMSR
SDM
SDM
[11. Request CCCM Certificates Management]
CA
2. Request Global Eligibility Info
6. Request Global Service Management
SP
AP
SEI
3. Request SE Info
[10. Request CA Info]
[15. OTA commands for Device management]
9. OTA commands for SE management
[13. OTA commands for SE personalization]
[18. OTA commands for SCWS management]
[13. OTA commands for SE personalization]
[18. OTA commands for SCWS management]
End-User
SE
Device
1. Request service subscription
A. Request service operation
9'. Commands for SE management
[13'. Commands for SE personalization]
[18'. Commands for SCWS management]
In the scope of this document
In the scope of a future version of this document
Out of scope of this document
[n. Xxxx] Optional step
[n.] Void step (kept for numbering)

Figure 2-11: Delegated mode with UICC. Example of roles to actors allocation.

1. The end-user requests for the subscription to a mobile-NFC service ("Service subscription" functions group)

Phase 1:

2. The AP should first request for the global eligibility of the end-user mobile environment to the TSM SDM ("Global Eligibility Info" functions group).

To determine the global eligibility to the mobile-NFC service:

3. The TSM SDM requests information from the SEI regarding the Secure Element capabilities ("SE Info" functions group).
4. The TSM SDM also requests information from the DMSR regarding the Device capabilities ("Device Info" functions group).

5. Finally, the TSM SDM requests authorization and information to the DMSR regarding the Mobile Subscription ("Mobile Subscription Info" functions group) to be used to reach the SE and the Device.

6. If global eligibility is confirmed, the AP now delegates the deployment of the mobile-NFC service to the TSM SDM ("Global Service Management" functions group)

The TSM SDM starts the global mobile-NFC service deployment by declaring the deployment operation (α. "Service Life Cycle Notification" functions group). Then the TSM SDM may request SE and Device information again:

3. [On the Secure Element capabilities ("SE Info" functions group)],
4. [On the Device capabilities ("Device Info" functions group)]

Phase 2:

7. *[Intentionally void step to harmonize enumeration of steps between the different cases: not relevant in Delegated mode]*
8. As the TSM SDM has the Delegated Management privilege, it generates the CCM commands and then asks the MNO SDM for the generation of the Token ("Delegate Management" functions group).]
9. The commands being generated are sent by the TSM SDM to the Secure Element through the Device ("OTA commands for SE management" and then "Commands for SE management").

Phase 3:

Security Domain personalization may require the involvement of the Controlling Authority, for confidential SD personalization:

10. [For such confidential SD personalization, the SDM requests the SEI to know which CA is present in the SE ("CA Info" functions group)…]
11. [The SDM requests the Controlling Authority to deliver a certificate ("CCCM Certificates Management" functions group)]

The TSM SDM or the AP generates the personalization commands for the SSD or for the NFC service application. Then, it sends the script OTA by itself or delegates the OTA operation to another SDM:

12. [If the TSM SDM does not have any OTA capability, it requests the OTA sending of the script to the MNO SDM ("Script Sending" functions group).]
13. [The personalization commands are sent OTA to the Secure Element ("OTA commands for SE personalization"), through the Device ("Commands for SE personalization").]

Phase 4:

Note: to be able to communicate with the Secure Element, it might be required to download a Device application. If so, steps 14 to 16 might be required to be performed in a prior phase.

14. [If a Device application is to be loaded, the SDM requests for the loading of the Device applications part of the service to the DMSR ("Device Application Management" functions group).]
15. [Commands are sent by the DMSR to the Device, for performing or just triggering the Device application loading ("OTA commands for Device management").]
16. [Then, if the Device application is to be bound to the SE application of the mobile-NFC service, the TSM SDM may request the MNO SDM to perform this binding ("Device Application Binding" functions group) if it does not have the capability and privileges to do it by itself.]
17. [If a SCWS Portal is to be loaded, and if the TSM SDM does not have the capability and privileges to do it by itself, the TSM SDM requests the MNO TSM for the loading of the SCWS portal of the service ("SCWS Management" functions group).]
18. [SCWS administration commands are sent by the MNO SDM or by the TSM SDM to the Secure Element, through the Device ("OTA commands for SCWS management" and then "Commands for SCWS management").]

Phase 5:

At the end of the global service delivery, the TSM SDM notifies of the global status of the service (α. "Service Life Cycle Notification" functions group).

Note that at any stage of the process, the TSM SDM may request the MNO SDM for an audit of the content of the Secure Element to get the available free memory in the SE or to get the status of the Applications, Security Domains or Executable Load Files (a. "SE Audit" functions group).

The functions of the functions groups presented in the above diagram are listed in Table 3-1. The notification functions are listed in Table 3-2. These functions and notification functions are defined in section 3.

2.2.2.2.2.2 ESE or SMC

The following diagram presents the various interactions between the roles of the actors of the mobile-NFC ecosystem, in the following example context:

- The SE is an ESE or a SMC and the MNO does not have the SEI role
- The Card Content Management Mode between the TSM SDM and the SE Provider SDM is the "Delegated mode" (see section 1.5.2.1), meaning that the SE Provider has the SDM role for Token generation, and the TSM has the SDM role for the CCM command generation and the Eligibility and Service Management.
- The mobile-NFC service is being deployed according to Service Deployment Mode #1 (see section 1.5.2.2). The Service Provider takes the AP role and delegates full operational deployment of the service to the TSM SDM (see section 2.2.1.2.3).

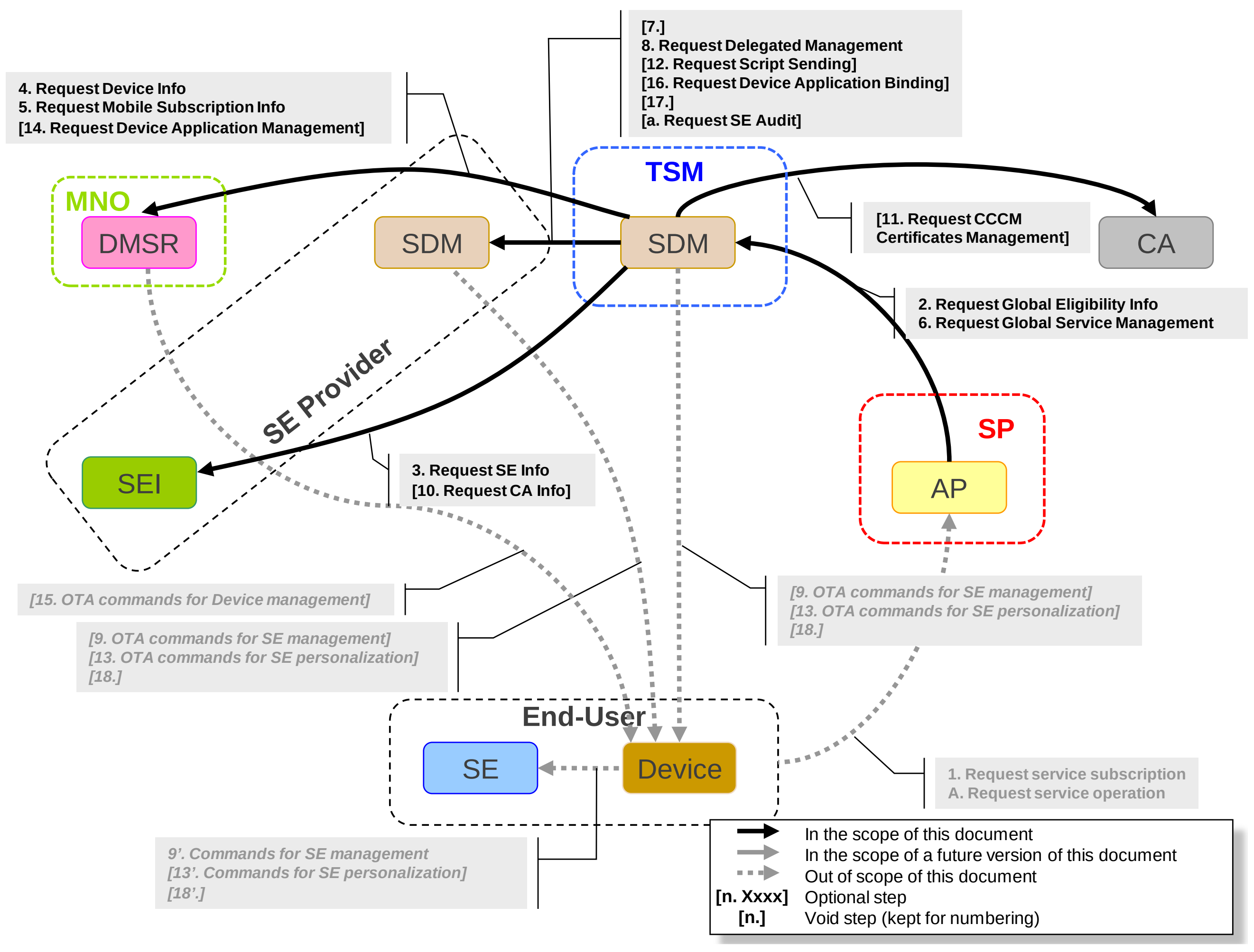

[7.]
8. Request Delegated Management
[12. Request Script Sending]
[16. Request Device Application Binding]
[17.]
[a. Request SE Audit]
4. Request Device Info
5. Request Mobile Subscription Info
[14. Request Device Application Management]
MNO
DMSR
SDM
TSM
SDM
CA
[11. Request CCCM Certificates Management]
2. Request Global Eligibility Info
6. Request Global Service Management
SE Provider
SEI
3. Request SE Info
[10. Request CA Info]
SP
AP
[15. OTA commands for Device management]
[9. OTA commands for SE management]
[13. OTA commands for SE personalization]
[18.]
[9. OTA commands for SE management]
[13. OTA commands for SE personalization]
[18.]
End-User
SE
Device
1. Request service subscription
A. Request service operation
9'. Commands for SE management
[13'. Commands for SE personalization]
[18'.]
In the scope of this document
In the scope of a future version of this document
Out of scope of this document
[n. Xxxx] Optional step
[n.] Void step (kept for numbering)

Figure 2-12: Delegated mode with ESE or SMC. Example of roles to actors allocation.

1. The end-user requests for the subscription to a mobile-NFC service ("Service subscription" functions group)

Phase 1:

2. The AP should first request for the global eligibility of the end-user mobile environment to the TSM SDM ("Global Eligibility Info" functions group).

To determine the global eligibility to the mobile-NFC service:

3. The TSM SDM requests information from the SEI regarding the Secure Element capabilities ("SE Info" functions group).
4. The TSM SDM also requests information from the DMSR regarding the Device capabilities ("Device Info" functions group).
5. Finally, the TSM SDM requests authorization and information to the DMSR regarding the Mobile Subscription ("Mobile Subscription Info" functions group) to be used to reach the SE and the Device.

6. If global eligibility is confirmed, the AP now delegates the deployment of the mobile-NFC service to the TSM SDM ("Global Service Management" functions group)

The TSM SDM starts the global mobile-NFC service deployment by declaring the deployment operation (α. "Service Life Cycle Notification" functions group). Then the TSM SDM may request SE and Device information again:

3. [On the Secure Element capabilities ("SE Info" functions group)],
4. [On the Device capabilities ("Device Info" functions group)]

Phase 2:

7. *[Intentionally void step to harmonize enumeration of steps between the different cases: not relevant in Delegated mode]*
8. As the TSM SDM has the Delegated Management privilege, it generates the CCM commands and then asks the SE Provider SDM for the generation of the Token ("Delegate Management" functions group).]
9. The commands being generated are sent by the TSM SDM to the Secure Element – through the Device ("OTA commands for SE management" and then "Commands for SE management"); or via the SE Provider SDM ("Script Sending" functions group).

Phase 3:

Security Domain personalization may require the involvement of the Controlling Authority, for confidential SD personalization:

10. [For such confidential SD personalization, the SDM requests the SEI to know which CA is present in the SE ("CA Info" functions group)…]
11. [The SDM requests the Controlling Authority to deliver a certificate ("CCCM Certificates Management" functions group)]

The TSM SDM or the AP generates the personalization commands for the SSD or for the NFC service application. Then, it sends the script OTA by itself or delegates the OTA operation to another SDM:

12. [If the TSM SDM does not have any OTA capability, it requests the OTA sending of the script to the SE Provider SDM (12. "Script Sending" functions group).]
13. [The personalization commands are sent OTA to the Secure Element ("OTA commands for SE personalization"), through the Device ("Commands for SE personalization").]

Phase 4:

Note: to be able to communicate with the Secure Element, it might be required to download a Device application. If so, steps 14 to 16 might be required to be performed in a prior phase.

14. [If a Device application is to be loaded, the SDM requests for the loading of the Device applications part of the service to the DMSR ("Device Application Management" functions group).]
15. [Commands are sent by the DMSR to the Device, for performing or just triggering the Device application loading ("OTA commands for Device management").]

16. [Then, if the Device application is to be bound to the SE application of the mobile-NFC service, the TSM SDM may request the SE Provider SDM to perform this binding ("Device Application Binding" functions group) if it does not have the capability and privileges to do it by itself.]
17. *[Intentionally void step to harmonize enumeration of steps between the different cases: not relevant for non UICC Secure Elements]*
18. *[Intentionally void step to harmonize enumeration of steps between the different cases: not relevant for non UICC Secure Elements]*

Phase 5:

At the end of the global service delivery, the TSM SDM notifies of the global status of the service (α. "Service Life Cycle Notification" functions group).

Note that at any stage of the process, the TSM SDM may request the SE Provider SDM for an audit of the content of the Secure Element to get the available free memory in the SE or to get the status of the Applications, Security Domains or Executable Load Files (a. "SE Audit" functions group).

The functions of the functions groups presented in the above diagram are listed in Table 3-1. The notification functions are listed in Table 3-2. These functions and notification functions are defined in section 3.

2.2.2.2.3 Example of Interactions in Dual Mode Between TSM and SE Provider

In the "Dual mode" relation between the TSM SDM and the SE Provider SDM, the TSM SDM has the full responsibility of its own area inside the Secure Element, and generates autonomously all the Card Content Management commands.

The OTA capability can either be provided by the TSM itself or thanks to the MNO, depending on the availability of the OTA capability in the TSM SDM, and on the business agreement between TSM and MNO.

Legend used in the following message flow diagrams is detailed in the introduction part of section 2.2.2.

2.2.2.2.3.1 UICC

The following diagram presents the various interactions between the roles of the actors of the mobile-NFC ecosystem, in the following example context:

- The SE is a UICC, so the MNO has the SEI role,
- The Card Content Management Mode between the TSM SDM and the SE Provider SDM is the "Dual mode" (see section 1.5.2.1), meaning that the TSM has the SDM role for the CCM command generation and for the Eligibility and Service Management.
- The mobile-NFC service is being deployed according to Service Deployment Mode #1 (see section 1.5.2.2). The Service Provider takes the AP role and delegates full operational deployment of the service to the TSM SDM (see section 2.2.1.2.3).

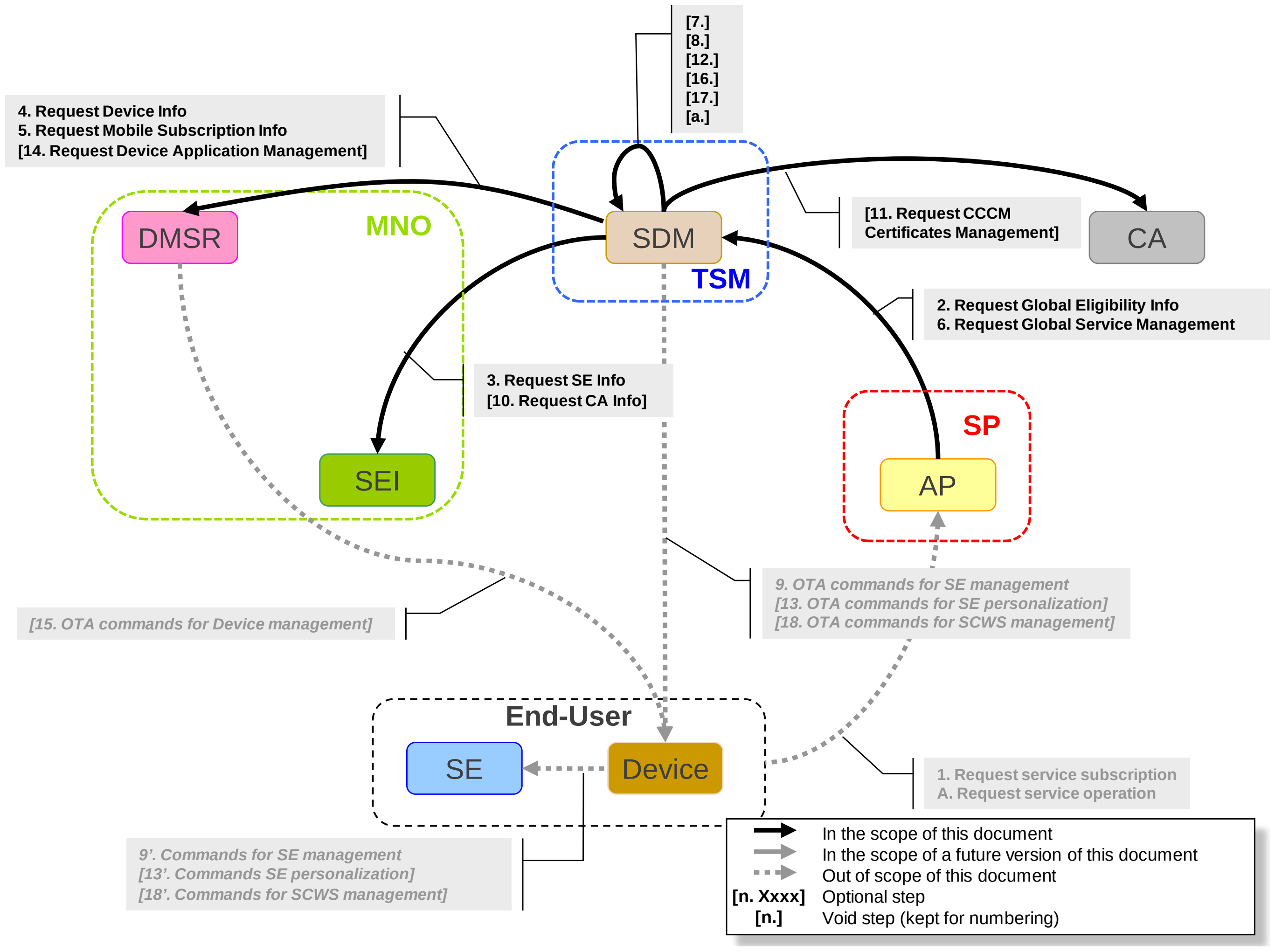

[7.]
[8.]
[12.]
[16.]
[17.]
[a.]
4. Request Device Info
5. Request Mobile Subscription Info
[14. Request Device Application Management]
DMSR
MNO
SDM
TSM
[11. Request CCCM Certificates Management]
CA
2. Request Global Eligibility Info
6. Request Global Service Management
3. Request SE Info
[10. Request CA Info]
SEI
SP
AP
9. OTA commands for SE management
[13. OTA commands for SE personalization]
[18. OTA commands for SCWS management]
[15. OTA commands for Device management]
End-User
SE
Device
1. Request service subscription
A. Request service operation
9'. Commands for SE management
[13'. Commands SE personalization]
[18'. Commands for SCWS management]
In the scope of this document
In the scope of a future version of this document
Out of scope of this document
[n. Xxxx] Optional step
[n.] Void step (kept for numbering)

Figure 2-13: Dual mode with UICC. Example of roles to actors allocation.

1. The end-user requests for the subscription to a mobile-NFC service ("Service subscription" functions group)

Phase 1:

2. The AP should first request for the global eligibility of the end-user mobile environment to the TSM SDM ("Global Eligibility Info" functions group).

To determine the global eligibility to the mobile-NFC service:

3. The TSM SDM requests information from the SEI regarding the Secure Element capabilities ("SE Info" functions group).
4. The TSM SDM also requests information from the DMSR regarding the Device capabilities ("Device Info" functions group).

5. Finally, the TSM SDM requests authorization and information to the DMSR regarding the Mobile Subscription ("Mobile Subscription Info" functions group) to be used to reach the SE and the Device.

6. If global eligibility is confirmed, the AP now delegates the deployment of the mobile-NFC service to the TSM SDM ("Global Service Management" functions group)

The TSM SDM starts the global mobile-NFC service deployment by declaring the deployment operation (α. "Service Life Cycle Notification" functions group). Then the TSM SDM may request SE and Device information again:

3. [On the Secure Element capabilities ("SE Info" functions group)],
4. [On the Device capabilities ("Device Info" functions group)]

Phase 2:

7. *[Intentionally void step to harmonize enumeration of steps between the different cases: not relevant for Dual mode]*
8. *[Intentionally void step to harmonize enumeration of steps between the different cases: not relevant for Dual mode]*
9. As the TSM SDM has the Authorized Management privilege, it generates the CCM commands and sends them to the Secure Element, through the Device ("OTA commands for SE management" and then "Commands for SE management").

Phase 3:

Security Domain personalization may require the involvement of the Controlling Authority, for confidential SD personalization:

10. [For such confidential SD personalization, the SDM requests the SEI to know which CA is present in the SE ("CA Info" functions group)…]
11. [The SDM requests the Controlling Authority to deliver a certificate ("CCCM Certificates Management" functions group)]

The TSM SDM or the AP generates the personalization commands for the SSD or for the NFC service application. Then, it sends the script OTA:

12. *[Intentionally void step to harmonize enumeration of steps between the different cases: internally managed by the SDM]*
13. [The personalization commands are sent OTA to the Secure Element ("OTA commands for SE personalization"), through the Device ("Commands for SE personalization").]

Phase 4:

Note: to be able to communicate with the Secure Element, it might be required to download a Device application. If so, steps 14 to 16 might be required to be performed in a prior phase.

14. [If a Device application is to be loaded, the SDM requests for the loading of the Device applications part of the service to the DMSR ("Device Application Management" functions group).]
15. [Commands are sent by the DMSR to the Device, for performing or just triggering the Device application loading ("OTA commands for Device management").]
16. *[Intentionally void step to harmonize enumeration of steps between the different cases: internally managed by the SDM]*
17. *[Intentionally void step to harmonize enumeration of steps between the different cases: internally managed by the SDM]*
18. [SCWS administration commands are sent by the TSM SDM to the Secure Element, through the Device ("OTA commands for SCWS management" and then "Commands for SCWS management").]

Phase 5:

At the end of the global service delivery, the TSM SDM notifies of the global status of the service (α. "Service Life Cycle Notification" functions group).

The functions of the functions groups presented in the above diagram are listed in Table 3-1. The notification functions are listed in Table 3-2. These functions and notification functions are defined in section 3.

2.2.2.2.3.2 ESE or SMC (for Service Deployment Mode #1)

The following diagram presents the various interactions between the roles of the actors of the mobile-NFC ecosystem, in the following example context:

- The SE is an ESE or a SMC and the MNO has not the SEI role
- The Card Content Management Mode between the TSM SDM and the SE Provider SDM is the "dual mode" (see section 1.5.2.1), meaning that the TSM has the SDM role for the CCM command generation and the Eligibility and Service Management.
- The mobile-NFC service is being deployed according to Service Deployment Mode #1 (see section 1.5.2.2). The Service Provider takes the AP role and delegates full operational deployment of the service to the TSM SDM (see section 2.2.1.2.3).

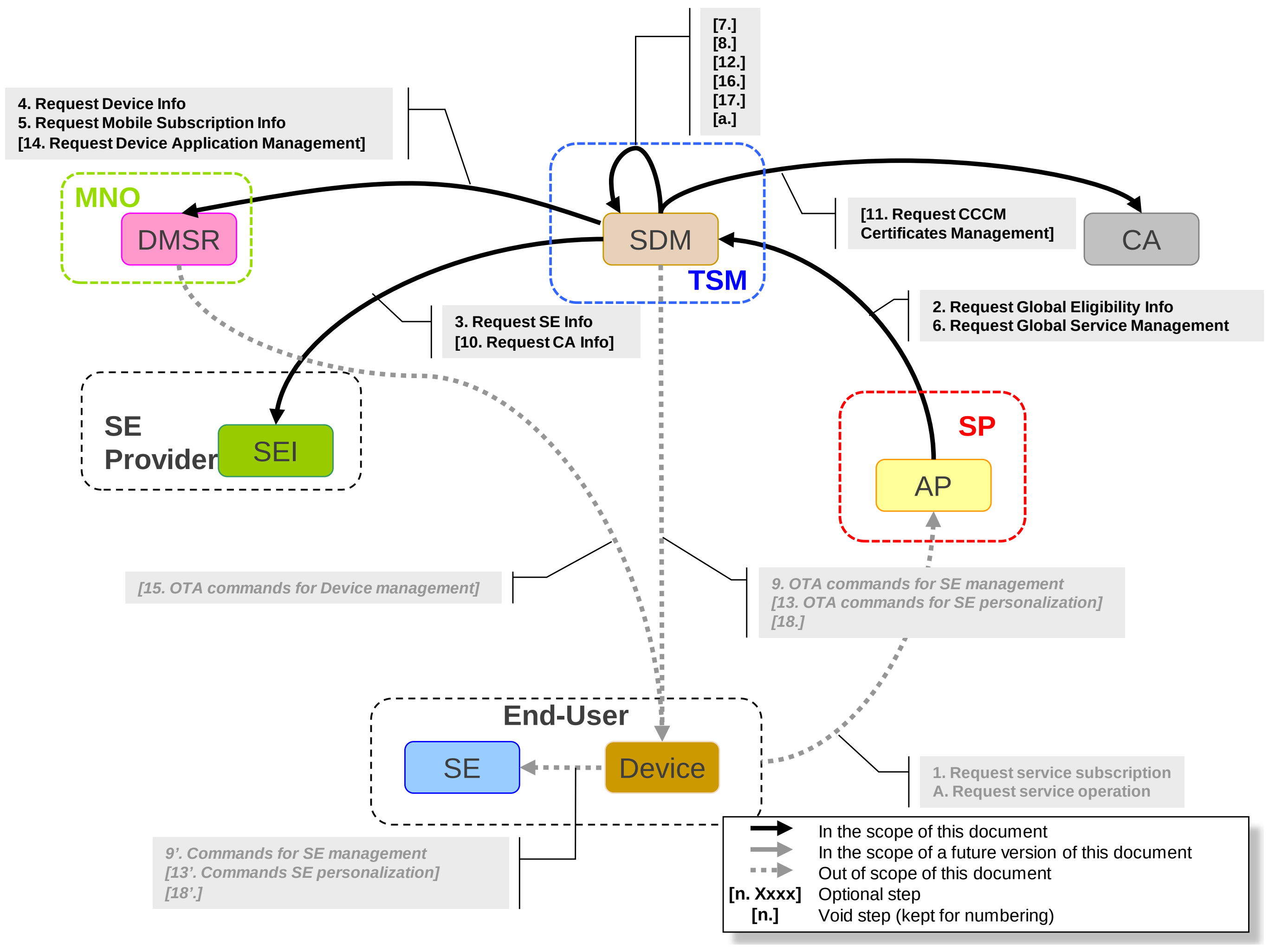

[7.]
[8.]
[12.]
[16.]
[17.]
[a.]
4. Request Device Info
5. Request Mobile Subscription Info
[14. Request Device Application Management]
MNO
DMSR
SDM
TSM
[11. Request CCCM Certificates Management]
CA
3. Request SE Info
[10. Request CA Info]
2. Request Global Eligibility Info
6. Request Global Service Management
SE Provider
SEI
SP
AP
[15. OTA commands for Device management]
9. OTA commands for SE management
[13. OTA commands for SE personalization]
[18.]
End-User
SE
Device
1. Request service subscription
A. Request service operation
9'. Commands for SE management
[13'. Commands SE personalization]
[18'.]
In the scope of this document
In the scope of a future version of this document
Out of scope of this document
[n. Xxxx] Optional step
[n.] Void step (kept for numbering)

Figure 2-14: Dual mode with ESE or SMC (Service Deployment Mode #1). Example of roles to actors allocation.

1. The end-user requests for the subscription to a mobile-NFC service ("Service subscription" functions group)

Phase 1:

2. The AP should first request for the global eligibility of the end-user mobile environment to the TSM SDM ("Global Eligibility Info" functions group).

To determine the global eligibility to the mobile-NFC service:

3. The TSM SDM requests information from the SEI regarding the Secure Element capabilities ("SE Info" functions group).
4. The TSM SDM also requests information from the DMSR regarding the Device capabilities ("Device Info" functions group).
5. Finally, the TSM SDM requests authorization and information to the DMSR regarding the Mobile Subscription ("Mobile Subscription Info" functions group) to be used to reach the SE and the Device.

6. If global eligibility is confirmed, the AP now delegates the deployment of the mobile-NFC service to the TSM SDM ("Global Service Management" functions group)

The TSM SDM starts the global mobile-NFC service deployment by declaring the deployment operation (α. "Service Life Cycle Notification" functions group). Then the TSM SDM may request SE and Device information again:

3. [On the Secure Element capabilities ("SE Info" functions group)],
4. [On the Device capabilities ("Device Info" functions group)]

Phase 2:

7. *[Intentionally void step to harmonize enumeration of steps between the different cases: not relevant for Dual mode]*
8. *[Intentionally void step to harmonize enumeration of steps between the different cases: not relevant for Dual mode]*
9. As the TSM SDM has the Authorized Management privilege, it generates the CCM commands and sends them by itself to the Secure Element – through the Device ("OTA commands for SE management" and then "Commands for SE management").

Phase 3:

Security Domain personalization may require the involvement of the Controlling Authority, for confidential SD personalization:

10. [For such confidential SD personalization, the SDM requests the SEI to know which CA is present in the SE ("CA Info" functions group)…]
11. [The SDM requests the Controlling Authority to deliver a certificate ("CCCM Certificates Management" functions group)]

The TSM SDM or the AP generates the personalization commands for the SSD or for the NFC service application. Then, it sends the script OTA:

12. *[Intentionally void step to harmonize enumeration of steps between the different cases: internally managed by the SDM]*
13. [The personalization commands are sent OTA to the Secure Element ("OTA commands for SE personalization"), through the Device ("Commands for SE personalization").]

Phase 4:

Note: to be able to communicate with the Secure Element, it might be required to download a Device application. If so, steps 14 to 16 might be required to be performed in a prior phase.

14. [If a Device application is to be loaded, the SDM requests for the loading of the Device applications part of the service to the DMSR ("Device Application Management" functions group).]
15. [Commands are sent by the DMSR to the Device, for performing or just triggering the Device application loading ("OTA commands for Device management").]

16. *[Intentionally void step to harmonize enumeration of steps between the different cases: internally managed by the SDM]*
17. *[Intentionally void step to harmonize enumeration of steps between the different cases: not relevant for non UICC Secure Elements]*
18. *[Intentionally void step to harmonize enumeration of steps between the different cases: not relevant for non UICC Secure Elements]*

Phase 5:

At the end of the global service delivery, the TSM SDM notifies about the global status of the service (α. "Service Life Cycle Notification" functions group).

The functions of the functions groups presented in the above diagram are listed in Table 3-1. The notification functions are listed in Table 3-2. These functions and notification functions are defined in section 3.

2.2.2.3 Example of Interactions for Other Service Deployment Modes

The following sub-sections give examples of roles to actors allocation in the context of other Service Deployment Modes:

- Section 2.2.2.3.1 presents a possible ecosystem in Service Deployment Mode #2,
- Section 2.2.2.3.2 presents a possible ecosystem in Service Deployment Mode #3.

The specificities of those modes are that several different actors perform Card Content Management, for some different steps of the service deployment. There can thus be several one-to-one CCM relations, each of them potentially following a different Card Content Management Mode.

<u>Legend used in the following message flow diagrams is detailed in the introduction part of section 2.2.2.</u>

2.2.2.3.1 Service Deployment Mode #2, for ESE or SMC

The following diagram presents the various interactions between the roles of the actors of the mobile-NFC ecosystem, in the following example context:

- The SE is an ESE or a SMC and the MNO has not the SEI role
- The mobile-NFC service is being deployed according to Service Deployment Mode #2 (see section 1.5.2.2).
    - In complement to the AP role, the Service Provider takes the SDM role for service personalization (see section 2.2.1.2.3): the Service Provider SDM then directly requests for SD creation to the TSM SDM, and then requests for the personalization script sending to the same TSM SDM.

- The Service Provider devolves the entire service delivery, including Eligibility and Service Management, and single CCM to the TSM SDM (see section 2.2.1.2.3): the service deployment (excluding the SD creation and SD personalization) is then realized in Dual mode relation between the TSM SDM and the SE Provider SDM (see section 1.5.2.1).

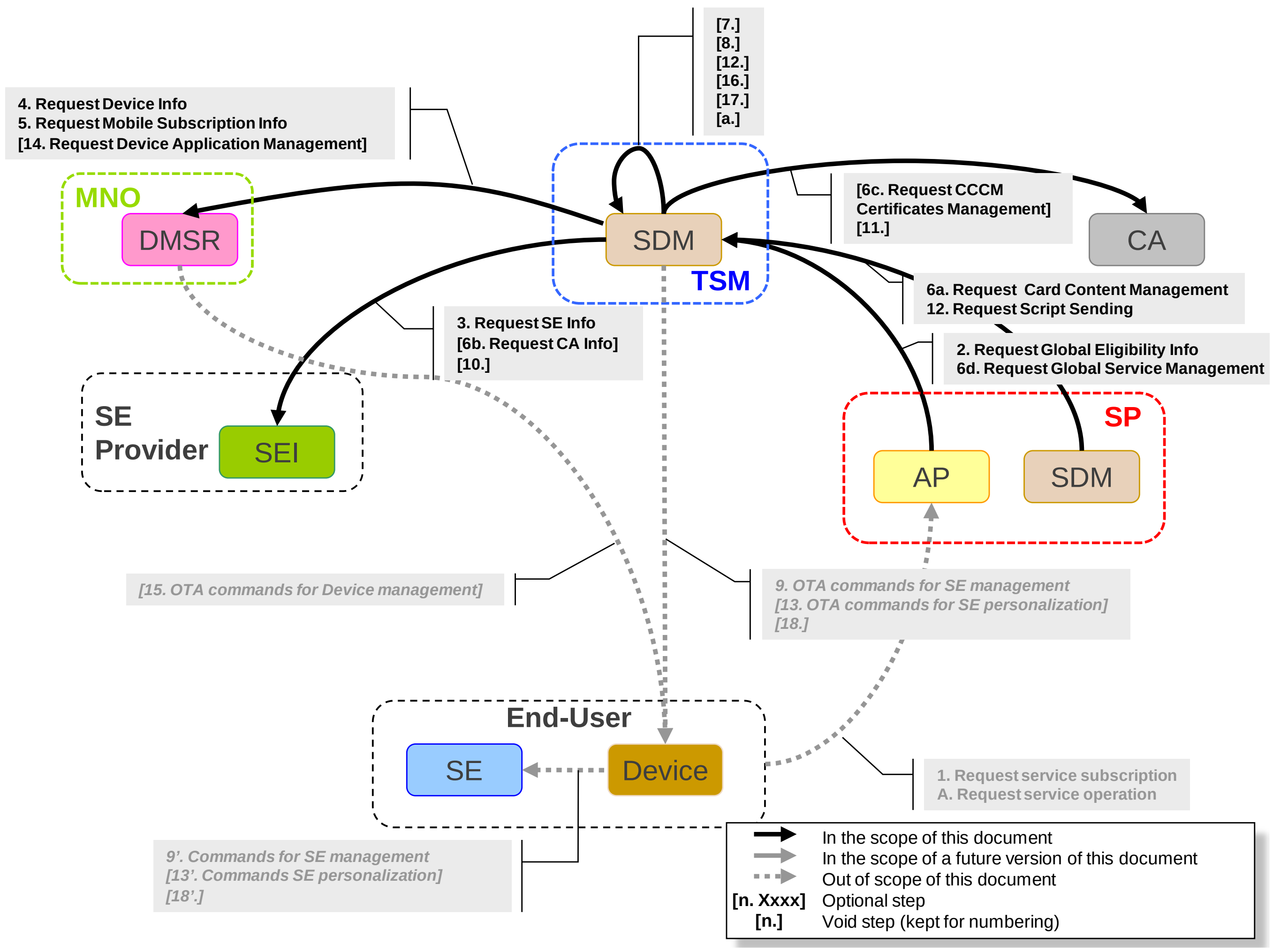

[7.]
[8.]
[12.]
[16.]
[17.]
[a.]
4. Request Device Info
5. Request Mobile Subscription Info
[14. Request Device Application Management]
MNO
DMSR
SDM
TSM
[6c. Request CCCM Certificates Management]
[11.]
CA
6a. Request Card Content Management
12. Request Script Sending
3. Request SE Info
[6b. Request CA Info]
[10.]
2. Request Global Eligibility Info
6d. Request Global Service Management
SE Provider
SEI
SP
AP
SDM
[15. OTA commands for Device management]
9. OTA commands for SE management
[13. OTA commands for SE personalization]
[18.]
End-User
SE
Device
1. Request service subscription
A. Request service operation
9'. Commands for SE management
[13'. Commands SE personalization]
[18'.]
In the scope of this document
In the scope of a future version of this document
Out of scope of this document
[n. Xxxx] Optional step
[n.] Void step (kept for numbering)

Figure 2-15: Dual mode with ESE or SMC (Service Deployment Mode #2). Example of roles to actors allocation.

1. The end-user requests for the subscription to a mobile-NFC service ("Service subscription" functions group)

Phase 1:

2. The AP should first request for the global eligibility of the end-user mobile environment to the TSM SDM ("Global Eligibility Info" functions group).

To determine the global eligibility to the mobile-NFC service:

3. The TSM SDM requests information from the SEI regarding the Secure Element capabilities ("SE Info" functions group).

4. The TSM SDM also requests information from the DMSR regarding the Device capabilities ("Device Info" functions group).
5. Finally, the TSM SDM requests authorization and information to the DMSR regarding the Mobile Subscription ("Mobile Subscription Info" functions group) to be used to reach the SE and the Device.

6. If global eligibility is confirmed:
   a. AP SDM request for the creation and personalization of its Security Domain to the TSM SDM ("Card Content Management " functions group)

   Security Domain personalization may require the involvement of the Controlling Authority, for confidential SD personalization:

   b. [For such confidential SD personalization, the SDM requests the SEI to know which CA is present in the SE ("CA Info" functions group)…]
   c. [The SDM requests the Controlling Authority to deliver a certificate ("CCCM Certificates Management" functions group)]

   Finally:

   d. AP then delegates the deployment of the mobile-NFC service in this Security Domain to the TSM SDM ("Global Service Management" functions group)

The TSM SDM starts the global mobile-NFC service deployment by declaring the deployment operation (α. "Service Life Cycle Notification" functions group). Then the TSM SDM may request SE and Device information again:

3. [On the Secure Element capabilities ("SE Info" functions group)],
4. [On the Device capabilities ("Device Info" functions group)]

Phase 2:

7. *[Intentionally void step to harmonize enumeration of steps between the different cases: not relevant for Dual mode]*
8. *[Intentionally void step to harmonize enumeration of steps between the different cases: not relevant for Dual mode]*
9. As the TSM SDM has the Authorized Management privilege, it generates the CCM commands and sends them by itself to the Secure Element – through the Device ("OTA commands for SE management" and then "Commands for SE management").

Phase 3:

10. *[Intentionally void step to harmonize enumeration of steps between the different cases] – Refer to item 6b.*
11. *[Intentionally void step to harmonize enumeration of steps between the different cases] – Refer to item 6c.*

The SP SDM generates the personalization commands for the SSD or for the NFC service application. Then, the SP SDM delegates the OTA operation to another SDM:

12. The SP SDM requests the OTA sending of the personalization script to the TSM SDM ("Script Sending" functions group).
13. [The personalization commands are sent OTA to the Secure Element ("OTA commands for SE personalization"), through the Device ("Commands for SE personalization").]

Phase 4:

14. [If a Device application is to be loaded, the SDM requests for the loading of the Device applications part of the service to the DMSR ("Device Application Management" functions group).]
15. [Commands are sent by the DMSR to the Device, for performing or just triggering the Device application loading ("OTA commands for Device management").]
16. *[Intentionally void step to harmonize enumeration of steps between the different cases: internally managed by the SDM]*
17. *[Intentionally void step to harmonize enumeration of steps between the different cases: not relevant for non UICC Secure Elements]*
18. *[Intentionally void step to harmonize enumeration of steps between the different cases: not relevant for non UICC Secure Elements]*

Phase 5:

At the end of the global service delivery, the TSM SDM notifies about the global status of the service (α. "Service Life Cycle Notification" functions group).

The functions of the functions groups presented in the above diagram are listed in Table 3-1. The notification functions are listed in Table 3-2. These functions and notification functions are defined in section 3.

2.2.2.3.2 Service Deployment Mode #3, for ESE or SMC

The following diagram presents the various interactions between the roles of the actors of the mobile-NFC ecosystem, in the following example context:

- The SE is an ESE or a SMC and the MNO has not the SEI role
- The mobile-NFC service is being deployed according to Service Deployment Mode #3 (see section 1.5.2.2).
    - In complement to the AP role, the Service Provider takes the SDM role for Eligibility and Service Management, as well as service personalization. The Service Provider delegates the single CCM operation to the TSM SDM (see section 2.2.1.2.3).
    - The relation between the TSM SDM and the SE Provider SDM may be any of the CCM modes (see section 1.5.2.1):
        - Either Delegated mode relation, the TSM SD having the Delegated Management privilege.

- Or Dual mode relation, the TSM SD having the Authorized Management privilege.
- Or a Simple mode relation, the TSM SD having no CCM privilege.

In the example below, it is assumed that the relation between the TSM SDM and the SE Provider SDM is in Dual mode.

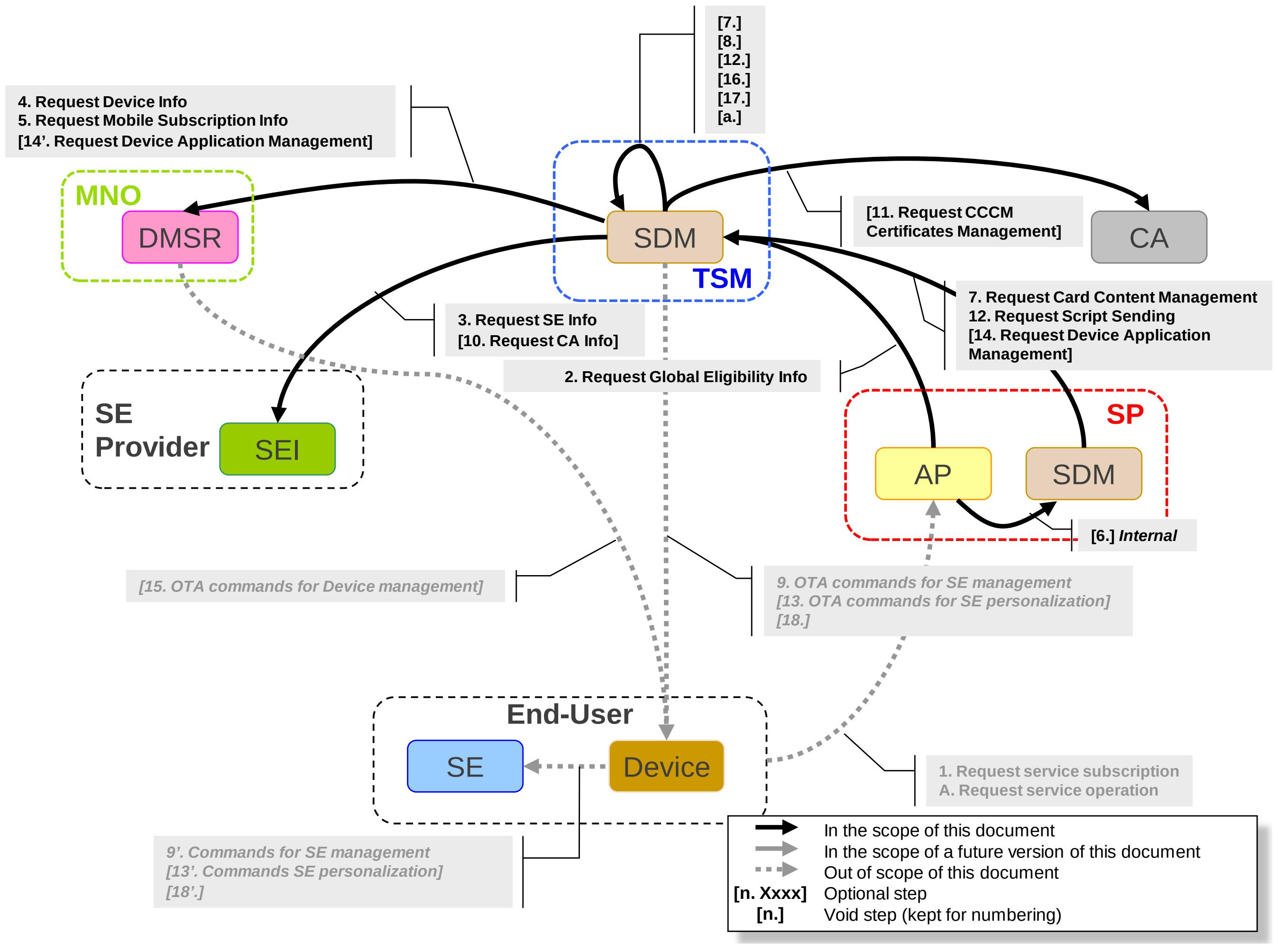

[7.]
[8.]
[12.]
[16.]
[17.]
[a.]
4. Request Device Info
5. Request Mobile Subscription Info
[14'. Request Device Application Management]
MNO
DMSR
SDM
TSM
[11. Request CCCM Certificates Management]
CA
3. Request SE Info
[10. Request CA Info]
7. Request Card Content Management
12. Request Script Sending
[14. Request Device Application Management]
2. Request Global Eligibility Info
SE Provider
SEI
SP
AP
SDM
[6.] Internal
[15. OTA commands for Device management]
9. OTA commands for SE management
[13. OTA commands for SE personalization]
[18.]
End-User
SE
Device
1. Request service subscription
A. Request service operation
9'. Commands for SE management
[13'. Commands SE personalization]
[18'.]
In the scope of this document
In the scope of a future version of this document
Out of scope of this document
[n. Xxxx] Optional step
[n.] Void step (kept for numbering)

Figure 2-16: Dual mode with ESE or SMC (Service Deployment Mode #3). Example of roles to actors allocation.

1. The end-user requests for the subscription to a mobile-NFC service ("Service subscription" functions group)

Phase 1:

2. The AP should first request for the global eligibility of the end-user mobile environment to the TSM SDM ("Global Eligibility Info" functions group).

To determine the global eligibility to the mobile-NFC service:

3. The TSM SDM requests information from the SEI regarding the Secure Element capabilities ("SE Info" functions group).
4. The TSM SDM also requests information from the DMSR regarding the Device capabilities ("Device Info" functions group).
5. Finally, the TSM SDM requests authorization and information to the DMSR regarding the Mobile Subscription ("Mobile Subscription Info" functions group) to be used to reach the SE and the Device.

6. If global eligibility is confirmed, the AP now internally requests the deployment of the mobile-NFC service to the SP SDM. (Since this call is internal, the "Global Service Management" functions group is not mandated.)

The SP SDM starts the global mobile-NFC service deployment by declaring the deployment operation (α. "Service Life Cycle Notification" functions group).

Phase 2:

The SP SDM does not have sufficient privileges on the SE and requests individual CCM from the TSM SDM having the required privileges:

7. The SP SDM does not have sufficient privileges on the SE, the SDM requests for CCM operations to the TSM SDM ("Card Content Management" functions group)].
8. *[Intentionally void step to harmonize enumeration of steps between the different cases: not relevant for Dual mode]*
9. As the TSM SDM has the Authorized Management privilege, it generates the CCM commands and sends them by itself to the Secure Element – through the Device ("OTA commands for SE management" and then "Commands for SE management").

Phase 3:

Security Domain personalization may require the involvement of the Controlling Authority, for confidential SD personalization:

10. [For such confidential SD personalization, the SDM requests the SEI to know which CA is present in the SE ("CA Info" functions group)…]
11. [The SDM requests the Controlling Authority to deliver a certificate ("CCCM Certificates Management" functions group)]

The SP SDM generates the personalization commands for the SSD or for the NFC service application. Then, the SP SDM delegates the OTA operation to another SDM:

12. The SP SDM requests the OTA sending of the personalization script to the TSM SDM ("Script Sending" functions group).
13. [The personalization commands are sent OTA to the Secure Element ("OTA commands for SE personalization"), through the Device ("Commands for SE personalization").]

Phase 4:

14. [If a Device application is to be loaded, the SP SDM requests for the loading of the Device applications part of the service to the TSM SDM, which forwards the call to the DMSR ("Device Application Management" functions group).]
15. [Commands are sent by the DMSR to the Device, for performing or just triggering the Device application loading ("OTA commands for Device management").]
16. *[Intentionally void step to harmonize enumeration of steps between the different cases: internally managed by the SDM]*
17. *[Intentionally void step to harmonize enumeration of steps between the different cases: not relevant for non UICC Secure Elements]*
18. *[Intentionally void step to harmonize enumeration of steps between the different cases: not relevant for non UICC Secure Elements]*

Phase 5:

At the end of the global service delivery, the SP SDM notifies about the global status of the service (α. "Service Life Cycle Notification" functions group).

The functions of the functions groups presented in the above diagram are listed in Table 3-1. The notification functions are listed in Table 3-2. These functions and notification functions are defined in section 3.

3 Functions Description (normative)

This section provides the functional definition of the interactions identified in section 2. These definitions represent role-to-role interactions and are consequently agnostic to:

- The role repartition between actors: as mentioned in section 2.2.1.2, these is not a unique role to actor allocation, and this is a deployment choice and business agreement that will define this allocation.
- The structure of the messages used for these interactions and the transport protocol used between the various actors to transport those messages.

Note however that section 4 of this document provides a mapping of the functions to specific messages, and section 5 a binding to Web Services (SOAP) over HTTP.

The following table presents the normative list of all the functions that are defined in this section. Functions are grouped into functions groups. Each functions group is provided by a unique role and corresponds to an autonomous and consistent functionality.

When a functions group is implemented by a role, all the functions associated to this functions group SHALL be implemented by this role. In other words, functions groups cannot be partially implemented; if a special function is requested, then all the functions of the corresponding functions group SHALL be implemented.

Request-response functions:

Functions group	Involved functions	Function provider role
Global Eligibility Info	• `CheckGlobalEligibility` (see section 3.2.1.1)	SDM
Global Service Management	• `LookupServiceInstanceReference` (see section 3.3.2.1) • `DeclareServiceInstanceReference` (see section 3.3.2.2) • `GetServiceInstanceReferenceDescriptor` (see section 3.3.2.3) • `GetServiceState` (see section 3.3.2.4) • `DeployService` (see section 3.3.2.5) • `UpgradeService` (see section 3.3.2.6) • `ExchangeServiceData` (see section 3.3.2.7) • `SuspendOrResumeService` (see section 3.3.2.8) • `TerminateService` (see section 3.3.2.9)	SDM
Card Content Management	• `SECommandsGenerationAndRemoteExecution` (see section 3.5.1.2)	SDM
Delegated Management	• `GenerateDMToken` (see section 3.5.2.1) • `VerifyDMReceipt` (see section 3.5.2.2)	SDM

Functions group	Involved functions	Function provider role
Script Sending	• `BeginConversation` (see section 3.5.3.1) • `SendScript` (see section 3.5.3.2) • `EndConversation` (see section 3.5.3.3)	SDM
SE Info	• `GetSECapabilityProfileId` (see section 3.2.1.3) • `GetSEMobileSubscriptionIdentifier` (see section 3.2.4.2)	SEI
SE Audit	• `GetApplicationOrELFStatus` (see section 3.5.4.1) • `GetSDFreeMemory` (see section 3.5.4.2)	SDM
CA Info	• `AuditCAInformation` (see section 3.4.1)	SEI
Device Info	• `GetDeviceCapabilityProfileId` (see section 3.2.1.2)	DMSR
Mobile Subscription Info	• `CheckMobileSubscriptionEligibility` (see section 3.2.1.4) • `GetMobileSubscriptionAlternateIdentifier` (see section 3.2.4.1) • `GetMobileSubscriptionSEIdentifiers` (see section 3.2.4.3)	DMSR
Device Application Management	• `LoadDeviceApplication` (see section 3.6.1) • `DeleteDeviceApplication` (see section 3.6.3)	DMSR
Device Application Binding	• `BindDeviceApplicationToSEApplication` (see section 3.6.2) • `UnbindDeviceApplicationToSEApplication` (see section 3.6.4)	SDM
SCWS Management	• `LoadSCWSServicePortal` (see section 3.5.5.1) • `DeleteSCWSServicePortal` (see section 3.5.5.2)	SDM
CCCM Certificates Management	• `EnrollSSDOwnerCertificate` (see section 3.4.2)	CA

Table 3-1: Functions of the different functions groups, and associated provider roles

Notification handler functions:

Functions group	Notification handler functions	Role providing the notification handler
Service Environment Change Notification	• `HandleServiceEnvironmentChangeNotification` (see section 3.3.3.1) • `HandleActionDoneOnServiceNotification` (see section 3.3.3.2)	AP

Functions group	Notification handler functions	Role providing the notification handler
Service Life Cycle Notification	• `HandleStartServiceStateChangeNotification` (see section 3.2.3.1) • `HandleEndServiceStateChangeNotification` (see section 3.2.3.2)	SDM
Service Life Cycle Notification (on SE)	• `HandleStartServiceStateChangeNotification` (see section 3.2.3.1) • `HandleEndServiceStateChangeNotification` (see section 3.2.3.2)	SEI
Service Life Cycle Notification (on Mobile Subscription)	• `HandleStartServiceStateChangeNotification` (see section 3.2.3.1) • `HandleEndServiceStateChangeNotification` (see section 3.2.3.2)	DMSR
Mobile Subscription Life Cycle Notification	• `HandleMobileSubscriptionIdentifierChangedNotification` (see section 3.2.4.4) • `HandleMobileSubscriptionStatusChangeNotification` (see section 3.2.4.5)	SDM
SE Life Cycle Notification	• `HandleSERenewalNotification` (see section 3.2.5.1) • `HandleSEDeviceChangedNotification` (see section 3.2.5.2) • `HandleSEDeviceStatusChangeNotification` (see section 3.2.5.3) • `HandleSEMobileSubscriptionChangedNotification` (see section 3.2.5.4) • `HandleSEStatusChangeNotification` (see section 3.2.5.5)	SDM
Device Application Life Cycle Notification	• `HandleActionDoneOnDeviceApplicationNotification` (see section 3.6.5)	SDM, DMSR

Table 3-2: Notification handler functions of the different functions groups, and associated notification handler roles

For each deployment, the actors of the mobile-NFC ecosystem will decide to play particular roles (one or several of them), and to implement particular features of these roles.

If it is not required for a given deployment, an actor might decide not to implement all the behavior of a role, i.e. to implement just some functions of this role.

If an actor plays several roles for this given deployment, this actor might decide not to implement functions that are in-between two of the roles it is playing.

In fact an actor will implement one or several functions groups as defined above, meaning that it will implement all the functions defined in this group.

As a prerequisite of each deployment, depending on the ecosystem and on the deployment context, the different actors shall define which role is played by which actor and which functions group is implemented by each role.

The following section provides a concrete example of how the different functions groups can be dispatched between the different actors.

In this example we have an ecosystem with 2 actors: a MNO and a TSM. The responsibilities between these 2 actors are split as follow:

- The MNO is the Secure Element Issuer. It has chosen the Delegated mode for the Card Content Management operations, and so it will play the SDM role. It will also rely on the Smart Card Web Server technology for the user interfaces of the mobile-NFC services.
- The TSM is responsible for global delivery of the mobile-NFC services. It will use the Delegated mode capability of the MNO and will perform the OTA operations (for Card Content Management as well as for personalization of the mobile-NFC services) with its own OTA capability.

In this context, the actors decide that:

- On the MNO side:
    - The MNO shall implement the "Delegated Management" functions group, to manage the Delegated mode.
    - As there is a wide range of Secure Elements and of NFC handsets deployed on the field, it is required that the MNO implements the "SE Info" functions group.
    - However, as the MNO has not imposed any constraint on the Mobile Subscription (no eligibility criteria due to Mobile Subscription, no alternate mobile subscription, etc.), the "Mobile Subscription Info" functions group will not be implemented.
    - For SCWS portal management, the MNO will also implement the "SCWS Management" functions group as it will be the only administrator of the SCWS.
- On the TSM side:
    - As Delegated mode is used, the TSM is client of the "Delegated Management" functions group provided by the MNO. He knows that this functions group is fully implemented by the MNO, as mandated by GlobalPlatform. It can then manage the DM Receipts.
    - As it has its own OTA capability, it neither needs the help of the Service Providers, nor of the MNO. The "Script Sending" functions group is then not required to be implemented, neither by the MNO nor by the TSM.

Additionally, if the UI of new mobile-NFC services is based on a Device application, then:

- If the TSM does not have its own Device Management server, then the MNO will also play the DMSR role, providing the "Device Application Management" functions group.
- Optionally, for the Devices that supports a particular binding technology between Device applications and SE applications, the MNO might also provide the "Device Application Binding" functions group.

Now imagine that another TSM actor is entering into the ecosystem. This TSM is however not autonomous and shall rely on the Card Content Management capability of the MNO (i.e. Simple mode) and on the MNO OTA capability for mobile-NFC services personalization. As a consequence:

- The MNO shall implement the "Card Content Management" functions group, for Simple mode management. The new TSM will be client of this "Card Content Management" functions group.
- The MNO shall also implement the "Script Sending" functions group, to send the personalization scripts to the Secure Elements.

If it is finally required to create and personalize supplementary Security Domains in a confidential way (on behalf of the new TSM), then:

- The CA will be a new actor of the ecosystem, and it shall implement the "CCCM Certificates Management" functions group, that will be used by the new TSM
- The MNO, as Secure Element Issuer, shall also implement the "CA Info" functions group.
- When it is required to update the initial key values, the new TSM will rely on the "Script Sending" capability of the MNO.

3.1 Functions Commonalities

Functions represent entry points that are provided by roles (function provider), and that can be called by other roles (function requester).

Request-response functions:

A function is to be seen as a simple request-response data exchange between roles. It may take input data and provide output data. A function may also deliver no output data.

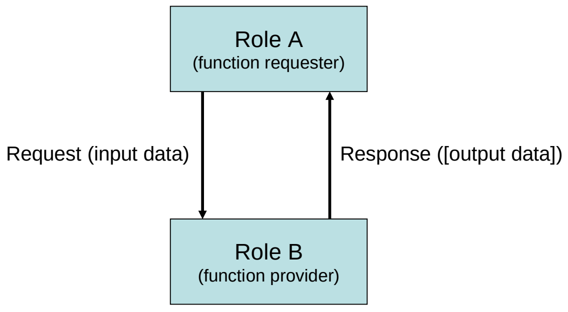


Figure 3-1: Functions as a request-response data exchange

At the function definition level, it is not defined if the function is synchronous or asynchronous. It is however the role of the message mapping and transport binding to specify this aspect, as mentioned in section 4 and 5.

Validity period:

When a function is called, the function provider takes the responsibility to execute all the single execution steps that are required to complete the function. Such processing may require some time to complete, but the function caller might want this processing duration to not exceed a specific amount of time called the "function validity period", as detailed in the following use cases:

- The function processing might no longer be valuable if it ends after the validity period. For example, a function is only valuable if it is executed within a minute. If more than a minute has elapsed, then it is no longer required to continue the function execution.
- Processing might not want to wait for an external event that might not occur before a very long time or an event that might even never occur at all. For example, it is possible when performing an OTA dialog that the device is unreachable (switched off, lost, etc.), or that an acknowledgement message coming from the device is lost on the network (e.g. the loss of a Proof of Receipt coming from a UICC). If so, it might not be acceptable to wait several days or weeks for the device to be switched on again, or even to wait forever for an acknowledgement message that will never come.
- It is desirable that the function provider system is not overloaded with requests that will be pending for a long period. The function caller would like to be notified as soon as possible that the function cannot be processed within a specific amount of time, and may then implement a calling side retry policy.

By providing a validity period, the function caller indicates a specific amount of time to the function provider to process the function. As a consequence, during this validity period, the function caller shall not issue the same request again as it might generate duplicate execution steps within the function provider system.

After the end of the validity period, the function provider shall no longer continue with new execution steps. It is only mandated to tell the function caller that the function processing has expired. It is then the caller responsibility to either:

- Request the same function again,
- Request the processing of the remainder of the execution steps,
- Request the processing of corrective execution steps,
- Or simply abandon the overall process into which the function was called.

Exceptions:

Note that during the processing of a function, an unexpected behavior may happen. This event, called an *exception* in this specification, may cause the function to be ended before the functional work to be completed (the exception is then considered as an *error*), or may let the functional work continue, but under specific conditions (the exception is then called a *warning*).

This is the function provider's responsibility to give information on any exception encountered during the processing of a function; however the behavior of the function caller when receiving this exception may depend on its own context (e.g. stop its current processing, or perform a retry attempt, or try a workaround processing, etc.).

Notification functions:

In some cases, functions are considered as notifications as they functionally correspond to events sent from one role to another. If so, the role that generates the notification is called the notification source or the notifier, and the role that receives the notification is called the notification destination or the notification recipient or notification handler. By definition, no validity period is applied for a notification, and no data can be returned back by the notification recipient to the notification source.

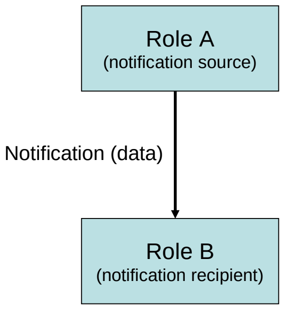


Figure 3-2: Notification as one way events

By convention, the name of the notification handler functions defined in this document are `Handle`*`Xxxx`*`Notification` as this is the notification recipient that acts as the function provider, i.e. that implements the particular processing related to the reception of the `Xxxx` notification.

This is the responsibility of the mapping to messages and of the binding to transport to specify how notifications are handled compared to classical request-response functions. See section 5 for the binding to Web Services over HTTP.

3.1.1 Simple Types and Identifiers

The functions specified in this document deal with the deployment and management of mobile-NFC services into a mobile environment, that includes a Secure Element, for an end-user that holds a Mobile Subscription.

The following entities consequently need to be referenced in most of the functions:

- The Mobile Subscription, i.e. the subscription to the mobile network
- The Secure Element,
- The mobile-NFC service,

Moreover, basic concepts such as Card Unique Data, ICCID, MSISDN, AID, TAR, etc. are also used in several of the functions.

The following sub-sections define those basic data as well as the main identifiers using in the functions.

For the openness of the specification to vertical businesses or to specific deployments, a generic type representing a "Name:Value pair" is also defined. This can be used in several contexts, to match business specific or deployment specific data.

3.1.1.1 Simple Types

The following tables present the simple types that define the basic data used in the functions.

Simple types related to Secure Element and Mobile Subscription identification:

Card Unique Data and ICCID are well known SE identifiers.

MSISDN is a well known Mobile Subscription identifier, but there is a requirement to use aliasing to prevent from using the MSISDN information all over the system.

As a consequence, the following simple types are used in the current specification for Secure Element and Mobile Subscription identification:

Type name	Description	Base type - Constraint
`Card Unique Data`	The Card Unique Data used to identify a Secure Element. Card Unique Data is build based on the IIN (Issuer Identification Number) and the CIN (Card Identification Number)	String - String representation of IIN + CIN concatenated, up to 32 hexadecimal digits.
`ICCID`	The ICCID (Integrated Circuit Card IDentifier) used to identify a Secure Element.	String - String representation of up to 20 hexadecimal digits.
`MSISDN`	The MSISDN (Mobile Station ISDN Number) used to identify a Mobile Subscription.	String - String representation of up to 15 decimal digits, as defined in ITU E.164 [27]
`Alias`	The alias used to identify a Mobile Subscription. An Alias is a free string that may be used in place of another identifier (such as the MSISDN) to identify a Mobile Subscription, for example to prevent from legal issues of using the original identifier all over the system (which might be considered as a private information, for example).	String - Free formatted string up to 255 characters.

Table 3-3 Simple types related to Secure Element and Mobile Subscription identification

Simple types related to Secure Element Application, Device Application, and SCWS Service Portal identification:

AID or TAR is used to identify a Secure Element Application.

There is no standard for a universal definition of a Device Application identifier.

There is no standard for a universal definition of a SCWS Portal identifier.

As a consequence, the following simple types are used in the current specification for SE Application, Device Application, and SCWS Service Portal identification:

Type name	Description	Base type - Constraint
`AID`	The AID (Application IDentifier) of an Executable Load File, an Executable Module, a Security Domain, or an Application.	String - String representation of 10 to 32 hexadecimal digits.
`TAR`	The TAR (Toolkit Application Reference) of a Security Domain or an Application.	String - String representation of exactly 6 hexadecimal digits.
`Device Application Identifier`	The identifier used to identify a Device Application.	String – Alphanumeric string of up to 255 characters. Allowed characters: `'A'` to `'Z'`, `'a'` to `'z'`, `'0'` to `'9'`.
`SCWS Portal Identifier`	The identifier used to identify a SCWS service portal.	String – Alphanumeric string of up to 255 characters. Allowed characters: `'A'` to `'Z'`, `'a'` to `'z'`, `'0'` to `'9'`.

Table 3-4 Simple types related to Secure Element and Mobile Subscription identification

3.1.1.2 Identifiers

The following tables define the identifiers that are used in the various functions as input or output parameters:

The `Mobile Subscription Identifier` type:

A Mobile Subscription can be primarily identified either by a MSISDN or an Alias, but it can be extended to other type of identifier.

Data name	Description	Type	MOC	No.
`MNO Name`	The name of the Mobile Network Operator that delivered the Mobile Subscription. Note that the MNO Name might be the name of a Virtual MNO (MVNO). Suggested format for the MNO name is the combination of Mobile Country Code (MCC) and Mobile Network Code (MNC) as specified by ITU E.212 [28]. Example: "XXXYYY", where: • XXX is the Mobile Country Code (MCC) • YYY is the Mobile Network Code (MNC) Any other format that all the parties can agree on MAY also be used as MNO Name.	`String`	O	1

Data name	Description	Type	MOC	No.
`Identifier`	The Mobile Subscription identifier. New types can be deployed, as a deployment choice.	`MSISDN` *or* `Alias` *or* `<any identifier type>`	M	1

Table 3-5 `Mobile Subscription Identifier` type

The `SE Identifier` type:

A Secure Element can be primarily identified either by a Card Unique Data or an ICCID, but it can be extended to other type of identifier.

Data name	Description	Type	MOC	No.
`Identifier`	The Secure Element identifier value. New types can be deployed, as a deployment choice.	`Card Unique Data` *or* `ICCID` *or* `<any identifier type>`	M	1

Table 3-6 `SE Identifier` type

The `Service Identifier` type:

A mobile-NFC service is identified by a `Service Id` and optionally a `Service Version`.

Data name	Description	Type	MOC	No.
`Service Id`	The technical identifier of the service.	`Integer`	M	1
`Service Version`	The version of the service. The `Service Version` type is described below.	`Service Version`	O	1

Table 3-7 `Service Identifier` type

Where a `Service Version` is:

Data name	Description	Type	MOC	No.
`Major Version`	The major version of the service.	`Integer`	M	1
`Minor Version`	The minor version of the service.	`Integer`	M	1
`Revision Version`	The revision version of the service. If not present, it implicitly corresponds to the latest revision of the 'major.minor' version.	`Integer`	O	1

Table 3-8 `Service Version` type

A mobile-NFC service is always identified in the context of the SDM that is responsible for its global delivery, and the `Service Identifier` shall be unique in this context. At the opposite, several `Service Identifiers` may have identical values as soon as the services are managed by different SDMs.

Note that the `Service Identifier` identifies the static definition of the mobile-NFC service. When deployed on an end-user mobile environment, it may be required to uniquely identify this particular instance that is under installation, and later, for post-delivery actions, to identify this particular instance that is to be suspended, resumed, or deleted. Then, a `Service Qualifier` information may be required as defined below.

A possible usage of such qualifier is the multiple deployment of the same service (same `Service Identifier`) on a particular mobile environment. For example:

- Multiple instantiation of the same banking application, each instance belonging to a particular Bank. The `Service Qualifier` information would qualify the particular instance (i.e. Bank) the instance of the service belongs to.
- Multiple instantiation of the same transport application, each instance corresponding to a particular town where the transport application is applicable. The `Service Qualifier` information would qualify the particular instance (i.e. town) the instance of the service belongs to.

Note that instantiating a service several times (and thus using a `Service Qualifier` to differentiate the instances) can only be done if the various instances of the service are exactly composed of the same SE and Device applications. Only the AID of the various applications and the content of the personalization data may change from one instance to another. Otherwise, it is to be considered as two different services.

The `Service Qualifier` contains a `Qualifier` information:

Data name	Description	Type	MOC	No.
`Qualifier`	Additional information that qualifies the mobile-NFC service. For instance, it may be used when a service is multi-instantiated, to uniquely identify the particular instance that is under installation, and later, for post-delivery actions, to identify the particular instance that is to be suspended, resumed, or deleted.	`Service Qualifier`	M	1

Table 3-9 `Service Qualifier` data

Where the `Service Qualifier` simple type is:

Type name	Description	Base type - Constraint
`Service Qualifier`	Additional information that qualifies the mobile-NFC service.	String – Alphanumeric string of up to 255 characters. Allowed characters: `'A'` to `'Z'`, `'a'` to `'z'`, `'0'` to `'9'`.

Table 3-10 `Service Qualifier` simple type

The `Service Qualifier` information is always considered in the context of the mobile-NFC service it is related to, i.e. it is always linked to a `Service Identifier` data presence.

Note that the usage of the `Service Qualifier` information SHALL be consistent and SHALL respect the following rule: as soon as this information is used once in conjunction with a particular `Service Identifier`, it SHALL be provided every time this `Service Identifier` is used. This is why the `Service Qualifier` information is mentional as a conditional data in all functions that take a `Service Identifier`.

The `Service Instance Reference` type:

Specifically for Application Provider, it might be difficult to know technical information such as the Secure Element identifier. It is also difficult for the AP to keep the exact value of the Mobile Subscription identifier over time, because this information might change at the initiative of the MNO or of the end-user, without the involvement of the Application Provider.

As a consequence, the identification of a mobile-NFC service deployed into a mobile environment, that includes a Secure Element, and for an end-user that holds a Mobile Subscription, is represented by the `Service Instance Reference` (SIR) for the AP to SDM relations.

The `Service Instance Reference` is an opaque (i.e. without standardized capability to retrieve information by just analyzing its value) representation of a specific realization of a mobile-NFC service for an end-user.

Data name	Description	Type	MOC	No.
`Reference owner`	The identifier of the actor that owns the SIR, i.e. who has generated the reference. In case the SIR is generated and returned by a SDM during the call of the `LookupServiceInstanceReference` function, then the `Reference owner` SHALL be filled with the OID of the `LookupServiceInstanceReference` function provider. In case the SIR is generated by an AP and declared by a call to the `DeclareServiceInstanceReference` function, then the `Reference owner` SHALL be filled with the OID of the `DeclareServiceInstanceReference` function requester.	`Object Identifier`	M	1
`Reference`	The `Service Instance Reference` value.	`String` Alphanumeric string of up to 255 characters. Allowed characters: `'A'` to `'Z'`, `'a'` to `'z'`, `'0'` to `'9'`.	M	1

Table 3-11 `Service Instance Reference` type

Where an `Object Identifier` is a string representation of an OID, i.e. of integers separated with dots (e.g.: '1.2', '3.4.5').

The `Service Instance Reference` lifetime is linked to the refered service instance, which life cycle is depicted in Figure 3-7. The `Service Instance Reference` can:

- Either be computed by the SDM, following a call to the `Lookup Service Instance Reference` function (see section 3.3.2.1),
- Or be provided by the AP and then declared by a call to the `DeclareServiceInstanceReference` function (see section 3.3.2.2).

The `Service Instance Reference` shall then be used in any of the AP to SDM further functions calls for identifying the instance of service that is to be managed.

Note that at the end of the `LookupServiceInstanceReference` or `DeclareServiceInstanceReference` functions calls, the reference exists, but the mobile-NFC service itself may not be deployed yet.

3.1.1.3 "Name Value pair" Type

The `Name Value Pair` type:

A "Name:Value pair" is a generic type that can be used to store business specific or context specific information, when allowed or requested in the functions.

Data name	Description	Type	MOC	No.
`Name`	The name of the "Name:Value pair ".	`String`	M	1
`Value`	The value associated to the "name". This value can be of any type. Having several occurrences of this field enables to specify a list of values for the corresponding name.	`<any value type>`	M	1..N

Table 3-12 `Name Value Pair` type

Here are some examples of such "Name:Value pairs", in order to illustrate possible usages:

- To represent the value of a counter:
  - Name = "Counter"
  - Value = 10 *(as an Integer)*
- To represent the name of an end-user:
  - Name = "Last Name"
  - Value = "Smith" *(as a String)*
- To represent a list of dates:
  - Name = "Birthday list"
  - Value = {"1965/01/12", "1971/05/22", "1983/08/01", "2012/02/28"} *(as a list of Dates)*
- To represent a list of "Name:Value pairs":
  - Name = "Person"

- o Value = { [*Name:Value pair:* Name="First Name", Value="John"], [*Name:Value pair:* Name="Last Name", Value="Smith"], [*Name:Value pair:* Name="Birth Date", Value="1986/06/26"] }

List can also be of mixed types.

3.1.2 Functions Input Header

All functions (request-response and notification handler) SHALL include the following header as part of the input data:

Input data name	**Description**	**Type**	**MOC**	**No.**
`Function Requester Identifier`	Identification of the function requester.	`String`	M	1
`Function Call Identifier`	Identification of the function call. This identifier enables to manage function call retry policies. When requesting for the execution of a function, the function caller shall provide a unique Function Call Identifier. Uniqueness is to be ensured in its own perimeter. In case the function caller wants to retry the same function, then it shall perform the same function call, providing the same Function Call Identifier. On function provider side, when receiving this retry attempt, if a call to a function if performed with an Function Call Identifier of a function already in process in its system, then the function provider shall refuse the new call If the function provider does not want to implement any retry policy, then it might ignore this field. The Function Call identifier is only mandatory for request-response functions. It SHALL NOT be present for notification functions	`String`	C	1
`Validity Period`	This field defines the length of the period (provided as a number of seconds) during which the request is valid. The period starts at the time the function call was received by the function provider and ends a number of seconds later. During this period, the function provider has the responsibility to execute all the single execution steps that are required to complete the function. However: • If the function provider immediately considers that the validity period is invalid (e.g. too long or too short) or cannot fulfill the requirements (i.e. cannot start the sequence of operations so that all of the	`Integer`	O	1

Input data name	Description	Type	MOC	No.
	operations are completed within the validity period), it SHALL NOT process the function and SHALL return a `Function Execution Status` output parameter with a `Status` field set to `'Failed'` and a `Subject` and `Reason` of the `Status Code Data` field set to 'Validity Period not accepted' (see section 3.1.3). The function provider SHALL also indicate to the function caller an acceptable amount of time into which the request could be fulfilled, by setting the `Acceptable Validity Period` field in the output header. • If the function provider considers the validity period as acceptable but failed in completing the execution of the function within the validity period (due to unforeseen delays), it SHOULD not engage any new execution steps and SHALL return an `'Expired'` value in the `Status` field of the `Function Execution Status` output parameter (see section 3.1.3). The Validity Period is only present (but optional) for request-response functions. If not specified, a default validity period value SHOULD be applied following business agreement between parties. The Validity Period SHALL NOT be present for notification functions.			

Table 3-13 Functions input header

Each function can define its own set of additional input data.

The following diagram presents the flow chart for the validity period management by the function provider:

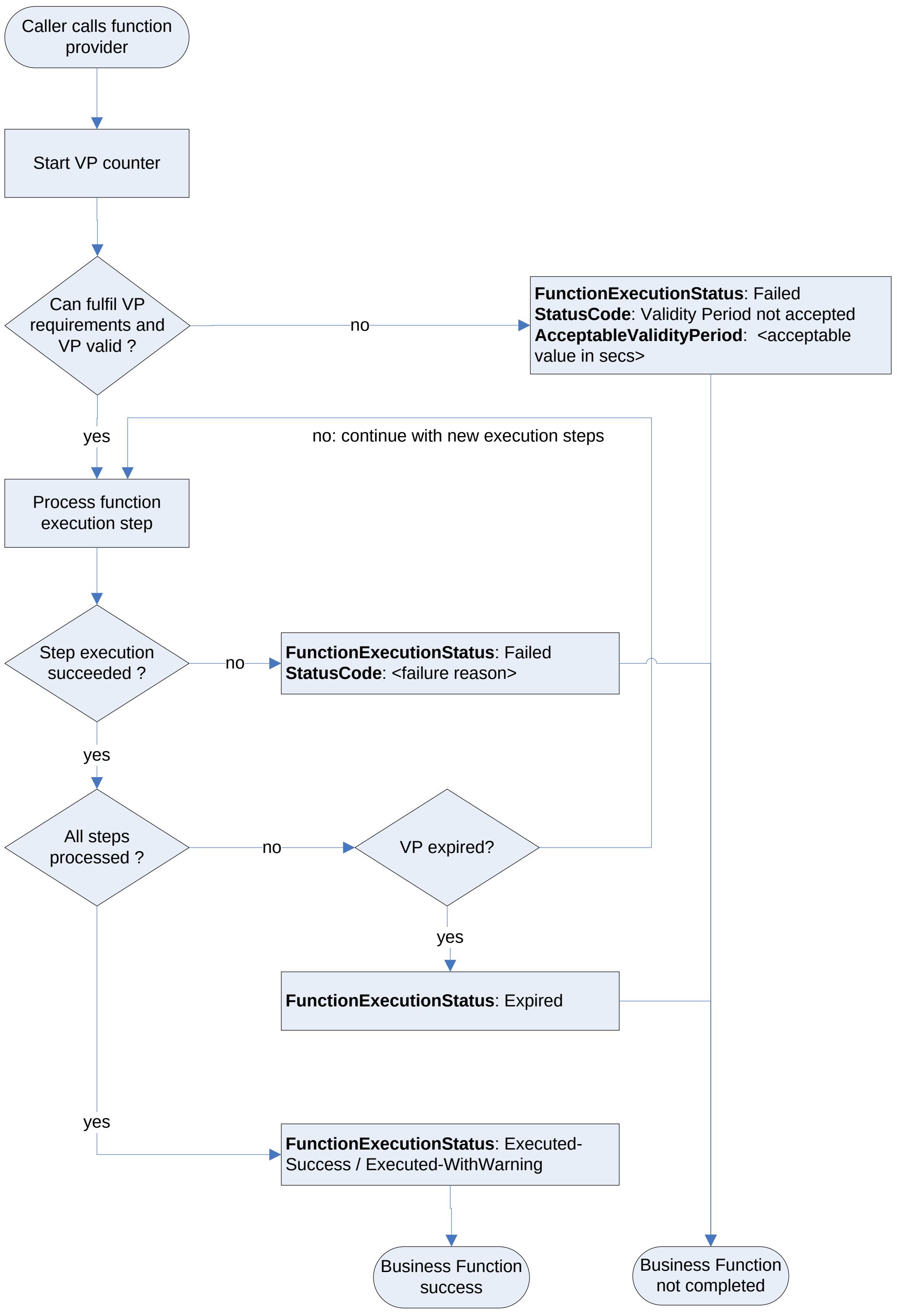

Caller calls function provider
Start VP counter
Can fulfil VP requirements and VP valid ?
no
FunctionExecutionStatus: Failed
StatusCode: Validity Period not accepted
AcceptableValidityPeriod: <acceptable value in secs>
yes
no: continue with new execution steps
Process function execution step
Step execution succeeded ?
no
FunctionExecutionStatus: Failed
StatusCode: <failure reason>
yes
All steps processed ?
no
VP expired?
yes
FunctionExecutionStatus: Expired
yes
FunctionExecutionStatus: Executed-Success / Executed-WithWarning
Business Function success
Business Function not completed

Figure 3-3: Validity period management flow chart

3.1.3 Functions Output Header

All functions (request-response) SHALL include the following header as part of the output data. Notifications don't have any output data.

Output data name	Description	Type	MOC	No.
`Processing Start`	The start time and date of the real processing of the function by the function provider (and not the time and date of reception of the request).	`DateTime`	O	1
`Processing End`	The function processing end time and date.	`DateTime`	O	1
`Acceptable Validity Period`	In case the validity period provided as input parameter is not acceptable, then the function provider SHALL return an acceptable value to the function caller (see section 3.1.2). The function caller might then call again the same function with a validity period that is more convenient (but that may however differ from the exact value of the `Acceptable Validity Period` field sent in response to the previous function call).	`Integer`	C	1
`Function Execution Status`	Indicates whether the processing has been completed correctly or not. If required, provides information to give details on the processing result (status code, status code reason, status message). The `Execution Status` type is described below.	`Execution Status`	M	1

Table 3-14 Functions output header

Where an `Execution Status` is:

Data name	Description	Type	MOC	No.

`Status`	It indicates whether the processing has been completed correctly or not.
	Value **`'Executed-Success'`** means that the function has been processed correctly. Application output data MAY optionally be part of the function response.
	Value **`'Executed-WithWarning'`** means that the function has been processed correctly, but that warnings have been generated during this execution. Application output data MAY optionally be part of the function response in order to provide details on the warnings.
	Value **`'Failed'`** means that the function execution has encountered errors during its processing. The `Status Code Data` output structure SHALL give the reason of error in the processing (values depend on the function and may be implementation dependant).
	Value **`'Expired'`** means that the validity period of the request has expired before the completion of the function processing. The `Status Code Data` output structure MAY give the reason of expiration of the function.
		`Enumeration {Executed-Success,`
		`Executed-WithWarning,`
		`Failed, Expired}`
			M	1

Data name	Description	Type	MOC	No.
Status Code Data	It provides the reason of the Status. Present only if the Status is 'Execute-WithWarning', 'Failed', or 'Expired'. The Status Code Data type is described below.	Status Code Data	O	1

Table 3-15 Execution Status type

Where a Status Code Data is:

Data name	Description	Type	MOC	No.
Subject	Represents the system element concerned by the exception.	Object Identifier	M	1
Reason	Represents the reason of the exception	Object Identifier	M	1
Subject Identifier	The identifier of the subject or any identification data of the subject that caused the exception (e.g. AID of the instance when the Subject is an "Application instance"). The possible values of the Subject Identifier depend on the function and may be implementation dependant.	String	O	1
Message	It provides a textual and human readable explanation of the exception. The Message value is implementation dependant.	String	O	1

Table 3-16 Status Code Data type

Where an Object Identifier is a string representation of an OID, i.e. of integers separated with dots (e.g.: '1.2', '3.4.5').

In case where the function processing has been performed correctly (Status is 'Executed-Success' or 'Executed-WithWarning'), the function output data SHALL include the output header data and MAY include additional output parameters. The mandatory (M), conditional (C), or optional (O) aspect of the output data SHALL then be respected.

In the case where the function processing has been performed incorrectly (Status is 'Failed' or 'Expired'), the output header data SHALL be returned, but the additional output data SHALL only be present if it is explicitly mentioned in the function description. This is for example the case for the "SE Commands Generation and Remote Execution" and the "Send Script" functions that MAY contain additional output data even if failed. For other functions, only the output data are present: the mandatory (M), conditional (C), or optional (O) aspect of the additional output SHALL be ignored (no additional output data is provided).

The exception mapping and binding to Web Services over HTTP is provided in section 5.

A normative list of exception codes is provided in section 6 but additional codes may be defined.

3.2 Control Functions

The following functions are described in this section:

- Eligibility checking functions:
    - `CheckGlobalEligibility` (see section 3.2.1.1)
    - `GetDeviceCapabilityProfileId` (see section 3.2.1.2)
    - `GetSECapabilityProfileId` (see section 3.2.1.3)
    - `CheckMobileSubscriptionEligibility` (see section 3.2.1.4)
- AID generation:
    - No function required (see section 3.2.2)
- Service management life cycle notifications:
    - `HandleStartServiceStateChangeNotification` (see section 3.2.3.1)
    - `HandleEndServiceStateChangeNotification` (see section 3.2.3.2)
- Mobile Subscription life cycle:
    - `GetMobileSubscriptionAlternateIdentifier` (see section 3.2.4.1)
    - `GetSEMobileSubscriptionIdentifier` (see section 3.2.4.2)
    - `GetMobileSubscriptionSEIdentifiers` (see section 3.2.4.3)
    - `HandleMobileSubscriptionIdentifierChangedNotification` (see section 3.2.4.4)
    - `HandleMobileSubscriptionStatusChangeNotification` (see section 3.2.4.5)
- Secure Element life cycle notifications:
    - `HandleSERenewalNotification` (see section 3.2.5.1)
    - `HandleSEDeviceChangedNotification` (see section 3.2.5.2)
    - `HandleSEDeviceStatusChangeNotification` (see section 3.2.5.3)
    - `HandleSEMobileSubscriptionChangedNotification` (see section 3.2.5.4)
    - `HandleSEStatusChangeNotification` (see section 3.2.5.5)

3.2.1 Eligibility Check

In order for the service delivery ecosystem to work, it is necessary for the Service Provider to be able to make a business decision on whether the combined capabilities of the device, secure element and the subscription meets the requirements that a mobile-NFC service demands.

There are mainly three areas that are of interest to be checked before issuing a service to an end-user. These are Device eligibility, SE eligibility and Mobile Subscription eligibility.

For Service Provider point of view, a global eligibility is required (the `CheckGlobalEligibility` function). This can for example be requested by the SP to a TSM.

Then, for the TSM to establish this eligibility, dedicated check of the Device (through the `GetDeviceCapabilityProfileId` function), the SE (through the `GetSECapabilityProfileId` function) and the Mobile Subscription (the `CheckMobileSubscriptionEligibility` function) will be performed.

3.2.1.1 The "Check Global Eligibility" Function

Name: `CheckGlobalEligibility`

Description:

This function enables to request for the global eligibility of the end-user mobile environment (identified by the `Mobile Subscription` and/or the `Secure Element` input data), for the delivery of a mobile NFC service (identified by the `Service` and the optional `Service Qualifier` input data). This check is useful when the AP wants to delegate the eligibility decision for mobile-NFC service deployment to the SDM. Most commonly, it covers the Secure Element eligibility, the Device eligibility, and the Mobile Subscription eligibility.

At least:

- Either the `Mobile Subscription Identifier`
- Or the `Secure Element Identifier`

SHALL be provided by the function caller: the `Mobile Subscription Identifier` is an information that should be easily known by the Application Provider, but the `Secure Element Identifier` is important when multiple Secure Elements exist on the device or in a non-UICC based deployment, to be able to specify on which Secure Element the eligibility check is requested. However, it should also be possible for the AP to fully delegate the SDM the decision of which SE to be used: the SDM will know the best fit of SE given a specific service.

If only one of those two identifiers is provided, the function provider SHALL retrieve the other one if it is required for the execution of the eligibility process. Retrieval MAY be done by the function provider using its own information system, or by asking for this data to another role (for example using the `GetSEMobileSubscriptionIdentifier` function (see section 3.2.4.2), or through the `GetMobileSubscriptionSEIdentifiers` function (see section 3.2.4.3)).

In case of non eligibility (the `Eligible` output data), the function provider SHALL provide as much non-eligibility reasons as possible (the `Non Eligibility Reason` output data list): non eligibility reason of the Secure Element, non eligibility reason on the Device, non eligibility reason on the Mobile Subscription. This enables the function caller to immediately be aware of (and inform the end-user of) the overall set of reasons, and not only the "first" one.

Function provider: SDM

Functions group: Global Eligibility Info

Service Deployment Modes: mode #1, mode #2, mode #3

Additional input data:

Input data name	Description	Type	MOC	No.
`Mobile Subscription`	The Mobile Subscription identifier through which the Secure Element is accessible.	`Mobile Subscription Identifier`	C	1
`Secure Element`	The Secure Element into which the mobile-NFC service may be deployed.	`SE Identifier`	C	1
`Service`	The mobile-NFC service.	`Service Identifier`	M	1
`Service Qualifier`	The qualifier of the mobile-NFC service. Only present if the `Service Identifier` is present.	`Service Qualifier`	C	1

Table 3-17 Additional input parameters for `CheckGlobalEligibility` function

Note: the `Mobile Subscription Identifier`, the `Secure Element Identifier`, the `Service Identifier` and the `Service Qualifier` types are defined in section 3.1.1.

Additional output data:

Output data name	Description	Type	MOC	No.
`Eligible`	If the device, the SE and the mobile subscription is eligible for mobile NFC service delivery.	`Boolean`	M	1
`Non Eligibility Reason`	A reason of non-eligibility. Present only if `Eligible` is set to `'False'.` Having several occurrences of this field enables to specify several non-eligibility reasons. The following values are pre-defined, but other values MAY be defined by the function provider: **1 ⇨ Mobile Subscription eligibility** **11 ⇨ SE capability** **21 ⇨ Device capability** **31 ⇨ Incompatible deployment conditions** (e.g. trying to act on a SE which Secure Element Issuer is not known by the function provider; function provider recognizes the Mobile Subscription Identifier but does not have the capability to use the relevant mobile network) **32 ⇨ Incompatible trust environment** (incompatible deployment condition due to a lack of trust between actors of the system. E.g.: access rights not granted or insufficient business agreement to perform the service management)	`Integer`	C	1..N

Table 3-18 Additional output parameters for `CheckGlobalEligibility` function

3.2.1.2 The "Get Device Capability Profile Id" Function

Name: `GetDeviceCapabilityProfileId`

Description:

This function enables to request for the identifier of a Device profile (the `Device Capability Profile Id` output data) that gives information about the Device currently associated to a particular Mobile Subscription (identified by the `Mobile Subscription` input data). This profile enables to know the various capabilities of the Device.

The output of this function is not the profile itself, but a reference (identifier) to a Profile Id. This Id is agreed in advance as a business agreement between the function provider and the function requesters. However a suggested and informative list of capabilities, that might be important to include in such a profile, is listed in appendix (see section 9).

The capabilities can be both of technical and business nature.

Function provider: DMSR

Functions group: Device Info

Additional input data:

Input data name	Description	Type	MOC	No.
`Mobile Subscription`	The Mobile Subscription through which the Device is accessible.	`Mobile Subscription Identifier`	M	1

Table 3-19 Additional input parameters for `GetDeviceCapabilityProfileId` function

Note: the `Mobile Subscription Identifier` type is defined in section 3.1.1.

Additional output data:

Output data name	Description	Type	MOC	No.
`Device Capability Profile Id`	The pre-defined Profile Identifier for Device capabilities.	`Integer`	M	1

Table 3-20 Additional output parameters for `GetDeviceCapabilityProfileId` function

3.2.1.3 The "Get SE Capability Profile Id" Function

Name: `GetSECapabilityProfileId`

Description:

This function enables to request for the identifier of a SE profile (the `SE Capability Profile Id` output data) that gives information about the SE (identified by the `Secure Element` input data). This profile enables to know the various capabilities of the Secure Element.

The output of this function is not the profile itself, but a reference (identifier) to a Profile Id. This Id is agreed in advance as a business agreement between the function provider and the function requesters. However a suggested and informative list of capabilities, that might be important to include in such a profile, is listed in appendix (see section 10).

The capabilities can be both of technical and business nature.

Function provider: SEI

Functions group: SE Info

Additional input data:

Input data name	Description	Type	MOC	No.
`Secure Element`	The Secure Element into which the mobile-NFC service is deployed/under deployment.	`SE Identifier`	M	1

Table 3-21 Additional input parameters for `GetSECapabilityProfileId` function

Note: the `SE Identifier` type is defined in section 3.1.1.

Additional output data:

Output data name	Description	Type	MOC	No.
`SE Capability Profile Id`	The pre-defined Profile Identifier for SE capabilities.	`Integer`	M	1

Table 3-22 Additional output parameters for `GetSECapabilityProfileId` function

3.2.1.4 The "Mobile Subscription Eligibility Check" Function

Name: `CheckMobileSubscriptionEligibility`

Description:

This function enables to request for the eligibility of the Mobile Subscription (identified by the `Mobile Subscription` input data) to the to-be-delivered mobile-NFC service (identified by the `Service` and the optional `Service Qualifier` input data). This check is useful to know if the Mobile Subscription is allowed to have a data plan and if the Mobile Subscription is setup to enable NFC services.

Function provider: DMSR

Functions group: Mobile Subscription Info

Additional input data:

Input data name	Description	Type	MOC	No.
`Mobile Subscription`	The Mobile Subscription to check.	`Mobile Subscription Identifier`	M	1
`Service`	The mobile-NFC service.	`Service Identifier`	M	1
`Service Qualifier`	The qualifier of the mobile-NFC service.	`Service Qualifier`	C	1

Table 3-23 Additional input parameters for `CheckMobileSubscriptionEligibility` function

Note: the `Mobile Subscription Identifier`, the `Service Identifier` and the `Service Qualifier` types are defined in section 3.1.1.

Additional output data:

Output data name	Description	Type	MOC	No.
`Eligible`	If the subscription is eligible for use, in the context of deployment of the particular mobile-NFC service.	`Boolean`	M	1
`Non Eligibility Reason`	The reason of non-eligibility. Reasons are business process dependent. Present only if `Eligible` is set to `'False'`.	`Integer`	C	1

Table 3-24 Additional output parameters for `CheckMobileSubscriptionEligibility` function

3.2.2 AID Generation

At mobile-NFC service deployment time, AID values shall be assigned to the Application instances to be deployed in the Secure Element. The determination of the value of these Application instances AIDs depends on the service AID policy, but also on the AID values already used in the Secure Element (AIDs are unique over the whole SE).

Such AID generation task may be complex as it does not just correspond to the generation of a simple Application IDentifier: this AID shall first be unique within the Secure Element, among all AID used in the SE, including all Executable Load Files AIDs and all Application instances AIDs.

Secondly, if the mobile-NFC service is deployed in a UICC, it may be required that a SE Application have a TAR (Toolkit Application Reference) in order to be reachable by OTA following the ETSI TS 102.226 [14] specification. An application may have several TAR values, but ETSI TS 101 220 [11] states that the primary TAR value is composed of bytes 13, 14 and 15 of the AID. As for AIDs, TARs shall be unique over the whole UICC.

AID (and thus the optional TAR) generation policy may be very dependent on the agreements between the various actors of a NFC deployment. As a consequence, GlobalPlatform considers in this version of the specification that only two basic models are applicable:

- Model #1: AID and TAR generation is fully under the responsibility of the SDM:

This may occur in the context of very structured business and markets, where AID and TAR rules are already defined by a standardization body and are detailed enough to prevent any AID collision for services deployed by several SDM. The SDM may then simply follow such rules in order to determine a unique AID and TAR value.

This is for example the case for banking services for which Application Owners (e.g. Visa or MasterCard) have defined their unique short AID for their Application. This means that all instances of a particular Visa (resp. MasterCard) Credit service shall have the same short AID, additional bytes of the long AID only differentiating the various instances of the Applications: EMVCo for example recommends using the BIN6 and an "instance number" to complement the short AID, in order to build a unique long AID.

This may also occur if a common agreement between all the actors of the ecosystem has been reached in order to allocate dedicated ranges of AIDs and TARs to each SDM. For example, a particular value of a particular byte of all AID values may be reserved for each SDM.

This is consequently the responsibility of each SDM to ensure uniqueness of the AIDs and TARs it generates by itself, global uniqueness with other SDM being ensured by the range usage.

- Model #2: AID and TAR generation is fully under the responsibility of the Secure Element Issuer:

  This may occur in less structured contexts where no AID rules have been defined, or if existing rules do not prevent AID collision in case of multiple SDM accessing the same Secure Element.

  The only entity that is really able to ensure AID and TAR uniqueness for a particular Secure Element is then the Secure Element Issuer itself. This is consequently the SEI that generates all AIDs and TARs, ensuring uniqueness of the values.

For both models, it is not required to have any real time interaction between the SDM and the SEI: only pre-agreement is required. As a consequence, no particular function is provided for AID/TAR generation or uniqueness checking. Application AID and TAR values are only exchanged (from the SDM to the SEI, or from the SEI to the MNO) at instantiation time, through the instantiation function (see section 3.5.1.3.2.3).

3.2.3 Service Management Life Cycle Notifications

Several global operations can be performed during the life of a mobile-NFC service: service delivery, service suspension, service deletion, etc. The processing of those operations lead the mobile-NFC service instance to follow a state diagram that is depicted in Figure 3-7 (see section 3.3.1).

In order to execute each of these global operations, and thus move from one state to another of this state diagram, a set of single actions may be needed. It is consequently difficult for roles having a partial visibility on the end-user mobile environment to be aware of the real service instance state.

In order to provide visibility on this mobile-NFC service instance state to other roles, the "start service state change" notification SHALL be sent prior performing any global operation on the NFC service, and the "end service state change" notification SHALL be sent at the end of the global operation execution. These notifications MAY be received by many roles of the ecosystem such as the SEI, the SDM, the DMSR, in order to know that the state of the service is changing, and then that it has been changed. As concrete examples:

- The "start" notification may be sent by the Service Provider (playing the Application Provider role) to the TSM (playing the SDM role), before requesting for the global deployment of the service. The "end" notification would then be sent at the end of the deployment process. Note that actions performed between the "start" and the "end" may encompass functions calls between the two parties, but also actions performed by the notifier side (such as the update of the SP backend, so that the service is notified as "deployed" only when it is really fully available to the end-user).

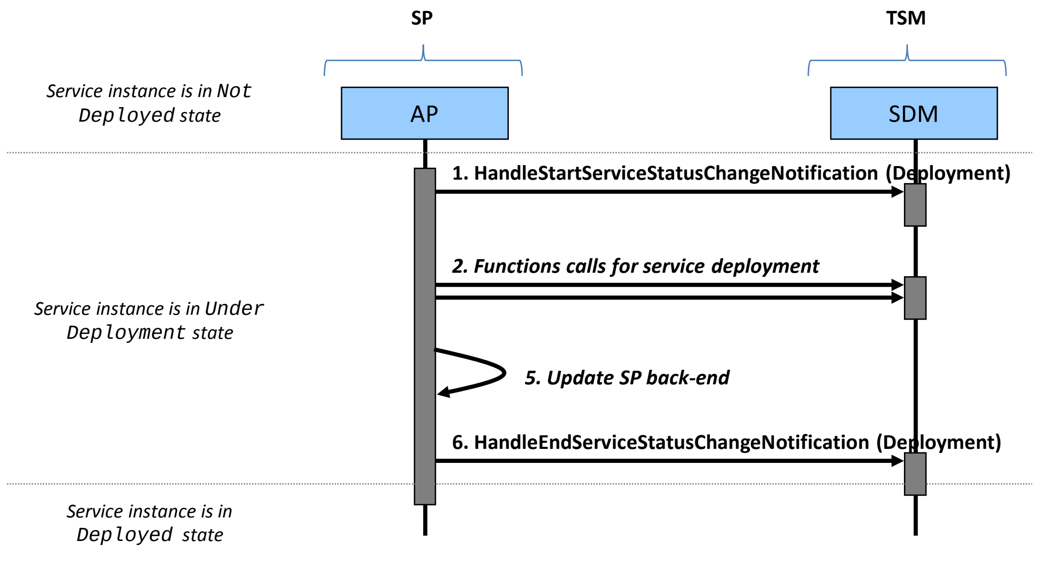

SP
TSM
Service instance is in Not Deployed state
AP
SDM
1. HandleStartServiceStatusChangeNotification (Deployment)
2. Functions calls for service deployment
Service instance is in Under Deployment state
5. Update SP back-end
6. HandleEndServiceStatusChangeNotification (Deployment)
Service instance is in Deployed state

Figure 3-4: AP Sends Service Life Cycle Notifications to TSM SDM

- The TSM (playing the SDM role, for Eligibility and Service Management) may also send those notifications to the SE Provider (playing also the SDM role, for SE Content Management) and to the DMSR (for Device application management), before (and after) calling the series of functions that takes part of the global service deployment.

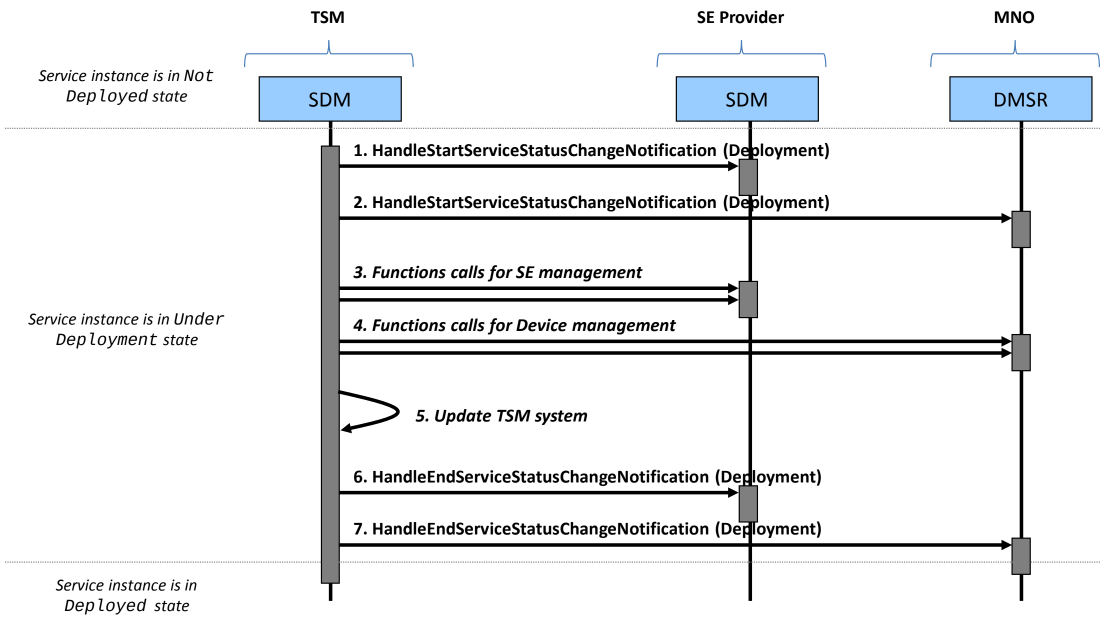

TSM
SE Provider
MNO
Service instance is in Not Deployed state
SDM
SDM
DMSR
1. HandleStartServiceStatusChangeNotification (Deployment)
2. HandleStartServiceStatusChangeNotification (Deployment)
3. Functions calls for SE management
Service instance is in Under Deployment state
4. Functions calls for Device management
5. Update TSM system
6. HandleEndServiceStatusChangeNotification (Deployment)
7. HandleEndServiceStatusChangeNotification (Deployment)
Service instance is in Deployed state

Figure 3-5: TSM SDM Sends Service Life Cycle Notifications to SE Provider SDM and DMSR

- Finally, as an example of the reverse way, the SE Provider (playing also the SDM role) may also send those notifications to the TSM (playing also the SDM role) if the SE Provider takes the decision to lock a SE application belonging to a service, following the loss of theft of the Secure Element.

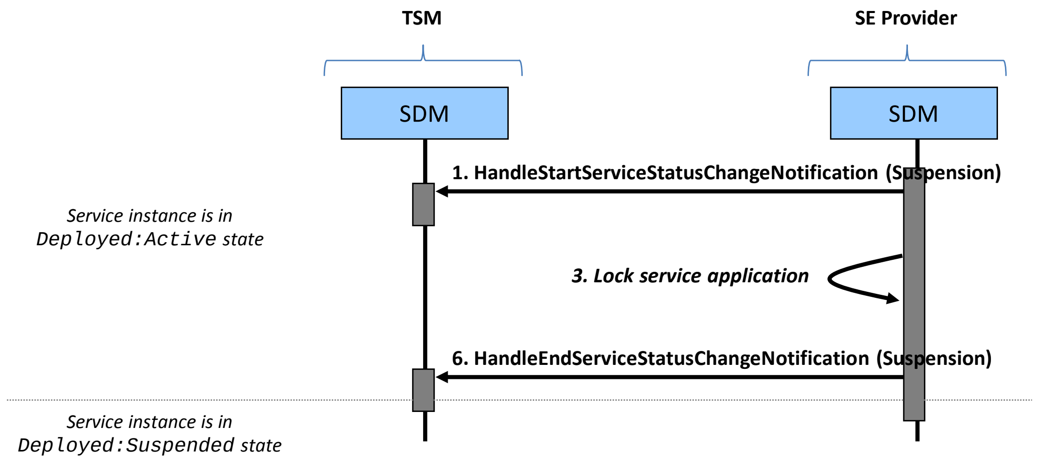

TSM
SE Provider
SDM
SDM
1. HandleStartServiceStatusChangeNotification (Suspension)
Service instance is in Deployed:Active state
3. Lock service application
6. HandleEndServiceStatusChangeNotification (Suspension)
Service instance is in Deployed:Suspended state

Figure 3-6: SE Provider SDM Sends Service Life Cycle Notifications to TSM SDM

The following global operations may be tracked. Note that these operations can be mapped to the service instance state diagram that is decipted in Figure 3-7.

- Service deployment: the initial installation, personalization and activation of a new service instance on a mobile environment (`Operation` input data set to `1`),
- Service instance re-deployment: the partial or full re-delivery of an already existing service instance. May be:
    - The full service instance re-deployment, for example following a mobile environment loss or renewal (`Operation` input data set to `101`),
    - The Secure Element applications re-deployment, for example following a SE renewal (`Operation` input data set to `102`),
    - The Device applications re-deployment, for example following a Device renewal (`Operation` input data set to `103`),
    - The Service applications re-personalization (`Operation` input data set to `104`), if the usage of the `ExchangeServiceData` function is not possible for updating the whole personalization data.
- Service upgrade: the upgrade of the service instance to another version of the same service (`Operation` input data set to `2`),
- Service update: the update of internal data of the service instance (validity date, individual personalization data, etc.), through exchange of service specific data between the Application Provider and the Secure Element, without impacting the version of the service (`Operation` input data set to `3`),
- Service suspension: temporary blocking the usage of the service instance (`Operation` input data set to `4`),

- Service resumption: unblocking the usage of the service instance (`Operation` input data set to 5),
- Service termination: the removal of the components of the service from the mobile environment (`Operation` input data set to 6).

Complementary operations may be tracked, upon ecosystem's actors' choice.

The "end service state change" notification SHALL indicate whether the state change has been performed correctly or not, and SHALL trigger most of the service instance states transitions, as shown in section 3.3.1.

Note that the "start service state change" and the "end service state change" notifications are different from the "Action Done On Service" notification depicted in section 3.3.3.2 because the first two declare the intention and completion of change in the service state, to other roles of the ecosystem. The last notification only targets the AP in order to inform it about actions performed on the service without its involment.

3.2.3.1 The "Start Service State Change" Notification Functions

Name: `HandleStartServiceStateChangeNotification`

Description:

This function SHALL be called to notify other actors of the ecosystem about the start of a global operation (defined by the `Operation` input data) on a particular instance of mobile-NFC service. Depending on the operation, the service instance state MAY change, as depicted in Figure 3-7 (see section 3.3.1).

The instance of mobile-NFC service MAY be identified through several ways, depending on the context and on the area of interest of the notification recipient. This influences the identifiers to be sent in the notification, which also depends on the operation that is notified:

- When the notification sender is the AP, the service instance SHALL be identified by the SIR.
  - In case this notification notifies a service deployment or a service re-deployment (for Device applications re-loading or service re-personalization, or following a Secure Element renewal or a Device change), then the corresponding SIR SHALL be provided in the `New Service Instance Reference` input data~~.~~

    The `Current Service Instance Reference` input data SHALL NOT be present.
  - In case this notification notifies a service update, suspension, resumption, or termination, then the reference to the service instance SHALL be provided in the `Current Service Instance Reference` input data.

    The `New Service Instance Reference` input data SHALL NOT be present.
  - In case this notification notifies a service upgrade, then the reference to the current service instance (the `Current Service Instance Reference` input data) and the new reference to the service instance (the `New Service Instance Reference` input data) SHALL be provided.

- When the notification sender is not the AP:
  - If the notification recipient is focusing on the Secure Element (e.g. the SEI), the service instance SHALL be identified by the service identifier and the SE identifier,
  - If the notification recipient is focusing on the Mobile Subscription (e.g. the DMSR), the service instance SHALL be identified by the service identifier and the Mobile Subscription identifier.

  If the recipient is interested in both the Secure Element and the Mobile Subscription (e.g. the SDM), then both the SE identifier and the Mobile Subscription identifier SHALL be provided.

  In all those use cases:

  - In case this notification notifies a service deployment or a service re-deployment (for Device applications re-loading or service re-personalization, or following a Secure Element renewal or a Device change), then the identification of the service to be (re-) installed (including the version and the qualifier of the service) SHALL be provided in the `New Service` and `New Service Qualifier` input data.

    The `Current Service` and `Current Service Qualifier` input data SHALL NOT be present.
  - In case this notification notifies a service update, suspension, resumption, or termination, then the identification of the service (including the version and the qualifier of the service) SHALL be provided in the `Current Service` and `Current Service Qualifier` input data.

    The `New Service` and `New Service Qualifier` input data SHALL NOT be present.
  - In case this notification notifies a service upgrade, then the identification (including the version and the qualifier of the service) of the current service SHALL be provided in the `Current Service` and `Current Service Qualifier` input data, and the identification of the new version of the service SHALL be provided in the `New Service` and `New Service Qualifier` input data.

The recipient of the notification MAY use this notification to update customer or service statuses in their internal systems, in order to provide up-to-date information to the customer care or the self-care interface.

Notification recipients:

- With Service Instance Reference: SDM
- With Secure Element identifier: SEI, SDM
- With Mobile Subscription identifier: DMSR, SDM

Functions group: Service Life Cycle Notification

Service Deployment Modes: mode #1, mode #2

Card Content Management modes: Simple mode, Delegated mode, Dual mode

Additional input data:

Input data name	Description	Type	MOC	No.
`Current Service Instance Reference`	The current reference to the mobile-NFC service instance.	`Service Instance Reference`	C	1
`New Service Instance Reference`	The new reference to the mobile-NFC service instance, in case of service (re-) deployment and upgrade.	`Service Instance Reference`	C	1
`Mobile Subscription`	The Mobile Subscription through which the mobile-NFC service is accessible.	`Mobile Subscription Identifier`	O	1
`Secure Element`	The Secure Element into which the mobile-NFC service is deployed/under deployment.	`SE Identifier`	O	1
`Current Service`	The current mobile-NFC service.	`Service Identifier`	C	1
`Current Service Qualifier`	The qualifier of the current mobile-NFC service.	`Service Qualifier`	C	1
`New Service`	The identifier of the new version of mobile-NFC service, in case of service (re-) deployment or upgrade.	`Service Identifier`	C	1
`New Service Qualifier`	The qualifier of the new version of mobile-NFC service, in case of service (re-) deployment or upgrade.	`Service Qualifier`	C	1
`Operation`	The operation to be tracked. The following values are pre-defined: **1 ⇨ Service deployment** **101 ⇨ Service re-deployment** **102 ⇨ Secure Element applications re-deployment** **103 ⇨ Device Applications re-deployment** **104 ⇨ Service re-personalization** **2 ⇨ Service upgrade** **3 ⇨ Service update** **4 ⇨ Service suspension** **5 ⇨ Service resumption** **6 ⇨ Service termination**	`Integer`	M	1

Table 3-25: Additional input parameters for `HandleStartServiceStateChangeNotification` functions

Note: the `Service Instance Reference`, the `Mobile Subscription Identifier`, the `SE Identifier`, the `Service Identifier` and the `Service Qualifier` types are defined in section 3.1.1.

This notification function does not have any output data.

3.2.3.2 The "End Service State Change" Notification Functions

Name: `HandleEndServiceStateChangeNotification`

Description:

This function SHALL be called to notify other actors of the ecosystem about the end of a global operation (defined by the `Operation` input data) on a particular instance of mobile-NFC service. Depending on the operation, the service instance state MAY change, as depicted in Figure 3-7 (see section 3.3.1).

The instance of mobile-NFC service MAY be identified through several ways, depending on the context and on the area of interest of the notification recipient. This influences the identifiers to be sent in the notification, which also depends on the operation that is notified:

- When the notification sender is the AP, the service instance SHALL be identified by the SIR.
    - In case this notification notifies a service deployment or a service re-deployment (for Device applications re-loading or service re-personalization, or following a Secure Element renewal or a Device change), then the corresponding SIR SHALL be provided in the `New Service Instance Reference` input data.

      The `Current Service Instance Reference` input data SHALL NOT be present.
    - In case this notification notifies a service update, suspension, resumption, or termination, then the reference to the service instance SHALL be provided in the `Current Service Instance Reference` input data.

      The `New Service Instance Reference` input data SHALL NOT be present.
    - In case this notification notifies a service upgrade, then the reference to the current service instance (the `Current Service Instance Reference` input data) and the new reference to the service instance (the `New Service Instance Reference` input data) SHALL be provided.

- When the notification sender is not the AP:
    - If the notification recipient is focusing on the Secure Element (e.g. the SEI), the service instance SHALL be identified by the service identifier and the SE identifier
    - If the notification recipient is focusing on the Mobile Subscription (e.g. the DMSR), the service instance SHALL be identified by the service identifier and the Mobile Subscription identifier

  If the recipient is interested in both the Secure Element and the Mobile Subscription (e.g. the SDM), then both the SE identifier and the Mobile Subscription identifier SHALL be provided.

  In all those use cases:
    - In case this notification notifies a service deployment or a service re-deployment (for Device applications re-loading or service re-personalization, or following a Secure Element renewal or a Device change), then the identification of the service to be (re-) installed (including the version and the qualifier of the service) SHALL be provided in the `New Service` and `New Service Qualifier` input data.

      The `Current Service` and `Current Service Qualifier` input data SHALL NOT be present.

- o In case this notification notifies a service update, suspension, resumption, or termination, then the identification of the service (including the version and the qualifier of the service) SHALL be provided in the `Current Service` and `Current Service Qualifier` input data.

  The `New Service` and `New Service Qualifier` input data SHALL NOT be present.
- o In case this notification notifies a service upgrade, then the identification (including the version and the qualifier of the service) of the current service SHALL be provided in the `Current Service` and `Current Service Qualifier` input data, and the identification of the new version of the service SHALL be provided in the `New Service` and `New Service Qualifier` input data.

The recipient of the notification MAY use this notification to update customer or service statuses in their internal systems, in order to provide up-to-date information to the customer care or the self care interface.

Notification recipients:

- With Service Instance Reference: SDM
- With Secure Element identifier: SEI, SDM
- With Mobile Subscription identifier: DMSR

Functions group: Service Life Cycle Notification

Service Deployment Modes: mode #1, mode #2

Card Content Management modes: Simple mode, Delegated mode, Dual mode

Additional input data:

Input data name	Description	Type	MOC	No.
`Current Service Instance Reference`	The current reference to the mobile-NFC service instance.	`Service Instance Reference`	C	1
`New Service Instance Reference`	The new reference to the mobile-NFC service instance, in case of service (re-) deployment and upgrade.	`Service Instance Reference`	C	1
`Mobile Subscription`	The Mobile Subscription through which the mobile-NFC service is accessible.	`Mobile Subscription Identifier`	O	1
`Secure Element`	The Secure Element into which the mobile-NFC service is deployed/under deployment.	`SE Identifier`	O	1
`Current Service`	The current mobile-NFC service.	`Service Identifier`	C	1
`Current Service Qualifier`	The qualifier of the current mobile-NFC service.	`Service Qualifier`	C	1
`New Service`	The identifier of the new version of mobile-NFC service, in case of service (re-) deployment or upgrade.	`Service Identifier`	C	1
`New Service Qualifier`	The qualifier of the new version of mobile-NFC service, in case of service (re-) deployment or upgrade.	`Service Qualifier`	C	1

Input data name	Description	Type	MOC	No.
`Operation`	The operation to be tracked. The following values are pre-defined: **1 ⇨ Service deployment** **101 ⇨ Service re-deployment** **102 ⇨ Secure Element applications re-deployment** **103 ⇨ Device Applications re-deployment** **104 ⇨ Service re-personalization** **2 ⇨ Service upgrade** **3 ⇨ Service update** **4 ⇨ Service suspension** **5 ⇨ Service resumption** **6 ⇨ Service termination**	`Integer`	M	1
`Operation Status`	The status of the operation performed on the instance of the service. The `Execution Status` type is re-used to specify the result of processing of the operation, and optionally to provide information on any encountered problem (status code, data/object that causes the status code, and message to provide textual and human readable explanation of the status code).	`Execution Status`	M	1

Table 3-26: Additional input parameters for `HandleEndServiceStateChangeNotification` functions

Note: the `Service Instance Reference`, the `Mobile Subscription Identifier`, the `SE Identifier`, the `Service Identifier` and the `Service Qualifier` types are defined in section 3.1.1.

Note: the `Execution Status` type is defined in section 3.1.3.

This notification function does not have any output data.

3.2.4 Mobile Subscription Life Cycle

Some events related to the Mobile Subscription, and that may have an impact on the mobile-NFC service lifecycle, may occur: the identifier of the Mobile Subscription has changed, the Mobile Subscription status has changed.

Such events might be known first by the DMSR, but other roles of the ecosystem may required to be aware about them to update their internal system, and potentially to perform some actions on the mobile-NFC service.

The following sections define the various notifications related to the Mobile Subscription lifecycle that MAY be generated.

3.2.4.1 The "Get Mobile Subscription Alternate Identifier" Function

Name: `GetMobileSubscriptionAlternateIdentifier`

Description:

This function enables to request for an alternate identifier (the `Alternate Mobile Subscription Identifier` output data) of a Mobile Subscription (identified by the `Mobile Subscription Identifier` input data).

This function MAY be used by the SDM for example to get an alias of the given Mobile Subscription identifier. This is especially useful for ensuring confidentiality over the overall system of a Mobile Subscription identifier initially provided by the end-user (e.g. the MSISDN, for example).

The identifier of the service is optionally provided (the `Service` and the optional `Service Qualifier`). If the alternate identifier computation is based on the `Service identifier` or on the `Service Qualifier`, this information MUST be present. Note that computing the alternate identifier by using the identifier of the service leads to specific behaviors on some notifications and some functions calls (e.g. sending several notifications instead of once, providing the service identifier as an additional data, etc.). Those specific behaviors are described in the corresponding functions description.

Note also that the usage of an alternate identifier in an ecosystem is a business agreement between two actors, and not only between two roles: as the goal is to ensure privacy of the original Mobile Subscription identifier, all the roles played by the actor requesting the alternate identifier SHALL use this alternate identifier when dialoging with any of the roles played by the actor that provided it.

In other words, for any further function call or notification between the actor that requested the alternate identifier and the actor that provided it, and that take a `Mobile Subscription Identifier` as input or output data, the alternate Mobile Subscription identifier SHALL be used instead.

Function provider: DMSR

Functions group: Mobile Subscription Info

Additional input data:

Input data name	Description	Type	MOC	No.
`Mobile Subscription Identifier`	The Mobile Subscription identifier.	`Mobile Subscription Identifier`	M	1
`Service`	The mobile-NFC service.	`Service Identifier`	O	1
`Service Qualifier`	The qualifier of the mobile-NFC service. Only present if the `Service Identifier` is present.	`Service Qualifier`	C	1

Table 3-27: Additional input parameters for `GetMobileSubscriptionAlternateIdentifier` function

Note: the `Mobile Subscription Identifier`, the `Service Identifier` and the `Service Qualifier` types are defined in section 3.1.1.

Additional output data:

Output data name	Description	Type	MOC	No.
`Alternate Mobile Subscription Identifier`	The alternate Mobile Subscription identifier to be used in any further function calls taking a Mobile Subscription identifier as input parameter (in place of the former Mobile Subscription identifier value).	`Mobile Subscription Identifier`	M	1

Table 3-28: Additional output parameters for `GetMobileSubscriptionAlternateIdentifier` function

Note: the `Mobile Subscription Identifier` type is defined in section 3.1.1.

3.2.4.2 The "Get SE Mobile Subscription Identifier" Function

Name: `GetSEMobileSubscriptionIdentifier`

Description:

This function enables to request for the identifier of the Mobile Subscription (the `Mobile Subscription Identifier` output data) that shall be used to target a particular Secure Element (identified by the `Secure Element` input data).

This function MAY be used by the SDM in case the Mobile Subscription identifier is not known by itself, or has not been provided by the AP or by another SDM.

In case the ecosystem has decided to use alternate Mobile Subscription identifiers computed based on service identifiers, the service (identified by the `Service` and the optional `Service Qualifier` input data) SHALL be provided. Otherwise, the `Service` and the `Service Qualifier` SHALL NOT be provided.

Function provider: SEI

Functions group: SE Info

Additional input data:

Input data name	Description	Type	MOC	No.
`Secure Element`	The Secure Element into which the mobile-NFC service is deployed/under deployment.	`SE Identifier`	M	1
`Service`	The Mobile-NFC Service that is concerned by the Mobile Subscription retrieval. This field SHALL only be present if the ecosystem has decided to use alternate Mobile Subscription identifiers computed based on service identifiers.	`Service Identifier`	C	1

Input data name	Description	Type	MOC	No.
`Service Qualifier`	Additional information that qualifies the mobile-NFC service that is concerned by the Mobile Subscription retrieval. This field MAY only be present if the ecosystem has decided to use alternate Mobile Subscription identifiers computed based on service identifiers.	`Service Qualifier`	C	1

Table 3-29: Additional input parameters for `GetSEMobileSubscriptionIdentifier` function

Note: the `SE Identifier` type is defined in section 3.1.1.

Additional output data:

Output data name	Description	Type	MOC	No.
`Mobile Subscription Identifier`	The Mobile Subscription identifier to be used to reach the Secure Element.	`Mobile Subscription Identifier`	M	1

Table 3-30: Additional output parameters for `GetSEMobileSubscriptionIdentifier` function

Note: the `Mobile Subscription` Identifier type is defined in section 3.1.1.

3.2.4.3 The "Get Mobile Subscription SE Identifiers" Function

Name: `GetMobileSubscriptionSEIdentifiers`

Description:

This function enables to request for the identifiers of all the Secure Elements (the `Secure Element` output data) that can be currently accessed through a provided Mobile Subscription (identified by the `Mobile Subscription Identifier` input data).

This function MAY be used by the SDM during the first step of the mobile-NFC service life-cycle management, in order to get information about the Secure Elements associated to a Mobile Subscription. This is a best effort function that will only return what the DMSR knows. Therefore not all Secure Elements may be returned.

Function provider: DMSR

Functions group: Mobile Subscription Info

Additional input data:

Input data name	Description	Type	MOC	No.
`Mobile Subscription`	The Mobile Subscription identifier through which the Secure Elements can be currently accessed.	`Mobile Subscription Identifier`	M	1

Table 3-31: Additional input parameters for `GetMobileSubscriptionSEIdentifiers` function

Note: the `Mobile Subscription Identifier` type is defined in section 3.1.1.

Additional output data:

Output data name	Description	Type	MOC	No.
`Secure Element`	A Secure Element that is accessible through the Mobile Subscription. Having several occurrences of this field enables to specify several Secure Elements that are accessible through the same Mobile Subscription.	`SE Identifier`	M	0..N

Table 3-32: Additional output parameters for `GetMobileSubscriptionSEIdentifiers` function

Note: the `SE Identifier` type is defined in section 3.1.1.

3.2.4.4 The "Mobile Subscription Identifier Changed" Notification Function

Name: `HandleMobileSubscriptionIdentifierChangedNotification`

Description:

This function SHALL be called to notify that the identification of the Mobile Subscription has changed. Note that the Mobile Subscription itself is still the same; it is only the identifier which has changed. This new identifier SHALL be used for any later function call taking a `Mobile Subscription Identifier` as input parameter (in place of the former `Mobile Subscription Identifier` value).

For consistency of the system, the "old" identifier (the `Mobile Subscription Old Identifier` input data) and the "new" identifier (the `Mobile Subscription New Identifier` input data) of the Mobile Subscription SHALL be of the same type (e.g. two MSISDN, two Aliases).

This notification is not related to a particular instance of mobile-NFC service.

No output data are expected to this notification.

Notification handler/recipient: SDM

Functions group: Mobile Subscription Life Cycle Notification

Additional input data:

Input data name	Description	Type	MOC	No.
`Mobile Subscription Old Identifier`	Old identifier of the Mobile Subscription.	`Mobile Subscription Identifier`	M	1
`Mobile Subscription New Identifier`	The new identifier of the Mobile Subscription.	`Mobile Subscription Identifier`	M	1

Table 3-33: Additional input parameters for `HandleMobileSubscriptionIdentifierChangedNotification` function

Note: the `Mobile Subscription Identifier` type is defined in section 3.1.1.

3.2.4.5 The "Mobile Subscription Status Change" Notification Function

Name: `HandleMobileSubscriptionStatusChangeNotification`

Description:

This function SHALL be called to notify of a global status change on the Mobile Subscription. Four possible statuses may be notified:

- **`Suspended`**: the Mobile Subscription is temporarily suspended and SHOULD temporarily not use its NFC services. Another notification may be received if the Mobile Subscription is reactivated or definitely terminated.

  An example of suspension is that the end-user has billing payment problems with the Mobile Subscription provider, which suspends the subscription for the end-user.
- **`Activated`**: the Mobile Subscription is available again. This notification follows a previous suspension or restriction of the Mobile Subscription.

  An example of reactivation is that the end-user has paid its bills to the MNO, so the DMSR reactivates it
- **`Terminated`**: the Mobile Subscription is definitely not available. Its NFC services SHALL NOT be used anymore. No other function or notification related to the end-user SHOULD be sent or received by any actor of the NFC ecosystem.

  This notification may have been sent directly, or after a previous suspension or restriction.

  An example of termination is that the end-user is no longer a subscriber of the MNO, so the DMSR terminates it.
- **`Restricted`**: the Mobile Subscription is restricted but the SE and the Device are still reachable OTA.

  Typically, this status is an intermediate status before subscription termination or suspension: the SE and the Device may not be reachable OTA when suspended or terminated, so the Mobile Subscription might be put in the restricted status to still enable OTA communication with the SE and the Device, for example to perform some application data backup actions.

An optional date (the `Date` input data) MAY be provided as input data of the notification to specify when the suspension, activation, restriction or termination has been or will be done:

- If the date is in the past, it represents the date when the suspension, activation, restriction or termination has been done,
- If the date is in the future, it represents the date when the suspension, activation, restriction or termination will be done. This enables the notification receiver to perform actions prior the suspension, activation, restriction or termination, if required.

This notification is generally not related to a particular instance of mobile-NFC service. It is up to the notification recipient to perform the related actions for each of the single mobile-NFC services that are deployed for this Mobile Subscription identifier. However, as alternate identifiers might have been requested for representing the Mobile Subscription (optionally using the Service Identifier as diversification data), the notification SHALL be sent as many times as there are different Mobile Subscription identifiers representing the Mobile Subscription.

No output data are expected to accompany this notification.

Notification handler/recipient: SDM

Functions group: Mobile Subscription Life Cycle Notification

Additional input data:

Input data name	Description	Type	MOC	No.
`Mobile Subscription`	The Mobile Subscription.	`Mobile Subscription Identifier`	M	1
`Date`	The date when the suspension, activation, restriction or termination has been or will be done.	`DateTime`	O	1

`New Status`	The new status of the Mobile Subscription.
	The values are:
	`'Suspended'`
	`'Activated'`
	`'Restricted'`
	`'Terminated'`
		`Enumeration {Suspended, Activated, Terminated, Restricted}`	M	1

`Reason`	The reason of the suspension, activation, restriction or termination. The following values are pre-defined, but other values MAY be defined by the function provider: (for suspension, restriction and termination) **1 ⇨ Subscription suspended** **2 ⇨ Subscription restricted** **3 ⇨ Subscription terminated** **4 ⇨ Subscription contract owner changed (e.g. a parent passes the contract on to a child who comes of age)** **11 ⇨ NFC suspended**	`Integer`	O	1

Input data name	Description	Type	MOC	No.
	12 ⇨ NFC terminated **21 ⇨ SE Stolen** **22 ⇨ SE Lost** **23 ⇨ SE Lost or Stolen (in case the notifier cannot differentiate Stolen and Lost process)** **24 ⇨ SE Broken** **31 ⇨ Device Stolen** **32 ⇨ Device Lost** **33 ⇨ Device Lost or Stolen (in case the notifier cannot differentiate Stolen and Lost process)** **34 ⇨ Device Broken** **41 ⇨ SE + Device Stolen** **42 ⇨ SE + Device Lost** **43 ⇨ SE + Device Lost or Stolen (in case the notifier cannot differentiate Stolen and Lost process)** (for reactivation) **101 ⇨ Subscription restored** **111 ⇨ NFC restored** **121 ⇨ SE recovered (following theft or loss)** **122 ⇨ SE renewed – No eligibility info** **123 ⇨ SE renewed – SE is eligible** **124 ⇨ SE renewed – SE is not eligible** **131 ⇨ Device recovered (following theft or loss)** **132 ⇨ Device changed – No eligibility info** **133 ⇨ Device changed – Device is eligible** **134 ⇨ Device changed – Device is not eligible** **141 ⇨ SE + Device recovered**			

Table 3-34: Additional input parameters for `HandleMobileSubscriptionStatusChangeNotification` function

Note: the `Mobile Subscription Identifier` type is defined in section 3.1.1.

3.2.5 Secure Element Life Cycle Notifications

During the overall life of the Secure Element, some events that have impact on the mobile-NFC service life cycle may occur: the Device hosting the SE UI has changed, the SE is renewed and replaced by a new one, the end-user handling the SE has changed of Mobile Subscription, etc.

Such events might be known first by the SEI or by the DMSR, but other roles of the ecosystem may required to be aware about them to update their internal system, and potentially to perform some actions on the mobile-NFC service.

The following sections define the various notifications related to the Secure Element life cycle that MAY be generated.

3.2.5.1 The "Secure Element Renewal" Notification Function

Name: `HandleSERenewalNotification`

Description:

This function SHALL be called to notify that the Secure Element has been renewed. Renewal means that the previous Secure Element is no longer alive and accessible.

Note that in the scope of this version of specification, this entry point does not address the change of Secure Element Issuer: the new SE is issued by the same Secure Element Issuer as the old one.

This notification is not related to a particular instance of mobile-NFC service. It is up to the notification recipient to perform the related actions for each of the single mobile-NFC services that are deployed on this Secure Element.

No output data are expected to this notification.

Notification handler/recipient: SDM

Functions group: SE Life Cycle Notification

Additional input data:

Input data name	Description	Type	MOC	No.
`Old Secure Element`	The identifier of the old Secure Element.	`SE Identifier`	M	1
`New Secure Element`	The identifier of the new Secure Element.	`SE Identifier`	M	1

Table 3-35: Additional input parameters for `HandleSERenewalNotification` function

Note: the `SE Identifier` type is defined in section 3.1.1.

3.2.5.2 The "Change SE Device" Notification Function

Name: `HandleSEDeviceChangedNotification`

Description:

This function SHALL be called to notify that the Device hosting the SE has changed. This notification corresponds only to a change of the device; the SE and the Mobile Subscription used to target the SE are unchanged:

- If the Secure Element is a UICC: this notification means that the UICC has been inserted in another Device. However, the Secure Element and the Device are still accessible OTA using the same Mobile Subscription Identifier.

- If the Secure Element is a SMC: this notification means that the SMC has been inserted in another Device. Potentially, the Secure Element and the Device are no longer accessible OTA using the same Mobile Subscription Identifier. If so, a `HandleSEMobileSubscriptionChangedNotification` function SHALL also be called.
- If the Secure Element is an ESE: this notification is not applicable as the SE is soldered in the Device.

This notification is generally not related to a particular instance of mobile-NFC service. It is up to the notification recipient to perform the related actions for each of the single mobile-NFC services that are deployed for this Secure Element.

Note that if the identifier of the Mobile Subscription through which the SE is accessible is known by the notifier, then it MAY provide it (in the `Mobile Subscription` input data): it might help the notification recipient to reach the Secure Element or the Device in case it is required to perform actions on them. However, as alternate identifiers might have been requested for representing the Mobile Subscription (optionally using the Service Identifier as diversification data), the notification SHALL be sent as many times as there are different Mobile Subscription identifiers representing the Mobile Subscription.

If Mobile Subscription identifier is not known by the notification sender, and if it is required by the notification recipient, then the notification recipient is responsible to retrieve the Mobile Subscription identifier based on the Secure Element identifier provided in the notification (by using its own information system or by asking another role for example through the `GetSEMobileSubscriptionIdentifier` function – see section 3.2.4.2).

No output data are expected to this notification.

Notification handler/recipient: SDM

Functions group: SE Life Cycle Notification

Additional input data:

Input data name	Description	Type	MOC	No.
`Secure Element`	The Secure Element.	`SE Identifier`	M	1
`Mobile Subscription`	The Mobile Subscription through which the Secure Element is accessible.	`Mobile Subscription Identifier`	O	1
`Old Device Capability Profile Id`	The identifier of the profile of the old Device the SE was inserted in.	`Integer`	M	1
`New Device Capability Profile Id`	The identifier of the profile of the new Device into which the SE has been inserted.	`Integer`	M	1

Table 3-36 Additional input parameters for `HandleSEDeviceChangedNotification` function

Note: the `SE Identifier` and the `Mobile Subscription Identifier` types are defined in section 3.1.1.

3.2.5.3 The "SE Device Status Change" Notification Function

Name: `HandleSEDeviceStatusChangeNotification`

Description:

This function SHALL be called to notify of an event related to the Device that is hosting a Secure Element: device lost, device stolen, etc. Note that the Device change event is managed by the `HandleSEDeviceChangedNotification` (see section 3.2.5.2) notification, so is out of scope of the `HandleSEDeviceStatusChangeNotification` notification function.

This notification is generally not related to a particular instance of mobile-NFC service. It is up to the notification recipient to perform the related actions for each of the single mobile-NFC services that are deployed on this Device.

Note that if the identifier of the Mobile Subscription through which the SE is accessible is known by the notifier, then it MAY provide it (in the `Mobile Subscription` input data): it might help the notification recipient to reach the Secure Element or the Device in case it is required to perform actions on them. However, as alternate identifiers might have been requested for representing the Mobile Subscription (optionally using the Service Identifier as diversification data), the notification SHALL be sent as many times as there are different Mobile Subscription identifiers representing the Mobile Subscription.

If Mobile Subscription identifier is not known by the notification sender, and if it is required by the notification recipient, then the notification recipient is responsible to retrieve the Mobile Subscription identifier based on the Secure Element identifier provided in the notification (by using its own information system or by asking another role for example through the `GetSEMobileSubscriptionIdentifier` function – see section 3.2.4.2).

No output data are expected to this notification.

Notification handler/recipient: SDM

Functions group: SE Life Cycle Notification

Additional input data:

Input data name	Description	Type	MOC	No.
`Secure Element`	The Secure Element.	`SE Identifier`	M	1
`Mobile Subscription`	The Mobile Subscription through which the Device is accessible.	`Mobile Subscription Identifier`	O	1
`Event`	The event that caused the Device status to change. The following values are pre-defined, but other values MAY be defined by the function provider: **31 ⇨ Device Stolen** **32 ⇨ Device Lost** **33⇨ Device Lost or Stolen (in case the notifier cannot differentiate Stolen and Lost processes)**	`Integer`	M	1

Input data name	Description	Type	MOC	No.
	34 ⇨ Device Broken **131 ⇨ Recovered (following theft or loss)**			

Table 3-37: Additional input parameters for `HandleSEDeviceStatusChangeNotification` function

Note: the `SE Identifier` and the `Mobile Subscription Identifier` type are defined in section 3.1.1.

3.2.5.4 The "SE Mobile Subscription Changed" Notification Function

Name: `HandleSEMobileSubscriptionChangedNotification`

Description:

This function SHALL be called to notify that the Mobile Subscription that is used for reaching by OTA the Secure Element and the Device has changed. It does not refer to a change of the identifier of the Mobile Subscription (notified by the `HandleMobileSubscriptionIdentifierChangedNotification` function); but, it really refers to a change of the Mobile Subscription itself.

This notification is generally not related to a particular instance of mobile-NFC service. It is up to the notification recipient to perform the related actions for each of the single mobile-NFC services that are deployed for this Mobile Subscription. However, as alternate identifiers might have been requested for representing the Mobile Subscription (optionally using the Service Identifier as diversification data), the notification SHALL be sent as many times as there are different Mobile Subscription identifiers representing the Mobile Subscription.

For consistency of the system, the identifier of the "old" Mobile Subscription (the `Old Mobile Subscription` input data) and the identifier of the "new" Mobile Subscription (the `New Mobile Subscription` input data) SHALL be of the same type (e.g. two MSISDN, two Aliases).

No output data are expected to this notification.

Notification handler/recipient: SDM

Functions group: SE Life Cycle Notification

Additional input data:

Input data name	Description	Type	MOC	No.
`Secure Element`	The Secure Element.	`SE Identifier`	M	1
`Old Mobile Subscription`	The old Mobile Subscription the Secure Element was accessible through.	`Mobile Subscription Identifier`	M	1

Input data name	Description	Type	MOC	No.
`New Mobile Subscription`	The new Mobile Subscription the Secure Element is now accessible through.	`Mobile Subscription Identifier`	M	1

Table 3-38: Additional input parameters for `HandleSEMobileSubscriptionChangedNotification` function

Note: the `SE Identifier` and the `Mobile Subscription Identifier` types are defined in section 3.1.1.

3.2.5.5 The "Secure Element Status Change" Notification Function

Name: `HandleSEStatusChangeNotification`

Description:

This function SHALL be called to notify of a global status change on the Secure Element. Three possible statuses may be notified:

- **`Suspended`**: the SE is temporarily not available. Another notification may make it be reactivated or definitely terminated.
- **`Activated`**: the SE is available again. This notification follows a previous suspension of the SE.
- **`Terminated`**: the SE is definitely not available. When terminated, only the `HandleSERenewalNotification` notification SHOULD be sent. No other function or notification related to this SE SHOULD be sent or received by any actor of the NFC ecosystem.

  This notification may have been sent after a previous suspension or not.

An optional date (the `Date` input data) MAY be provided as input data of the notification to specify when the suspension, activation or termination has been or will be done:

- If the date is in the past, it represents the date when the suspension, activation or termination has been done,
- If the date is in the future, it represents the date when the suspension, activation or termination will be done. This enables the notification receiver to perform actions prior the suspension, activation or termination, if required, as the SE and the Device may not be reachable OTA when suspended or terminated (UICC case).

This notification is not related to a particular instance of mobile-NFC service. It is up to the notification recipient to perform the related actions for each of the single mobile-NFC services that are deployed on this Secure Element.

No output data are expected to this notification.

Notification handler/recipient: SDM

Functions group: SE Life Cycle Notification

Additional input data:

Input data name	Description	Type	MOC	No.
`Secure Element`	The Secure Element.	`SE Identifier`	M	1
`Date`	The date when the suspension, activation or termination will be done.	`DateTime`	O	1

`New Status`	The new status of the Secure Element.
	The values are:
	`'Suspended'`
	`'Activated'`
	`'Terminated'`
		`Enumeration {Suspended, Activated, Terminated}`	M	1

`Reason`	The reason of the suspension, activation or termination. The following values are pre-defined, but other values MAY be defined by the function provider: (for suspension and termination) **11 ⇨ NFC suspended** **21 ⇨ Stolen** **22 ⇨ Lost** **23⇨ Lost or Stolen (in case the notifier cannot differentiate Stolen and Lost processes)** **31 ⇨ Locked** (for reactivation) **111 ⇨ NFC restored** **121 ⇨ Recovered (following theft or loss)** **131 ⇨ Un-locked**	`Integer`	O	1

Table 3-39: Additional input parameters for `HandleSEStatusChangeNotification` function

Note: the `SE Identifier` type is defined in section 3.1.1.

3.3 Functions for Mobile-NFC Service Management

The mobile-NFC service deployment is a complex workflow that implies an important knowledge on the Secure Element and on the Device structure and content, and that requires a sequencing of various single actions on these SE and Device. This is the SDM role responsibility to manage this global mobile-NFC service management activity, and to hide this complexity behind a simple "service management" API that is mainly called by Application Providers.

The following functions, related to global mobile-NFC service management, are described in this section:

- Management of the SIR:
    - `LookupServiceInstanceReference` (see section 3.3.2.1)
    - `DeclareServiceInstanceReference` (see section 3.3.2.2)
    - `GetServiceInstanceReferenceDescriptor` (see section 3.3.2.3)
- Management of the services:
    - `GetServiceState` (see section 3.3.2.4)
    - `DeployService` (see section 3.3.2.5)
    - `UpgradeService` (see section 3.3.2.6)
    - `ExchangeServiceData` (see section 3.3.2.7)
    - `SuspendOrResumeService` (see section 3.3.2.8)
    - `TerminateService` (see section 3.3.2.9)
- Notifications on/around the services:
    - `HandleServiceEnvironmentChangeNotification` (see section 3.3.3.1)
    - `HandleActionDoneOnServiceNotification` (see section 3.3.3.2)

Note that:

- `DeployService` function could either be used for:
    - The initial installation, personalization and activation a new service instance on a mobile environment,
    - The service instance re-deployment: the partial or full re-delivery of an already existing service instance. May be:
        - The full service instance re-deployment, for example following a mobile environment loss or renewal,
        - The Secure Element applications re-deployment, for example following a SE renewal,
        - The Device applications re-deployment, for example following a Device renewal,
        - The service instance re-personalization, if the usage of the `ExchangeServiceData` function is not possible for updating the whole personalization data.

- `UpgradeService` function is to be used for upgrading a service instance to another version of the same service.
- `ExchangeServiceData` function is to be used for updating internal data of the service instance, without impacting the version of the service (e.g. resetting counters).

3.3.1 Service Instance Life Cycle

The "mobile-NFC management" API provides high level functions for service management. In this particular context, the `Service Instance Reference` is used to identify the exact mobile-NFC service instance that is deployed into a mobile environment (see section 3.1.1.2).

The service "instance" represents the overall set of mobile-NFC service components that are deployed (or under deployment) into a mobile environment for a specific user. The components of the service and the default states are defined by the Application Provider.

This service instance has a specific life cycle that is defined below:

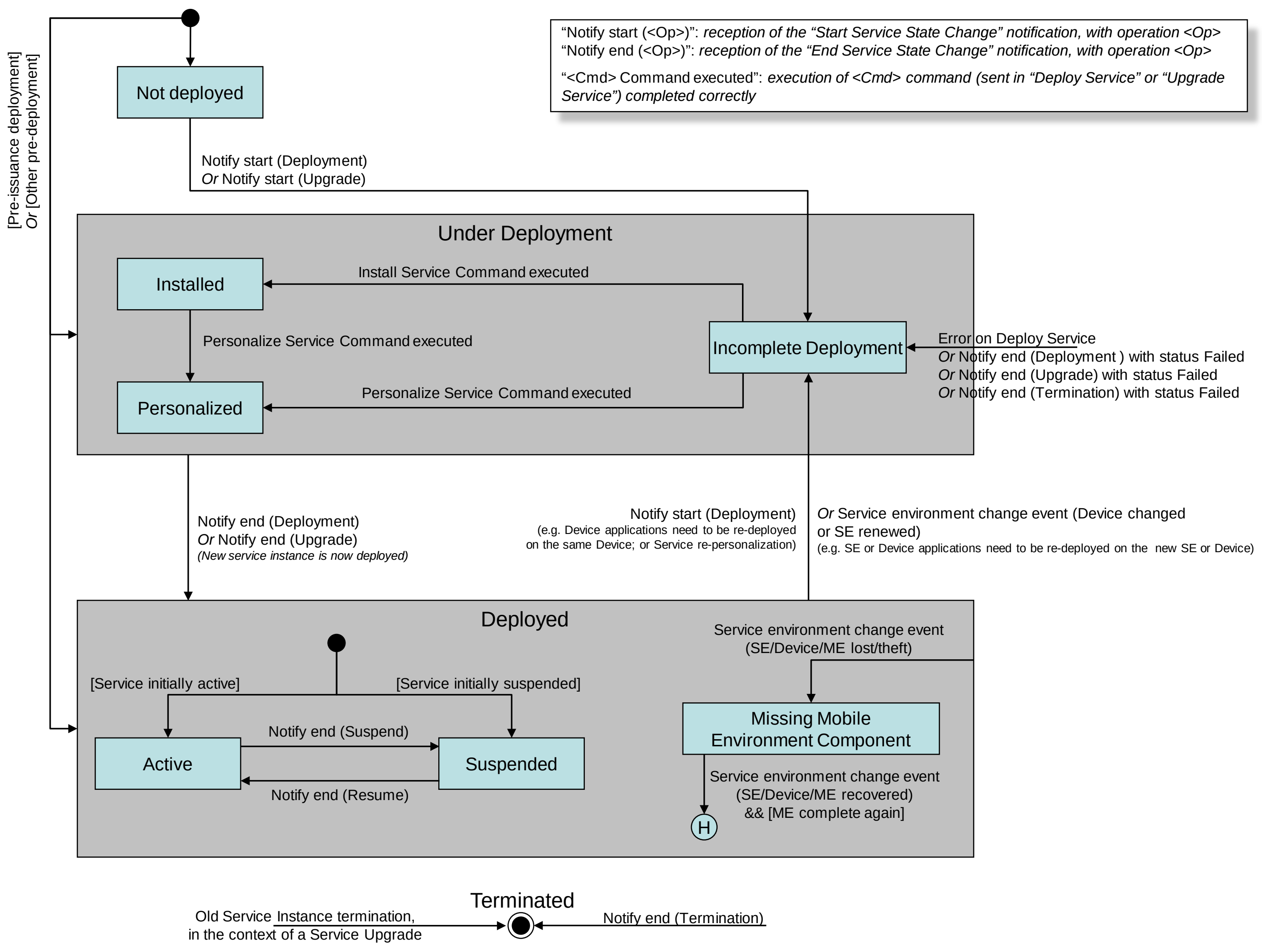

"Notify start (<Op>)": reception of the "Start Service State Change" notification, with operation <Op>
"Notify end (<Op>)": reception of the "End Service State Change" notification, with operation <Op>
"<Cmd> Command executed": execution of <Cmd> command (sent in "Deploy Service" or "Upgrade Service") completed correctly
[Pre-issuance deployment] Or [Other pre-deployment]
Not deployed
Notify start (Deployment) Or Notify start (Upgrade)
Under Deployment
Installed
Install Service Command executed
Personalize Service Command executed
Incomplete Deployment
Error on Deploy Service Or Notify end (Deployment) with status Failed Or Notify end (Upgrade) with status Failed Or Notify end (Termination) with status Failed
Personalized
Personalize Service Command executed
Notify end (Deployment) Or Notify end (Upgrade) (New service instance is now deployed)
Notify start (Deployment) (e.g. Device applications need to be re-deployed on the same Device; or Service re-personalization)
Or Service environment change event (Device changed or SE renewed) (e.g. SE or Device applications need to be re-deployed on the new SE or Device)
Deployed
Service environment change event (SE/Device/ME lost/theft)
[Service initially active]
[Service initially suspended]
Notify end (Suspend)
Active
Suspended
Notify end (Resume)
Missing Mobile Environment Component
Service environment change event (SE/Device/ME recovered) && [ME complete again]
H
Terminated
Old Service Instance termination, in the context of a Service Upgrade
Notify end (Termination)

Figure 3-7: Mobile-NFC service instance life cycle

Initial Service deployment:

The initial state of the service instance is `Not Deployed`. In this state, none of the components of the mobile-NFC service (Secure Element application or Device application) is deployed. However, the `Service Instance Reference` might have been already created, following a call to the `LookupServiceInstanceReference` function (see section 3.3.2.1) or the `DeclareServiceInstanceReference` (see section 3.3.2.2).

This SIR will remain valid whatever the events on the mobile environment (Device changed, Secure Element renewed, Mobile Subscription changed, etc.). Valid SIR means the corresponding service instance is not in the `Terminated` state.

The SIR only becomes invalid:

- When the service instance is explicitely terminated (through a call to the `TerminateService` function, see section 3.3.2.9),
- Or if the service is upgraded (through a call to the `UpgradeService` function): the current service instance is then terminated (and the corresponding SIR is then invalidated) and a new service instance corresponding to the new version of the mobile-NFC service is immediately created (with a new declared or looked up SIR).

Invalid SIR means the corresponding service instance is in the `Terminated` state.

The notification of the beginning of the "Service deployment" operation (through the call to the `HandleStartServiceStateChangeNotification`, see section 3.2.3.1) makes the service instance go into the `Under Deployment:Incomplete Deployment` state. Invocation of the `DeployService` function with relevant Commands (see section 3.3.2.5) really starts the deployment of the service instance:

- The service instance reaches the `Under Deployment:Installed` state when the `Install Service Command` execution is completed, i.e. when the SE applications and Device applications as defined by the AP are loaded and instantiated.

  Note that at that stage, the applications are neither personalized nor activated yet. For that, the personalization and activation steps have to be executed.
- Personalization of the service is done through the execution of the `Personalize Service Command`. This Command might have been provided in the same call to `DeployService`, or through a separated `DeployService` function call. The service instance state then becomes `Under Deployment:Personalized` at the end of execution of this Command.
- The service instance may then directly become `Deployed`, or a final activation step may be required in order the mobile-NFC service to be operational. If required, the execution of the `Activate Service Command` finalizes the mobile-NFC service deployment. Again, this Command might have been provided in the same call to `DeployService`, or through a separated `DeployService` function call.

The service instance finally reaches the `Deployed` state when the notification of the end of the Deployment operation is received (through the call to the `HandleEndServiceStateChangeNotification`, see section 3.2.3.2). Depending on the agreed upon initial `Deployed` state to be reached at the end of the deployment process, the service instance will go into the `Deployed:Active` or into the `Deployed:Suspended` state. This agreed upon initial `Deployed` sub-state is to be specified by the Application Provider, as a business agreement.

Note that the personalization as well as the activation steps are both optional, and the service instance might become `Deployed` directly at the end of the installation step (when receiving the `HandleEndServiceStateChangeNotification` notification).

Service suspension/resumption:

When in the `Deployed:Active` state, a call to the `SuspendOrResumeService` function (see section 3.3.2.8) changes the state of the service instance to `Deployed:Suspended`; another call to the `SuspendOrResumeService` function brings the service instance back to `Deployed:Active`.

Mobile environment events:

During the overall life of the service instance, there might be several changes in the mobile environment of the service:

- If the Secure Element is renewed or the Device is changed, then the service instance reaches the `Under Deployment:Incomplete Deployment` state. In this state, the service instance waits for a call to the `DeployService` function for the re-deployment of the missing service components: Secure Element applications or Device applications.

  In this call to `DeployService`, the AP might need to provide new service personalization information (mainly if SE has changed), or might want to specify that only the Device application should be re-installed (mainly if only the Device has been renewed), if this information is known by the function caller. Otherwise, the function provider should analyze the mobile environment content and re-deploy what is required.

- If either the Secure Element, or the Device, or the overall mobile environment is lost or stolen, the mobile-NFC service is no longer fully operational from end-user point of view. The service instance is then set in the `Deployed:Missing Mobile Environment Component` state, waiting for:

  - Either the broken {Secure Element, Device} pair to be re-associated (for example because the lost Device is recovered)

    If the mobile environment is complete again, then the mobile-NFC service is considered as operational again: the service instance comes back to the exact state it was in before reaching the `Deployed:Missing Mobile Environment Component` state.

  - Or the Secure Element to be renewed: the SE renewal process described above applies.

  - On the Device to be changed: the Device change process described above applies.

Service upgrade:

It is possible to upgrade the service instance to another version of the same service, through a call to the `UpgradeService` function (see section 3.3.2.6). If so, a new service instance (the service in its new version) will replace the old one (the currently available service instance): a new state chart will represent the state of the new service instance:

- The old state chart corresponds to the current service instance. It reaches the `Terminated` state.

- The new state chart corresponds to the new service instance. This new state chart starts and enters in the `Under Deployment:Incomplete Deployment` state as soon as the `HandleStartServiceStateChangeNotification` notification with the "Service upgrade" operation is received.

  The state of this new service instance then evolves according to the Service Commands provided in the `UpgradeService` function call.

Note that the exact termination time of the current (i.e. old) service instance may happen at any time during the overall upgrade process: at reception of the `HandleStartServiceStateChangeNotification` notification, or at reception of the `UpgradeService` function call, or during the `UpgradeService` function execution, or even at the reception of the `HandleEndServiceStateChangeNotification` with the "Service upgrade" operation. Such timing is implementation dependent, and depends on the service components technology, on which components need to be upgraded, on how the SDM performs the overall upgrade process, etc. For instance, in case the service upgrade leads to the upgrade of a Secure Element application, then the current application and Executable Load File will most probably be removed from the SE at the very beginning of the process (at least before the loading of the new Executable Load File): the current service will then be terminated at the beginning of the `UpgradeService` function execution.

In any case, all parties shall ensure that the current service instance has reached the `Terminated` state when the `HandleEndServiceStateChangeNotification` notification sending (for the notification sender) and processing (for the notification recipient) are completed.

Service Termination:

Whatever the service instance state, the termination of the Service can be requested through a call to the `TerminateService` function (see section 3.3.2.9). The service instance however does not reach the `Terminated` state immediately, as there might be some complementary actions required for this termination (on the caller back-end system, for example).

The service instance state then only changes to `Terminated` when the `HandleEndServiceStateChangeNotification` function is received (notification of the end of processing of the "Terminate service" operation). The corresponding `Service Instance Reference` then becomes invalid.

Miscellaneous remarks:

- In case the mobile-NFC service is already deployed in the mobile environment in pre-issuance, then the service instance is directly in one of the `Under Deployment` or `Deployed` sub-states, depending on what has been done in factory.
- In case part or all of the service has been already deployed in the mobile environment following a business agreement between parties, then the service instance might be in one of the `Under Deployment` or `Deployed` sub-states.
- In case an error occurs during the processing of any call to the `DeployService` or `UpgradeService` function, or in case of notification of the end of the "Service deployment" or "Service upgrade" operations with a failed status, the service may become in the `Under Deployment:Incomplete Deployment` state if some service components are missing in the mobile environment of the end-user (either in the Secure Element or in the Device).

- From any of the `Deployed` sub-states it is possible to notify of the start of the "Service deployment" operation and to call the `DeployService` function with the `DeviceOnly` boolean set to '`True`'. This is especially the case if the Application Provider has been directly notified by the end-user of a Device change, and wants to force the re-delivery of the Device applications of the service instance. If so, the service instance is going to the `Under Deployment:Incomplete Deployment` state, and then moves to the `Deployed` sub-state when the end of the "Service deployment" operation is notified. It can also go through the intermediary `Under Deployment:Installed` or the `Under Deployment:Personalized` states depending on the remaining actions to be done for the re-deployment of the device applications.
- In case of notification of the end of the "Service termination" operation with a failed status, the service becomes in the `Under Deployment:Incomplete Deployment` state as the termination is explicitely declared as non completed. The SIR also remains valid.

  From this state, a re-deployment is possible (call to the `DeployService` function), as well as a new notification of the service termination with a succeeded status.
- When a SIR is invalidated, its value is considererd as free to be re-used. It means that:
  - It is possible for the AP to use the exact same SIR value in a new call to the `DeclareServiceInstanceReference` function, regardless if it is for the same mobile-NFC service (in the same or a different version) or for another mobile-NFC service. Only applicable if the SIR was previously declared by the AP.
  - It is possible for the SDM to use the exact same SIR value in a new call to the `LookupServiceInstanceReference` function, regardless if it is for the same mobile-NFC service (in the same or a different version) or for another mobile-NFC service. Only applicable if the SIR was previously generated by the SDM.

Note that the current status of a service instance can be checked via the `GetServiceState` function (see section 3.3.2.4).

3.3.2 Global Service Management

3.3.2.1 The "Lookup Service Instance Reference" Function

Name: `LookupServiceInstanceReference`

Description:

For Application Provider point of view, a mobile-NFC service deployed into a mobile environment is identified by the SIR. Such a reference SHALL be used for all the Global Service Management functions described in section 3.3.2.

This reference remains valid whatever events that may happen during the life of the service instance: Secure Element renewal, Device change, Mobile Subscription change, etc.

When calling the `LookupServiceInstanceReference` function, the function caller:

- Specifies the service that is to be managed (identified by the `Service` and the optional `Service Qualifier` input data),
- And identifies the Secure Element into which the service is or should be deployed:
    - Either directly, by providing the SE identifier (the `Secure Element` input data),
    - Or indirectly, by providing the Mobile Subscription through which the SE can be currently reached (the `Mobile Subscription` input data). In that case, the mapping between the Mobile Subscription and the Secure Element identifier has to be known by the function provider. In case several Secure Elements might be accessible through the given Mobile Subscription, the AP should accept to delegate the SDM the decision of which SE to be used: the SDM will know the best fit of SE given a specific service, as mentioned in the `CheckGlobalEligibility` function (see section 3.2.1.1).

  It is mandated to provide either the identifier of the Secure Element or the identifier of the Mobile Subscription as input data.

The function provider SHALL then determine and return a SIR (the `Service Instance Reference` output data) and the corresponding state of the service instance (the `Service Instance State` output data).

The function provider SHALL check that the service and the Secure Elements are known and defined before determining the SIR. Otherwise an error SHOULD be returned to the function caller: the `Status` field of the `Function Execution Status` SHOULD then returned with a value set to `'Executed-Failed'`, and with a relevant `Status Code` explaining the error.

The function provider SHALL store the determined reference as the reference of the service instance.

Any subsequent calls to any other global mobile-NFC service management functions require this `Service Instance Reference`.

Note that:

- In case no component of the service is present in the mobile environment at the `LookupServiceInstanceReference` function calling time, then the `Service Instance State` SHALL be set to `'Not Deployed'`.
- In case service components are already deployed in the mobile environment at the `LookupServiceInstanceReference` function calling time (either during a pre-issuance phase of the SE or of the Device, or by a previous call to service management functions), then the `Service Instance State` SHALL represent the current service instance state.

Note also that the `LookupServiceInstanceReference` function can be called several times by the same function requester, providing either the same input data, or updated input data following events on the mobile environment (e.g. a new SE Id, following a SE renewal). In both cases, the function provider SHALL retrieve the service instance corresponding to the provided input data, and SHALL return the existing SIR and the state of the service instance. Calls with old data (e.g. using the former SE Id) are considered as a lookup for a new service.

Function provider: SDM

Functions group: Global Service Management

Service Deployment Modes: mode #1, mode #2

Additional input data:

Input data name	Description	Type	MOC	No.
`Service`	The Mobile-NFC Service to be managed.	`Service Identifier`	M	1
`Service Qualifier`	Additional information that qualifies the mobile-NFC service.	`Service Qualifier`	C	1
`Secure Element`	The Secure Element hosting the mobile-NFC service instance.	`SE Identifier`	C	1
`Mobile Subscription`	The Mobile Subscription that can be used to currently reach the Secure Element which will host the mobile-NFC service instance.	`Mobile Subscription Identifier`	C	1

Table 3-40 Additional input parameters for `LookupServiceInstanceReference` function

Note: the `Service Identifier`, `Service Qualifier`, `SE Identifier` and `Mobile Subscription Identifier` types are defined in section 3.1.1.

Additional output data:

Output data name	Description	Type	MOC	No.
`Service Instance Reference`	The reference to the mobile-NFC service instance. If no instance is deployed yet, a new reference is created.	`Service Instance Reference`	M	1
`Service Instance State`	The current state of the service instance. The possible values are: **1 ⇨ Not Deployed** **11 ⇨ Installed** **12 ⇨ Personalized** **13 ⇨ Incomplete Deployment** **21 ⇨ Active** **22 ⇨ Suspended** **23 ⇨ Missing Mobile Environment Component**	`Integer`	M	1

Table 3-41 Additional output parameters for `LookupServiceInstanceReference` function

Note: the `Service Instance Reference` type is defined in section 3.1.1.

3.3.2.2 The "Declare Service Instance Reference" Function

Name: `DeclareServiceInstanceReference`

Description:

Contrary to the `LookupServiceInstanceReference` function, the `DeclareServiceInstanceReference` function enables the Application Provider to provide the value of the SIR (the `Service Instance Reference` input data) to be used for any subsequent call to any other global mobile-NFC service management function. This enables the Application Providers to use an own business identifier already managed in their information system, in place of a reference generated by the SDM, and that they would have to store in their information systems.

This reference remains valid regardless of what events that may happen during the life of the service instance: Secure Element renewal, Device change, Mobile Subscription change, etc.

When calling this `DeclareServiceInstanceReference` function, the function caller still:

- Specifies the service that is to be managed (identified by the `Service` and the optional `Service Qualifier` input data),
- And identifies the Secure Element into which the service is or should be deployed:
    - Either directly, by providing the SE identifier (the `Secure Element` input data),
    - Or indirectly, by providing the Mobile Subscription through which the SE can be currently reached (the `Mobile Subscription` input data). In that case, the mapping between the Mobile Subscription and the Secure Element identifier has to be known by the function provider. In case several Secure Elements might be accessible through the given Mobile Subscription, the AP should accept to delegate the SDM the decision of which SE to be used: the SDM will know the best fit of SE given a specific service, as mentioned in the `CheckGlobalEligibility` function (see section 3.2.1.1).

  It is mandated to provide either the identifier of the Secure Element or the identifier of the Mobile Subscription as input data.

The function caller SHALL ensure that the provided reference is unique within its own scope.

The function provider SHALL check that the service and the Secure Elements are known and defined before accepting the SIR declaration. Otherwise an error SHOULD be returned to the function caller: the `Status` field of the `Function Execution Status` SHOULD then returned with a value set to `'Executed-Failed'`, and with a relevant `Status Code` explaining the error.

The function provider SHALL store the provided reference as the reference of the service instance.

Any subsequent calls to any other global mobile-NFC service management functions require this `Service Instance Reference`.

Note that:

- In case no component of the service is present in the mobile environment at the `DeclareServiceInstanceReference` function calling time, then the `Service Instance State` SHALL be set to `'Not Deployed'`.

- In case service components are already deployed in the mobile environment at the `DeclareServiceInstanceReference` function calling time (either during a pre-issuance phase of the SE or of the Device, or by a previous call to service management functions), then the `Service Instance State` SHALL represent the current service instance state.

Note that the `DeclareServiceInstanceReference` function can be called several times using an existing SIR value, providing either the same input data (Service identification, Secure Element identification) or updated input data following events on the mobile environment (e.g. a new SE Id, following a SE renewal). In both cases, the function provider SHALL retrieve the service instance corresponding to the provided SIR and SHALL check that the other input data exactly correspond to the current mobile environment context of the service instance. Otherwise, the function provider SHOULD consider that this is an inconsistent re-declaration of a reference and an error SHOULD be returned to the function caller: the `Status` field of the `Function Execution Status` SHOULD then be returned with a value set to `'Executed-Failed'`, with a relevant `Status Code` explaining the error.

If the `DeclareServiceInstanceReference` function is called with a new SIR value, but with the same input data values as the ones already used in the declaration of a different and still active SIR, then the function provider SHOULD also consider that this is an inconsistent declaration of two different active references on the same service instance. An error SHOULD be returned to the function caller: the Status field of the `Function Execution Status` SHOULD then be returned with a value set to `'Executed-Failed'`, with a relevant `Status Code` explaining the error.

However, if the SIR was invalidated through service instance termination according to the service instance state diagram depicted in Figure 3-7, then a call to the `DeclareServiceInstanceReference` function with a re-used SIR value and the same or different input data values SHOULD be considered as a valid function call.

Function provider: SDM

Functions group: Global Service Management

Service Deployment Modes: mode #1, mode #2

Additional input data:

Input data name	**Description**	**Type**	**MOC**	**No.**
`Service Instance Reference`	The reference to the mobile-NFC service instance.	`Service Instance Reference`	M	1
`Service`	The Mobile-NFC Service to be managed.	`Service Identifier`	M	1
`Service Qualifier`	Additional information that qualifies the mobile-NFC service.	`Service Qualifier`	C	1
`Secure Element`	The Secure Element hosting the mobile-NFC service instance.	`SE Identifier`	C	1
`Mobile Subscription`	The Mobile Subscription that can be used to currently reach the Secure Element which will host the mobile-NFC service instance.	`Mobile Subscription Identifier`	C	1

Table 3-42 Additional input parameters for `DeclareServiceInstanceReference` function

Note: the `Service Identifier`, `Service Qualifier`, `SE Identifier` and `Mobile Subscription Identifier` and `Service Instance Reference` types are defined in section 3.1.1.

Additional output data:

Output data name	Description	Type	MOC	No.
`Service Instance State`	The current state of the service instance. The possible values are: **1 ⇨ Not Deployed** **11 ⇨ Installed** **12 ⇨ Personalized** **13 ⇨ Incomplete Deployment** **21 ⇨ Active** **22 ⇨ Suspended** **23 ⇨ Missing Mobile Environment Component**	`Integer`	M	1

Table 3-43 Additional output parameters for `DeclareServiceInstanceReference` function

3.3.2.3 The "Get Service Instance Reference Descriptor" Function

Name: `GetServiceInstanceReferenceDescriptor`

Description:

The SIR is an opaque (i.e. without standardized capability to retrieve information by just analyzing its value) representation of a specific realization of a mobile-NFC service for an end-user.

When calling the `LookupServiceInstanceReference` or `DeclareServiceInstanceReference` functions to obtain this SIR, various data are provided as input data. In particular, the Secure Element can not only be identified through a SE identifier, but may be identified through the Mobile Subscription through which the SE can be currently reached.

The Application Provider may not be willing to persistently store those data. Moreover, some of them may change during the life of the mobile-NFC service instance, following some environmental change. For example, the Mobile Subscription or the Secure Element might be changed.

The `GetServiceInstanceReferenceDescriptor` function consequently allows retrieving the value of the data that are represented by the SIR, at the time the `GetServiceInstanceReferenceDescriptor` function is called.

In the above examples, the latest Mobile Subscription Identifier and/or Secure Element Identifier SHALL be returned, even if former identifiers have been provided at the initial call to the `LookupServiceInstanceReference` or `DeclareServiceInstanceReference` functions.

The function caller simply provides the reference to the service instance (the `Service Instance Reference` input data), and the function provider returns the identification of the service (the `Service` and the optional `Service Qualifier` output data), and optionally the identifier of the Secure Element on which the service is deployed (the `Secure Element` output data) and the current identifier of Mobile Subscription that is used to reach the Secure Element (the `Mobile Subscription` output data), if available.

Note that the `GetServiceInstanceReferenceDescriptor` function MAY also provide information on a `Terminated` service instance, i.e. accepting a SIR that is no longer valid. If so, the returned data SHOULD correspond to the latest known values at the time of service instance termination. Function providers are NOT REQUIRED to maintain such a history of invalid SIR information, and lifetime of this history information is implementation dependent. As a consequence, the `Status` field of the `Function Execution Status` MAY also be returned with a value set to `'Executed-Failed'`, with a relevant `Status Code` explaining the error (unknown Service Instance Reference).

In case the SIR value has been re-used for another service instance (in between the service termination and the call to `GetServiceInstanceReferenceDescriptor`), then the data of the new service instance SHALL be returned instead.

Function provider: SDM

Functions group: Global Service Management

Service Deployment Modes: mode #1, mode #2

Additional input data:

Input data name	Description	Type	MOC	No.
`Service Instance Reference`	The reference to the mobile-NFC service instance.	`Service Instance Reference`	M	1

Table 3-44 Additional input parameters for `GetServiceInstanceReferenceDescriptor` function

Note: the `Service Instance Reference` type is defined in section 3.1.1.

Additional output data:

Output data name	Description	Type	MOC	No.
`Service`	The Mobile-NFC Service that is referenced.	`Service Identifier`	M	1
`Service Qualifier`	Additional information that qualifies the mobile-NFC service.	`Service Qualifier`	C	1
`Secure Element`	The Secure Element hosting the mobile-NFC service instance.	`SE Identifier`	O	1
`Mobile Subscription`	The Mobile Subscription that is currently used to reach the Secure Element which hosts the mobile-NFC service instance.	`Mobile Subscription Identifier`	O	1

Table 3-45 Additional output parameters for `GetServiceInstanceReferenceDescriptor` function

Note: the `Service Identifier`, `Service Qualifier`, `SE Identifier` and `Mobile Subscription Identifier` types are defined in section 3.1.1.

3.3.2.4 The "Get Service State" Function

Name: `GetServiceState`

Description:

The `GetServiceState` function allows retrieving the state of a mobile-NFC service instance. Service instance states are depicted in Figure 3-7.

The function caller simply provides the reference to the service instance (the `Service Instance Reference` input data), and the function provider returns the current state of the service instance (the `Service Instance State` output data).

State transitions are either triggered explicitly by calls to the service deployment or upgrade functions (see sections 3.3.2.5 and 3.3.2.6) or implicitly by changes to the mobile environment (see section 3.3.3.1).

The resulting state of the mobile-NFC service is consequently also provided as output parameter of those `DeployService` and `UpgradeService` functions, or via the `HandleServiceEnvironmentChangeNotification` notification. However, note that the state of the service might not be obvious to the function caller, especially when a timestamp marks the occurrence of a future life cycle state transition (see section 3.3.3.1).

Note that the `GetServiceState` function MAY indicate that the service is terminated, meaning that the provided SIR is no longer valid. Function providers are however NOT REQUIRED to maintain such a history of invalid SIR information, and lifetime of this history information is implementation dependent. As a consequence, the `Status` field of the `Function Execution Status` MAY also be returned with a value set to `'Executed-Failed'`, with a relevant `Status Code` explaining the error (unknown Service Instance Reference).

In case the SIR value has been re-used for another service instance (in between the service termination and the call to `GetServiceInstanceReferenceDescriptor`), then the state of the new service instance will be returned instead.

Function provider: SDM

Functions group: Global Service Management

Service Deployment Modes: mode #1, mode #2

Additional input data:

Input data name	Description	Type	MOC	No.
`Service Instance Reference`	The reference to the mobile-NFC service instance.	`Service Instance Reference`	M	1

Table 3-46 Additional input parameters for `GetServiceState` function

Note: the `Service Instance Reference` type is defined in section 3.1.1.

Additional output data:

Output data name	Description	Type	MOC	No.
`Service Instance State`	The state of the service instance. The possible values are: **1 ⇨ Not Deployed** **11 ⇨ Installed** **12 ⇨ Personalized** **13 ⇨ Incomplete Deployment** **21 ⇨ Active** **22 ⇨ Suspended** **23 ⇨ Missing Mobile Environment Component** **31 ⇨ Terminated**	`Integer`	M	1

Table 3-47 Additional output parameters for `GetServiceState` function

3.3.2.5 The Service Deployment Function

3.3.2.5.1 The Service Command Concept

The full deployment of a mobile-NFC service may be composed of up to three steps: installation, personalization (optional), and activation, as depicted in Figure 3-7.

The launching of these single steps can be independently requested by the Application Provider. However, in order for example to optimize the communication between the SE and the administration server, or simply to fully delegate the service deployment within a unique function call, it might be required that the AP requests for the execution of several steps at a time.

For that purpose, a generic function for mobile-NFC service deployment, called "Deploy Service", is provided by the SDM role. This function takes a sequence of single "Service Commands" as input parameter, so that the function provider can optimize the script generation and thus the OTA exchanges if required. This function is defined in section 3.3.2.5.2.

Each "Service Command" represents a particular service deployment action among the ones defined in section 3.3.2.5.3. Prior to performing any action, the SDM MAY check that the function caller has the right to perform the requested command. The number and the granularity of such verifications are SDM dependent and are out of scope of this specification.

Even if several Service Commands can be provided at a time:

- Only one Service Command per type can be provided in a single function call.
- Order of the Service Commands is relevant. The function caller SHALL provide the commands in the right order so that the service instance state diagram depicted in Figure 3-7 is respected. The function provider SHOULD also check that the provided commands are correctly ordered.

3.3.2.5.2 The "Deploy Service" Function

Name: `DeployService`

Description:

This function enables to request for the deployment of a service instance which reference (the `Service Instance Reference` input data) has been previously obtained through a call to the `LookupServiceInstanceReference` function, or declared through the call of the `DeclareServiceInstanceReference` function.

The `DeployService` function enables to request one or several "Service Commands" (contained in the list of `Service Command` input data). The possible Service Commands are:

- `Install Service Command` (see section 3.3.2.5.3.1),
- `Personalize Service Command` (see section 3.3.2.5.3.2),
- `Activate Service Command` (see section 3.3.2.5.3.3).

The function caller may request to perform:

- A single `DeployService` call providing the three `Install Service Command`, `Personalize Service Command`, and `Activate Service Command`,
- Or three calls to the `DeployService` function, providing the `Install Service Command` first, then the `Personalize Service Command`, and finally the `Activate Service Command`,
- Or call the `DeployService` function providing any of the service commands (one or several) that fits the service instance state diagram depicted in section 3.3.1.

Only one occurrence of each Service Command type can be present in a call.

The function provider SHALL verify that the service instance is in an allowed state (according to the service instance life cycle depicted in Figure 3-7) before proceeding to the service deployment. Note that, following this state diagram, the `DeployService` function might be called for requesting the re-deployment of the service after a service environment change (e.g. Secure Element renewed or Device changed), or in case the Application Provider wants to force the (re-)deployment of the Device application, or in case the Application Provider wants to force the re-personalization of a service (that might lead, depending on the SE application, to a card application removal and then re-instantiation, if the direct re-personalization of the card application is not possible). Such optional removal and re-instantiation should remain transparent to the function caller, who functionally only requested re-personalization of an already deployed service). As a consequence, the function provider SHALL check if the provided `Service Instance Reference` corresponds to an already deployed service instance. This can for example be detected by checking the following criteria:

- If the same reference has been used for a previous call to the `DeployService` function,
- If the provided reference corresponds to the same service previously deployed (with the same or another reference) on another Secure Element, both former and current Secure Element being linked by a SE renewal event (previously received by the function provider).

In case of successful execution of the `DeployService` function, and depending on the last executed Service Command, the service instance state SHALL be set to the `Installed`, `Personalized`, `Active` or `Suspended` state.

In case of failed execution of the function, the service instance state SHOULD be set to the `Partially Deployed` state if a partial change has occurred to any of the service components in the mobile environment. Otherwise, the service instance status SHOULD remain in the state of the last successfully executed Service Command.

Function provider: SDM

Functions group: Global Service Management

Service Deployment Modes: mode #1, mode #2

Additional input data:

Input data name	Description	Type	MOC	No.
`Service Instance Reference`	The `Service Instance Reference` corresponding to the service instance to be deployed.	`Service Instance Reference`	M	1

Input data name	Description	Type	MOC	No.
`Service Command`	The Service Command to be executed, among the following: • `InstallServiceCommand` • `PersonalizeServiceCommand` • `ActivateServiceCommand` Having several occurrences of this field enables to specify several Service Commands to be executed in a single call. However: • Only one Service Command per type can be provided in a single function call. • Order of the Service Commands is relevant. The function caller SHALL provide the commands in the right order so that the service instance state diagram depicted in Figure 3-7 is respected. The function provider SHOULD also check that the provided commands are correctly ordered. The structure of each single Service Command is described in section 3.3.2.5.3.	`Service Command`	M	1..3
`Device Only`	States whether the function processing should be limited to actions on the Device only, or addresses both the Secure Element and the Device. This is especially useful if the function caller wants to force the re-delivery of the Device applications of the service instance (because for example it has been directly notified by the end-user of a Device change). If so, the service instance is not going through the `Incomplete Deployment` state. If `'True'`, the function processing SHALL be limited to actions on the Device only If `'False'`, the function processing MAY act on the Secure Element as well as on the Device If the field is not present, the function processing MAY act on the Secure Element as well as on the Device.	`Boolean`	O	1

Table 3-48: Additional input parameters for `DeployService` function

Note: the `Service Instance Reference` type is defined in section 3.1.1.

Additional output data:

Output data name	Description	Type	MOC	No.
`Service Command Result`	Provide the single result of execution of a corresponding Service Command. Sequence of result SHALL follow the sequence of `Service Command` provided in the function input data, as specified in section 3.3.2.5.2.1. The `Service Command Result` basic structure is described below. This structure can be complemented by additional output data described for each `Service Command` of section 3.3.2.5.3.	`Service Command Result`	M	1..3
`Service Instance State`	The state of the service instance at the end of execution of the `DeployService` function. The possible values are: **1 ⇨ Not Deployed** **11 ⇨ Installed** **12 ⇨ Personalized** **13 ⇨ Incomplete Deployment** **21 ⇨ Active** **22 ⇨ Suspended** **23 ⇨ Missing Mobile Environment Component**	`Integer`	M	1

Table 3-49: Additional output parameters for `DeployService` function

Where a `Service Command Result` is:

Data name	Description	Type	MOC	No.

`Command Execution Status`	Indicates whether the `Service Command` has been executed or not.
	
	The values are:
	`'Executed-Success'`
	`'Executed-Failed'`
	`'NotExecuted'`
	`'DeliveredWithNoResponse'`
	
	A detailed description of the status is provided in section 3.3.2.5.2.1.
		`Enumeration {Executed-Success,`
		`Executed-Failed,`
		`NotExecuted,`
		`DeliveredWithNoResponse}`
			M	1

Data name	Description	Type	MOC	No.
`Command Status Code Data`	Provides the reason of the `Command Execution Status` value when the `Command Execution Status` is `'Executed-Failed'`. The possible values depend on the `Service Command`. A normative list of codes is provided in section 6 but additional codes may be defined.	`Command Status Code Data`	C	1

Table 3-50: `Service Command Result` type

Where a `Command Status Code Data` is:

Data name	Description	Type	MOC	No.
`Subject`	Represents the system element concerned by the exception.	`Object Identifier`	M	1
`Reason`	Represents the reason of the exception.	`Object Identifier`	M	1

Table 3-51 `Command Status Code Data` type

Optionally, additional output data MAY be returned in each `Service Command Result`. Refer each single `Service Command` definition.

3.3.2.5.2.1 Error Management in Service Commands Processing

The output data of the "Deploy Service" and "Upgrade Service" functions contains a list of Service Command Result: for each single Service Command contained in the function input data, a corresponding Service Command Result MAY be provided in the output data.

The basic principles of the processing of the "Deploy Service" and "Upgrade Service" functions are the following:

- Order of the Service Commands is relevant. The function caller SHALL provide the commands in the right order so that the service instance state diagram depicted in Figure 3-7 is respected. The function provider SHOULD also check that the provided commands are correctly ordered.
- The processing of the "Deploy Service" and "Upgrade Service" functions SHALL stop as soon as an error is encountered in the processing of a Service Command.

Within each single Service Command Result, a command execution status (the `Command Execution Status` output data) SHALL specify whether the overall processing of the Service Command (i.e. up to the Secure Element update) has been processed correctly (`'Executed-Success'`), with error (`'Executed-Failed'`), or not processed at all (`'NotExecuted'`).

The reliability of the OTA connectivity is such that it may happen that even if data has been received and processed by the SE or the Device, the execution status sent back by the SE or the Device over the network may get lost. In such case where the status of execution cannot be determined, the command execution status SHALL be `'DeliveredWithNoResponse'`.

As for any function, a global status is specified in the output parameters: the `Status` field of the `Function Execution Status` output data. Its value, together with the `Status Code Data` of the function (in case of an error or the expiration), depends on the overall statuses of all the `Service Command Result`:

- The global function processing status SHALL be set to `'Executed-Success'` if the execution status of all the Service Command is `'Executed-Success'`. The `Command Status Code Data` of the `Service Command Result` and the `Status Code Data` of the function SHALL NOT be present.
- The global function processing status SHALL be set to `'Failed'` if at least one of the Service Commands has an execution status set to `'Executed-Failed'`.

  The `Command Status Code Data` of the Service Commands in error SHALL detail the reason that caused the Service Command error. This code SHALL also be set in the `Status Code Data` of the function.
- The global function processing status SHALL be set to `'Expired'` if at least one of the Service Command has an execution status set to `'DeliveredWithNoResponse'`.

  The `Command Status Code Data` of the Service Commands in expiration SHALL detail the reason that caused the Service Command expiration. The `Subject` and `Reason` fields of the `Status Code` of the function SHALL be set to the relevant error code (for example, 'OTA transport error' code –see section 6).

Those rules imply that the Service Command Results with `'NotExecuted'` status can only be located at the end of the `Service Command Result` list. As a consequence, it is optional to explicitly put those Service Command Results in the `Service Command Result` list. This is especially the case if an error occurs before the processing of any of the Service Command: the list of Service Command Result MAY either be empty or MAY contain as many Service Command Results as Service Commands, each of them having the `'NotExecuted'` status.

Note that the function validity period may expire:

- If expiration is raised before the concrete execution of actions in the SE or in the Device for the first Service Command, then the corresponding `Service Command Result` SHALL be set to `'NotExecuted'` and this is the responsibility of the function requester system to cancel all server-side actions done for this Service Command.
- If expiration is raised after the concrete execution of actions in the SE or in the Device for the first Service Command, then the corresponding `Service Command Result` SHALL be set to `'DeliveredWithNoResponse'` and the confirmation or cancellation of the actions on the function requester system is function requester-implementation dependant.

As stated before, in both cases, the `Status` field of the `Function Execution Status` output parameter of the function SHALL be set to `'Expired'`.

The following figures give example of the possible statuses of the function execution:

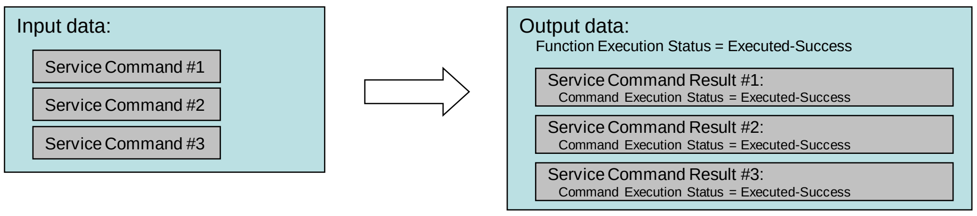

Input data:
Service Command #1
Service Command #2
Service Command #3
Output data:
Function Execution Status = Executed-Success
Service Command Result #1:
Command Execution Status = Executed-Success
Service Command Result #2:
Command Execution Status = Executed-Success
Service Command Result #3:
Command Execution Status = Executed-Success

Figure 3-8: All Service Commands processed successfully

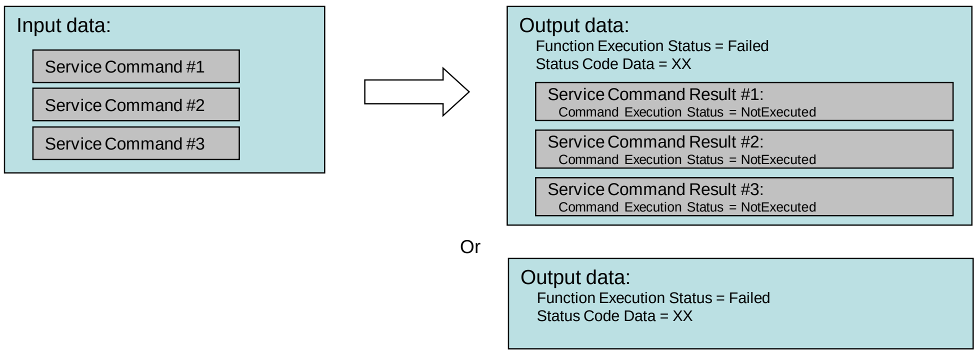

Input data:
Service Command #1
Service Command #2
Service Command #3
Output data:
Function Execution Status = Failed
Status Code Data = XX
Service Command Result #1:
Command Execution Status = NotExecuted
Service Command Result #2:
Command Execution Status = NotExecuted
Service Command Result #3:
Command Execution Status = NotExecuted
Or
Output data:
Function Execution Status = Failed
Status Code Data = XX

Figure 3-9: Error XX before processing any Service Command

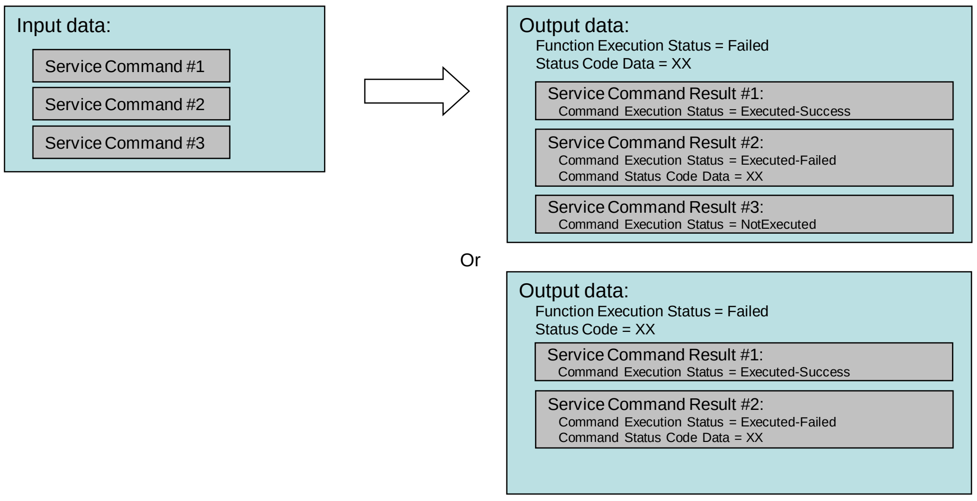

Input data:
Service Command #1
Service Command #2
Service Command #3
Output data:
Function Execution Status = Failed
Status Code Data = XX
Service Command Result #1:
Command Execution Status = Executed-Success
Service Command Result #2:
Command Execution Status = Executed-Failed
Command Status Code Data = XX
Service Command Result #3:
Command Execution Status = NotExecuted
Or
Output data:
Function Execution Status = Failed
Status Code = XX
Service Command Result #1:
Command Execution Status = Executed-Success
Service Command Result #2:
Command Execution Status = Executed-Failed
Command Status Code Data = XX

Figure 3-10: The 2nd Service Command fails with Error 'XX'

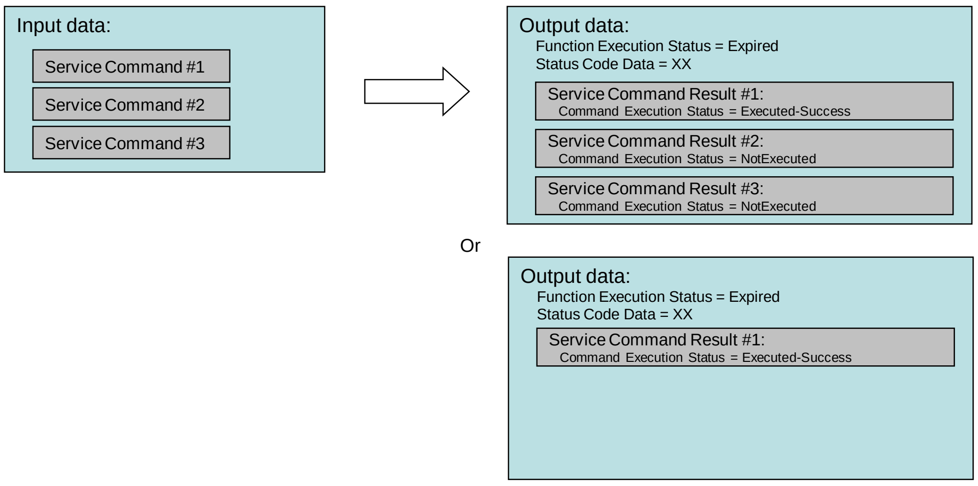

Input data:
Service Command #1
Service Command #2
Service Command #3
Output data:
Function Execution Status = Expired
Status Code Data = XX
Service Command Result #1:
Command Execution Status = Executed-Success
Service Command Result #2:
Command Execution Status = NotExecuted
Service Command Result #3:
Command Execution Status = NotExecuted
Or
Output data:
Function Execution Status = Expired
Status Code Data = XX
Service Command Result #1:
Command Execution Status = Executed-Success

Figure 3-11: The 2nd Service Command expires before the first Service Command execution in the SE or Device

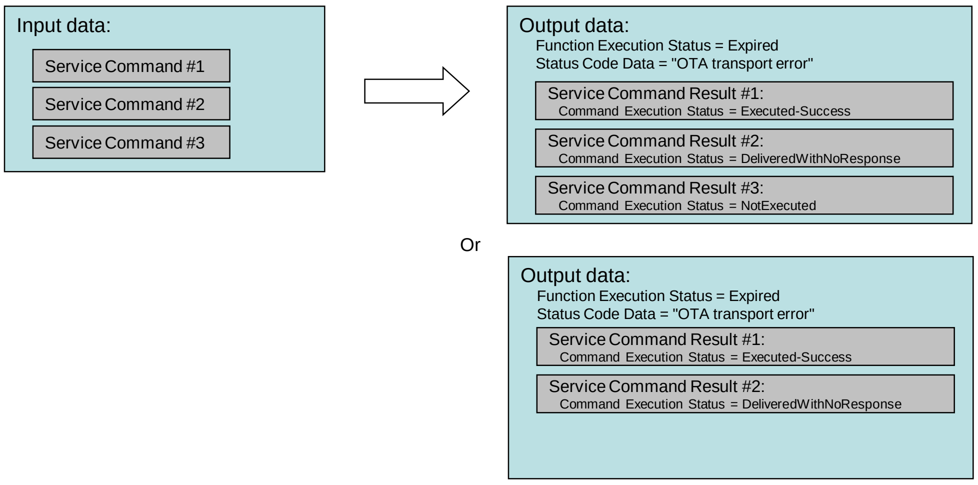

Input data:
Service Command #1
Service Command #2
Service Command #3
Output data:
Function Execution Status = Expired
Status Code Data = "OTA transport error"
Service Command Result #1:
Command Execution Status = Executed-Success
Service Command Result #2:
Command Execution Status = DeliveredWithNoResponse
Service Command Result #3:
Command Execution Status = NotExecuted
Or
Output data:
Function Execution Status = Expired
Status Code Data = "OTA transport error"
Service Command Result #1:
Command Execution Status = Executed-Success
Service Command Result #2:
Command Execution Status = DeliveredWithNoResponse

Figure 3-12: The 2nd Service Command expires after the first Service Command execution in the SE or Device

3.3.2.5.3 The Service Commands

3.3.2.5.3.1 Installing a Service

Command Name: `InstallServiceCommand`

Description:

This Service Command requests the loading and installation of the service components, as defined by the AP, into the mobile equipment. For example:

- SD creation and keys management on the SE,
- Loading of the SE applications,
- Instantiation of the SE applications (installation, extradition),
- Loading and installation of the Device applications.

The list of actions to be performed is service dependent and is defined as a business agreement with the AP.

In order to achieve the service installation, it MAY be required for the SDM to perform some specific actions to setup the mobile environment, such as creating a SDM SSD.

The `InstallServiceCommand` has no data.

3.3.2.5.3.2 Personalizing a Service

Command Name: `PersonalizeServiceCommand`

Description:

This Service Command requests the personalization of the service.

Installed service may require to be personalized, i.e. that additional information is sent to the applications in order to complete its installation (customer's data, diversification data, initial parameters, etc.).

Personalization scripts or messages are prepared by the function provider, based on the data provided by the function caller (the `Personalization Data` input data). Format of `Personalization Data` is out of the scope of this specification.

The `PersonalizeServiceCommand` is composed of the following data:

Input data name	Description	Type	MOC	No.
`Personalization Data`	Personalization data sent by the AP so that the function provider could compute the personalization scripts or messages. This data can be of any type. In particular, the "Name:Value pair" data type defined in section 3.1.1.3 could be used.	`<any data type>`	O	1

Input data name	Description	Type	MOC	No.
`New Computed Data`	States whether the provided `Personalization Data` corresponds to a set of newly computed data. This is especially useful when calling the `DeployService` function while the service instance state is `Incomplete Deployment` or `Missing Mobile Environment Component`, or when calling the `UpgradeService` function, as it could help the function provider determine if the personalization of the service application should be re-executed or not. If `'True'`, the application SHALL execute the personalization If `'False'`, the application MAY execute the personalization, if required	`Boolean`	O	1

Table 3-52: Data of the `PersonalizeServiceCommand` Service Command

When the Service Command processing has been performed correctly (`Command Execution Status` is `'Executed-Success'`), the `Service Command Result` output data (detailed in section 3.3.2.5.2) SHALL include the following output parameters.

Else, none of these complementary output data will be returned.

Output data name	Description	Type	MOC	No.
`Personalization Output Data`	Data that may be required to be delivered back to the AP following the personalization step. This data can be of any type. In particular, the "Name:Value pair" data type defined in section 3.1.1.3 could be used.	`<any>`	O	1

Table 3-53: Output parameters for the `PersonalizeServiceCommand` Service Command

3.3.2.5.3.3 Activating a Service

Command Name: `ActivateServiceCommand`

Description:

This Service Command requests the activation of the service. Service activation makes the service be operational for the end-user. This step completes the deployment process.

Setting the application state to "SELECTABLE" (following the GlobalPlatform Card Specification [1]) is an example of what can be done during an activation process. Enabling the end-user access of the service in a Device application is another example of service activation.

The ActivateServiceCommand is composed of the following data:

Input data name	Description	Type	MOC	No.
Suspension Control	The AP can specify whether the service instance should be set in Active state or in Suspended state at the end of the activation process. If 'True', the application SHALL be Suspended If 'False', the application SHALL be Active If the field is not present, the service instance state is defined by the AP as a business agreement.	Boolean	O	1

Table 3-54: Data of the ActivateServiceCommand Service Command

3.3.2.6 The "Upgrade Service" Function

Name: UpgradeService

Description:

This function enables to request for the deployment of a new version (identified by the New Service Instance Reference input data) of an already deployed service instance (identified by the Current Service Instance Reference input data).

Prior to calling this function, it is the function caller responsibility to obtain the new SIR by calling the LookupServiceInstanceReference function, or to declare it through the call of the DeclareServiceInstanceReference function, providing the new version of the service within the Service Identifier field (see section 3.1.1.2).

It is also the function caller responsibility to request for a global eligibility checking of the mobile environment for the new service version by calling the CheckGlobalEligibility function.

The implementation of the UpgradeService function highly depends on business agreements. Anyway, it is the function caller responsibility to provide the necessary information required for upgrading the service to the new version, and it is the function provider responsibility to analyze the content of the Secure Element and of the Device, and to determine which components should be updated and which ones could be kept as they are.

As an example, the function caller might provide all the Service Commands (in the Service Command input data list) that would be required for an initial deployment of the service instance, but the function provider would only need to consider the Service Commands that require to be processed.

For the specific personalization step, the New Computed Data field of the Personalize Service Command (see section 3.3.2.5.3.2) might help the function provider know whether the re-personalization of unchanged application is required or not.

The function provider SHALL verify that the New Service Instance Reference really corresponds to another version of the same service.

It SHALL also verify that the current service instance is in an allowed state (according to the service instance life cycle depicted in Figure 3-7) before proceeding to the service upgrade.

After the `UpgradeService` function has been executed successfully:

- The initial service instance is `Terminated` (according to the service instance life cycle depicted in Figure 3-7), and the corresponding SIR is no longer valid.
- The new service instance has started a new life cycle and reaches the `Installed`, `Personalized`, `Activated` or `Suspended` state, depending on the last executed Service Command.

In case of failed execution of the function, the service instance state SHOULD be set to the `Incomplete Deployment` state and the initial SIR remains valid.

Function provider: SDM

Functions group: Global Service Management

Service Deployment Modes: mode #1, mode #2

Additional input data:

Input data name	Description	Type	MOC	No.
`Current Service Instance Reference`	The reference to the currently deployed service instance	`Service Instance Reference`	M	1
`New Service Instance Reference`	The reference to the new (version of the) service. It SHALL correspond to another version of the same service (`Service Id` and `Service Qualifier`) that is referenced by the `Current Service Instance Reference` input data.	`Service Instance Reference`	M	1

Input data name	Description	Type	MOC	No.
`Service Command`	The Service Command to be executed, among the following: • `InstallServiceCommand` • `PersonalizeServiceCommand` • `ActivateServiceCommand` Having several occurrences of this field enables to specify several Service Commands to be executed in a single call. However: • Only one Service Command per type can be provided in a single function call. • Order of the Service Commands is relevant. The function caller SHALL provide the commands in the right order so that the service instance state diagram depicted in Figure 3-7 is respected. The function provider SHOULD also check that the provided commands are correctly ordered. The structure of each single Service Command is described in section 3.3.2.5.3.	`Service Command`	M	1..N

Table 3-55: Additional input parameters for `UpgradeService` function

Note: the `Service Instance Reference` type is defined in section 3.1.1.

Additional output data:

Output data name	Description	Type	MOC	No.
`Service Command Result`	Provide the single result of execution of a corresponding Service Command. Sequence of result SHALL follow the sequence of `Service Command` provided in the function input data, as specified in section 3.3.2.5.2.1. The `Service Command Result` basic structure is described in section 3.3.2.5.2. This structure can be complemented by additional output data described for each `Service Command` of section 3.3.2.5.3.	`Service Command Result`	M	1..N

Output data name	Description	Type	MOC	No.
`Service Instance State`	The state of the new service instance at the end of execution of the `UpgradeService` function. The possible values are: **1 ⇨ Not Deployed** **11 ⇨ Installed** **12 ⇨ Personalized** **13 ⇨ Incomplete Deployment** **21 ⇨ Active** **22 ⇨ Suspended** **23 ⇨ Missing Mobile Environment Component**	`Integer`	M	1

Table 3-56: Additional output parameters for `UpgradeService` function

3.3.2.7 The "Exchange Service Data" Function

Name: `ExchangeServiceData`

Description:

The purpose of this function is to enable exchange of service specific data between the Application Provider and the Secure Element, for a particular mobile-NFC service (identified by the `Service Instance Reference` input data), as depicted in the "mobile-NFC Service Data Exchange/Update" use case (see section 2.1.1.6). As examples, this function MAY be used in vertical specific use cases such as PIN code reset, counter reset, value top-up, etc.

Due to the diversity of vertical specific use cases, this function has a generic input and output structure type (the `Service Data` field of the `Exchange Service Data Command` input data, and the `Service Output Data` field of the `Exchange Service Data Command Result` output data). It is however not the intention to prefer using this function over more specialized functions for other purposes than exchange of service data. Whenever a more specialized function is available, the function caller SHALL use that instead. For example, for loading an applet into a security domain, the function caller SHALL use the `SECommandsGenerationAndRemoteExecution` function with the `LoadELFCommand` command.

Each call to this function contains one or more `Exchange Service Data Command`. Each of these commands defines an action (the `Action` field) and data (the `Service Data` field) to be executed by the function provider. As output of the function, for each `Exchange Service Data Command`, an `Exchange Service Data Commands Result` provide the result of execution of the command. Each of such command result MAY contain command output data (the `Service Output Data` field).

The valid service instance states (as depicted in section 3.3.1) for when an `Exchange Service Data Command` can be executed shall be decided by the AP and SDM as a separate business agreement. It is the function provider responsibility to verify that the current service state allows execution of the command. The execution of this function SHALL NOT alter the state of the service instance.

The implementation of this function highly depends on business agreements. It is however the function provider responsibility to validate the `Service Data` input and its values according to the requested `Action`.

This function can target any item of the mobile NFC service environment, including application data in the SE as well as in the Device.

Function provider: SDM

Functions group: Global Service Management

Service Deployment Modes: mode #1, mode #2

Additional input data:

Input data name	Description	Type	MOC	No.
`Service Instance Reference`	The `Service Instance Reference` of the service with which the function caller wishes to exchange data.	`Service Instance Reference`	M	1
`Exchange Service Data Command`	The `Exchange Service Data Command` to be executed. Having several occurrences of this field enables to specify several `Exchange Service Data Commands` to be executed in a single call.	`Exchange Service Data Command`	M	1..N

Table 3-57: Additional input parameters for `ExchangeServiceData` function

Note: the `Service Instance Reference` type is defined in section 3.1.1.

Where an `Exchange Service Data Command` is:

Data name	Description	Type	MOC	No.
`Action`	The reason or context or action for the service data exchange. E.g. "PINchange" or "PurchaseTicket".	`String`	M	1
`Service Data`	The data related to the action. This data can be of any type. In particular, the "Name:Value pair" data type defined in section 3.1.1.3 could be used. Having several occurrences of this field enables to specify several data for the given action.	`<any data type>`	O	1..N

Table 3-58 `Exchange Service Data Command` type

Additional output data:

Output data name	Description	Type	MOC	No.
Exchange Service Data Command Result	Provides the single result of execution of the corresponding Exchange Service Data Command. Sequence of result SHALL follow the sequence of Exchange Service Data Command provided in the function input data. The Exchange Service Data Command Result basic structure is described below.	Exchange Service Data Command Result	M	1..N

Table 3-59: Additional output parameters for ExchangeServiceData function

Where an Exchange Service Data Command Result is:

Data name	Description	Type	MOC	No.
Command Execution Status	Indicates whether the Exchange Service Data Command has been executed or not. The values are: **'Executed-Success'** **'Executed-WithWarning'** **'Executed-Failed'** **'NotExecuted'** **'DeliveredWithNoResponse'** A detailed description of the status is provided in section 3.3.2.7.1.	Enumeration {Executed-Success, Executed-WithWarning, Executed-Failed, NotExecuted, DeliveredWithNoResponse}	M	1
Command Status Code Data	Provides the reason of the Command Execution Status value when the Command Execution Status is 'Executed-WithWarning' or 'Executed-Failed'. The possible values depend on the Exchange Service Data Command. A normative list of codes is provided in section 6 but additional codes may be defined.	Command Status Code Data	C	1
Service Output Data	When the Exchange Service Data Command has been executed (Command Execution Status equal to 'Executed-Success', 'Executed-WithWarning' or 'Executed-Failed'), it provides output data of the executed command. This data can be of any type. In particular, the "Name:Value pair" data type defined in section 3.1.1.3 could be used. Having several occurrences of this field enables to specify several output data.	<any data type>	O	1..N

Table 3-60 Exchange Service Data Command Result type

Where a `Command Status Code Data` is:

Data name	Description	Type	MOC	No.
`Subject`	Represents the system element concerned by the exception.	`Object Identifier`	M	1
`Reason`	Represents the reason of the exception.	`Object Identifier`	M	1

Table 3-61 `Command Status Code Data` type

3.3.2.7.1 Error Management in Exchange Service Data Commands Processing

The output data of the "Exchange Service Data" function contains a list of Exchange Service Data Command Result: for each single Exchange Service Data Command in the function input data, a corresponding Exchange Service Data Command Result MAY be provided in the output data.

The basic principles of the processing of the `ExchangeServiceData` function are the following:

- Order of the Exchange Service Data Commands is relevant. The function provider SHALL execute the commands in the same order as given by the function caller.
- The processing of the "Exchange Service Data" function SHALL stop as soon as an error is encountered in the processing of an Exchange Service Data Command.

Within each single Exchange Service Data Command Result, a command execution status (the `Command Execution Status` output data) SHALL specify whether the overall processing of the Exchange Service Data Command (i.e. up to the Secure Element or Device update) has been processed correctly (`'Executed-Success'`), correctly with some warnings (`'Executed-WithWarning'`), with error (`'Executed-Failed'`), or not processed at all (`'NotExecuted'`).

The reliability of the OTA connectivity is such that it may happen that even if data has been received and processed by the SE or by the Device, the execution status sent back by the SE or by the Device over the network may get lost. In such case where the status of execution cannot be determined, the command execution status SHALL be `'DeliveredWithNoResponse'`.

As for any function, a global status is specified in the output parameters: the `Status` field of the `Function Execution Status` output data. Its value, together with the `Status Code Data` of the function (in case of an error, a warning or the expiration), depends on the overall statuses of all the Exchange Service Data Command Results:

- The global function processing status SHALL be set to `'Executed-Success'` if the execution status of all the Exchange Service Data Commands is `'Executed-Success'`. The `Command Status Code Data` of the Exchange Service Data Command Result and the `Status Code Data` of the function SHALL NOT be present.
- The global function processing status SHALL be set to `'Executed-WithWarning'` if at least one of the Exchange Service Data Commands has an execution status set to `'Executed-WithWarning'`, the execution status of all the other Exchange Service Data Commands being `'Executed-Success'`.

- The global function processing status SHALL be set to `'Failed'` if at least one of the Exchange Service Data Command has an execution status set to `'Executed-Failed'`.

  The `Command Status Code Data` of the Exchange Service Data Commands in error SHALL detail the reason that caused the Exchange Service Data Command error. This code SHALL also be set in the `Status Code Data` of the function.

- The global function processing status SHALL be set to `'Expired'` if at least one of the Exchange Service Data Commands has an execution status set to `'DeliveredWithNoResponse'`.

  The `Command Status Code Data` of the Exchange Service Data Commands in expiration SHALL detail the reason that caused the Exchange Service Data Command expiration. The `Subject` and `Reason` fields of the `Status Code` of the function SHALL be set to the relevant error code (for example, 'OTA transport error' code –see section 6)

Those rules imply that the Exchange Service Data Command Results with `'NotExecuted'` status can only be located at the end of the `Exchange Service Data Command Result` list. As a consequence, it is optional to explicitly put those Exchange Service Data Command Results in the `Exchange Service Data Command Result` list. This is especially the case if an error occurs before the processing of any of the Exchange Service Data Commands: the list of Exchange Service Data Command Results MAY either be empty or MAY contain as many Exchange Service Data Command Results as Exchange Service Data Commands, each of them having the `'NotExecuted'` status.

Note that the function validity period may expire:

- If expiration is raised before the concrete execution of actions in the SE or in the Device for the first Exchange Service Data Command, then the corresponding `Exchange Service Data Command Result` SHALL be set to `'NotExecuted'` and this is the responsibility of the function caller system to cancel all server-side actions done for this Exchange Service Data Command.
- If expiration is raised after the concrete execution of actions in the SE or in the Device for the first Exchange Service Data Command, then the corresponding `Exchange Service Data Command Result` SHALL be set to `'DeliveredWithNoResponse'` and the confirmation or cancellation of the actions on the function caller side is function requester-implementation dependant.

As stated before, in both cases, the `Status` field of the `Function Execution Status` output parameter of the function SHALL be set to `'Expired'`.

3.3.2.8 The "Suspend Or Resume Service" Function

Name: `SuspendOrResumeService`

Description:

This function enables to suspend or resume the deployed service instance (identified by the `Service Instance Reference` input data). Using this function, the function caller can block or unblock the usage of the service. However, the behavior in case of service being suspended is service specific (NFC service access is disabled, UI access disabled, etc.).

The way a service is suspended by the function provider can be different depending on the service components. For example a service could be suspended by changing the SE application(s) state(s) to LOCKED (see GP 2.2 Card Specifications [1]), or by using service-specific management features of a Device application. The exact behavior of this function consequently depends on the service and is established as a business agreement.

At the end of the function execution, the service instance state SHALL be set to `Suspended`.

When a service is resumed, the service instance state SHALL be set to `Active`.

The function provider SHALL verify that the service instance is in an allowed state (according to the service instance life cycle depicted in Figure 3-7) before proceeding to the service suspension or resumption.

In case of failed execution of the function, the service instance state SHALL stay in its current state.

Function provider: SDM

Functions group: Global Service Management

Service Deployment Modes: mode #1, mode #2

Additional input data:

Input data name	Description	Type	MOC	No.
`Service Instance Reference`	The `Service Instance Reference` of the service to be suspended or resumed.	`Service Instance Reference`	M	1
`Suspension Control`	This field specifies the requested state transition. If `'True'`, the service SHALL be suspended If `'False'`, the service SHALL be resumed	`Boolean`	M	1

Table 3-62: Additional input parameters for `SuspendOrResumeState` function

Note: the `Service Instance Reference` type is defined in section 3.1.1.

No additional response data complement the output header defined for functions and detailed in section 3.1.3.

3.3.2.9 The "TerminateService" Function

Name: `TerminateService`

Description:

This function enables to terminate the service, i.e. remove the components of the service (identified by the `Service Instance Reference` input data) from the mobile environment.

The `TerminateService` function consists of un-installing the service components as defined by the AP from the mobile equipment, such as:

- Deleting or deactivating the SE applications instances
- Deleting the SE elementary load files
- Deleting the SD from the SE, or removing the SD content
- Deleting or deactivating the Device applications

The list of actions to be performed is service dependent and is defined as a business agreement with the AP.

After the `TerminateService` function has been executed successfully, the service instance is `Terminated` (according to the service instance life cycle depicted in Figure 3-7), and the SIR is no longer valid. The SIR value is then available for reuse by a new call to the `LookupServiceInstanceReference` or `DeclareServiceInstanceReference` function (see sections 3.3.2.1 and 3.3.2.2).

Depending on the function provider implementation, calls to the `GetServiceInstanceReferenceDescriptor` and `GetServiceState` functions using the just invalidated SIR might however still be possible (see sections 3.3.2.3 and 3.3.2.4).

Function provider: SDM

Functions group: Global Service Management

Service Deployment Modes: mode #1, mode #2

Additional input data:

Input data name	Description	Type	MOC	No.
`Service Instance Reference`	The `Service Instance Reference` of the service to terminate.	`Service Instance Reference`	M	1

Table 3-63: Additional input parameters for `TerminateService` function

Note: the `Service Instance Reference` type is defined in section 3.1.1.

No additional response data complement the output header defined for functions and detailed in section 3.1.3.

3.3.3 Service Environment Change

3.3.3.1 The "Service Environment Change" Notification Function

Name: `HandleServiceEnvironmentChangeNotification`

Description:

This function SHALL be called to notify that the environment of the mobile-NFC service instance (identified by the `Service Instance Reference` input data) has changed. Environmental change (defined by the `Event` input data) might correspond to a change related to the Secure Element (SE stolen, lost, recovered, renewed, etc.), or to the Device (Device stolen, lost, recovered, renewed, etc.), or to the Mobile Subscription (suspended, terminated, restored, etc.), etc.

An optional date (the `Date` input data) MAY be provided as input data of the notification to specify when the environment change has or will occur:

- If the date is in the past, it represents the date when the environment change has occurred,
- If the date is in the future, it represents the date when the environment change will occur. This enables the notification receiver to perform actions prior the change, if required.

This notification SHALL also indicate the new state of the service instance (the `Service Instance State` input data), according to the service instance state diagram depicted in Figure 3-7, in section 3.3.1. It enables in particular the notification recipient to know whether the service instance is:

- Still operational: Device and Secure Element applications still available and functional, in end-users hands
- No longer operational: either Device or SE no longer in end-users hands –`Missing Mobile Environment Component` state–, or Device or SE applications no longer operational or missing –`Incomplete Deployment` state–.

This notification is related to a particular instance of mobile-NFC service, meaning that, in case for example of an environment change related to a Secure Element, there SHALL be as many notifications as service instances deployed on this Secure Element.

The recipient of the notification MAY use this notification to update customer or service states in their internal systems, in order to provide up-to-date information to the customer care or the self-care interface.

No output data are expected to this notification.

Notification recipients: AP

Functions group: Service Environment Change Notification

Service Deployment Modes: mode #1, mode #2

Additional input data:

Input data name	Description	Type	MOC	No.
`Service Instance Reference`	The reference to the service instance	`Service Instance Reference`	M	1
`Event`	The event that causes the Service environment change. The following values are pre-defined, but other values MAY be defined by the function provider: (for issues) **1 ⇨ Mobile subscription suspended** **2 ⇨ Mobile subscription restricted** **3 ⇨ Mobile subscription terminated** **11 ⇨ NFC mobile option suspended** **12 ⇨ NFC mobile option terminated** **21 ⇨ SE Stolen** **22 ⇨ SE Lost** **23 ⇨ SE Lost or Stolen (in case the notifier cannot differentiate Stolen and Lost process)** **24 ⇨ SE Broken** **31 ⇨ Device Stolen** **32 ⇨ Device Lost** **33 ⇨ Device Lost or Stolen (in case the notifier cannot differentiate Stolen and Lost process)** **34 ⇨ Device Broken** **41 ⇨ SE + Device Stolen** **42 ⇨ SE + Device Lost** **43 ⇨ SE + Device Lost or Stolen (in case the notifier cannot differentiate Stolen and Lost process)** (for corrections) **101 ⇨ Mobile subscription restored** **102 ⇨ Mobile subscription identifier changed (same subscription, but identifier changed)** **103 ⇨ Mobile subscription changed** **111 ⇨ NFC mobile option restored** **121 ⇨ SE recovered (following theft or loss)** **122 ⇨ SE renewed – No eligibility info** **123 ⇨ SE renewed – SE is eligible** **124 ⇨ SE renewed – SE is not eligible** **131 ⇨ Device recovered (following theft or loss)** **132 ⇨ Device changed – No eligibility info** **133 ⇨ Device changed – Device is eligible**	`Integer`	M	1

Input data name	Description	Type	MOC	No.
	134 ➪ Device changed – Device is not eligible **141 ➪ SE + Device recovered**			
`Date`	The date when the event has or will occur.	`DateTime`	O	1
`New Service Instance State`	The new state of the service instance. The possible values are: **1 ➪ Not Deployed** **11 ➪ Installed** **12 ➪ Personalized** **13 ➪ Incomplete Deployment** **21 ➪ Active** **22 ➪ Suspended** **23 ➪ Missing Mobile Environment Component** **31 ➪ Terminated**	`Integer`	M	1

Table 3-64: Additional input parameters for `HandleServiceEnvironmentChangeNotification` function

Note: the `Service Instance Reference` type is defined in section 3.1.1.

This notification function does not have any output data.

3.3.3.2 The "Action Done On Service" Notification Function

Name: `HandleActionDoneOnServiceNotification`

Description:

This function SHALL be called to notify to the AP that a global action has been performed on the mobile-NFC service instance (identified by the `Service Instance Reference` input data). The action (the `Action` input data) might have been directly executed by the notification sender, or simply passed through by the notification sender.

This notification can for example be used in conjunction with the `HandleServiceEnvironmentChangeNotification` function in case some corrective or preventive actions have been executed on the service when detecting the environment change. For example:

- Corrective action: the re-load of a Device applications requested by the SDM, following a Device change notification
- Preventive action: the lock of the SE applications performed by the SEI, following a SE lost notification

Note that the `HandleActionDoneOnServiceNotification` notification is different from the `HandleStartServiceStateChangeNotification` notification and the `HandleEndServiceStateChangeNotification` notification depicted in section 3.2.3 because the first notification only targets the AP in order to inform it about actions performed on the service without its involment. The other two declare the intention and completion of change in the service state, to other roles of the ecosystem.

The result of execution of each action SHALL be specified in the `Action Execution Status` field. If an action has been processed correctly, the `Action Execution Status` SHALL be `'Executed-Success'`. If an error occurred during the processing of an action, the `Action Execution Status` SHALL be `'Executed-Failed'`.

The reliability of the OTA connectivity is such that it may happen that even if data has been received and processed by the SE or the Device, the execution status sent back by the SE or the Device over the network may get lost. In such case where the status of execution cannot be determined, the `Action Execution Status` SHALL be `'DeliveredWithNoResponse'`.

There SHALL be as many execution results as actions. The order of the results SHALL respect the order of the actions.

This notification is related to a particular instance of mobile-NFC service, meaning that, in case for example of an environment change related to a Secure Element, there SHALL be as many notifications as service instances deployed on this Secure Element.

The recipient of the notification MAY use this notification to update customer or service states in their internal systems, in order to provide up-to-date information to the customer care or the self-care interface.

No output data are expected to this notification.

Notification recipients: AP

Functions group: Service Environment Change Notification

Service Deployment Modes: mode #1, mode #2

Additional input data:

Input data name	Description	Type	MOC	No.
`Service Instance Reference`	The reference to the service instance.	`Service Instance Reference`	M	1
`Action`	An action that has been executed on the mobile-NFC service instance. Having several occurrences of this field enables to specify several executed actions. The following values are pre-defined, but other values MAY be defined by the function provider: (for preventive actions)	`Integer`	M	1..N

Input data name	Description	Type	MOC	No.
	1 ⇨ Delete Device application **11 ⇨ Lock SE application** **12 ⇨ Delete SE application** **21 ⇨ Lock SE** (for corrective actions) **101 ⇨ Re-deploy Device application** **111 ⇨ Un-lock SE application** **121 ⇨ Un-lock SE**			

`Action Execution Status`	Indicates the execution status of the `Action`.
	There SHALL be as many `Action Execution Status` as `Action`.
	
	The values are:
	`'Executed-Success'`
	`'Executed-Failed'`
	`'DeliveredWithNoResponse'`
		`Enumeration {Executed-Success,`
		`Executed-Failed,`
		`DeliveredWithNoResponse}`
			M	1..N

`New Service Instance State`	The new state of the service instance. The possible values are: **1 ⇨ Not Deployed** **11 ⇨ Installed** **12 ⇨ Personalized** **13 ⇨ Incomplete Deployment** **21 ⇨ Active** **22 ⇨ Suspended** **23 ⇨ Missing Mobile Environment Component** **31 ⇨ Terminated**	`Integer`	M	1

Table 3-65: Additional input parameters for `HandleActionDoneOnServiceNotification` function

Note: the `Service Instance Reference` type is defined in section 3.1.1.

This notification function does not have any output data.

3.4 Functions for CCCM Certificate Management

Security Domains are particular GlobalPlatform Applications that are instantiated and deleted like any other classical Application. However, the creation of keys within a Supplementary SD (i.e. the SSD personalization process) is a specific mechanism that highly depends on the level of confidentiality to ensure to those keys.

Basic mechanism for key management uses the GlobalPlatform PUT KEY command, and relies on the security provided by the secure channel used to dialog with the Secure Element (SCP80 or SCP81 for UICC; SCP02 for ESE/SMC). It means that the keys of a Supplementary Security Domain are known of course by the SSD owner, but also by the actor that provided the secure channel for the SSD personalization. All the keys of the keyset shall be independently created using the PUT KEY command.

Amendment A of GP 2.2 Card Specification [3] defined several Confidential Card Content Management (CCCM) scenarios to ensure full confidentiality of the global key management of a SSD, so that only the SSD owner will know the key values. Amendment A defines several scenarios in order to create keysets in a SSD; all of these scenarios are based on the presence of a Controlling Authority (CA) that is a trusted third party that ensures the confidentiality of the key values. The specific behavior of these modes is defined in UICC Card Configuration document [7] and Amendment E of GP 2.2 Card Specification [6].

- Scenario #1 (Pull Model), using PK scheme: either the final key values or a unique master key (called the Randomly Generated Key (RGK) master key) are randomly generated through the On Board Key Generation (OBKG) mechanism.

  The generated key values or the RGK are returned to the off-card server. Prior to be sent OTA, they are encrypted with the public key of the Application Provider/SSD owner using an asymmetric encryption scheme.

  In case of usage of a RGK Master Key, the keys of the keyset are then derived from this RGK both on card and on server side.
- Scenario #1 (Pull Model), using Non-PK scheme: this mode is similar to the previous one except that the final key values or the RGK master key are encrypted with a secret key KS shared between the SSD and the SSD owner using a symmetric encryption scheme. The key KS is established by means of secret keys shared between CA and CASD.
- Scenario #2.A (Push Model with Application Provider Certificate): the final key values or the RGK master key to be used to derivate the keys of the keyset are generated off card. Before being sent OTA to the card and used for the diversification, they are ciphered through a PK scheme using asymmetric keys belonging to the CASD and signed by the Application Provider/SSD owner.
- Scenario #2.B (Push Model without Application Provider Certificate): the process is similar to the one of the previous mode except that the final key values or the master key are not signed by the Application Provider/SSD owner.
- Scenario #3 (Elliptic Curve Key Agreement - ElGamal scheme): a shared secret is established in a key agreement protocol where both on card and off-card entities contribute to the secret. The public key scheme is based on Elliptic Curve Cryptography and there are different options of key lengths (256, 384, 512 and 521 bits) as is further detailed in [6].

The GlobalPlatform STORE DATA command is used for this confidential key management mechanism.

Note that this version of the specification only considers the basic key management mechanism and the following Confidential Card Content Management scenarios defined in GP 2.2 Card Specification Amendment A [3]: CCCM Scenario #1 using PK scheme, CCCM Scenario #2.A, CCCM Scenario #2.B, and CCCM Scenario #3.

Note also that CCCM Scenario #2.B can be implemented using two different ways:

- Either, following the UICC Configuration document [7], in two consecutive steps:
    - The SSD key owner (acting as a "Off-card Application Provider Platform") asks a SDM (acting as the "LPO SD") to create the temporary keyset by calling the `SE Commands Generation and Remote Execution` function with a `Create First SSD keyset` SE Command in `Basic Random Create` mode
    - Then, the SSD key owner uses the created keys as secure channel to autonomously (or through the call to the `Send Script` function on the SDM) update the temporary keyset with the final keys values
- Or, using the `Create First SSD keyset` SE Command, in one single step from function caller point of view, but still being compliant with the DGIs defined in GP 2.2 Card Specification Amendment A [3] and in UICC Configuration document [7]:

  The SSD key owner calls the `Generate SE Command and Remote Execution` function with a `Create First SSD keyset` SE Command in `CCCM Scenario #2.B` mode. The function provider SDM then:
    - Creates the temporary keyset in the Supplementary Security Domain
    - Using this temporary keyset as secure channel, updates the temporary keyset with final keys values

Both flows are fully equivalent in terms of security as SSD keys are in both cases secured by the Controlling Authority SD keys.

Once a first keyset of a SSD is created using the Amendment A mechanism, it is possible for the SSD owner to open a secure channel with its SSD, using this keyset. The SSD owner shall then use this particular secure channel to create all the other keysets required to finalize the personalization of the SSD (using the basic key management PUT KEY command).

In other words, Simple mode functions for key management are theoretically only required for the creation of the very first keyset. After that step, the function requester is fully autonomous for the creation of other keysets and for the update of key values: it no longer requires the call of the Simple mode key management functions to do so. However, in order to enable OTA sessions optimization when several types of keysets need to be created in a Security Domain through Simple mode functions (e.g. a SCP80 together with a SCP81 keyset), this specification allows to call the key management functions for the creation of the first keyset for each of the keyset types of a Security Domain.

The following functions are used for the first keyset creation:

- The `EnrollSSDOwnerCertificate` function for the generation, by the Controlling Authority, of a certificate for a SSD owner (see section 3.4.2).
- The `AuditCAInformation` function for the retrieval of information on the CASD of a given Secure Element (see section 3.4.1).

- The `CreateFirstSSDKeysetCommand` SE Command (see section 3.5.1.1 for the explanation of what a SE Command is) for the true creation of the first keyset for each of the keyset types of a SSD (see section 3.5.1.3.1.2).

The following functions are described in this section:

- `AuditCAInformation` (see section 3.4.1)
- `EnrollSSDOwnerCertificate` (see section 3.4.2)

3.4.1 The "Audit CA Information" Function

Name: `AuditCAInformation`

Description:

This function queries about information regarding the Controlling Authority Security Domain of a Secure Element (identified by the `Secure Element` input data). It gives the list of CCCM modes that are supported for key management (the `Supported CCCM Mode` output data) and returns the CASD certificate of the SE (the `CA Certificate` output data). Note that despite the name, the function queries exclusively about information regarding the CASD.

Note that if the identifier of the Mobile Subscription through which the SE is accessible is known by the function caller, then it SHALL provide it (in the `Mobile Subscription` input data): it might help the function provider to reach the Secure Element in case it is required to dialog with the Secure Element to get the CA information. If this information is not known by the function caller, and if it is required by the function provided, then the function provider is responsible to retrieve the Mobile Subscription identifier based on the Secure Element identifier (by using its own information system or by asking another role for example through the `GetSEMobileSubscriptionIdentifier` function – see section 3.2.4.2).

Note also that in case the CASD certificate is expired or is known as revoked by the function provider, the `Status` field of the `Function Execution Status` SHOULD then returned with a value set to `'Executed-Failed'`, and with a relevant `Status Code` explaining the error (expired or revoked).

Function provider: SEI

Functions group: CA Info

Additional input data:

Input data name	Description	Type	MOC	No.
`Secure Element`	The Secure Element into which the mobile-NFC service is deployed/under deployment.	`SE Identifier`	M	1
`Mobile Subscription`	The Mobile Subscription through which the Secure Element is accessible.	`Mobile Subscription Identifier`	O	1

Table 3-66: Additional input parameters for `AuditCAInformation` function

Note: the `SE Identifier` and the `Mobile Subscription Identifier` types are defined in section 3.1.1.

Additional output data:

Output data name	Description	Type	MOC	No.

`Supported CCCM Mode`	A CCCM mode supported by the CA SD in the Secure Element.
	Having several occurrences of this field enables to specify several supported modes.
	No occurrence means that no CA SD is present in the SE, or no CCCM mode is supported.
	
	The values are:
	`'CCCM Scenario #1, using PK scheme'`
	`'CCCM Scenario #2.A'`
	`'CCCM Scenario #2.B'`
	`'CCCM Scenario #3, using ECC-256'`
	`'CCCM Scenario #3, using ECC-384'`
	`'CCCM Scenario #3, using ECC-512'`
	`'CCCM Scenario #3, using ECC-521'`
		`Enumeration {Scenario 1-PK,`
		`Scenario 1-Non PK,`
		`Scenario 2A,`
		`Scenario 2B,`
		`Scenario 3-ECC-256,`
		`Scenario 3-ECC-384,`
		`Scenario 3-ECC-512,`
		`Scenario 3-ECC-521}`
			O	1..N

`CA Certificate`	The CA SD certificate. The format of this field is a byte array which content corresponds to the full content of tag `'7F21'` (including the two `'7F21'` bytes) defined in table 9-4 "CASD Certificate Structure (Certificate with Message Recovery)" of the UICC Card Configuration [7]. Only present if the `Supported CCCM Mode` is not `'None'`.	`Byte[]`	C	1

Table 3-67: Additional output parameters for `AuditCAInformation` function

3.4.2 The "Enroll SSD Owner" Function

Name: `EnrollSSDOwnerCertificate`

Description:

This function enables a SSD owner to request for the enrollment of a certificate to the CA, for later confidential SSD key management. This function is based on RFC 2986 [19], providing an ASN.1 CertificationRequest (the `Certification Request` input data) that is Distinguished Encoding Rules (DER) encoded.

The Subject Identifier as defined in UICC Card Configuration [7] should be submitted as an attribute in the CertificationRequestInfo field inside the CertificationRequest.

The SSD owner certificate is returned (the `SSD Owner Certificate` output data), as described in the UICC Card Configuration [7].

Function provider: CA

Functions group: CCCM Certificates Management

Additional input data:

Input data name	Description	Type	MOC	No.
`Certification Request`	A DER encoded CertificationRequest according to RFC 2986 [19].	`Byte[]`	M	1

Table 3-68: Additional input parameters for `EnrollSSDOwnerCertificate` function

Additional output data:

Output data name	Description	Type	MOC	No.
`SSD Owner Certificate`	The certificate of the SSD owner. The format of this field is a byte array which content corresponds to the full content of tag `'7F21'` (including the two `'7F21'` bytes) defined in table 10-2 "Structure of AP Public Key Certificate with Message Recovery" of the UICC Card Configuration [7].	`Byte[]`	M	1

Table 3-69: Additional output data for `EnrollSSDOwnerCertificate` function

3.5 Functions for the Secure Element Content Management

The delivery of a NFC services requires the execution of commands on the Secure Element. Those commands are mainly Card Content Management commands such as SD creation, Executable Load File loading, and Application instantiation and personalization.

Depending on its privileges, the SDM SHALL do one of the following:

- Generate by itself the Card Content Management commands and send them to the SE (Dual mode – does not imply any API between roles),
- Generate by itself the Card Content Management commands but ask another SDM for a Delegated Management Token before sending them to the SE (Delegated Management mode defined in section 3.5.2),
- Ask another SDM to fully generate and send the Card Content Management commands to the SE (Simple mode defined in section 3.5.1).

Additional operations may be required on the Secure Element to complete the delivery of the mobile-NFC service:

- Sending script to perform application personalization: in case the builder of the script does not have any OTA capability to reach the SE, he needs to ask a SDM to send the script OTA (see section 3.5.3).
- Uploading a service portal: in case the User Interface of the service is built using the Smart Card Web Server (SCWS) technology, and if the SDM does not have its own capability to update the SCWS content, then he needs to ask another SDM to update the SCWS content (see section 3.5.5).

Note that the bearer used to send all those commands might depend on the type of secure element (UICC vs. Embedded Secure Element vs. Secure Memory Card) and of the size of the payload to be sent (e.g. SMS bearer vs. CAT-TP bearer vs. HTTP bearer for UICC).

The following functions related to SE Content Management are described in this section:

- Card Content Management functions:
  - SE Card Content Management functions for Simple mode
    - `SECommandsGenerationAndRemoteExecution` (see section 3.5.1.2)
  - Delegated Management
    - `GenerateDMToken` (see section 3.5.2.1)
    - `VerifyDMReceipt` (see section 3.5.2.2)
- Additional SE operations:
  - Scripts sending
    - `BeginConversation` (see section 3.5.3.1)
    - `SendScript` (see section 3.5.3.2)

        - `EndConversation` (see section 3.5.3.3)
    - Secure Element audit
        - `GetApplicationOrELFStatus` (see section 3.5.4.1)
        - `GetSDFreeMemory` (see section 3.5.4.2)
    - SCWS service portal
        - `LoadSCWSServicePortal` (see section 3.5.5.1)
        - `DeleteSCWSServicePortal` (see section 3.5.5.2)

3.5.1 SE Card Content Management Functions for Simple Mode

In Simple mode, the TSM SDM asks the SE Provider SDM to perform single CCM operations on its behalf.

Such request is made through the functions defined in this section, and that SHOULD be provided by the SE Provider SDM. Unless explicitly mentioned, the "SDM" role used in this section refers to the SE Provider SDM, the TSM SDM being referred as the "function caller" or "function requester".

3.5.1.1 The SE Command Concept

Whatever the communication channel used to dialog with the Secure Element, it may be required to optimize the communication between the SE and the administration server. As an example, these optimizations become almost mandatory when communicating with a UICC:

- Using the SMS bearer, to prevent the several commands from being sent over several SMS when they can be regrouped into a single SMS,
- Using the CAT-TP bearer or the HTTP bearer, to prevent from a systematic opening and closing of the channel for each command (as the opening of the channel implies to send a triggering SMS). The preferred scheme is to perform a unique channel opening, then the execution of the different commands, and at the end a unique final closing of the channel.

As these optimizations need to be done when building the CCM secure script, it is not possible to rely on single and independent functions (such as "Create SD", "Load Executable Load File" or "Instantiate Application"). The SDM would not be able to correlate and to identify a sequence of CCM actions to be regrouped for optimization. Therefore, a generic function is defined.

This generic function, called "SE Commands Generation and remote Execution", takes a sequence of single "SE commands" as input parameter, so that the function provider can optimize the secure script generation and thus the OTA exchanges. This function is defined in section 3.5.1.2.

Each "SE command" of a sequence represents a particular Card Content Management action, as defined in section 3.5.1.3. Prior to any script generation or OTA dialog initiation, the SDM MAY check that the function caller has the right to perform the requested command. The number and the granularity of such verifications are SDM dependent and are out of scope of this specification.

When receiving a sequence of SE commands, the SDM SHOULD provide mechanisms to optimize the generation of the secure script and the following communication with the SE. However, the optimization capabilities are SDM-dependant and are out of scope of this specification.

As an example, the caller may request the SDM to generate the CCM script to instantiate and make selectable a Security Domain in a UICC. If the two associated commands are requested using a unique function, the SDM can optimize the script to send it in a single SMS.

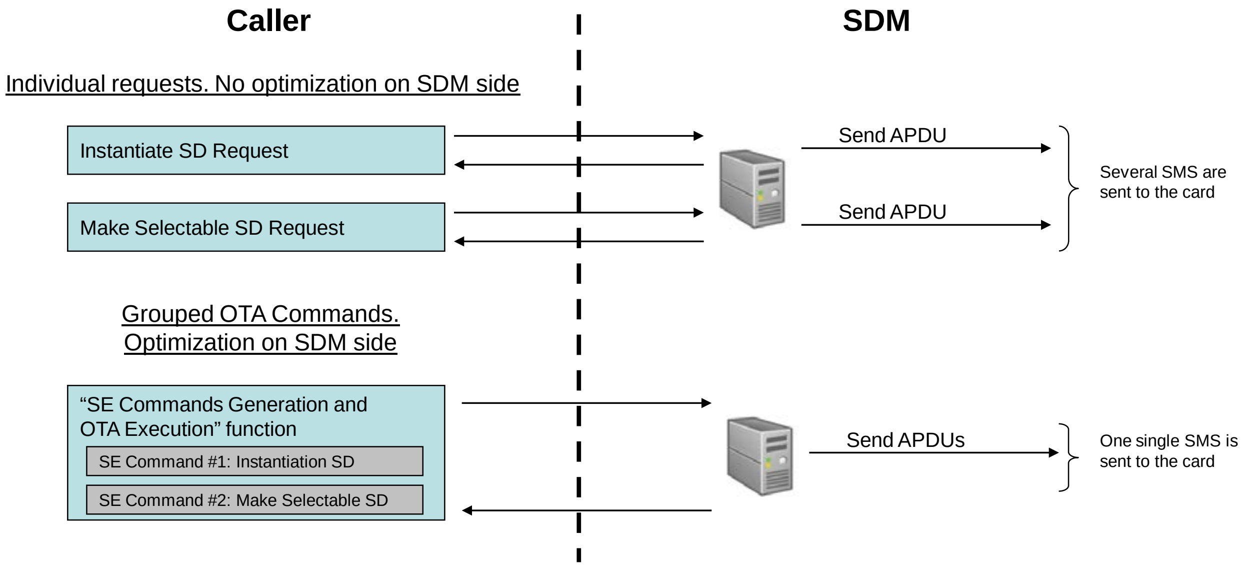

Caller
SDM
Individual requests. No optimization on SDM side
Instantiate SD Request
Make Selectable SD Request
Send APDU
Send APDU
Several SMS are sent to the card
Grouped OTA Commands. Optimization on SDM side
"SE Commands Generation and OTA Execution" function
SE Command #1: Instantiation SD
SE Command #2: Make Selectable SD
Send APDUs
One single SMS is sent to the card

Figure 3-13: OTA optimization on the SDM-side using SMS bearer

Later on, the function requester may require the SDM to load an Executable Load File, then to instantiate an Application, to extradite it and to make it selectable. If, those four commands are requested through a single "SE Commands Generation and remote Execution" function, they can be optimized by the SDM to build a single secure script sent using a single CAT-TP or HTTP session.

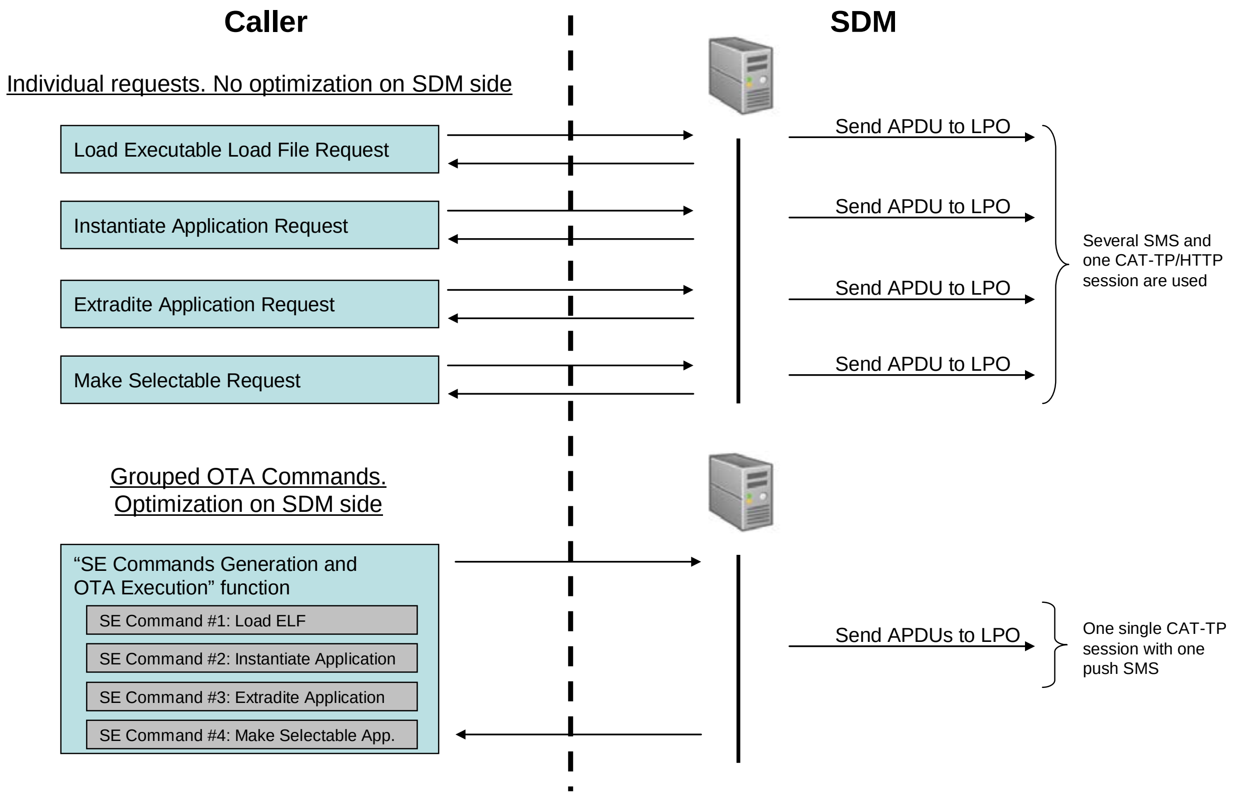

Caller
SDM
Individual requests. No optimization on SDM side
Load Executable Load File Request
Instantiate Application Request
Extradite Application Request
Make Selectable Request
Send APDU to LPO
Send APDU to LPO
Send APDU to LPO
Send APDU to LPO
Several SMS and one CAT-TP/HTTP session are used
Grouped OTA Commands. Optimization on SDM side
"SE Commands Generation and OTA Execution" function
SE Command #1: Load ELF
SE Command #2: Instantiate Application
SE Command #3: Extradite Application
SE Command #4: Make Selectable App.
Send APDUs to LPO
One single CAT-TP session with one push SMS

Figure 3-14: OTA Optimization on the SDM-side using CAT-TP or HTTP bearer

3.5.1.2 The "SE Commands Generation and Remote Execution" Function

Name: `SECommandsGenerationAndRemoteExecution`

Description:

This function enables to request for the generation and the OTA execution of a sequence of Card Content Management commands (contained in the list of `SE Command` input data) for a Secure Element, in the context of the management of a particular instance of mobile-NFC service (identified by the `Service` and the optional `Service Qualifier` input data) through a particular Mobile Subscription (identified by the `Mobile Subscription` input data).

Note that if the identifier of the Mobile Subscription through which the SE is accessible is known by the function caller, then it SHALL provide it (in the `Mobile Subscription` input data). If this information is not known by the function caller, then the function provider is responsible to retrieve the Mobile Subscription identifier based on the Secure Element identifier (by using its own information system or by asking another role for example through the `GetSEMobileSubscriptionIdentifier` function – see section 3.2.4.2).

The function provider SHOULD determine the appropriate bearer to be used to reach the targeted Secure Element, according to the type of Secure Element and to the concordance between the bearer capabilities and the requested commands to be sent.

As an example, the HTTP bearer is probably the bearer to be preferred to reach an ESE or a SMC whereas it is possible to use the SMS, the CAT-TP or the HTTP bearer to reach a UICC. However, in case of a UICC, if one of the commands contains an important payload (such as to load an Executable Load File), the CAT-TP or HTTP bearer should probably be used.

Note that this function is to be used:

- For all Card Content Management actions of Simple mode,
- But also for initial creation of Supplementary Security Domains (which might be associated the Delegated Management or the Authorized Management privilege).

  If so, this function is still considered as being used in Simple mode, even if the rest of the Card Content Management actions will be in Delegated or Dual mode, using the new SSD capabilities.

As this function is used in Simple mode, the function provider has the capability to determine which Application/Security Domain to target in the SE in order to properly execute the commands. There is consequently no need for the function requester to provide such AID or TAR information.

Function provider: (SE Provider's) SDM

Functions group: Card Content Management

Card Content Management Modes: Simple mode

Service Delivery modes: mode #2, mode #3

Additional input data:

Input data name	Description	Type	MOC	No.
`Secure Element`	The Secure Element into which the mobile-NFC service is deployed/under deployment.	`SE Identifier`	M	1
`Mobile Subscription`	The Mobile Subscription through which the Secure Element is accessible.	`Mobile Subscription Identifier`	O	1
`Service`	The mobile-NFC service.	`Service Identifier`	M	1
`Service Qualifier`	The qualifier of the mobile-NFC service.	`Service Qualifier`	C	1
`SE Command`	The SE Commands to be executed. If several SE Commands are provided, they SHALL be processed in the sequence order. The structure of each single SE Command is described in section 3.5.1.3.	`SE Command`	M	1..N

Input data name	Description	Type	MOC	No.
Stop On Warning	Specifies whether the processing of SE Commands shall stop ('True') or continue ('False') if a warning is encountered. If not present, parameter is considered to be set to 'False'.	Boolean	O	1

Table 3-70: Additional input parameters for SECommandsGenerationAndRemoteExecution function

Note: the SE Identifier, the Mobile Subscription Identifier, the Service Identifier and the Service Qualifier types are defined in section 3.1.1.

Additional output data:

Output data name	Description	Type	MOC	No.
SE Command Result	Provide the single result of the *on-card* execution of a corresponding SE Command. Sequence of result SHALL follow the sequence of SE Command provided in the function input data, as specified in section 3.5.1.2.1. The SE Command Result basic structure is described below, This structure can be complemented by additional output data described for each SE Command of section 3.5.1.3.	SE Command Result	M	1..N
Used Bearer	The OTA bearer really used for the execution of the secure script. The following values are pre-defined, but other values MAY be defined by the function provider: **1 ⇨ SMS to UICC** **2 ⇨ CAT-TP to UICC** **3 ⇨ HTTP to UICC** **11 ⇨ SMS to ESE/SMC** **12 ⇨ HTTP to ESE/SMC**	Integer	O	1

Table 3-71: Additional output parameters for SECommandsGenerationAndRemoteExecution function

Where a `SE Command Result` is:

Data name	Description	Type	MOC	No.
`Command Execution Status`	Indicates whether the `SE Command` has been executed or not. The values are: **'Executed-Success'** **'Executed-WithWarning'** **'Executed-Failed'** **'NotExecuted'** **'DeliveredWithNoResponse'** A detailed description of the status is provided in section 3.5.1.2.1.	`Enumeration {Executed-Success, Executed-WithWarning, Executed-Failed, NotExecuted, DeliveredWithNoResponse}`	M	1
`Command Status Code Data`	Provides the reason of the `Command Execution Status` value when the `Command Execution Status` is `'Executed-WithWarning'` or `'Executed-Failed'`. The possible values depend on the `SE Command`. A normative list of codes is provided in section 6 but additional codes may be defined.	`Command Status Code Data`	C	1
`Response APDU`	When the `SE Command` has been executed (`Command Execution Status` = `'Executed-Success'` or `'Executed-WithWarning'` or `'Executed-Failed'`), this field MAY contain (function provider decision) the response APDU of the last APDU executed by the SE. This might be the last APDU of the script (in case of correct execution of the script), or the first failed APDU (in case an error is encountered during the script execution). If this `Response APDU` field is provided, whatever the script format used by the function provider to send the APDU script (either Compact or Expanded – following ETSI TS 102 226 [14]), this field SHALL contain the APDU response data first, followed by the APDU status word (without any separator). In some cases, the `Response APDU` field might not be relevant. For example, for the loading of an Executable Load File that has not generated any communication up to the Secure Element because the ELF is already present in the SE, then the Response APDU field is obviously not present (as nothing has been executed by the Secure Element). In any case, even if data are sent back by the Secure Element, this is the function provider decision to provide the `Response APDU` field or not.	`Byte[]`	O	1

Table 3-72: `SE Command Result` type

Where a `Command Status Code Data` is:

Data name	Description	Type	MOC	No.
`Subject`	Represents the system element concerned by the exception.	`Object Identifier`	M	1
`Reason`	Represents the reason of the exception.	`Object Identifier`	M	1

Table 3-73 `Command Status Code Data` type

Optionally, additional output data MAY be returned in each `SE Command Result`. Refer each single `SE Command` definition.

3.5.1.2.1 Error Management in SE Commands Processing

The output data of the "SE Commands generation and remote execution" function contains a list of SE Command Result: for each single SE Command contained in the function input data, a corresponding SE Command Result MAY be provided in the output data.

The basic principles of the processing of the "SE Commands generation and remote execution" function are the following:

- The SE Command order SHALL be respected, even if it is grouped into a single OTA transport message or session for optimization purpose.
- Except if the `Stop On Warning` input data if set to `'True'`, the processing of the "SE Commands generation and remote execution" function SHALL continue if a warning is encountered in the processing of a SE Command. Otherwise, the processing of the "SE Commands generation and remote execution" function SHALL stop as soon as a warning is encountered.
- The processing of the "SE Commands generation and remote execution" function SHALL stop as soon as an error is encountered in the processing of a SE Command.

As a consequence, when a warning (`Stop On Warning` input data being set to `'True'`) or an error occurs on a SE command, all the following SE Commands will not be processed.

Within each single SE Command Result, a command execution status (the `Command Execution Status` output data) SHALL specify whether the overall processing of the SE Command (i.e. the APDU script generation, then the OTA communication, and finally the card execution) has been processed:

- Correctly (`'Executed-Success'`): APDUs have been generated, sent and executed correctly,
- Correctly with some warnings (`'Executed-WithWarning'`): the objective of the SE Command execution has been reached, even if some actions or APDUs have not been fully executed as normally planned (thus the "warning" aspect). For example, prior to sending the ELF loading APDU to a Secure Element, the function provider may check in its database whether the load file is already present in the SE, or may perform an on-line audit of the SE. If the ELF already exists in the Secure Element, then no APDU will be sent to the SE, and the `'Executed-WithWarning'` status will be returned to the function caller: initial request (ELF loading) is done, but without performing all the actions.

- With error (`'Executed-Failed'`): SE Command execution has failed, meaning that the Secure Element has rejected the given APDUs, and the objective of the SE Command execution has not been reached (the state of the Secure Element does not comply with what the SE Command initially requested),
- Or not processed at all (`'NotExecuted'`): the SE Command has not been executed at all. This happens for example if the execution of a previous SE Command has failed.

The reliability of the OTA connectivity is so that, it may append that even if data have been sent to a SE, the status of execution sent back by the SE on the network is lost. In this case, the command execution status SHALL be `'DeliveredWithNoResponse'`.

As for any function, a global status is specified in the output parameters: the `Status` field of the `Function Execution Status` output data. Its value, together with the `Status Code Data` of the function (in case of an error, a warning or the expiration), depends on the overall statuses of all the `SE Command Result`:

- The global function processing status SHALL be set to `'Executed-Success'` if the execution status of all the SE Command is `'Executed-Success'`. The `Command Status Code Data` of the `SE Command Result` and the `Status Code Data` of the function SHALL NOT be present.
- The global function processing status SHALL be set to `'Executed-WithWarning'` if at least one of the SE Commands has an execution status set to `'Executed-WithWarning'`, the execution status of all the other SE Commands being `'Executed-Success'`.

  If the `Stop On Warning` input data was set to `'True'`, then the `Command Status Code Data` of the `SE Command Result` SHALL detail the reason of warning (and optionally the `Command Output Data` MAY provide the output data sent by the SE). The `Subject` and `Reason` fields of the `Status Code Data` of the function SHALL be set to the `'Stopped on warning'` warning code (see section 6).

  If the `Stop On Warning` input data was set to `'False'`, then the `Command Status Code Data` of all the SE Commands that raised a warning SHALL detail the reason of their respective warning (and MAY provide the output data sent by the SE in the `Command Output Data`). The `Status Code Data` of the function SHALL NOT be present.
- The global function processing status SHALL be set to `'Failed'` if at least one of the SE Commands has an execution status set to `'Executed-Failed'`.

  The `Command Status Code Data` of the SE Commands in error SHALL detail the reason that caused the SE Command error. This code SHALL also be set in the `Status Code Data` of the function. Optionally, the `Command Output Data` MAY provide the output data sent by the SE.
- The global function processing status SHALL be set to `'Expired'` if at least one of the SE Commands has an execution status set to `'DeliveredWithNoResponse'`.

  The `Command Status Code Data` of the SE Commands in expiration SHALL detail the reason that caused the SE Command expiration. The `Subject` and `Reason` fields of the `Status Code` of the function SHALL be set to the 'OTA transport error' code (see section 6).

Those rules imply that the `SE Command Results` with `'NotExecuted'` status can only be located at the end of the `SE Command Result` list. As a consequence, it is optional to explicitly put those `SE Command Results` in the `SE Command Result` list. This is especially the case if an error occurs before the processing of any of the SE Command: the list of SE Command Result MAY either be empty or MAY contain as many `SE Command Result` as `SE Command`, each of them having the `'NotExecuted'` status.

Note that the function validity period may expire:

- If expiration is raised before the execution of the first SE Command in the SE, then the corresponding `SE Command Result` SHALL be set to `'NotExecuted'` and this is the responsibility of the function requester system to cancel all server-side actions done for this SE Command.
- If expiration is raised after the execution of the first SE Command in the SE, then the corresponding `SE Command Result` SHALL be set to `'DeliveredWithNoResponse'` and the confirmation or cancellation of the actions on the function requester system is function requester-implementation dependant.

As stated before, in both cases, the `Status` field of the `Function Execution Status` output parameter of the function SHALL be set to `'Expired'`.

The following figures give example of the possible statuses of the function execution:

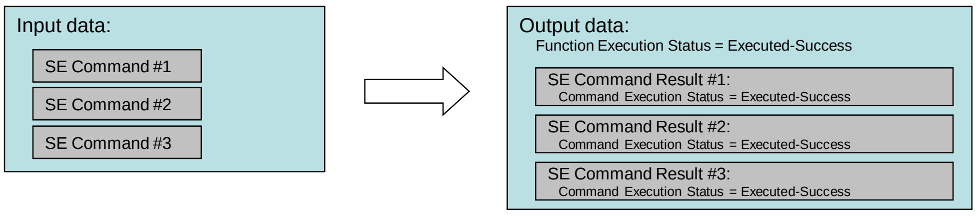

Input data:
SE Command #1
SE Command #2
SE Command #3
Output data:
Function Execution Status = Executed-Success
SE Command Result #1:
Command Execution Status = Executed-Success
SE Command Result #2:
Command Execution Status = Executed-Success
SE Command Result #3:
Command Execution Status = Executed-Success

Figure 3-15: All SE Commands processed successfully

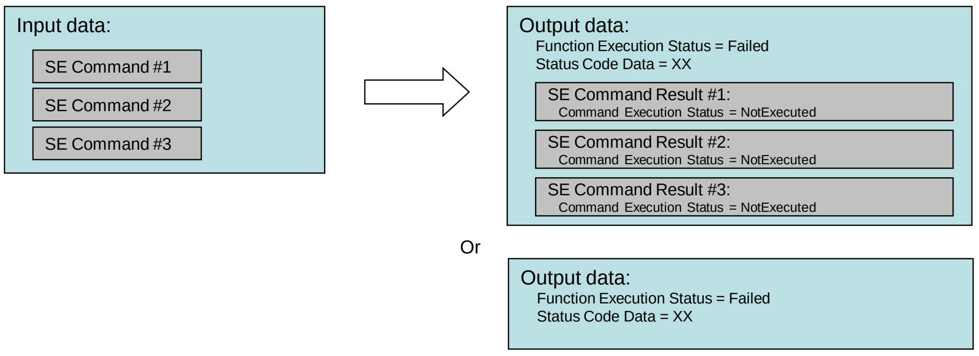

Input data:
SE Command #1
SE Command #2
SE Command #3
Output data:
Function Execution Status = Failed
Status Code Data = XX
SE Command Result #1:
Command Execution Status = NotExecuted
SE Command Result #2:
Command Execution Status = NotExecuted
SE Command Result #3:
Command Execution Status = NotExecuted
Or
Output data:
Function Execution Status = Failed
Status Code Data = XX

Figure 3-16: Error XX before processing any SE Command

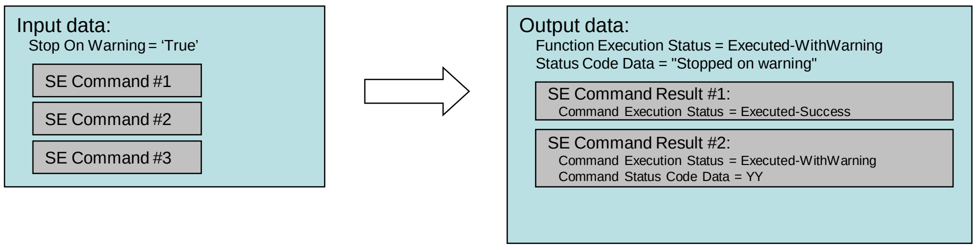

Input data:
Stop On Warning = 'True'
SE Command #1
SE Command #2
SE Command #3
Output data:
Function Execution Status = Executed-WithWarning
Status Code Data = "Stopped on warning"
SE Command Result #1:
Command Execution Status = Executed-Success
SE Command Result #2:
Command Execution Status = Executed-WithWarning
Command Status Code Data = YY

Figure 3-17: The 2nd SE Command had a Warning 'YY' and `Stop On Warning` was set to `'True'`

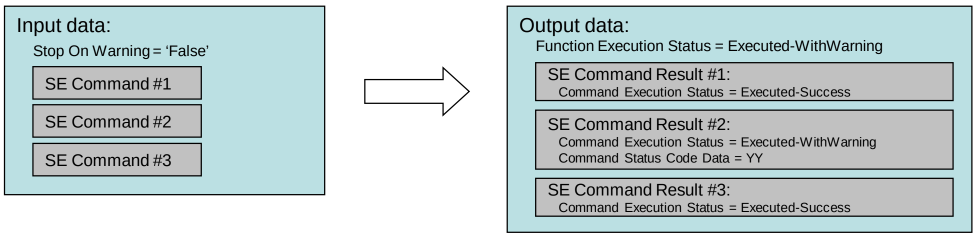

Input data:
Stop On Warning = 'False'
SE Command #1
SE Command #2
SE Command #3
Output data:
Function Execution Status = Executed-WithWarning
SE Command Result #1:
Command Execution Status = Executed-Success
SE Command Result #2:
Command Execution Status = Executed-WithWarning
Command Status Code Data = YY
SE Command Result #3:
Command Execution Status = Executed-Success

Figure 3-18: The 2nd SE Command had a Warning 'YY' and `Stop On Warning` was set to `'False'`

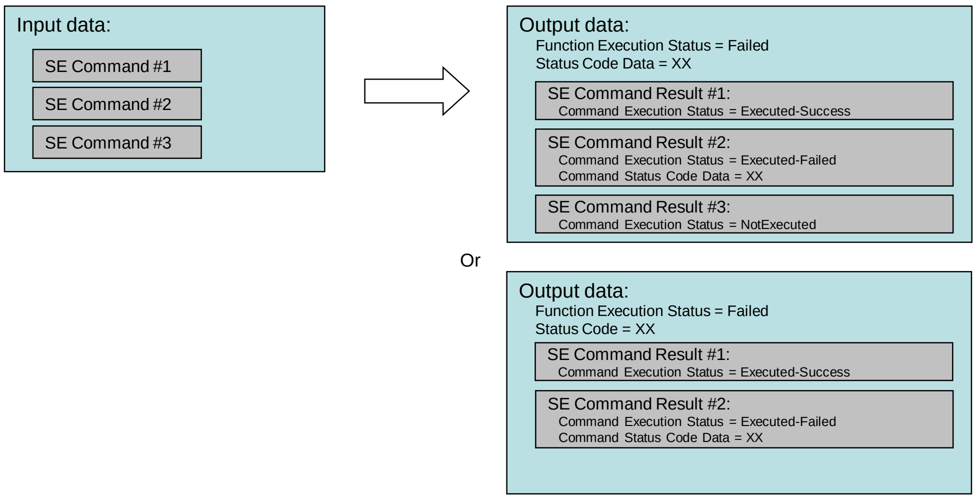

Input data:
SE Command #1
SE Command #2
SE Command #3
Output data:
Function Execution Status = Failed
Status Code Data = XX
SE Command Result #1:
Command Execution Status = Executed-Success
SE Command Result #2:
Command Execution Status = Executed-Failed
Command Status Code Data = XX
SE Command Result #3:
Command Execution Status = NotExecuted
Or
Output data:
Function Execution Status = Failed
Status Code = XX
SE Command Result #1:
Command Execution Status = Executed-Success
SE Command Result #2:
Command Execution Status = Executed-Failed
Command Status Code Data = XX

Figure 3-19: The 2nd SE Command fails with Error 'XX'

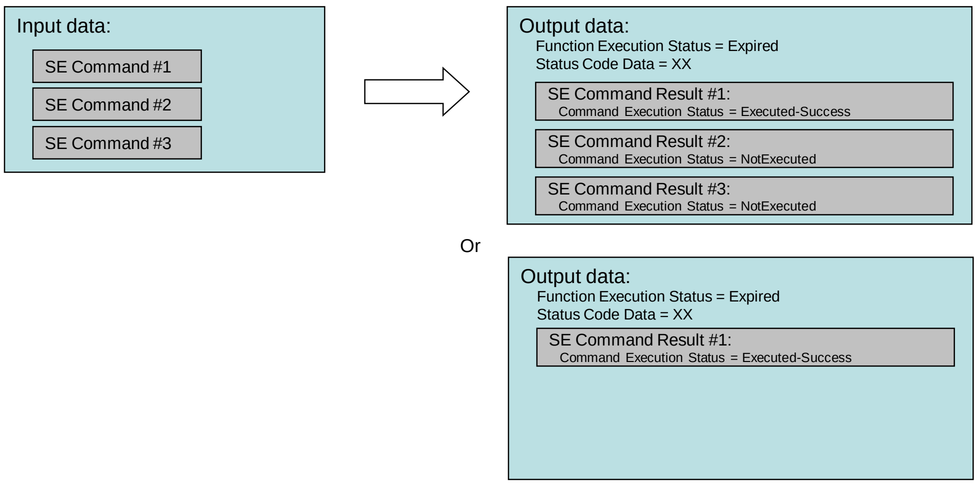

Input data:
SE Command #1
SE Command #2
SE Command #3
Output data:
Function Execution Status = Expired
Status Code Data = XX
SE Command Result #1:
Command Execution Status = Executed-Success
SE Command Result #2:
Command Execution Status = NotExecuted
SE Command Result #3:
Command Execution Status = NotExecuted
Or
Output data:
Function Execution Status = Expired
Status Code Data = XX
SE Command Result #1:
Command Execution Status = Executed-Success

Figure 3-20: The 2nd SE Command expires before the first SE Command execution in the SE

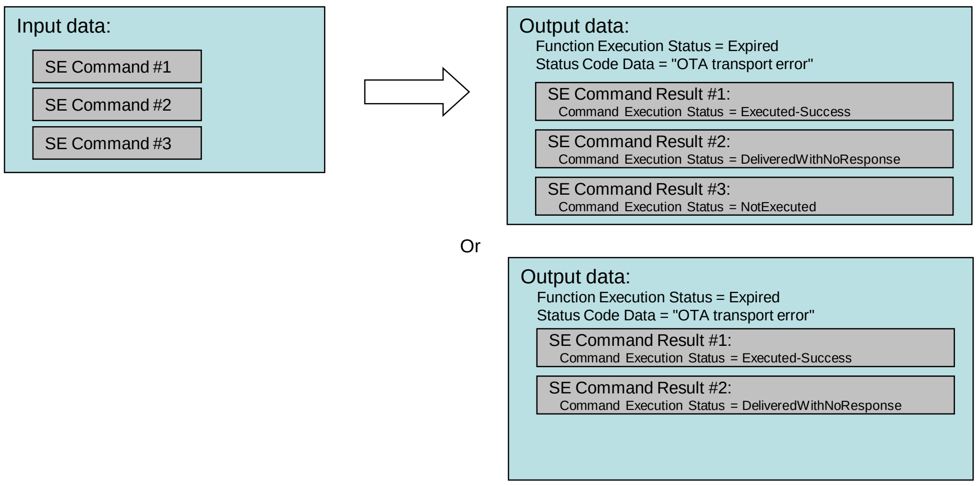

Input data:
SE Command #1
SE Command #2
SE Command #3
Output data:
Function Execution Status = Expired
Status Code Data = "OTA transport error"
SE Command Result #1:
Command Execution Status = Executed-Success
SE Command Result #2:
Command Execution Status = DeliveredWithNoResponse
SE Command Result #3:
Command Execution Status = NotExecuted
Or
Output data:
Function Execution Status = Expired
Status Code Data = "OTA transport error"
SE Command Result #1:
Command Execution Status = Executed-Success
SE Command Result #2:
Command Execution Status = DeliveredWithNoResponse

Figure 3-21: The 2nd SE Command expires after the first SE Command execution in the SE

3.5.1.3 The SE Commands

3.5.1.3.1 SE Commands for Supplementary Security Domain Management

3.5.1.3.1.1 Creating a Security Domain

Security Domains are GlobalPlatform Applications that are instantiated like any other classical Application. As a consequence, the `InstantiateApplicationCommand` (see section 3.5.1.3.2.3) SHALL directly be used for SSD creation but the `Executable Load File AID` and `Executable Module AID` input data are optional (only set if known by the function requester).

Note that in a very specific use case, it is possible that a (set of) SSD has been instantiated at the Secure Element manufacturing time, but not yet allocated to any SDM. As a consequence, if the function caller requests for the instantiation of a SSD, and if this is the function provider responsibility to manage AID generation (model #2, as mentioned in section 3.2.2), then the function provider MAY allocate one of the pre-created SSD to the function caller, without performing any OTA dialog up to the SE.

The `Command Execution Status` SHALL be set to `'Executed-With Warning'` and the `Subject` and `Reason` fields of the `Command Status Code Data` to 'Application is already instantiated'. The AID and the TAR (if any) of the allocated SSD SHALL be returned as output data (the `Application AID` and `Application TAR` output data of the `SE Command Result`).

3.5.1.3.1.2 Creating the First KeySet of a Security Domain

Command Name: `CreateFirstSSDKeysetCommand`

Description:

This SE Command requests the creation of the first keyset of a Supplementary Security Domain. The description of the keyset to be created is provided as input parameter (the `KeySet Description` input data).

In order to enable OTA sessions optimization when several types of keysets need to be created in a Security Domain through Simple mode functions (e.g. a SCP80 together with a SCP81 keyset), the `CreateFirstSSDKeysetCommand` function can be called for the creation of the first keyset of each of the keyset types of a Security Domain.

The Supplementary Security Domain to be personalized is identified by its AID (the `SSD AID` input data).

Using this AID, the function provider MAY for example check that the given AID is allowed for the service, and that the requester is allowed to handle such SSD. If not, the function provider SHOULD reject the function call. These verifications depend on the implementation of the function provider and are out of scope of this specification.

The SSD AID MAY be excluded by the function caller. If so, this is the responsibility of the function provider to determine it, according to business agreement with the function caller (e.g. get the SSD AID of a preceding InstantiateApplicationCommand of the same SE Commands series).

The mode to be used for the key provisioning SHALL be selected: choice of the mode implies different types of input data (the `Key Provisioning Data` input data) and output data (the `Key Provisioning Output Data` output data) that implicitly specify the key provisioning mode.

The possible key provisioning modes are:

- Basic Create: All keys of the keyset are provided by the function requester (the `Key Version Number` and the `KeySet` input data). The key values are encrypted using a transport key which has been previously shared between the function provider and the function caller (only the transport key index is then provided as input data: `Transport Key Index`).

  Optionally, the initial value of the sequence counter MAY be provided as input data.

- Basic Random Create. Key values of the keyset are randomly generated by the function provider. Only the Key Version Number and the requested initial value of the synchronization counter of the to-be-created keyset are provided by the function caller (the `Key Version Number` input data).

  The keyset is then returned to the function requester (the `KeySet` output data) together with the true value of the synchronization counter (see below the use case where SSD keys have already been created in factory). The key values are encrypted using a transport key that has been previously shared between the function provider and the function caller (only the transport key index is then provided as output data: `Transport Key Index`).

- Basic Diversified Create: Key values are diversified by the function provider, using a master key and a diversification algorithm that has been previously shared between the function provider and the function requester (out of scope of this specification).

  No key value is sent nor received through the `CreateFirstSSDKeysetCommand` SE Command as the function requester is also able to re-generate the key values for using them directly, or for updating them autonomously.

  The function caller however provides the Key Version Number (the `Key Version Number` input data) and the requested initial value of the synchronization counter of the to-be-created keyset, together with the identifier of the Master Key to be used for keys generation (the `Master Key Index` input data).

- CCCM Scenario #1 (Pull Model), using PK scheme. Key values of the keyset are generated on-card. Those key values or the Randomly Generated Key used to generate those keys are returned ciphered by OTA to the function requester (the `Encrypted Key Value` output data). The key values or the RGK Master Key have been ciphered on-card using the SSD owner certificate public key provided by the function requester (the `SSD Owner Certificate` input data), and signed using the CASD certificate private key.

  Optionally, the requested initial value of the synchronization counter MAY be provided as input data.

- CCCM Scenario #2.A (Push Model with Application Provider Certificate). Key values of the keyset are generated off-card, either randomly, or from a RGK Master Key. Those key values or the RGK master key are provided by the function requester (the `Encrypted Key Value` input data) in order to be sent OTA to the Secure Element by the function provider. They are provided ciphered using the CASD certificate public key and signed by the SSD owner certificate private key. The SSD owner certificate (the `SSD Owner Certificate` input data) is also loaded into the SSD.

  Optionally, the requested initial value of the synchronization counter MAY be provided as input data.

- CCCM Scenario #2.B (Push Model without Application Provider Certificate). Key values of the keyset are generated off-card, either randomly, or from a RGK Master Key. Either those key values or the RGK master key used to generate those keys are provided (ciphered) by the function requester (the `Encrypted Key Value` input data). No SSD owner certificate is loaded into the SSD.

  Optionally, the requested initial value of the synchronization counter MAY be provided as input data.
- CCCM Scenario #3 (Elliptic Curve Key Agreement - ElGamal scheme): secure channel base key or full key set is derived both in the SE and by the SSD owner using jointly provided data: the SSD owner ephemeral (one-time) public key (the `SSD Owner Ephemeral Public Key` input data), the CASD static public key (in the CASD certificate), and optionally a random number provided by the SSD (the `Derivation Random` (DR) output data).

Whatever the mode that is used, the keys created by the `CreateFirstSSDKeysetCommand` function are the ownership of the function caller. As a consequence, the function provider SHOULD NOT store the created key values in its information system; only the function caller SHOULD be able to use those keys.

This rule is obvious for confidential key management as the function caller requests for the key creation in a confidential way among the function provider, but the rule is also true for the Basic Create, Basic Random Create and Basic Diversified Create modes.

Transport key indexes required in the Basic Create and Basic Random Create modes are determined as a business agreement between the function caller and the function provider. This single index SHALL implicitely determine the transport key value, the algorithm (3DES, AES…), the mode of operation (CBC, ECB…) and any other key data to be used for the transport security.

Note that a usual recommendation is that the transport key strength is higher or equal to the transported key strength.

Moreover, in case of usage of a specific transport key leading to a padding of the key values, the `KeySet Description` profile shall provide the exact key length so that the function caller or function provider is able to remove the padding and retrieve the exact key values.

Note that in a very specific use case, it is possible that a SSD has been instantiated and personalized at the Secure Element manufacturing time, and thus known by the function provider. As a consequence, if the function caller requests for the creation of a first keyset of a keyset type for this SSD, then the function provider MAY return the pre-created SSD key values to the function caller according to the following rules, and then delete the keys from its information system (see also section 3.5.1.3.1.1).

- If the first keyset creation request is in Basic Random Create mode with a `Key Version Number` that corresponds to the existing keyset, then the keyset value SHALL be returned to the function requester (in the `KeySet` output data), without any OTA dialog up to the SE.

  In case the optional `Initial Sequence Counter` provided as input parameter does not match the current value present in the SE, then the real value SHALL also be returned in the `Sequence Counter` output data.

  The `Command Execution Status` SHALL be set to `'Executed-WithWarning'` and the `Subject` and `Reason` fields of the `Command Status Code Data` to 'First keyset is already created'.

- If the first keyset creation request is in Basic Random Create mode with a Key Version Number that is different from the ones existing in the SE, then the function provider SHALL perform the OTA keyset creation into the SSD, by using the existing keyset. The keyset created in factory will then remain unknown by the function requester.

  The Command Execution Status SHALL be set to 'Executed-Success'.

  Note that depending of the type of key that has been created in factory (SCP80, SCP81, SCP02, etc.), the new keyset creation OTA process might not be technically possible. If so, the Command Execution Status SHALL be set to 'NotExecuted' and the Subject and Reason fields of the Command Status Code Data to 'First keyset is already created'.

- If the first keyset creation request is in Basic Create mode with a Key Version Number that corresponds to the existing keyset, then the function provider SHALL update the keyset value of the SE with the provided keyset value.

  The Command Execution Status SHALL be set to 'Executed-Success'.

  If the keyset update is not possible, then key values provided in input parameters SHALL be ignored and the existing keyset value SHALL be returned as output parameter (the KeySet output data). The Command Execution Status SHALL be set to 'Executed-WithWarning' and the Subject and Reason fields of the Command Status Code Data to 'First keyset is already created'.

- If the first keyset creation request is in Basic Create mode with a Key Version Number that is different from the ones existing in the SE, then the function provider SHALL perform the OTA keyset creation into the SSD, by using the existing keyset. The keyset created in factory will then remain unknown by the function requester.

  The Command Execution Status SHALL be set to 'Executed-Success'.

  Note that depending of the type of key that has been created in factory (SCP80, SCP81, SCP02, etc.), the new keyset creation OTA process might not be technically possible. If so, the Command Execution Status SHALL be set to 'NotExecuted' and the Subject and Reason fields of the Command Status Code Data to 'First keyset is already created'.

- If the first keyset creation request is in Basic Diversified Create mode with a Key Version Number and a Master Key Index that corresponds to the existing keyset, then no OTA dialog up to the SE should be done.

  In case the optional Initial Sequence Counter provided as input parameter does not match the current value present in the SE, then the real value SHALL be returned in the Sequence Counter output data.

  The Command Execution Status SHALL be set to 'Executed-WithWarning' and the Subject and Reason fields of the Command Status Code Data to 'First keyset is already created'.

- If the first keyset creation request is in Basic Diversified Create mode with a Key Version Number that corresponds to the existing keyset, but with a different Master Key Index, then the function provider SHALL update the keyset value of the SE with the provided keyset value.

  The Command Execution Status SHALL be set to 'Executed-Success'.

  If the keyset update is not possible, then the Command Execution Status SHALL be set to 'NotExecuted' and the Subject and Reason fields of the Command Status Code Data to 'First keyset is already created'.

- If the first keyset creation request is in Basic Diversified Create mode with a `Key Version Number` that is different from the ones existing in the SE, then the function provider SHALL perform the OTA keyset creation into the SSD, by using the existing keyset. The keyset created in factory will then remain unknown by the function requester.

  The `Command Execution Status` SHALL be set to `'Executed-Success'`.

  Note that depending of the type of key that has been created in factory (SCP80, SCP81, SCP02, etc.), the new keyset creation OTA process might not be technically possible. If so, the `Command Execution Status` SHALL be set to `'NotExecuted'` and the `Subject` and `Reason` fields of the `Command Status Code Data` to `'First keyset is already created'`.

- If the first keyset creation request is in any of the Confidential Card Content Management scenario, then the `Command Execution Status` SHALL be set to `'NotExecuted'` and the `Subject` and `Reason` fields of the `Command Status Code Data` to 'First keyset is already created'.

The `CreateFirstSSDKeysetCommand` is composed of the following data:

Input data name	**Description**	**Type**	**MOC**	**No.**
`SSD AID`	AID of the SSD into which keys are provisioned.	`AID`	O	1
`KeySet Description`	The description of the keyset to be created. In this version of the specification, it could be a `Simple KeySet Profile` (providing a simple definition of SCP02, SCP03 and SCP80 keysets) and a `SCP81 KeySet Profile` (providing a description of SCP81 keysets), but other type structure may be defined in the future, for example for a more descriptive and key per key description of the keyset type. This field is optional to ensure backward compatibility with the previous version of the specification. In case this field is not provided by the function caller, this is the function provider responsibility to determine the type of keyset to be created, based on business agreement.	`Simple KeySet Profile` *or* `SCP81 KeySet Profile` *or RFU*	O	1
`Key Version Number`	The Key Version Number of the to-be-created keyset. From `'01'` to `'7F'`, i.e. 1 to 127.	`Integer`	O	1
`Initial Sequence Counter`	The initial value of the Sequence Counter of the keyset. Does not apply to SCP81 keysets.	`Integer`	O	1

Input data name	Description	Type	MOC	No.
`Key Provisioning Data`	The key provisioning data (KPD). The key provisioning mode is implicitly specified by the choice of key provisioning data. The `Basic KPD`, `Basic Random KPD`, `Basic Diversified Create KPD`, `CCCM Scenario 1 using PK KPD`, `CCCM Scenario 2A KPD`, `CCCM Scenario 2B KPD`, and `CCCM Scenario 3 KPD` are defined below.	`Basic KPD` *or* `BasicRandom KPD` *or* `BasicDiversified KPD` *or* `CCCM Scenario1 using PK KPD` *or* `CCCM Scenario 2A KPD` *or* `CCCM Scenario 2B KPD` *or* `CCCM Scenario 3 KPD`	M	1

Table 3-74: Data of the `CreateFirstSSDKeysetCommand` SE Command

Note: the `AID` simple type is defined in section 3.1.1.1.

Where a `Simple KeySet Profile` is:

Data name	Description	Type	MOC	No.
`Profile`	The simple description of the keyset, specified as a profile. The following values are pre-defined: **1 ⇨ SCP02 16 Bytes** **11 ⇨ SCP03 16 Bytes** **12 ⇨ SCP03 24 Bytes** **13 ⇨ SCP03 32 Bytes** **21 ⇨ SCP80 DES 16 Bytes** **22 ⇨ SCP80 DES 24 Bytes** **23 ⇨ SCP80 AES 16 Bytes** **24 ⇨ SCP80 AES 24 Bytes** **25 ⇨ SCP80 AES 32 Bytes**	`Integer`	M	1

Table 3-75: `Simple KeySet Profile` type

Where a `SCP81 KeySet Profile` is:

Data name	Description	Type	MOC	No.
`PSK-TLS Key Length`	The length of the PSK-TLS key, in bytes. It SHALL be inferior or equal to 64 bytes.	`Integer`	M	1

Data name	Description	Type	MOC	No.
`DEK Profile`	The description of the Data Encryption Key (DEK) of the SCP81 keyset, specified as a profile. The following values are pre-defined: **1 ⇨ DES 16 Bytes** **2 ⇨ DES 24 Bytes** **11 ⇨ AES 16 Bytes** **12 ⇨ AES 24 Bytes** **13 ⇨ AES 32 Bytes**	`Integer`	M	1

Table 3-76: `SCP81 KeySet Profile` type

Where a `Basic KPD` is:

Data name	Description	Type	MOC	No.
`KeySet`	The value of the keys of a keyset. The `KeySet` type is described below.	`KeySet`	M	1

Table 3-77: `Basic KPD` type

Where a `KeySet` is:

Data name	Description	Type	MOC	No.
`Key`	The value of a single key to be created within the keyset, encrypted by the transport key referenced by the `Transport Key Index`. Having several occurrences of this field enables to specify several keys. The `SD Key` type is described below.	`SD Key`	M	1..N
`Transport Key Index`	The index of the transport key that has been used to cipher the Key value.	`Integer`	M	1

Table 3-78: `KeySet` type

Where a `SD Key` is:

Data name	Description	Type	MOC	No.
`Key Identifier`	The Key Identifier within the keyset.	`Byte`	M	1
`Encrypted Key Value`	The key value, encrypted with the Transport Key.	`Byte[]`	M	1

Data name	Description	Type	MOC	No.
KCV	The Key Check Value. The Key Check Value computation depends on the type of key (DES, AES, PSK…) and is depicted in the Global Platform Card Specification [1] and in the Amendment B of the Card Specification [4]. This field is optional to ensure backward compatibility with the previous version of the specification, but it is strongly recommended to use it. In case this field is not provided by the function caller, this is the function provider responsibility to compute the Key Check Value if required.	Byte[]	O	1

Table 3-79: SD Key type

Where a Basic Random KPD does not provide any additional input data

Where a Basic Diversified KPD is:

Data name	Description	Type	MOC	No.
Master Key Index	The index of the Master Key that is to be used to diversify the key values. This field is optional as the Master Key Index choice may be under function provider responsibility.	Integer	O	1

Table 3-80: Basic Diversified KPD type

Where a CCCM Scenario 1 Using PK KPD is:

Data name	Description	Type	MOC	No.
SDIN	The SSD's Security Domain Image Number.	String	O	1
SSD Owner Certificate	The PK certificate of the SSD Owner. The format of this field exactly corresponds to the output data of the EnrollSSDOwnerCertificate function. It is a byte array which content corresponds to the full content of tag '7F21' (including the two '7F21' bytes) defined in table 10-2 "Structure of AP Public Key Certificate with Message Recovery" of the UICC Card Configuration document [7].	Byte[]	M	1

Table 3-81: CCCM Scenario 1 Using PK KPD type

Where a CCCM Scenario 2A KPD is:

Data name	Description	Type	MOC	No.
SDIN	The SSD's Security Domain Image Number.	String	O	1

Data name	Description	Type	MOC	No.
`SSD Owner Certificate`	The PK certificate of the SSD Owner. The format of this field exactly corresponds to the output data of the `EnrollSSDOwnerCertificate` function. It is a byte array which content corresponds to the full content of tag `'7F21'` (including the two `'7F21'` bytes) defined in table 10-2 "Structure of AP Public Key Certificate with Message Recovery" of the UICC Card Configuration document [7].	`Byte[]`	M	1

`Key Generation Mode`	Specifies the mode of generation of the keyset keys values. This field is linked with the `Encrypted Key Value` input data:
	• If the keyset key values are directly provided by the function caller, then the `Key Generation Mode` field SHALL be set to `'3 secure channel key'`, and the `Encrypted Key Value` SHALL contain the concatenation of the encrypted key values of the keyset.
	• If only the RGK master key is to be provided by the function caller an sent to the Secure Element, then the `Key Generation Mode` field SHALL be set to `'1 secure channel base key'`, and the `Encrypted Key Value` SHALL contain the RGK master key encrypted value.
	Refer to Amendment A of GP 2.2 Card Specification [3].
	This field is optional to ensure backward compatibility with the previous version of the specification. As a consequence, in case this field is not provided by the function caller, the provided `Encrypted Key Value` SHALL be considered as a single RGK Master Key.
	Note however that the provided value SHOULD be consistent with values allowed in the GlobalPlatform UICC Configuration [7] and the SE Configuration [8] documents.
		`Enumeration {1 secure channel base key,`
		`3 secure channel key}`
			O	1

`Encrypted Key Value`	The value of the key to be sent to the Secure Element, ciphered with the CASD public key. This could be the single RGK Master Key value, or the concatenation of the encrypted key values of the keyset, according to the `Key Generation Mode`.	`Byte[]`	M	1
`Signature`	The signature associated to the key creation command. The signature is computed off-card by the SSD owner with its SSD owner certificate private key – refer to section "Pushing AP Secure Channel Keys" of the UICC Card Configuration [7].	`Byte[]`	M	1

Table 3-82: `CCCM Scenario 2A KPD` type

Where a `CCCM Scenario 2B KPD` is:

Data name	Description	Type	MOC	No.
`SDIN`	The SSD's Security Domain Image Number.	`String`	O	1

`Key Generation Mode`	Specifies the mode of generation of the keyset keys values. This field is linked with the `Encrypted Key Value` input data:
	• If the keyset key values are directly provided by the function caller, then the `Key Generation Mode` field SHALL be set to `'3 secure channel key'`, and the `Encrypted Key Value` SHALL contain the concatenation of the encrypted key values of the keyset.
	• If only the RGK master key is to be provided by the function caller an sent to the Secure Element, then the `Key Generation Mode` field SHALL be set to `'1 secure channel base key'`, and the `Encrypted Key Value` SHALL contain the RGK master key encrypted value.
	Refer to Amendment A of GP 2.2 Card Specification [3].
	This field is optional to ensure backward compatibility with the previous version of the specification. As a consequence, in case this field is not provided by the function caller, the provided `Encrypted Key Value` SHALL be considered as a single RGK Master Key.
		`Enumeration {1 secure channel base key,`
		`3 secure channel key}`
			O	1

`Encrypted Key Value`	The value of the key to be sent to the Secure Element, ciphered with the CASD public key. This could be the single RGK Master Key value, or the concatenation of the encrypted key values of the keyset, according to the `Key Generation Mode`.	`Byte[]`	M	1

Table 3-83: `CCCM Scenario 2B KPD` type

Where a `CCCM Scenario 3 KPD` is:

Data name	Description	Type	MOC	No.
`SDIN`	The SSD's Security Domain Image Number.	`String`	O	1

`Key Generation Mode`	Specifies the mode of generation of the keyset keys values.
	Refer to Amendment E of GP 2.2 Card Specification [6].
		`Enumeration {1 secure channel base key,`
		`3 secure channel key}`
			M	1

`ECC Key Length`	The length of the Elliptic Curve Cryptography keys.	`Enumeration {ECC-256,`
		`ECC-384,`
		`ECC-512,`
		`ECC-521}`
			M	1

`Scenario Parameter`	Scenario parameter as defined in Table 4-17 of the Amendment E of GP 2.2 Card Specification [6].	`Byte`	M	1

Data name	Description	Type	MOC	No.
`Host ID`	HostID as defined in Table 4-15 of the Amendment E of GP 2.2 Card Specification [6], excluding tag '84'.	`Byte[]`	C	1
`SSD Owner Ephemeral Public Key`	An ephemeral public key of the SSD Owner. It is a byte array which content corresponds to the content of tag `'7F49'`, excluding tag `'7F49'`.	`Byte[]`	M	1

Table 3-84: `CCCM Scenario 3 KPD` type

When the SE Command processing has been performed correctly (`Command Execution Status` is `'Executed-Success'` or `'Executed-WithWarning'`), the `SE Command Result` output data (detailed in section 3.5.1.2) SHALL include the following output parameters:

Else, none of these complementary output data will be returned.

Output data name	Description	Type	MOC	No.
`Key Provisioning Output Data`	The key provisioning output data (KPOD). The `Basic KPOD`, `Basic Random KPOD`, `Basic Diversified KPOD`, `CCCM Scenario 1 using PK KPOD`, `CCCM Scenario 2A KPOD`, `CCCM Scenario 2B KPOD`, and `CCCM Scenario 3 KPOD` are defined below.	`Basic KPOD` *or* `Basic Random KPOD` *or* `Basic Diversified KPOD` *or* `CCCM Scenario1 using PK KPOD` *or* `CCCM Scenario 2A KPOD` *or* `CCCM Scenario 2B KPOD` *or* `CCCM Scenario 3 KPOD`	M	1
`Sequence Counter`	The value of the sequence counter of the keyset. If not set, value is to be considered as 0.	`Integer`	O	1

Table 3-85: Output parameters for the `CreateFirstSSDKeysetCommand` SE Command

Where a `Basic KPOD` is:

Data name	Description	Type	MOC	No.
`KeySet`	The value of the keys of a keyset. The `KeySet` type is described above. Only present in the very specific case when keyset is already created in factory.	`KeySet`	O	1

Table 3-86: `Basic KPOD` type

Where a `Basic Random KPOD` is:

Data name	Description	Type	MOC	No.
`KeySet`	The value of the keys of a keyset. The `KeySet` type is described above.	`KeySet`	M	1

Table 3-87: `Basic Random KPOD` type

Where a `Basic Diversified KPOD` is:

Data name	Description	Type	MOC	No.
`Master Key Index`	The true index of the Master Key that has been used to diversify the key values. This index may be different from the `Master Key Index` provided as input data, in case for example : • This is the function provider responsibility to determine the Master Key index (so the `Master Key Index` input data was empty) • The function provider wants to force the rollover of the Master Key over a period of time (so the `Master Key Index` output data is different from the provided `Master Key Index` input data)	`KeySet`	M	1

Table 3-88: `Basic Diversified KPOD` type

Where a `CCCM Scenario 1 Using PK KPOD` is:

Data name	Description	Type	MOC	No.

`Key Generation Mode`	Specifies the mode of generation of the keyset keys values. This field is linked with the `Encrypted Key Value` input data:
	• If the keyset key values are directly sent back by the Secure Element, then the `Key Generation Mode` field SHALL be set to `'3 secure channel key'`, and the `Encrypted Key Value` SHALL contain the concatenation of the encrypted key values of the keyset.
	• If only the RGK master key is sent back by the Secure Element, then the `Key Generation Mode` field SHALL be set to `'1 secure channel base key'`, and the `Encrypted Key Value` SHALL contain the RGK master key encrypted value.
	Refer to Amendment A of GP 2.2 Card Specification [3].
	This field is optional to ensure backward compatibility with the previous version of the specification. As a consequence, in case this field is not provided by the function caller, the provided `Encrypted Key Value`
		`Enumeration {1 secure channel base key,`
		`3 secure channel key}`
			O	1

Data name	Description	Type	MOC	No.
	SHALL be considered as a single RGK Master Key.			
`Encrypted Key Value`	The value of the key sent back by the Secure Element, ciphered with the SSD owner certificate public key. This could be the single RGK Master Key value, or the concatenation of the encrypted key values of the keyset, according to the `Key Generation Mode`.	`Byte[]`	M	1
`Signature`	The signature associated to the key creation command. The signature is computed on-card by the CASD using its private key, including the Remainder Data – refer to section "Pulling AP Secure Channel Keys" of the UICC Card Configuration [7].	`Byte[]`	M	1

Table 3-89: `CCCM Scenario 1 Using PK KPOD` type

Where a `CCCM Scenario 2A KPOD` does not provide any additional output data.

Where a `CCCM Scenario 2B KPOD` does not provide any additional output data.

Where a `CCCM Scenario 3 KPOD` is:

Data name	Description	Type	MOC	No.
`Stored Data`	CRT TLV with all sub TLVs as provided in the STORE DATA DGI '00A6' command, including the '00A6' tag.	`Byte[]`	M	1
`Derivation Random`	A random number generated in the SE for additional entropy in the key derivation process.	`Byte[]`	C	1
`Receipt`	A Message Authentication Code (MAC) calculated with the secure channel MAC over TLVs in the command and DR (if used), as specified in section 4.8 "Confidential Setup of Secure Channel Keys using ECKA" of Amendment E of GP 2.2 Card Specification [6].	`Byte[]`	M	1

Table 3-90: `CCCM Scenario 3 KPOD` type

Note that if the `Scenario Parameter` input data for scenario #3 request for a key derivation process using the Host ID and the Card ID according to Amendment E of GP 2.2 Card Specification [6], then the function caller needs the Card Unique Data value in order to calculate the secure channel keys. This Card Unique Data can for example be obtained using the `GetMobileSubscriptionSEIdentifiers` function; however remember that a Secure Element can be identified by several types of identifiers. As a consequence, if CCCM Scenario #3 is to be used, then it is required that the ecosystem is being setup so that the `GetMobileSubscriptionSEIdentifiers` returns Card Unique Data as Secure Element Identifiers.

3.5.1.3.1.3 Quota Management

Editor's note: this part will be specified in a future version of the specification.

3.5.1.3.1.4 Deleting a Supplementary Security Domain

Security Domains are GlobalPlatform Applications that are deleted like any other Application. As a consequence, the `DeleteCommand` (see section 3.5.1.3.2.7) SHALL directly be used for the SD deletion.

3.5.1.3.2 SE Commands for Application Management

SE Commands for Application management enables to:

- Load an Executable Load File
- Extradite an Application or an Executable Load File
- Instantiate an Application
- Make selectable an Application
- Update the registry of an Application
- Lock and Unlock an Application
- Delete an Application or an Executable Load File

Following the GlobalPlatform Card Specification [1], and for some of these commands, a set of specific parameters might be set in complement to the functional input parameters such as Executable Load File AID or Application AID. Restricted to the Simple mode, this is the case for:

- Executable Load File loading: tag `'EF'` of the Load Parameters fields (tag `'B6'` is used only for Delegated mode)
- Application instantiation: Application privileges and tags `'C9'`, `'EF'`, `'EA'` of the Install Parameters fields (tag `'B6'` is used only for Delegated mode)
- Making selectable an Application: Application privileges and tag `'EF'` of Make Selectable Parameters fields (tag `'B6'` is used only for Delegated mode)
- Application registry update: System specific parameters corresponding to the `'EF'` tag of the install parameters field.

In most of the case (except for the `'C9'` tag of the Install Parameters fields -Application Specific Parameters-), this information is static and can be managed by the function provider as part of its definition of the mobile-NFC service.

As a consequence, the "SE Commands generation and remote execution" function provider SHALL manage such parameters through parameters profiles. A parameters profile SHALL contain a consistent set of Parameters fields' content for Executable Load File loading, Application instantiation, and for making selectable an Application. Several profiles may be associated to a particular Application of a mobile-NFC Service, as the Application may be loaded, instantiated, made selectable or have its registry updated with different behaviors depending on the context and on the final usage of the Application/mobile-NFC service.

For each of the three SE Commands mentioned above, the function requester MAY then specify which parameters profile shall be used for the loading, instantiation, making selectable, and registry update (the `Parameters Profile Identifier` input data). If no profile is specified, the function provider SHALL select a default set of parameters.

Note that for the specific case of the Application Specific Parameters (tag `'C9'`) of Application instantiation, the function provider still has the opportunity to overwrite the parameters values (and only them) as it may be required to dynamically generate them at Application instantiation time: usage of values contained in a static profile would not be convenient in this case.

3.5.1.3.2.1 Loading of an Executable Load File

Command name: `LoadELFCommand`

Description:

This SE Command requests the loading of an Executable Load File.

The Executable Load File to load is identified by its AID (the `Executable Load File AID` input data). The ELF byte code (the `Byte Code` input data) MAY be provided as an input parameter of the function. If not provided as an input data, the byte code SHALL be retrieved by the function provider in its own information system, thanks to the ELF AID.

The Executable Load File loading may require setting Load Parameters. This is the function provider responsibility to manage such parameters set through parameters profiles (see section 3.5.1.3.2). The function requester MAY then specify which parameters profile shall be used for the ELF loading (the `Parameters Profile Identifier` input data). If no profile is specified, the function provider SHALL select a default set of parameters.

If this Executable Load File has been encrypted using the `'D4'` key (refer to Amendment A of GP 2.2 Card Specification [3]), then it SHALL be specified (the `Byte Code Encrypted` input data) in order the function provider to correctly generate the command for the SE.

Optionally, DAP blocks MAY be given (the `DAP Block` input data list).

Optionally, the Executable Load File MAY require to be immediately extradited to a SSD (the `SSD AID For Extradition` input data).

Using this information, the function provider MAY for example check that the given AIDs are allowed for the service, that the target SSD is an allowed SSD, and that the requester is allowed to handle such Executable Load File. If not, the function provider SHOULD reject the function call. Such verifications depend on the implementation of the function provider and are out of scope of this specification.

The `LoadELFCommand` is composed of the following data:

Data name	Description	Type	MOC	No.
`Executable Load File AID`	AID of the Executable Load File (i.e. package) to be loaded.	`AID`	M	1
`Byte Code`	The byte code of the Executable Load File, excluding the Tags and Length specified in the GP 2.2 Card Specification [1] and in the Amendment A of the GP 2.2 Card Specification [3] ('C4' or 'D4').	`Byte[]`	O	1

Data name	Description	Type	MOC	No.
`Byte Code Encrypted`	When the byte code is provided as an input data, this field specifies whether the byte code is encrypted or not (according to the Amendment A of the GP 2.2 Card Specification). If this field is not present, the Executable Load File SHALL be considered as not encrypted. If no byte code is provided as an input data, this field SHALL NOT be present or SHOULD be ignored if present.	`Boolean`	O	1
`Parameters Profile Identifier`	Specifies the identifier of the parameters profile to be used. For Executable Load File loading, a parameters profile defines the content of tag `'EF'` of the Load Parameters fields. If not present, default parameters values SHALL be used by the function provider.	`Integer`	O	1
`DAP Block`	DAP Block of the Executable Load File. The `DAP Block` type is described below. Having several occurrences of this field enables to specify several DAP blocks.	`DAP Block`	O	0..N
`SSD AID For Extradition`	AID of the SD to which the Executable Load File will be immediately extradited, if required. If not present, no extradition SHALL be done.	`AID`	O	1

Table 3-91: Data of the `LoadELFCommand` SE Command

Note: the `AID` simple type is defined in section 3.1.1.

Where a `DAP Block` is:

Data name	Description	Type	MOC	No.
`Security Domain AID`	AID of the Security Domain that SHALL verify the Data Block Signature.	`AID`	M	1
`Data Block Signature`	The value of the Data Block Signature.	`String`	M	1

Table 3-92: `DAP Block` type

3.5.1.3.2.2 Extraditing Application or Executable Load File

Command name: `ExtraditeCommand`

Description:

This SE Command requests the extradition of an Application or an Executable Load File to a targeted Security Domain.

The Application, the Executable Load File to be extradited and the targeted SSD are identified by their AIDs (the `Application or Executable Load File AID` input data and the `SSD AID For Extradition` input data).

Using these AIDs, the function provider MAY for example check that the given AIDs are allowed for the service, that the target SSD is an allowed SSD, and that the requester is allowed to handle such command. If not, the function provider SHOULD reject the function call. Such verifications depend on the implementation of the function provider and are out of scope of this specification.

The `ExtraditeCommand` is composed of the following data:

Data name	Description	Type	MOC	No.
`Application or Executable Load File AID`	AID of the Application (either a simple Application or a Supplementary Security Domain) or Executable Load File to be extradited.	`AID`	M	1
`SSD AID For Extradition`	AID of the SD to which extraditing the Application or Executable Load File.	`AID`	M	1

Table 3-93: Data of the `ExtraditeCommand` SE Command

Note: the `AID` simple type is defined in section 3.1.1.

3.5.1.3.2.3 Instantiating an Application

Command name: `InstantiateApplicationCommand`

Description:

This SE Command requests the instantiation of a particular Application (either a Supplementary Security Domain or a simple Application).

The Executable Module to be instantiated is identified by its AID (the `Executable Module AID` input data) and by the AID of the Executable Load File to which it belongs to (the `Executable Load File AID` input date). Note that this information are mandatory for simple Applications creation, but they are optional is for the instantiation of Supplementary Security Domains (only set if known by the function requester).

For simple Applications, using these AIDs, the function provider MAY for example check that the given AIDs are allowed AIDs for the service, and that the requester is allowed to handle such Executable Load File and Application. If not, the function provider SHOULD reject the function call. Such verifications depend on the implementation of the function provider and are out of scope of this specification.

As mentioned in section 3.2.2, two modes are supported by GlobalPlatform for Application AID management:

- Model #1: Application AID generation is fully under the responsibility of the SDM (function caller): the function caller SHALL then provide the AID of the Application instance (the `Application AID` input data).

  In case the SE is a UICC, this is also the function caller responsibility to generate a TAR if required to have a direct OTA dialog with the Application. This TAR is not directly provided to the function provider (using an explicit input parameter); but it can be indirectly determined using the AID of the Application instance (extracting bytes 13 to 15 from the Application AID).

- Model #2: Application AID generation is fully under the responsibility of the Secure Element Issuer (function provider): the function caller will then not provide any Application instance AID (the `Application AID` input data is not present): the function provider will generate this AID instead and will return its value in the `Application AID` output data.

  In case the SE is a UICC, this is also the function provider responsibility to generate a TAR if required. As a TAR is not always required, the function requester SHALL explicitly request for its generation by using the `Generate TAR` input data). If so, the TAR generated by the function provider SHALL be returned in the `Application TAR` output data.

As an acceptable limitation, it is assumed that only a single TAR is managed for an Application instance.

Note that the AID of the parent Security Domain (the `Parent AID` output data) MAY be returned by the function provider in order for the function caller to locate where the newly created Application is instantiated.

The instantiation of an Application requires to define the Application privileges and to provide a set of install parameters: the system specific parameters, the application specific parameters and the UICC system specific parameters (if the SE is a UICC). This is the function provider responsibility to manage such privileges and parameters set through parameters profiles (see section 3.5.1.3.2). The function requester MAY then specify which parameters profile shall be used for the Application instantiation (the `Parameters Profile Identifier` input data). If no profile is specified, the function provider SHALL select a default set of parameters.

However, the function provider still has the opportunity to overwrite the application specific parameters (and only them) as they may be dynamically determined at Application instantiation time. In this case, the function requester SHALL provide the `'C9'` tag of the Install Parameter fields (using the `Application Specific Parameters` input data).

Finally, the function requester MAY specify that the Application shall be made selectable immediately after its instantiation (the `Make Selectable` input data).

The `InstantiateApplicationCommand` is composed of the following data:

Data name	Description	Type	MOC	No.
`Executable Load File AID`	AID of the Executable Load File that contains the Executable Module to be instantiated. Mandatory for simple Application instantiation; optional for SSD creation.	`AID`	C	1
`Executable Module AID`	AID of the Executable Module to be instantiated. Mandatory for simple Application instantiation; optional for SSD creation.	`AID`	C	1
`Application AID`	AID to be used for the Application instance.	`AID`	O	1
`Parameters Profile Identifier`	Specifies the identifier of the parameters profile to be used. For Application instantiation, a parameters profile defines: • The privileges of the Application • The install parameters: o Application specific installation parameters corresponding to the `'C9'` tag of the Install Parameters field. o System specific parameters corresponding to the `'EF'` tag of the Install Parameters field. o UICC System specific parameters corresponding to the `'EA'` tag of the Install Parameters field (only if the SE is a UICC). If not present, default privileges and install parameters SHALL be used by the function provider.	`Integer`	O	1
`Application Specific Parameters`	Application specific installation parameters corresponding to the `'C9'` tag of the install parameters field (including the `'C9'` tag itself). To be used to force the Application specific parameters. The overall content of the `'C9'` tag shall be provided. It globally supersedes the `'C9'` tag content that may be specified in the selected instantiation profile.	`Byte[]`	O	1
`Generate TAR`	This field requests for the generation of a TAR for the Application.	`Boolean`	O	1
`Make Selectable`	This field specifies whether the Application should be immediately made selectable or not. If `'True'`, the Application SHALL be made selectable. If `'False'`, the Application SHALL NOT be made selectable. If not present, the Application SHALL NOT be made selectable.	`Boolean`	O	1

Table 3-94: Data of the `InstantiateApplicationCommand` SE Command

Note: the `AID` simple type is defined in section 3.1.1.

When the SE Command processing has been performed correctly (`Command Execution Status` is `'Executed-Success'` or `'Executed-WithWarning'`), the `SE Command Result` output data (detailed in section 3.5.1.2) SHALL include the following output parameters.

Else, none of these complementary output data will be returned.

Output data name	**Description**	**Type**	**MOC**	**No.**
`Application AID`	The AID of the instantiated Application.	`AID`	O	1
`Application TAR`	The TAR of the instantiated Application.	`TAR`	O	1
`Parent AID`	The AID of the parent Security Domain under which the Application is instantiated.	`AID`	O	1

Table 3-95: Output parameters for the `InstantiateApplicationCommand` SE Command

Note: the `AID` and `TAR` simple types are defined in section 3.1.1.

3.5.1.3.2.4 Make Selectable an Application

Command name: `MakeSelectableApplicationCommand`

Description:

This SE Command requests for making an Application selectable.

The Application to be made selectable is identified by its AID (the `Application AID` input data).

Using this AID, the function provider MAY for example check that the given AID is allowed for the service, and that the requester is allowed to handle such Application. If not, the function provider SHOULD reject the function call. Such verifications depend on the implementation of the function provider and are out of scope of this specification.

The Application make selectable may require to set Make Selectable Parameters. This is the function provider responsibility to manage such parameters set through parameters profiles (see section 3.5.1.3.2). The function requester MAY then specify which parameters profile shall be used for the Application make selectable (the `Parameters Profile Identifier` input data). If no profile is specified, the function provider SHALL select a default set of parameters.

The `MakeSelectableApplicationCommand` is composed of the following data:

Data name	**Description**	**Type**	**MOC**	**No.**
`Application AID`	AID of the Application to be made selectable.	`AID`	M	1

Data name	Description	Type	MOC	No.
`Parameters Profile Identifier`	Specifies the identifier of the parameters profile to be used. For making selectable an Application, a parameters profile defines the content of the Application privileges and of the tag `'EF'` of the Make Selectable fields. If not present, default parameters values SHALL be used by the function provider.	`Integer`	O	1

Table 3-96: Data of the `MakeSelectableApplicationCommand` SE Command

Note: the `AID` simple type is defined in section 3.1.1.

3.5.1.3.2.5 Registry Update for an Application

Command name: `ApplicationRegistryUpdateCommand`

Description:

This SE Command requests the registry update for an Application.

The Application to which the registry is updated is identified by its AID (the `Application AID` input data).

Using this AID, the function provider MAY for example check that the given AID is allowed for the service, and that the requester is allowed to handle such Application. If not, the function provider SHOULD reject the function call. Such verifications depend on the implementation of the function provider and are out of scope of this specification.

Thanks to Amendment C of the GP 2.2 Card Specifications [5], this SE Command MAY be used to make an Application be available/unavailable on a communication interface, to activate/deactivate an application on the Contactless interface, and to assign Cumulative Granted Memory.

As for the Application Instantiation, the Application registry update requires a profile defining the system specific parameters to be used.

The Application parameters that can be updated are the system-specific parameters (the `Parameters Profile Identifier` input data). If no profile is specified, the function provider SHALL select a default set of parameters.

The `ApplicationRegistryUpdateCommand` is composed of the following data:

Data name	Description	Type	MOC	No.
`Application AID`	AID of the Application which registry is to be updated.	`AID`	M	1

Data name	Description	Type	MOC	No.
`Parameters Profile Identifier`	Specifies the identifier of the parameters profile to be used. For Application registry update, a parameters profile defines the content of the System specific parameters profile corresponding to the `'EF'` tag of the install parameters field.	`Integer`	O	1

Table 3-97: Data of the `ApplicationRegistryUpdateCommand` SE Command

Note: the `AID` simple type is defined in section 3.1.1.

3.5.1.3.2.6 Locking and Unlocking an Application

Command name: `SetStatusCommand`

Description:

This SE Command requests a change of the Application Life Cycle State (either a simple Application or a Supplementary Security Domain).

The Application is identified by its AID (the `Application AID` input data).

Using this AID, the function provider MAY for example check that the given AID is allowed for the service, and that the requester is allowed to handle such Application. It MAY also check that the Application Life Cycle State transition is compliant with SE Application current state. If not, the function provider SHOULD reject the function call. Such verifications depend on the implementation of the function provider and are out of scope of this specification.

The `SetStatusCommand` is composed of the following data:

Data name	Description	Type	MOC	No.
`Application AID`	AID of the Application (either a simple Application or a Supplementary Security Domain).	`AID`	M	1
`Lock Control`	This field specifies the requested state transition. If `'True'`, the application SHALL be locked If `'False'`, the application SHALL be unlocked	`Boolean`	M	1

Table 3-98: Data of the `SetStatusCommand` SE Command

Note: the `AID` simple type is defined in section 3.1.1.

3.5.1.3.2.7 Deleting an Application or Executable Load File

Command name: `DeleteCommand`

Description:

This SE Command requests the deletion of an Application (either a simple Application or a Supplementary Security Domain) or an Executable Load File.

The Application or the Executable Load File to be deleted is identified by its AID (the `Application or Executable Load File AID` input data).

Using this AID, the function provider MAY for example check that the given AID is allowed for the service, and that the requester is allowed to handle such command. If not, the function provider SHOULD reject the function call. In case of deletion of an Executable Load File, the function provider MAY also check that it is not shared with another service. Such verifications depend on the implementation of the function provider and are out of scope of this specification.

Optionally, the objects related to an Executable Load File (i.e. the existing Applications instantiated from this Executable Load File) MAY also be deleted during this action (the `Delete Related Objects` input data).

The `DeleteCommand` is composed of the following data:

Data name	Description	Type	MOC	No.
`Application or Executable Load File AID`	AID of the Application (either a simple Application or a Supplementary Security Domain) or Executable Load File to be deleted.	`AID`	M	1
`Delete Related Objects`	**– Deprecated –** This field is the former way to specify the scope of the deletion. It is now deprecated since the Amendment C of the GP 2.2 Card Specifications [5] has introduced additional behavior. It is now replaced by the `SE Object Deletion Mode` input data. This field is kept in order to ensure the backward compatibility with the previous version of the specification. If present together with the `SE Object Deletion Mode` input data, then the `SE Object Deletion Mode` input data SHALL be considered instead, and the `Delete Related Objects` SHALL be ignored. If present, but the `SE Object Deletion Mode` input data being not set, then the behavior SHALL be the following: • If set to `'True'`, the `'Delete object and related object'` mode mentioned in the `SE`	`Boolean`	O	1

Data name	Description	Type	MOC	No.
	Object Deletion Mode input data SHALL be used. • If set to 'False', the 'Delete object' mode mentioned in the SE Object Deletion Mode input data SHALL be used. If specified with an Application AID in the Application or Executable Load File AID input data, this field SHALL be ignored. If none of the SE Object Deletion Mode and Delete Related Objects is provided, then the deletion mode SHALL be considered as 'Delete object and related object'.			
SE Object Deletion Mode	The deletion mode, according to the Amendment C of the GP 2.2 Card Specifications [5]. The values are: **'Delete object'** **'Delete object and related object'** **'Delete a root Security Domain and all associated Applications'** This field supersedes the former Delete Related Objects input data. It is optional to ensure backward compatibility with the previous version of the specification. As a consequence, in case this field is not provided by the function caller, the Delete Related Objects input data SHALL be taken into account. If none of the SE Object Deletion Mode and Delete Related Objects is provided, then the deletion mode SHALL be considered as 'Delete object and related object'. If specified with an Application AID in the Application or Executable Load File AID input data, this field SHALL be ignored.	Enumeration {Delete Object, Delete Object And Related Objects, Delete Root SD And Associated Applications }	O	1

Table 3-99: Data of the DeleteCommand SE Command

Note: the AID simple type is defined in section 3.1.1.1.

3.5.2 Delegated Management Functions

In Delegated Management mode, the TSM SDM generates the CCM operations and asks the SE Provider SDM to generate relevant Delegated Management Tokens.

This request is made through the functions defined in this section, and that SHOULD be provided by the SE Provider SDM. Unless explicitly mentioned, the "SDM" role used in this section refers to the SE Provider SDM, the TSM SDM being referred as the "function caller" or "function requester".

Tokens and receipts have no clear identifier that would enable to easily match a Receipt among a set of previously generated Tokens.

As a consequence, the following rules SHOULD be respected by the function caller:

- For a given SE, the function requester SHOULD call the `VerifyDMReceipt` function after each single `GenerateDMToken` call. In other words, it SHOULD always verify Receipts before asking for new Tokens.
- The `GenerateDMToken` function caller MAY provide a list of Card Content Management commands and then request for the generation of a series of Tokens in one unique function call.

  When verifying Receipts through the `VerifyDMReceipt` function, the list of provided Card Content Management command statuses SHALL then contain exactly as many items as the corresponding CCM command list of the `GenerateDMToken` function. In other words: All Tokens obtained through a unique call to the `GenerateDMToken` function SHALL be verified through a unique call to the `VerifyDMReceipt` function.

  Specific use case when a Receipt could have been lost is managed by the usage of the CCM Command Status structure detailed in section 3.5.2.2: the `Command Execution Status` data of the `Receipt Data` structure enables to specify that a Card Content Management command:

  - o Has been executed correctly: the Receipt is provided as input data of the `VerifyDMReceipt` function.
  - o Has been executed, but processing of the command failed: no Receipt is then provided by the SE,
  - o Has not been executed because a previous command has failed or a validity period has expired on the system: no Receipt has been generated by the SE,
  - o Has been delivered and executed in the SE, but without any response from the SE (or response has been lost on the network): no Receipt can be provided to the `VerifyDMReceipt` function.

3.5.2.1 The "Token Generation" Function

Name: `GenerateDMToken`

Description:

This function enables to request for the generation of one to several Delegated Management Tokens, in the context of the management of a particular instance of mobile-NFC service (identified by the `Service` and the optional `Service Qualifier` input data) on a particular Secure Element (identified by the `Secure Element` input data).

The Card Content Management commands SHALL be given as input parameters (the `CCM Command` input data), in the order they are intended to be delivered/executed in the Secure Element.

A Token generation implies giving the authorization to the caller to execute this particular CCM command on the SE. So the function provider SHOULD perform some check before delivering the Token. For instance, the command format and the consistency between the CCM command type and the service MAY be checked. `'B6'` tag (Control Reference Template for Digital Signature) MAY also be checked.

This command part can also be updated by the function provider to comply with security rules (for instance, add "Card Image Number" or "Token identifier" in order to insure Token diversification).

If all the Tokens are generated properly, the function returns the CCM commands updated with the generated Tokens (the `Updated CCM Command` output data), as specified in GP 2.2 Card Specifications [1].

If the Tokens generation fails, no command is returned at all.

Function provider: (SE Provider) SDM

Functions group: Delegated Management

Card Content Management Modes: Delegated mode

Additional input data:

Input Data Name	Description	Type	MOC	No.
`Secure Element`	The Secure Element into which the mobile-NFC service is deployed/under deployment.	`SE Identifier`	M	1
`Service`	The mobile-NFC service.	`Service Identifier`	M	1
`Service Qualifier`	The qualifier of the mobile-NFC service.	`Service Qualifier`	C	1

Input Data Name	Description	Type	MOC	No.
`CCM Command`	The Card Content Management command for which the Token has to be generated. A CCM command may be composed to one to several APDUs. Having several occurrences of this field enables to provide several CCM commands. A Token will be generated for each command. If so, the CCM commands must be given in the order they are intended to be delivered/executed in the Secure Element. The `CCM Command` type is described below.	`CCM Command`	M	1..N

Table 3-100: Additional input parameters for `GenerateDMToken` function

Note: the `SE Identifier`, the `Service Identifier` and the `Service Qualifier` types are defined in section 3.1.1.

Where a `CCM Command` is:

Data Name	Description	Type	MOC	No.
`APDU`	The single APDUs of a CCM command. For all Install CCM commands (Install for load, for install, for extradition, for make selectable, for registry update), the Token length SHALL be present in the APDU and set to `'0'` (because this Length is mandatory in the GP 2.2 Card Specification [1]). For the Delete CCM command, the Token TLV SHALL NOT be present (because this TLV is optional in the 2.2 Card Specification [1]). Having several occurrences of this field enables to specify several APDUs in the command.	`Byte[]`	M	1..N

Table 3-101: `CCM Command` Type

Additional output data:

Output Data Name	Description	Type	MOC	No.
`Updated CCM Command`	The CCM command, updated with the generated Token. A CCM command may be composed of one or more APDUs (the number of APDUs of the updated CCM command being potentially higher than the number of APDU of the original one, due to the inclusion of the Token itself).	`CCM Command`	C	1..N

Table 3-102: Additional output parameters for `GenerateDMToken` function

3.5.2.2 The "Receipt Verification" Function

Name: `VerifyDMReceipt`

Description:

This function enables to check the Delegated Management Receipts returned by the Secure Element, in the context of the management of a particular instance of mobile-NFC service (identified by the `Service` and the optional `Service Qualifier` input data) of a particular Secure Element (identified by the `Secure Element` input data).

The function provider SHOULD check the Receipt values and also that each Receipt corresponds to a Token previously requested through the `GenerateDMToken` function.

The list of statuses of the Card Content Management commands (the `CCM Command Status` input data list) SHALL contain exactly as many items as the corresponding CCM command list of the `GenerateDMToken` function (in the `CCM Command` input data list). In other words: all Tokens obtained through a unique call to the `GenerateDMToken` function SHALL be verified through a unique call to the `VerifyDMReceipt` function.

If a CCM Command has been processed correctly by the Secure Element, then the Receipt (the `Receipt` field of the `CCM Command Status` input data) and the confirmation data (the `Confirmation Data` field of the `CCM Command Status` input data) SHALL be provided. The `Command Execution Status` field of the `CCM Command Status` input data SHALL be set to `'Executed-Success'`.

If the function caller is absolutely sure that the corresponding CCM Command has been executed correctly but cannot prove it because neither the Receipt nor the confirmation data can be provided to the function provider, then the `Command Execution Status` field of the `CCM Command Status` input data SHOULD be set to `'Executed-WithWarning'`. Receipt and confirmation data may then not be provided.

- One example of such `'Executed-WithWarning'` usage is in case the action requested to be performed on the SE was indeed already performed (for example, the request for the loading of an ELF that is already present in the SE). Nothing is to be done on the SE, but the final state of the SE is compliant with the initially requested command.

- Another example is if the SE lacks support for receipt generation: the command was successfully executed, but no receipt was returned from the SE.

In both cases above, the function caller SHOULD set the `Command Execution Status` field to `'Executed-WithWarning'`.

If a CCM Command has not been processed correctly (processing failed) by the Secure Element, then the `Command Execution Status` field of the `CCM Command Status` input data SHALL be set to `'Executed-Failed'`. Neither the Receipt nor the confirmation data are provided.

If a CCM Command has been processed but the SE response has been lost (function caller not being able to state whether the execution has been terminated or not, and correctly or not), the `Command Execution Status` field of the `CCM Command Status` input data SHALL be set to `'DeliveredWithNoResponse'`. Neither the Receipt nor the confirmation data are provided.

If a CCM Command has not been processed at all (because a previous command processing has failed, excluding the `'Executed-WithWarning'` use case mentioned above), then the `Command Execution Status` field of the `CCM Command Status` input data SHALL be set to `'NotExecuted'`. Neither the Receipt nor the confirmation data are provided.

Function provider: (SE Provider's) SDM

Functions group: Delegated Management

Card Content Management Modes: Delegated mode

Additional input data:

Input data name	Description	Type	MOC	No.
`Secure Element`	The Secure Element into which the mobile-NFC service is deployed/under deployment.	`SE Identifier`	M	1
`Service`	The mobile-NFC service.	`Service Identifier`	M	1
`Service Qualifier`	The qualifier of the mobile-NFC service.	`Service Qualifier`	C	1
`CCM Command Status`	The status of execution of the Card Content Management commands Having several occurrences of this field enables to specify a list of statuses exactly corresponding to the list of CCM commands previously processed by the `GenerateDMToken` function, i.e. for which Tokens have been generated. The CCM Command Status must be given in the same order as the delivered/execution order in the Secure Element, i.e. the same order as the CCM Command list provided as input data of the `GenerateDMToken` function. The `CCM Command Status` type is described below.	`CCM Command Status`	M	1..N

Table 3-103: Additional input parameters for `VerifyDMReceipt` function

Note: the `SE Identifier`, the `Service Identifier` and the `Service Qualifier` types are defined in section 3.1.1.

Where a `CCM Command Status` is:

Data name	Description	Type	MOC	No.
`Command Execution Status`	Indicates whether the CCM Command has been executed or not. The values are: **'Executed-Success'** **'Executed-WithWarning'** **'Executed-Failed'** **'NotExecuted'** **'DeliveredWithNoResponse'**	`Enumeration {Executed-Success, Executed-WithWarning, Executed-Failed, NotExecuted, DeliveredWithNoResponse}`	M	1
`Receipt`	The receipt to verify. SHALL be present only if `Command Execution Status` is **'Executed-Success'**.	`Byte[]`	C	1
`Confirmation Data`	The confirmation data of the command to verify as defined in GP 2.2 Card Specifications [1]. SHALL be present only if `Command Execution Status` is **'Executed-Success'**.	`Byte[]`	C	1

Table 3-104: `CCM Command Status` type

Note: the `Command Execution Status` definition is the same as the one provided for SE Command execution status and detailed in section 3.5.1.2.

After verification, if all the provided Receipts are correct, the `Status` field of the `Function Execution Status` is simply returned with a value set to `'Executed-Success'`. Otherwise, `'Executed-Failed'` is returned with a relevant `Status Code` explaining the error, and specifying in the `Subject Identifier` of the `Status Code Data` the item number of the Receipt (in the `CCM Command Status` list) for which the verification has failed.

3.5.3 Script Sending Functions

During the overall life of a mobile-NFC service, it may be required to send application data to some of the service components located in the Secure Element: for example, for personalization of the service, or for post-personalization actions such as application data update (banking counter management, transportation ticket reloading; etc.).

In the case this data is generated by a particular actor but is requested to be sent to the SE by another actor, it is required particular functions for the sending of the data to the SE.

As a consequence, the here-below scripts sending functions enable to transport a set of APDUs to an Application or Security Domain of a Secure Element. In the following sub-sections, a set of APDUs is referred to as a script. These APDUs can be anything from Card Content Management commands to completely application specific commands.

The script sending functions assume the separation of the creation of the APDUs from the delivery of the script of the SE:

- To achieve the APDUs creation, some capabilities of the Secure Element and of the OTA bearer to be used shall be know by the entity creating the APDUs. These properties are indeed returned by a call to the Begin Conversation function.
- Then, the scripts are always to be sent to a specific SE Application or Security Domain in the Secure Element. This is then the responsibility of the entity sending the script to use the right bearer and the right secure channel for targeting the final Application, and for security reasons, this is also its responsibility to ensure that the script is executed by the correct Application and not another one.

A set of scripts is called a conversation. Conversations are started by calling the Begin Conversation function and ended by calling the End Conversation function. In a conversation, scripts are sent using the Send Script function, each call to send representing the transfer and execution of one script to the targeted Application/Security Domain.

The following functions are described in this section:

- `BeginConversation` (see section 3.5.3.1)
- `SendScript` (see section 3.5.3.2)
- `EndConversation` (see section 3.5.3.3)

3.5.3.1 The "Begin Conversation" Function

Name: `BeginConversation`

Description:

This function enables the function requester to indicate its intention to perform an OTA conversation with an Application or Security Domain of a Secure Element (the `Secure Element` input data), in the context of the management of a particular instance of mobile-NFC service (identified by the `Service` and the optional `Service Qualifier` input data) through a particular Mobile Subscription (identified by the `Mobile Subscription` input data).

Note that if the identifier of the Mobile Subscription through which the SE is accessible is known by the function caller, then it SHALL provide it (in the `Mobile Subscription` input data). If this information is not known by the function caller, then the function provider is responsible to retrieve the Mobile Subscription identifier based on the Secure Element identifier (by using its own information system or by asking another role for example through the `GetSEMobileSubscriptionIdentifier` function – see section 3.2.4.2).

The function requester SHALL declare with which Application or the Security Domain the conversation shall be done, i.e. which Application or Security Domain will execute the script (the `Target Identifier` input data that may be either an AID or a TAR, depending on the `Target Identifier Type` input data). Based in this information, this is the responsibility of the function provider to:

- Check that the function requester is allowed to access to the targeted Application/Security Domain. If not, the function provider SHOULD reject the function call.

  Such verifications depend on the implementation of the function provider and are out of scope of this specification.

- Determine the appropriate bearer and the secure channel to be used to send the script to the targeted Application. To do so, the function requester SHALL provide requested characteristics (the `Requested Script Format` and `Estimated APDU Script Size` input data).

  However, these parameters are only informative and the function provider MAY configure the conversation in a different manner than what is requested. The function provider SHALL then return the true characteristics of the OTA channel (the `Used Bearer`, the `APDU Script Size`, `Number of APDUs`, `Script Format`, and `APDU Response Script Size` output data) so that the function requester MAY end the conversation if the returned values deviate from the values specified by the function requester or are not compatible with what the function requester wants to perform OTA.

Note that it is possible, into a single conversation, to target several Applications or Security Domains as soon as they are accessible through the same OTA path (bearer, security, etc.). The `Target Identifier` specified in the `BeginConversation` consequently act as the default application to be targeted, and can be overwritten by the `Target Identifier` field of the `SendScript` function.

The function provider may optionally set a period of validity for the conversation (the `Time To Live` output data): any new `SendScript` requests on this conversation will then be refused after expiration of this period.

Note that this is an implementation choice of the function provider to start the OTA communication during the `BeginConversation` processing, or later at the first `SendScript` call.

Note also that a correct processing of the `BeginConversation` function does not ensure that a following call to `SendScript` will succeed; it merely informs the function provider about the intention of the calling role to have an OTA conversation with a specific Secure Element, and simply returns the properties needed for script generation and remote execution.

Function provider: SDM

Functions group: Script Sending

Service Deployment Modes: mode #2, mode #3

Card Content Management Modes: Simple mode, Delegated mode, Dual mode

Additional input data:

Input data name	Description	Type	MOC	No.
`Secure Element`	The Secure Element into which the mobile-NFC service is deployed/under deployment.	`SE Identifier`	M	1
`Mobile Subscription`	The Mobile Subscription through which the Secure Element is accessible.	`Mobile Subscription Identifier`	O	1
`Service`	The mobile-NFC service.	`Service Identifier`	M	1
`Service Qualifier`	The qualifier of the mobile-NFC service.	`Service Qualifier`	C	1
`Target Identifier`	Specifies the AID or the TAR (for UICC only) of the Application or Security Domain that shall execute the script. The function provider is responsible for selecting or by other means sending the script to the targeted entity. Note that this field represents the default application to be targeted when calling the `SendScript` function, but can be overwritten by the `Target Identifier` field of the `SendScript` function in order to target several Applications or the Security Domains within the same conversation.	`AID or TAR`	M	1

`Requested Script Format`	Indicates the requested script format to be used for the conversation (following ETSI TS 102 226 [14]). The values are:
	`'Compact'`
	`'Expanded'`
	
	Note that the script format influences the number of APDU responses that are returned by the Secure Element when executing scripts (see `SendScript` function).
		`Enumeration {Compact, Expanded}`	M	1

`Estimated APDU Script Size`	This is an indicative value of the rough size of the total amount of bytes to be sent during this conversation.	`Integer`	M	1

Table 3-105: Additional input parameters for `BeginConversation` function

Note: the `AID` and `TAR` simple types are defined in section 3.1.1.1.

Note: the `SE Identifier`, `Mobile Subscription Identifier`, the `Service Identifier` and the `Service Qualifier` types are defined in section 3.1.1.

Additional output data:

Output data name	Description	Type	MOC	No.
`Conversation ID`	ID of the conversation initiated by this call to `BeginConversation`. This ID is to be used in consecutive calls to `SendScript` and `EndConversation`.	`Integer`	M	1
`Used Bearer`	Indicates the OTA bearer that will be used to send the script. The following values are pre-defined, but other values MAY be defined by the function provider: **1 ⇨ SMS to UICC** **2 ⇨ CAT-TP to UICC** **3 ⇨ HTTP to UICC** **11 ⇨ SMS to ESE/SMC** **12 ⇨ HTTP to ESE/SMC**	`Integer`	M	1
`APDU Script Size`	Size of the maximum script that can be transferred over this bearer, i.e. the sum of the size of all APDUs that can be sent by one `SendScript` operation. To compute this value, for some bearers, it is needed to know the number of APDUs to be transferred, since each APDU may add a number of bytes to the size of the script. This has to be taken into account by the function provider when calculating this value. A value of 0 means no limitation.	`Integer`	M	1
`Number of APDUs`	Maximum number of APDUs that can be sent in one script. Only returned in case of Expanded format. A value of 0 means no limitation.	`Integer`	C	1

`Script Format`	Indicates the format that will be used by the function provider of the Send Script function to format the given APDU command script, according to ETSI TS 102 226 [14].
	The values are:
	`'Compact'`
	`'Expanded'`
		`Enumeration {Compact, Expanded}`	M	1

Output data name	Description	Type	MOC	No.
`APDU Response Script Size`	The maximum size of all APDU responses that can be delivered by the SE in a single script execution. This size represents the size of the application response payload, excluding any additional security and/or transport header under function provider responsibility. In other words, for the computation of this data, the function provider SHALL take into account any additional security or transport header of its own responsibility that may be added to form the final message sent by the SE. As the size of the responses directly depends on the content of the script to be sent, the function provider cannot verify that responses will fit into the buffers of the SE: the script execution will fail if the script is not built to match the response buffers constraints. It is consequently required that the function caller builds its scripts so that the sum of the size of all responses generated by the SE will not be greater than this `APDU Response Script Size` value. This field SHALL be provided if the function provider is able to compute/provide this information. Note also that for some commands such as GET STATUS, it is not possible to evaluate the size of the response that will be returned. This `APDU Response Script Size` value is therefore of a more informative nature.	`Integer`	C	1
`Time To Live`	The time to live of the conversation, in seconds. The function provider SHALL NOT accept further Send Script requests on this conversation after this period has elapsed.	`Integer`	O	1

Table 3-106: Additional output parameters for `BeginConversation` function

3.5.3.2 The "Send Script" Function

Name: `SendScript`

Description:

This function enables to request for the sending of a set of APDUs (the `Command APDUs` input data) using an already declared conversation with a Secure Element (the `Conversation ID` input data).

The APDU Script Size and the Number of APDUs values returned at the conversation declaration (as output data of the `BeginConversation` function) SHALL be respected by the function requester, when creating the script. If the function requester is not respecting the return values of the `BeginConversation` function call, the function provider SHALL NOT send any APDU to the Secure Element and SHALL return an exception.

The basic principles of the processing of the `SendScript` function are the following:

- The Command APDU order SHALL be respected, even if it is grouped into a single OTA transport message or session for optimization purpose.
- The processing of the `SendScript` function SHALL stop as soon as an exception is encountered in the processing of a Command APDU.
- The `SendScript` function SHALL provide the Response APDU (the `Response APDU` output data) returned by the Secure Element, in the same order than executed by the SE. The Response APDU ***MAY*** contain either the simple Status Word of the command or a complex application response.

Note that in the Compact mode, only the response of the last executed APDU will be returned by the Secure Element, and SHALL be returned to the function caller. If any, the index of the Command APDU that failed SHALL be returned (in the `Failed APDU Index` output data).

In the Expanded mode, the response of all the executed APDU (all correct responses, plus optionally the failed one) will be returned by the Secure Element, and SHALL be returned to the function caller.

Note that it is possible into a single conversation to target several Applications or Security Domains as soon as they are accessible through the same OTA path (bearer, security, etc.). The `Target Identifier` specified in the `BeginConversation` consequently act as the default application to be targeted, and can be overwritten by the `Target Identifier` field of the `SendScript` function.

Function provider: SDM

Functions group: Script Sending

Service Deployment Modes: mode #2, mode #3

Card Content Management Modes: Simple mode, Delegated mode, Dual mode

Additional input data:

Input data name	**Description**	**Type**	**MOC**	**No.**
`Conversation ID`	ID of a conversation opened by the `BeginConversation` function.	`Integer`	M	1

Input data name	Description	Type	MOC	No.
`Target Identifier`	Specifies the AID or the TAR (for UICC only) of the Application or Security Domain that shall execute the script, in place of the Application or Security Domain specified by the `Target Identifier` field provided in the `BeginConversation`. The function provider is responsible for checking that the OTA path (bearer, security keys, etc.) used for this new application is similar to the one selected at `BeginConversation` time. If this filed is not present, then the Application or Security Domain that shall execute the script is the one specified by the `Target Identifier` field provided in the `BeginConversation`.	`AID or TAR`	O	1
`Command APDU`	A list of Command APDUs (such as commands specified in GlobalPlatform Card Specification [1]) to be sent and executed by the targeted application. If either the number of APDUs is greater than the number returned by the `BeginConversation` function or the concatenated size of all the APDUs is bigger than the script size returned by `BeginConversation`, this function must return an exception indicating this. In this case the SDM must not send or execute any APDU in the script.	`Byte[]`	M	1..N

Table 3-107: Additional input parameters for `SendScript` function

Additional output data:

Output data name	Description	Type	MOC	No.
`Response APDU`	The list of Response APDUs that are returned by the Secure Element, when executing the script. In Expanded mode: • If the execution of all `Command APDU` has succeeded, the `Response APDU` list contains exactly the same number of items than the `Command APDU` list (one response per command). Note that in case the Le byte of the corresponding `Command APDU` is not set, then the `Response APDU` byte array for this `Command APDU` MAY be empty. • If the execution of one `Command APDU` has failed, the list contains first the response of all previous commands (they have been correctly executed), and then the response/status word of the one that failed. No response is provided for non executed `Command APDUs`. In Compact mode: • The `Response APDU` list contains only the	`Byte[]`	C	1..N

Output data name	Description	Type	MOC	No.
	response of the last executed `Command APDU` (executed correctly or not). If the execution of one `Command APDU` has failed, then the `Failed APDU Index` provides the index of the failed APDU in the `Command APDU` list. Whatever the script format used by the function provider to send the APDU script (either Compact or Expanded – following ETSI TS 102 226 [14]), the `Response APDU` SHALL contain the APDU response data first, followed by the APDU status word (without any separator). In case none of the Command APDU have been executed, then the `Response APDU` field SHALL NOT be present.			
`Failed APDU Index`	If one Command APDU execution has failed, this field contains the index (in the input `Command APDU` list) of this APDU. Only provided in Compact mode, when one of the Command APDU has failed. E.g. If the execution of the first APDU of the list has failed, then the `Failed APDU Index` SHALL be set to `'1'`. If the execution of the third APDU of the list has failed, then the `Failed APDU Index` SHALL be set to `'3'`. If the full ADPU execution succeeded, then this field SHALL NOT be present. If none of the Command APDU have been executed, then the `Failed APDU Index` field SHALL NOT be present.	`Integer`	C	1

Table 3-108: Additional output parameters for `SendScript` function

The `Status` field of the `Function Execution Status` output data SHALL be set to `'Executed-Success'` if all Command APDU have been executed successfully.

The `Status` field of the `Function Execution Status` output data SHALL be set to `'Failed'` if one of the Command APDU execution has failed. The `Response APDU` list SHALL provide the result of the executed Command APDU. If none of the Command APDU have been executed, then neither the `Response APDU` nor the `Failed APDU Index` fields SHALL be present.

The `Status` field of the `Function Execution Status` output data SHALL be set to `'Expired'` if the script could not be delivered to the Secure Element, or if the response was not delivered by the SE. The `Response APDU` list SHALL provide the result of the executed Command APDU.

3.5.3.3 The "End Conversation" Function

Name: `EndConversation`

Description:

This function indicates that a conversation (represented by the `Conversation ID` input data) is no longer intended to be used by the function requester.

A call to this function SHALL invalidate the conversation ID, which can no longer be used for sending scripts to the Secure Element.

Function provider: SDM

Functions group: Script Sending

Service Deployment Modes: mode #2, mode #3

Card Content Management Modes: Simple mode, Delegated mode, Dual mode

Additional input data:

Input data name	Description	Type	MOC	No.
`Conversation ID`	ID of a conversation opened by the `BeginConversation` function.	`Integer`	M	1

Table 3-109: Additional input parameters for `EndConversation` function

No additional response data complement the output header defined for functions and detailed in section 3.1.3.

3.5.4 Secure Element Audit

Auditing the status of an Application, a Security Domain, or an Executable Load File might be required in several situations such as for checking the existence of a component inside the Secure Element, for regularly synchronizing a own server-side view of the Secure Element content, or for getting a status on the SE content following an OTA processing which completion status is unknown by the server.

3.5.4.1 The "Get Status of an Application or an Executable Load File" Function

Name: `GetApplicationOrELFStatus`

Description:

This function is used to request information about components present on a Secure Element (identifier by the `Secure Element` input data).

Note that if the identifier of the Mobile Subscription through which the SE is accessible is known by the function caller, then it SHALL provide it (in the `Mobile Subscription` input data). If this information is not known by the function caller, then the function provider is responsible to retrieve the Mobile Subscription identifier based on the Secure Element identifier (by using its own information system or by asking another role for example through the `GetSEMobileSubscriptionIdentifier` function – see section 3.2.4.2).

The command is used to audit one or many components among:

- Applications: the status of the Application is returned
- Security Domains; the status of the Security Domain and all the Applications and Executable Load Files contained in this Security Domain are returned. If the Security Domain contains other Security Domains, their content MAY be recursively returned.
- Executable Load Files; The status of the ELF will be returned

Note that Executable Modules are not audited.

Each component to be audited is identified by an AID (the `Component AID` input data). Several component AID can be specified.

Two different types of audit are defined:

- *Offline audit*; this type of audit returns the view from the function provider Card Management System, without having to go Over The Air. The function provider MAY however return the OTA view of the Applications and Executable Load Files even in this audit type, if it is considers necessary to perform this OTA dialog.
- *Online audit*; this type of audit specifically requests information to be retrieved through an Over The Air dialog with the Secure Element.

Before performing the operation, the function provider SHALL verify that the function caller is authorized to access to the to-be-audited Executable Load Files, Applications or Security Domains. This can mean that the Application, SD or ELF belong to the function caller, or it can mean that the Application, SD or ELF comply with a specific function provider filtering rule. For example the function provider might know that the function caller is subject to deploy banking mobile-NFC services, so it should authorize the audit of the PPSE Application or ELF to this function caller.

The function provider SHOULD also verify the GP card state of the Secure Element (as specified in the GlobalPlatform Card Specification [1]). If the SE state is CARD_LOCKED or TERMINATED, the `Command Execution Status` SHALL be set to `'Executed-Failed'` and the `Subject` and `Reason` fields of the `Command Status Code Data` to 'Secure Element a the wrong GP Card State'.

As output parameters, the function SHALL provide a list of complex structures representing the audit data (the `Component Status` output data). If one of the provided AIDs does not match any component in the Secure Element, or if the function caller is not allowed to query the status of a particular AID, then the global function processing status SHALL be set to `'Executed-WithWarning'` and the `Component Status` corresponding to this AID SHALL simply NOT be returned into the list. All other `Component Status` corresponding to existing and allowed components SHALL be returned.

Function provider: SDM

Functions group: SE Audit

Card Content Management Modes: Simple mode, Delegated mode, Dual mode

Additional input data:

Input data name	Description	Type	MOC	No.
`Secure Element`	The Secure Element that is audited.	`SE Identifier`	M	1
`Mobile Subscription`	The Mobile Subscription through which the Secure Element can be audited.	`Mobile Subscription Identifier`	O	1
`Application or Executable Load File AID`	AID of components to be audited. Only the components which AIDs exactly match the provided AIDs will be considered. In other words, providing a partial AID does not enable to get information on all Applications having this partial AID. Having several occurrences of this field enables to audit several Applications, Security Domains or Executable Load Files.	`String`	M	1..N

`Audit Type`	The type of audit to be performed.
	
	Values are:
	`'Offline'`
	`'Online'`
		`Enumeration {Offline, Online}`	M	1

Table 3-110: Additional input parameters for `GetApplicationOrELFStatus` function

Note: the `SE Identifier` and `Mobile Subscription Identifier` types are defined in section 3.1.1.

Additional output data:

Output data name	Description	Type	MOC	No.
`Component Status`	This complex structure contains the information about an audited component. Having several occurrences of this field enables to specify several components statuses. The `Command Status` type is defined below.	`Component Status`	M	0..N

Table 3-111: Additional output parameters for `GetApplicationOrELFStatus` function

Where a `Component Status` is:

Data name	Description	Type	MOC	No.
`Application or Executable Load File AID`	AID of the audited component.	`String`	M	1

`Component Type`	The type of the component.
	Values are:
	`'Application'`
	`'Security Domain'`
	`'ELF'`
		`Enumeration {`
		`Application,`
		`Security Domain,`
		`ELF}`
			M	1

`Life Cycle State`	Represents the GlobalPlatform life cycle state of the Application, Security Domain or Executable Load File.
	The values are defined according to the `Component Type` value:
	
	• For `'Application'`, values are:
	`'Installed'`
	`'Selectable'`
	`'Application Specific'`
	`'Locked'`
	
	• For `'Security Domain'`, values are:
	`'Installed'`
	`'Selectable'`
	`'Personalized'`
	`'Locked'`
	
	• For `'ELF'`, values are:
		`Enumeration {Loaded,`
		`Installed,`
		`Selectable,`
		`Personalized,`
		`Application Specific,`
		`Locked}`
			M	1..N

Data name	Description	Type	MOC	No.
	'Loaded' The Life Cycle State can be represented by more than one value. Having several occurrences of this field enables to specify several sattes for the component. For example, an Application could be in the 'LOCKED from SELECTABLE' state. So, the audit would return a 'Selectable' and a 'Locked' state. A state can only be provided once in the list.			
Application Specific Bits	This field represents the content of the bits b4 to b7 of the Application life cycle state byte. These bits SHALL be returned as a full byte into which bits b1 to b3 and b8 SHALL be set to '0'. The Application Specific Bits SHALL only be present if Content Type is set to 'Application' and if the Life Cycle State field contains the 'Application Specific' value.	Byte	C	1
Descendant Component	A component contained in the Security Domain. Having several occurrences of this field enables to specify several components within the Security Domain. Only present if Content Type is set to 'Security Domain'.	Command Status	C	0..N

Table 3-112: `Command Status type`

3.5.4.2 The "Get SD Free Memory" Function

Name: `GetSDFreeMemory`

Description:

This function is used to get information on the amount of non-volatile free memory available in a Security Domain on a Secure Element (identifier by the `Secure Element` input data).

In a first usage, the function provider SHALL return a memory size (the `Free Memory` output data). The function provider MAY respond either with the actual amount of available memory, or with the amount of memory which the function caller is allowed to use, or which the function provider is ready to allocate to the function caller.

In a second usage of this function, the function caller SHALL provide a requested amount of memory (the `Requested Memory` input data), and the function provider SHALL simply return a `'Yes'` or `'No'` response, depending on whether there is at least such available amount or not (the `Memory Available` output data).

Note that if the identifier of the Mobile Subscription through which the SE is accessible is known by the function caller, then it SHALL provide it (in the `Mobile Subscription` input data). If this information is not known by the function caller, then the function provider is responsible to retrieve the Mobile Subscription identifier based on the Secure Element identifier (by using its own information system or by asking another role for example through the `GetSEMobileSubscriptionIdentifier` function – see section 3.2.4.2).

If the `Requested Memory` input data is set, then the function provider SHALL only return the `Memory Available` output data. The `Free Memory` SHALL NOT be returned.

Two different types of audit are defined:

- *Online audit*: this type of audit specifically requests information to be retrieved through an Over The Air dialog with the Secure Element.
- *Offline audit*: this type of audit returns the view from the function provider Card Management System, without having to go Over The Air. The function provider MAY however return the OTA view of the available memory on the SE even in this audit type, if the function provider considers that it is necessary to perform this OTA dialog.

Depending on its privileges, different SDs of the Secure Element may be accessible to the function caller. For example, the function caller may have an agreement with the function provider that it is allowed to get information about the ISD. But it may also be agreed that the function caller only has access to one or more sub-SDs. Before performing the operation, the function provider SHALL verify that the function caller is authorized to access the memory of the requested Security Domain. If authorization is not granted, the function fails and the appropriate status and reason codes of the output header shall be set by the function provider.

This function can be used to query about the amount of free memory available in a SD which already exists in the Secure Element, or to query for the amount of free memory available for creating a new SD which does not yet exist in the SE. If the SD already exists, the SD AID SHALL be provided as input data by the function caller (the `Security Domain AID` input data). The output data then refers to the memory available in this particular SD.

If the SD does not exist yet, the function caller has two alternatives for defining the location in the SD hierarchy for which the amount of available memory is queried. The function caller can either provide the AID of the parent SD under which the function caller wants to create the new SD (still in the `Security Domain` input data). Alternatively, the function caller can provide the identifier of the mobile-NFC service which the function caller wishes to install (identifier by the `Service` and the optional `Service Qualifier` input data). If the service identifier is given as input data, it is the function provider responsibility to map this identifier to a parent SD under which the new SD shall be created. It is assumed that this mapping is defined in a business agreement between the function caller and the function provider, and it is outside the scope of this specification.

Either the SD identifier or the Service identifier SHALL be provided. It is not allowed to provide both pieces of information at the same time.

Note that this function does not give the function caller any possibility to get information about the volatile memory. Information about the volatile memory could be obtained through the GetSECapabilityProfileId function.

Function provider: SDM

Functions group: SE Audit

Card Content Management Modes: Simple mode, Delegated mode, Dual mode

Additional input data:

Input data name	**Description**	**Type**	**MOC**	**No.**
`Secure Element`	The Secure Element that is audited.	`SE Identifier`	M	1
`Mobile Subscription`	The Mobile Subscription through which the Secure Element can be audited.	`Mobile Subscription Identifier`	O	1
`Security Domain AID`	AID of the Security Domain to which the request refers to. Providing the `Security Domain` is REQUIRED if the `Service` is not provided. Providing both the `Security Domain` ad the `Service` is not allowed.	`AID`	C	1
`Service`	The mobile-NFC service. Providing the `Service` is REQUIRED if the `Security Domain AID` is not provided. Providing both the `Security Domain` ad the `Service` is not allowed.	`Service Identifier`	C	1
`Service Qualifier`	The qualifier of the mobile-NFC service. If Service is provided, then providing the `Service Qualifier` is optional. If `Service` is not provided, then `Service Qualifier` SHALL NOT be sent.	`Service Qualifier`	C	1

`Audit Type`	The type of audit to be performed.
	
	Values are:
	`'Offline'`
	`'Online'`
		`Enumeration {Offline, Online}`	M	1

`Requested Memory`	The amount of free memory, in bytes, that is requested. This is a non negative integer.	`Integer`	C	1

Table 3-113: Additional input parameters for `GetSDFreeMemory` function

Note: the `SE Identifier`, the `Mobile Subscription Identifier`, the `Service Identifier`, and the `Service Qualifier` types are defined in section 3.1.1.

Note: the `AID` simple type is defined in section 3.1.1.1.

Additional output data:

Output data name	Description	Type	MOC	No.
`Free Memory`	The available free memory, in bytes, on the SE relative to the requested SD or the service. Only available if the `Requested Memory` input data is not present.	`Integer`	C	1
`Memory Available`	Specifies if the `Requested Memory` amount of memory is available in the SD or not. Only available if the `Requested Memory` input data is present. Depending on implementation specific behavior of memory management in the SE, there is a risk that even if this output parameter is set to `'True'`, the function caller MAY NOT be able to install a service if its size is on the limit to fit into the requested amount of memory.	`Boolean`	C	1

Table 3-114: Additional output parameters for `GetSDFreeMemory` function

3.5.5 SCWS Service Portal Functions

The global delivery of a mobile-NFC service may include the delivery of User Interface applications. If the Secure Element is a UICC, this UI application may be a Web portal deployed in the Smart Card Web Server [17] (SCWS) application hosted by the UICC.

The OTA administration of the SCWS does not require any GlobalPlatform privileges and is not handled by GlobalPlatform commands. However, the SCWS portal can be spread into several area of responsibility, each area being associated to a particular Security Domain in the card.

This is the business choice of the owner of the SCWS (i.e. the one which owns the root hierarchy of the portal) to either keep the full control of the overall SCWS content, or to authorize third party actors to administrate their own service portals.

In case the SDM deploying the mobile-NFC service does not have its own capability to administrate a specific area of the SCWS, it is required to ask the SCWS owner from managing the service portal.

The following functions are described in this section:

- `LoadSCWSServicePortal` (see section 3.5.5.1)
- `DeleteSCWSServicePortal` (see section 3.5.5.2)

3.5.5.1 The "Load SCWS Service Portal" Function

Name: `LoadSCWSServicePortal`

Description:

This function enables to request for the loading of the Smart Card Web Server (SCWS) service portal (identified by the `Service Portal Identifier` input data), in the context of the management of a particular instance of mobile-NFC service (identified by the `Service` and the optional `Service Qualifier` input data) through a particular Mobile Subscription (identified by the `Mobile Subscription` input data).

The Smart Card Web Server application is defined by the OMA SCWS specification [17], and is at this stage only available on the UICC.

Note that if the identifier of the Mobile Subscription through which the SE is accessible is known by the function caller, then it SHALL provide it (in the `Mobile Subscription` input data). If this information is not known by the function caller, then the function provider is responsible to retrieve the Mobile Subscription identifier based on the Secure Element identifier (by using its own information system or by asking another role for example through the `GetSEMobileSubscriptionIdentifier` function – see section 3.2.4.2).

The service portal may be composed of static resources only, or of both static and dynamic resources.

The root URI to which attaching the service portal in the global SCWS hierarchy, as well as the URI to which mapping the dynamic resources SHALL be defined in advance as a business agreement between the function provider and the function requester.

Similarly, if required, the potential update of the SCWS Home Page to make the mobile-NFC service User Interface be easily accessible MAY be determined at business agreement time. These actions SHOULD be performed transparently by the function provider.

Note that the call to the `LoadSCWSServicePortal` function MAY generate an upgrade of the Service Portal if this portal is already present in the UICC.

Function provider: SDM

Functions group: SCWS Management

Card Content Management Modes: Simple mode, Delegated mode, Dual mode

Additional input data:

Input data name	Description	Type	MOC	No.
`Secure Element`	The Secure Element into which the mobile-NFC service is deployed/under deployment.	`SE Identifier`	M	1
`Mobile Subscription`	The Mobile Subscription through which the Secure Element is accessible.	`Mobile Subscription Identifier`	O	1
`Service`	The mobile-NFC service.	`Service Identifier`	M	1
`Service Qualifier`	The qualifier of the mobile-NFC service.	`Service Qualifier`	C	1
`Service Portal Identifier`	Identifier of the SCWS service portal to be loaded. This field is optional as, following business agreements, the function provider may be able to know exactly which SCWS service portal shall be loaded for a given Service Identifier.	`SCWS Portal Identifier`	O	1

Table 3-115: Additional input parameters for `LoadSCWSServicePortal` function

Note: the `SCWS Portal Identifier` simple type is defined in section 3.1.1.1.

Note: the `Mobile Subscription Identifier`, the `Service Identifier`, and the `Service Qualifier` types are defined in section 3.1.1.

No additional response data complement the output header defined for functions and detailed in section 3.1.3.

3.5.5.2 The "Delete SCWS Service Portal" Function

Name: `DeleteSCWSServicePortal`

Description:

This function enables to request for the deletion of the Smart Card Web Server (SCWS) service portal (identified by the `Service Portal Identifier` input data), in the context of the management of a particular instance of mobile-NFC service (identified by the `Service` and the optional `Service Qualifier` input data) through a particular Mobile Subscription (identified by the `Mobile Subscription` input data).

The Smart Card Web Server application is defined by the OMA SCWS specification [17], and is at this stage only available on the UICC.

Note that if the identifier of the Mobile Subscription through which the SE is accessible is known by the function caller, then it SHALL provide it (in the `Mobile Subscription` input data). If this information is not known by the function caller, then the function provider is responsible to retrieve the Mobile Subscription identifier based on the Secure Element identifier (by using its own information system or by asking another role for example through the `GetSEMobileSubscriptionIdentifier` function – see section 3.2.4.2).

If required, the potential update of the SCWS Home Page to remove the mobile-NFC service User Interface entry point MAY be determined at business agreement time. These actions SHOULD be performed transparently by the function provider.

Function provider: SDM

Functions group: SCWS Management

Card Content Management Modes: Simple mode, Delegated mode, Dual mode

Additional input data:

Input data name	Description	Type	MOC	No.
`Secure Element`	The Secure Element into which the mobile-NFC service is deployed/under deployment.	`SE Identifier`	M	1
`Mobile Subscription`	The Mobile Subscription through which the Secure Element is accessible.	`Mobile Subscription Identifier`	O	1
`Service`	The mobile-NFC service.	`Service Identifier`	M	1
`Service Qualifier`	The qualifier of the mobile-NFC service.	`Service Qualifier`	C	1

Input data name	Description	Type	MOC	No.
`Service Portal Identifier`	Identifier of the SCWS service portal to be deleted. This field is optional as, following business agreements, the function provider may be able to know exactly which SCWS service portal shall be deleted for a given Service Identifier.	`SCWS Portal Identifier`	O	1

Table 3-116: Additional input parameters for `DeleteSCWSServicePortal` function

Note: the `SCWS Portal Identifier` simple type is defined in section 3.1.1.1.

Note: the `Mobile Subscription Identifier`, the `Service Identifier` and the `Service Qualifier` types are defined in section 3.1.1.

No additional response data complement the output header defined for functions and detailed in section 3.1.3.

3.6 Functions for Device Application Management

The global delivery of a mobile-NFC service may include the delivery of applications in the Device, in complement to the applications hosted by the Secure Element.

Moreover, depending on the technology of the Device, it may be required to explicitly grant the access of a particular Device application to a particular SE application, i.e. to bind each other.

The following functions are described in this section:

- Device application management
  - o `LoadDeviceApplication` (see section 3.6.1)
  - o `DeleteDeviceApplication` (see section 3.6.3)
- Security binding of a Device application to a SE application
  - o `BindDeviceApplicationToSEApplication` (see section 3.6.2)
  - o `UnbindDeviceApplicationToSEApplication` (see section 3.6.4)
- Actions on Device application
  - o `HandleActionDoneOnDeviceApplicationNotification` (see section 3.6.5)

3.6.1 The "Load Device Application" Function

Name: `LoadDeviceApplication`

Description:

This function enables to request for the loading of a Device application (identified by the `Device Application Identifier` input data), in the context of the management of a particular instance of mobile-NFC service (identified by the `Service` and the optional `Service Qualifier` input data) through a particular Mobile Subscription (identified by the `Mobile Subscription` input data).

The behavior of the download process highly depends on the Device technology and on the loading technology. However, this process has a direct impact on the end-user experience (whether the download process is visible by the end-user, or even requires actions to be done by the end-user) and on the accountability of the function provider (is the function provider accountable for the overall process or only of the trigger of the download process).

As a consequence, the function requester MAY request for a *preferred* installation mode (the `Preferred Installation Mode` input data) that matches its expectation, but this is the function provider last choice to determine the exact mode to be used, depending on the Device technology and on the service level agreement it can ensure to the service requester. The following aspects MAY be requested:

- Accountability (the `Accountability` input data): the function provider MAY either take the responsibility of the completion of the application download process, or MAY only be accountable for the trigger of the Device application downloading process.

In case the function provider has taken the responsibility of the full download process, the `LoadDeviceApplication` function response SHALL be sent when the Device Application loading process is completed.

In case the function provider is only responsible for the triggering of the download process, the `LoadDeviceApplication` function response SHALL be sent as soon as the Device has been triggered. A further `HandleActionDoneOnDeviceApplicationNotification` notification (see section 3.6.5) will be received to notify the completion of the loading process.

- Visibility to the end-user (the `Silent` input data): specifies whether the Device application download SHOULD be made visible or invisible for the end-user, on the Device screen. Remember that the support of the silent mode depends on the Device technology and capabilities.

Note that the call to the `LoadDeviceApplication` function MAY generate an upgrade of the Service Portal if this portal is already present in the UICC.

Function provider: DMSR

Functions group: Device Application Management

Additional input data:

Input data name	Description	Type	MOC	No.
`Mobile Subscription`	The Mobile Subscription through which the Device is accessible.	`Mobile Subscription Identifier`	M	1
`Service`	The mobile-NFC service.	`Service Identifier`	M	1
`Service Qualifier`	The qualifier of the mobile-NFC service.	`Service Qualifier`	C	1
`Device Application Identifier`	Identifier of the Device application to be loaded. This field is optional as, following business agreements, the function provider may be able to know exactly which Device application shall be loaded for a given Service Identifier.	`Device Application Identifier`	O	1
`Preferred Installation Mode`	The preferred installation mode. If not present, a default installation mode shall be used. This default installation mode is to be agreed as a business agreement between parties. The `Device Installation Mode` type is described below.	`Device Installation Mode`	O	1

Table 3-117: Additional input parameters for `LoadDeviceApplication` function

Note: the `Device Application Identifier` simple type is defined in section 3.1.1.1.

Note: the `Mobile Subscription Identifier`, the `Service Identifier` and the `Service Qualifier` types are defined in section 3.1.1.

Where an `Device Installation` Mode is:

Data name	Description	Type	MOC	No.
`Accountability`	The accountability that is requested for installing the Device application. Pre-defined accountabilities are: **1 ⇨ Trigger**: the function provider is only responsible for the trigger of the Device application downloading process. It cannot ensure the completion of the overall download process (because, for example, it cannot implement retry mechanisms). **2 ⇨ Full download**: the function provider is responsible for the full download process. It is responsible for both the trigger and for all the potential actions (such as retry mechanisms) to ensure that the download process is completed.	`Integer`	M	1
`Silent`	Indicates whether the installation shall be silent (no interaction with the end-user) or not. Capability to have an installation process that is fully transparent to the end-user depends on the Device and on the download technology.	`Boolean`	M	1

Table 3-118: `Device Installation Mode` type

Additional output data:

Output data name	Description	Type	MOC	No.
`Used Installation Mode`	The installation mode used for the loading of the Device Application. The `Device Installation Mode` type is described above.	`Device Installation Mode`	M	1

Table 3-119: Additional output parameters for `LoadDeviceApplication` function

3.6.2 The "Security Binding of a Device Application to a SE Application" Function

Name: `BindDeviceApplicationToSEApplication`

Description:

Device applications belonging to a mobile-NFC service may need to dialog with the service's SE applications. Depending on the Device technology, some security mechanisms may exist to prevent from a SE application to be accessed by any Device application. This is for example if the GlobalPlatform Secure Element Access Control [10] technology is used: access rules enable restricting Device applications access to Secure Element applications. Such rules can be set through the Access Rule Application Master (ARA-M) application, the Access Rule Application Client (ARA-C) application, or the Access Rule File (ARF) support.

As another example, the Access Control Files (ACF) mechanism defined in the MIDP 2.0 environment enables to specify which MIDlet is able to access to a particular application in the SE.

This function consequently enables to request for the binding of a Device application (identified by the `Device Application Identifier` input data) to a SE application (identified by the `SE Application AID` input data), in the context of the management of a particular instance of mobile-NFC service (identified by the `Service` and the optional `Service Qualifier` input data) through a particular Mobile Subscription (identified by the `Mobile Subscription` input data).

Note that if the identifier of the Mobile Subscription through which the SE is accessible is known by the function caller, then it SHALL provide it (in the `Mobile Subscription` input data). If this information is not known by the function caller, then the function provider is responsible to retrieve the Mobile Subscription identifier based on the Secure Element identifier (by using its own information system or by asking another role for example through the `GetSEMobileSubscriptionIdentifier` function – see section 3.2.4.2).

Function provider: (SE Provider's) SDM

Functions group: Device Application Management

Additional input data:

Input data name	Description	Type	MOC	No.
`Secure Element`	The Secure Element into which the mobile-NFC service is deployed/under deployment.	`SE Identifier`	M	1
`Mobile Subscription`	The Mobile Subscription through which the Secure Element is accessible.	`Mobile Subscription Identifier`	O	1
`Service`	The mobile-NFC service.	`Service Identifier`	M	1
`Service Qualifier`	The qualifier of the mobile-NFC service.	`Service Qualifier`	C	1
`Device Application Identifier`	Identifier of the Device application to be bound. This field is optional as, following business agreements, the function provider may be able to know exactly which Device application shall be bound for a given Service Identifier.	`Device Application Identifier`	O	1
`SE Application AID`	AID of the SE application to be bound. This field is optional as, following business agreements, the function provider may be able to know exactly which SE application shall be bound for a given Service Identifier.	`AID`	O	1

Table 3-120: Additional input parameters for `BindDeviceApplicationToSEApplication` function

Note: the `AID` and `Device Application Identifier` simple types are defined in section 3.1.1.1.

Note: the `SE Identifier`, the `Mobile Subscription Identifier`, the `Service Identifier` and the `Service Qualifier` types are defined in section 3.1.1.

No additional response data complement the output header defined for functions and detailed in section 3.1.3.

3.6.3 The "Delete Device Application" Function

Name: `DeleteDeviceApplication`

Description:

This function enables to request for the deletion of a Device application (identified by the `Device Application Identifier` input data), in the context of the management of a particular instance of mobile-NFC service (identified by the `Service` and the optional `Service Qualifier` input data) through a particular Mobile Subscription (identified by the `Mobile Subscription` input data).

The capability to remotely delete a Device application and the behavior of the deletion process highly depends on the Device technology. However, this process has a direct impact on the end-user experience (whether the deletion process is visible by the end-user, or even requires actions to be done by the end-user) and on the accountability of the function provider (is the function provider accountable for the overall process or only of the trigger of the deletion process).

As a consequence, the function requester MAY request for a *preferred* deletion mode (the `Preferred Deletion Mode` input data) that matches its expectation, but this is the function provider last choice to determine the exact mode to be used, depending on the Device technology and on the service level agreement it can ensure to the service requester. The following aspects MAY be requested:

- Accountability (the `Accountability` input data): the function provider MAY either have the full responsibility of the completion of the application deletion process, or MAY only be accountable for the trigger of the Device application deletion process.

  In case the function provider has taken the responsibility of the full deletion process, the `DeleteDeviceApplication` function response SHALL be sent when the Device Application deletion process is completed.

  In case the function provider is only responsible for the triggering of the deletion process, the `DeleteDeviceApplication` function response SHALL be sent as soon as the Device has been triggered. A further `HandleActionDoneOnDeviceApplicationNotification` notification (see section 3.6.5) will be received to notify the completion of the deletion process.
- Visibility to the end-user (the `Silent` input data): specifies whether the Device application deletion SHOULD be made visible or invisible for the end-user, on the Device screen. Remember that the support of the silent mode depends on the Device technology and capabilities.

Note that the function caller has the responsibility to unbind (if previously bound) the Device Application from any SE Application before requesting the Device Application deletion. In case unbinding has not been done before requesting the Device Application deletion, the function provider MAY take the decision to perform the unbinding by itself (if able to do so) together with the Device Application deletion.

Function provider: DMSR

Functions group: Device Application Management

Additional input data:

Input data name	Description	Type	MOC	No.
`Mobile Subscription`	The Mobile Subscription through which the Device is accessible.	`Mobile Subscription Identifier`	M	1
`Service`	The mobile-NFC service.	`Service Identifier`	M	1
`Service Qualifier`	The qualifier of the mobile-NFC service.	`Service Qualifier`	C	1
`Device Application Identifier`	Identifier of the Device application to be deleted. This field is optional as, following business agreements, the function provider may be able to know exactly which Device application shall be deleted for a given Service Identifier.	`Device Application Identifier`	O	1
`Preferred Deletion Mode`	The preferred deletion mode. If not present, a default deletion mode shall be used. This default deletion mode is to be agreed as a business agreement between parties. The `Device Deletion Mode` type is described below.	`Device Deletion Mode`	O	1

Table 3-121: Additional input parameters for `DeleteDeviceApplication` function

Note: the `Device Application Identifier` simple type is defined in section 3.1.1.1.

Note: the `Mobile Subscription Identifier`, the `Service Identifier` and the `Service Qualifier` types are defined in section 3.1.1.

Where a `Device Deletion Mode` is:

Data name	Description	Type	MOC	No.
`Accountability`	The accountability that is requested for deleting the Device application. Pre-defined accountabilities are: **1 ⇨ Trigger**: the function provider is only responsible for the trigger of the Device application deletion automatic process. It cannot ensure the completion of the overall deletion process (because, for example, it cannot implement retry mechanisms). **2 ⇨ Full deletion**: the function provider is responsible for the full deletion process. It is responsible for both the trigger and for all the potential actions (such as retry mechanisms) to ensure that the deletion process is completed. **3 ⇨ Request end-user**: the function provider is only responsible to request the end-user for a manual deletion of the Device application (because no automatic deletion mechanism exists or can be triggered on the Device).	`Integer`	M	1

Data name	Description	Type	MOC	No.
Silent	Indicates whether the deletion shall be silent (no interaction with the end-user) or not. It is ignored in case the Accountability is set to 'Request end-user' (that implies an interaction with the end-user). Capability to have a deletion process that is fully transparent to the end-user depends on the Device and on the deletion technology.	Boolean	M	1

Table 3-122: Device Deletion Mode type

Additional output data:

Output data name	Description	Type	MOC	No.
Used Deletion Mode	The deletion mode used for the deletion of the Device Application. The Device Deletion Mode type is described above.	Device Deletion Mode	M	1

Table 3-123: Additional output parameters for DeleteDeviceApplication function

3.6.4 The "Security Unbinding of a Device Application to a SE Application" Function

Name: UnbindDeviceApplicationToSEApplication

Description:

This function enables to request for the unbinding of a Device application (identified by the Device Application Identifier input data) from a SE application (identified by the SE Application AID input data), in the context of the management of a particular instance of mobile-NFC service (identified by the Service and the optional Service Qualifier input data) through a particular Mobile Subscription (identified by the Mobile Subscription input data).

Note that if the identifier of the Mobile Subscription through which the SE is accessible is known by the function caller, then it SHALL provide it (in the Mobile Subscription input data). If this information is not known by the function caller, then the function provider is responsible to retrieve the Mobile Subscription identifier based on the Secure Element identifier (by using its own information system or by asking another role for example through the GetSEMobileSubscriptionIdentifier function – see section 3.2.4.2).

This function makes the reverse processing as the one done by the BindDeviceApplicationToSEApplication function.

Function provider: (SE Provider's) SDM

Functions group: Device Application Management

Additional input data:

Input data name	Description	Type	MOC	No.
`Secure Element`	The Secure Element into which the mobile-NFC service is deployed/under deployment.	`SE Identifier`	M	1
`Mobile Subscription`	The Mobile Subscription through which the Secure Element is accessible.	`Mobile Subscription Identifier`	O	1
`Service`	The mobile-NFC service.	`Service Identifier`	M	1
`Service Qualifier`	The qualifier of the mobile-NFC service.	`Service Qualifier`	C	1
`Device Application Identifier`	Identifier of the Device application to be unbound. This field is optional as, following business agreements, the function provider may be able to know exactly which Device application shall be unbound for a given Service Identifier.	`Device Application Identifier`	O	1
`SE Application` AID	AID of the SE application to be unbound. This field is optional as, following business agreements, the function provider may be able to know exactly which SE application shall be unbound for a given Service Identifier.	`AID`	O	1

Table 3-124: Additional input parameters for `UnbindDeviceApplicationToSEApplication` function

Note: the `AID` and `Device Application Identifier` simple types are defined in section 3.1.1.1.

Note: the `SE Identifier`, the `Mobile Subscription Identifier`, the `Service Identifier` and the `Service Qualifier` types are defined in section 3.1.1.

No additional response data complement the output header defined for functions and detailed in section 3.1.3.

3.6.5 The "Action Done On Device Application" Notification Function

Name: `HandleActionDoneOnDeviceApplicationNotification`

Description:

This function SHALL be called to notify that an action has been performed on a Device application (identified by the `Device Application Identifier` input data), in the context of the management of a particular instance of mobile-NFC service (identified by the `Service` and the optional `Service Qualifier` input data) through a particular Mobile Subscription (identified by the `Mobile Subscription` input data).

Two possible actions (the `Action` input data) may be notified:

- Device application loading: the loading process of a Device application has been executed.

  Note that this loading process may have been previously initiated:

  - By a DMSR, if the 'Trigger' mode has been used during the `LoadDeviceApplication` function execution,
  - By a SDM, if this SDM has the capability to trigger the Device application loading process by its own,
  - By the end-user itself, through an application store.

- Device application deletion: the deletion process of a Device application has been executed.

  Note that this loading process may have been previously initiated:

  - By a DMSR, if the `'Trigger'` or `'Request end-user'` modes have been used during the `DeleteDeviceApplication` function execution,
  - By a SDM, if this SDM has the capability to trigger the Device application deletion process by its own,
  - By the end-user itself.

The result of execution of the action SHALL be specified in the `Action Execution Status` field. If the action has been processed correctly, the `Action Execution Status` SHALL be `'Executed-Success'`. If an error occurred during the processing of the action, the `Action Execution Status` SHALL be `'Executed-Failed'`.

The reliability of the OTA connectivity is such that it may happen that even if data has been received and processed by the Device, the execution status sent back by the Device over the network may get lost. In such case where the status of execution cannot be determined, the `Action Execution Status` SHALL be `'DeliveredWithNoResponse'`.

The action might have been directly executed by the notification sender, or simply passed through by the notification sender.

The recipient of the notification MAY use this notification to update customer or service states in their internal systems, in order to provide up-to-date information to the customer care or the self-care interface.

No output data are expected to this notification.

Notification handler/recipient: SDM, DMSR

Functions group: Device Application Life Cycle Notification

Additional input data:

Input data name	**Description**	**Type**	**MOC**	**No.**
`Mobile Subscription`	The Mobile Subscription through which the Device is accessible.	`Mobile Subscription Identifier`	M	1

Input data name	Description	Type	MOC	No.
Service	The mobile-NFC service to which the Device application belongs to.	Service Identifier	M	1
Service Qualifier	The qualifier of the mobile-NFC service.	Service Qualifier	C	1
Device Application Identifier	Identifier of the Device application which status has changed. This field is optional as, following business agreements, the function provider may be able to know exactly which Device application is concerned.	Device Application Identifier	O	1
Action	An action that has been executed on the Device application. The following values are pre-defined, but other values MAY be defined by the function provider: **1 ⇨ Load Device application** **2 ⇨ Delete Device application**	Integer	M	1
Action Execution Status	Indicates the execution status of the Action. The values are: **'Executed-Success'** **'Executed-Failed'** **'DeliveredWithNoResponse'**	Enumeration {Executed-Success, Executed-Failed, DeliveredWithNoResponse}	M	1

Table 3-125: Additional input parameters for HandleActionDoneOnDeviceApplicationNotification function

Note: the Device Application Identifier simple type is defined in section 3.1.1.1.

Note: the Mobile Subscription Identifier, the Service Identifier and the Service Qualifier types are defined in section 3.1.1.

4 Message Mapping (Normative)

This section provides the mapping of the functions defined in section 3 into messages to be exchanged between the roles.

Messages defined in this document are compliant with the GP messaging structure defined in section 5 of document [2], using a `GPHeader` and a `GPBody` element.

The messages defined in document [2] that are used in the context of mobile-NFC service management are: `CardCustomization` and `CardAuditTrail`. New messages have been created to match new needs.

Note that any technology can be used to transport those messages (mail, file, Web Services, etc.) as soon as it is agreed between the sender and the receiver.

However, for interoperability purpose, section 5 of this document specifies the particular binding to the Web Service technology, following the OASIS and W3C WS-* standard.

4.1 Namespaces

Namespace naming rules detailed in section 5.1 of document [2] are fully applicable.

In the context of this specification, a new version of the system messaging namespace is used:

- http://namespaces.globalplatform.org/systems-messaging/2.1.0

The XML schema defined in this specification refers to other namespaces, as mentioned in section 4.3.

4.2 Data Types and String Encoding

Data types and string encoding detailed in section 5.2 of document [2] are fully applicable.

The following table presents additional rules and additional data types:

Data	Description
Integer	Integer values SHALL be in the range of a signed 64-bit Integer: from -(2^{63}) to 2^{63}-1.
Internationalizable strings	When an element is a language dependent string, it SHOULD have an attribute `xml:lang="xx"` where xx is the language identifier as specified in RFC 4646 [20]. If no xml:lang attribute is present, implementations MUST assume the language to be English as defined by setting the attribute value to "en" (e.g. xml:lang="en").

Data	Description
Date	The date SHALL be expressed as a dateTime in "canonical representation" (refer to [21]). Implementations SHOULD NOT rely on time resolution finer than milliseconds and MUST NOT generate time instants that specify leap seconds.
Versions	The string representation of version follows the "<major>.<minor>.<revision>" format. The major, minor and revision numbers SHALL be treated as separate integers and each number MAY be incremented higher than a single digit. Thus, version "2.4.3" would be a lower version than version "2.13.1", which in turn would be lower than version "12.3.0". Leading zeros (e.g., version "6.01.04") MUST be ignored by recipients and MUST NOT be sent. Generally speaking, the major version number of an entity should be incremented only if the entity has changed so dramatically that an older version implementation would not be able to interoperate with a newer version. The minor version number indicates new capabilities, and MUST be ignored by an entity with a smaller minor version number, but used for informational purposes by the entity with the larger minor version number. The revision version number most commonly contains bug corrections of the relevant "major.minor" version. It MUST be ignored by an entity with the same major version number but a smaller revision version number, but used for informational purposes by the entity with the same major version number but a larger minor version number.

Table 4-1: Additional data types

4.3 XML Schema References

The XML schema defined in this specification refers to the following namespaces:

- ds: W3C XML-Signature Syntax and Processing, W3C Recommendation
- gpm: GlobalPlatform Messaging Specification, v 2.1.0
- gpp: GlobalPlatform Systems Profiles Specification, v 1.1.1
- xsd: Extensible Markup Language (XML) 1.0, W3C Recommendation

4.4 Message Header

The `GPHeader` structure defined in section 6 of document [2] is fully applicable.

The mapping between function defined in section 3 and the `GPHeader` attributes and elements is the following:

GPHeader attribute	Mapping
`TransactionID`	The `TransactionID` attribute has no particular mapping with any function input data specified in section 3. However note that the `TransactionID` is used to correlate a request message with a response message. As a consequence: • It SHALL be generated by the function requester, • It SHOULD be unique, within a dialog between a message sender and a message receiver. This specification does not describe the behavior of the function provider in case of non uniqueness of the `TransactionID`.
`Source`	The `Source` attribute is mapped to the Function Requester Identifier data of the function input header (See section 3.1.2). Identification of the function requester SHALL be the OID of the function requester.
`Destination`	The `Destination` attribute has no particular mapping with any function input data specified in section 3. However note that the `Destination` attribute SHOULD be filled with the OID of the function provider.
`Type`	New values of the Type attribute enable to represent the various messages corresponding to the functions described in section 3: • `CheckGlobalEligibilityRequest` • `CheckGlobalEligibilityResponse` • `GetDeviceCapabilityProfileIdRequest` • `GetDeviceCapabilityProfileIdResponse` • `GetSECapabilityProfileIdRequest` • `GetSECapabilityProfileIdResponse` • `CheckMobileSubscriptionEligibilityRequest` • `CheckMobileSubscriptionEligibilityResponse` • `HandleStartServiceStateChangeNotificationRequest` • `HandleEndServiceStateChangeNotificationRequest` • `GetMobileSubscriptionAlternateIdentifierRequest` • `GetMobileSubscriptionAlternateIdentifierResponse` • `GetSEMobileSubscriptionIdentifierRequest`

GPHeader attribute	Mapping
	• `GetSEMobileSubscriptionIdentifierResponse` • `GetMobileSubscriptionSEIdentifiersRequest` • `GetMobileSubscriptionSEIdentifiersResponse` • `HandleMobileSubscriptionIdentifierChangedNotificationRequest` • `HandleMobileSubscriptionStatusChangeNotificationRequest` • `HandleSERenewalNotificationRequest` • `HandleSEDeviceChangedNotificationRequest` • `HandleSEDeviceStatusChangeNotificationRequest` • `HandleSEMobileSubscriptionChangedNotificationRequest` • `HandleSEStatusChangeNotificationRequest` • `LookupServiceInstanceReferenceRequest` • `LookupServiceInstanceReferenceResponse` • `DeclareServiceInstanceReferenceRequest` • `DeclareServiceInstanceReferenceResponse` • `GetServiceInstanceReferenceDescriptorRequest` • `GetServiceInstanceReferenceDescriptorResponse` • `GetServiceStateRequest` • `GetServiceStateResponse` • `DeployServiceRequest` • `DeployServiceResponse` • `UpgradeServiceRequest` • `UpgradeServiceResponse` • `ExchangeServiceDataRequest` • `ExchangeServiceDataResponse` • `SuspendOrResumeServiceRequest` • `SuspendOrResumeServiceResponse` • `TerminateServiceRequest` • `TerminateServiceResponse` • `HandleServiceEnvironmentChangeNotificationRequest` • `HandleActionDoneOnServiceNotificationRequest` • `EnrollSSDOwnerCertificateRequest` • `GetCAInformationRequest` Note that for backward compatibility, the `GetCAInformationRequest` type SHALL be used (instead of the *`Audit`*`CAInformationRequest` type). • `GetCAInformationResponse` Note that for backward compatibility, the `GetCAInformationResponse` type SHALL be used (instead of the *`Audit`*`CAInformationResponse`

GPHeader attribute	Mapping
	type). • `SECommandsGenerationAndRemoteExecutionRequest` • `SECommandsGenerationAndRemoteExecutionResponse` • `GenerateDMTokenRequest` • `GenerateDMTokenResponse` • `VerifyDMReceiptRequest` • `VerifyDMReceiptResponse` • `BeginConversationRequest` • `BeginConversationResponse` • `SendScriptRequest` • `SendScriptResponse` • `EndConversationRequest` • `EndConversationResponse` • `GetApplicationOrELFStatusRequest` • `GetApplicationOrELFStatusResponse` • `GetSDFreeMemoryRequest` • `GetSDFreeMemoryResponse` • `LoadSCWSServicePortalRequest` • `LoadSCWSServicePortalResponse` • `DeleteSCWSServicePortalRequest` • `DeleteSCWSServicePortalResponse` • `LoadDeviceApplicationRequest` • `LoadDeviceApplicationResponse` • `BindDeviceApplicationToSEApplicationRequest` • `BindDeviceApplicationToSEApplicationResponse` • `LoadDeviceApplicationRequest` • `LoadDeviceApplicationResponse` • `UnbindDeviceApplicationToSEApplicationRequest` • `UnbindDeviceApplicationToSEApplicationResponse` • `HandleActionDoneOnDeviceApplicationNotificationRequest`
`MessageVersion`	"2.1.0"
`ErrataVersion`	"0" or not present

Table 4-2: `GPHeader` attributes mapping

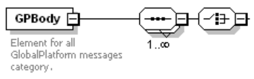
GPHeader elements	Mapping
`Signature`	The `Signature` element has no particular mapping with any function input data specified in section 3. `Signature` element is an optional element. It is up to message sender to provide this information, or up to the message receiver to mandate it. This specification does not describe the behavior of both parties in case the `Signature` is mandated but not provided.

Table 4-3: `GPHeader` elements mapping

4.5 Message Body

The `GPBody` structure follows the one defined in section 7 of document [2].

The `GPBody` is the element that contains the core of the message. In the context of this specification, it MAY be composed of one or several ordered single elements of the following types:



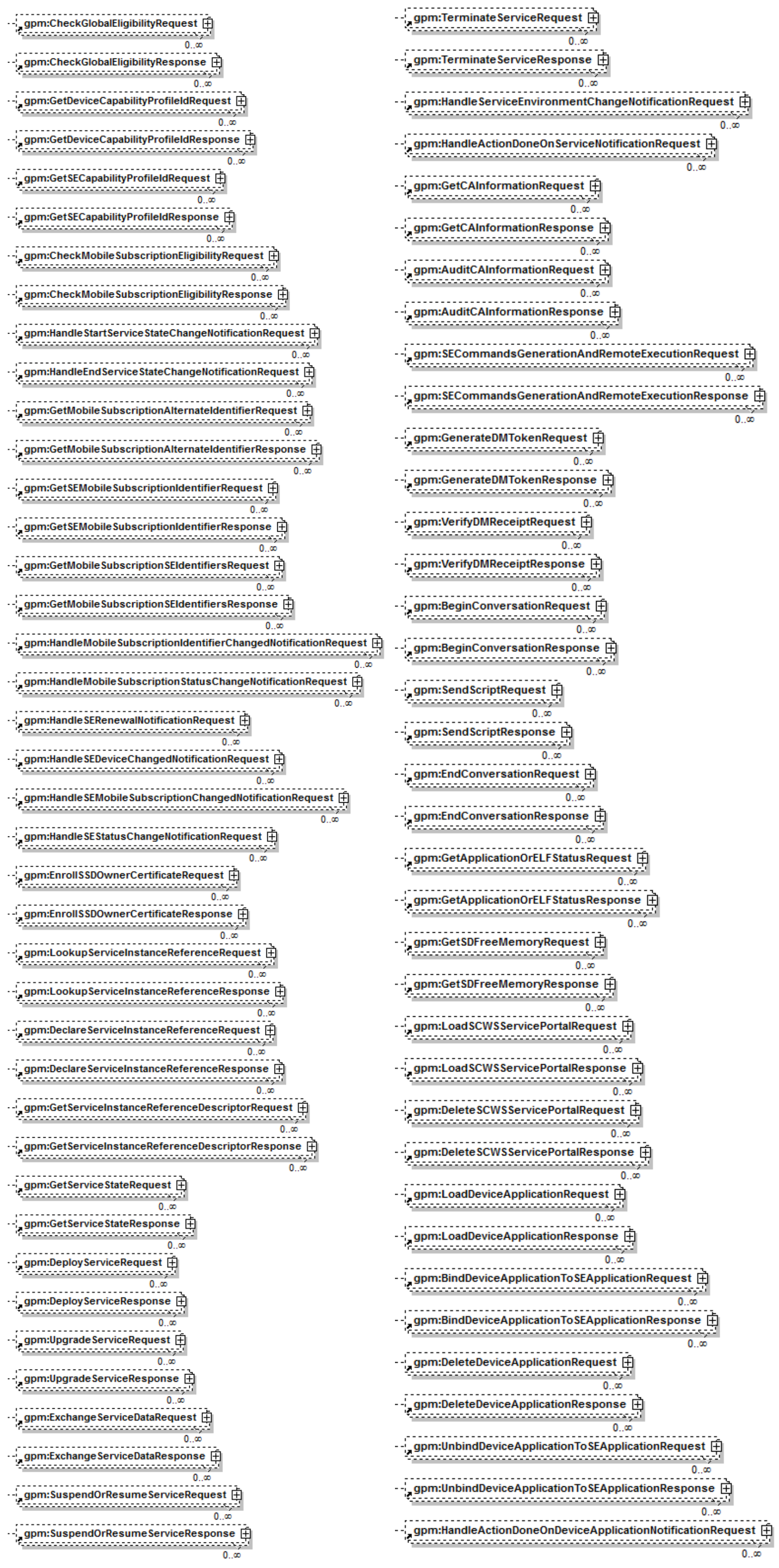

gpm:CheckGlobalEligibilityRequest
gpm:CheckGlobalEligibilityResponse
gpm:GetDeviceCapabilityProfileIdRequest
gpm:GetDeviceCapabilityProfileIdResponse
gpm:GetSECapabilityProfileIdRequest
gpm:GetSECapabilityProfileIdResponse
gpm:CheckMobileSubscriptionEligibilityRequest
gpm:CheckMobileSubscriptionEligibilityResponse
gpm:HandleStartServiceStateChangeNotificationRequest
gpm:HandleEndServiceStateChangeNotificationRequest
gpm:GetMobileSubscriptionAlternateIdentifierRequest
gpm:GetMobileSubscriptionAlternateIdentifierResponse
gpm:GetSEMobileSubscriptionIdentifierRequest
gpm:GetSEMobileSubscriptionIdentifierResponse
gpm:GetMobileSubscriptionSEIdentifiersRequest
gpm:GetMobileSubscriptionSEIdentifiersResponse
gpm:HandleMobileSubscriptionIdentifierChangedNotificationRequest
gpm:HandleMobileSubscriptionStatusChangeNotificationRequest
gpm:HandleSERenewalNotificationRequest
gpm:HandleSEDeviceChangedNotificationRequest
gpm:HandleSEMobileSubscriptionChangedNotificationRequest
gpm:HandleSEStatusChangeNotificationRequest
gpm:EnrollSSDOwnerCertificateRequest
gpm:EnrollSSDOwnerCertificateResponse
gpm:LookupServiceInstanceReferenceRequest
gpm:LookupServiceInstanceReferenceResponse
gpm:DeclareServiceInstanceReferenceRequest
gpm:DeclareServiceInstanceReferenceResponse
gpm:GetServiceInstanceReferenceDescriptorRequest
gpm:GetServiceInstanceReferenceDescriptorResponse
gpm:GetServiceStateRequest
gpm:GetServiceStateResponse
gpm:DeployServiceRequest
gpm:DeployServiceResponse
gpm:UpgradeServiceRequest
gpm:UpgradeServiceResponse
gpm:ExchangeServiceDataRequest
gpm:ExchangeServiceDataResponse
gpm:SuspendOrResumeServiceRequest
gpm:SuspendOrResumeServiceResponse
gpm:TerminateServiceRequest
gpm:TerminateServiceResponse
gpm:HandleServiceEnvironmentChangeNotificationRequest
gpm:HandleActionDoneOnServiceNotificationRequest
gpm:GetCAInformationRequest
gpm:GetCAInformationResponse
gpm:AuditCAInformationRequest
gpm:AuditCAInformationResponse
gpm:SECommandsGenerationAndRemoteExecutionRequest
gpm:SECommandsGenerationAndRemoteExecutionResponse
gpm:GenerateDMTokenRequest
gpm:GenerateDMTokenResponse
gpm:VerifyDMReceiptRequest
gpm:VerifyDMReceiptResponse
gpm:BeginConversationRequest
gpm:BeginConversationResponse
gpm:SendScriptRequest
gpm:SendScriptResponse
gpm:EndConversationRequest
gpm:EndConversationResponse
gpm:GetApplicationOrELFStatusRequest
gpm:GetApplicationOrELFStatusResponse
gpm:GetSDFreeMemoryRequest
gpm:GetSDFreeMemoryResponse
gpm:LoadSCWSServicePortalRequest
gpm:LoadSCWSServicePortalResponse
gpm:DeleteSCWSServicePortalRequest
gpm:DeleteSCWSServicePortalResponse
gpm:LoadDeviceApplicationRequest
gpm:LoadDeviceApplicationResponse
gpm:BindDeviceApplicationToSEApplicationRequest
gpm:BindDeviceApplicationToSEApplicationResponse
gpm:DeleteDeviceApplicationRequest
gpm:DeleteDeviceApplicationResponse
gpm:UnbindDeviceApplicationToSEApplicationRequest
gpm:UnbindDeviceApplicationToSEApplicationResponse
gpm:HandleActionDoneOnDeviceApplicationNotificationRequest
0..∞

Figure 4-1: `GPBody` element

4.5.1 Common Types

4.5.1.1 Extensions Type

The `ExtensionsType` type defines a way to perform schema extension without breaking the XML validation process. Extensions SHALL be defined with explicit namespace.

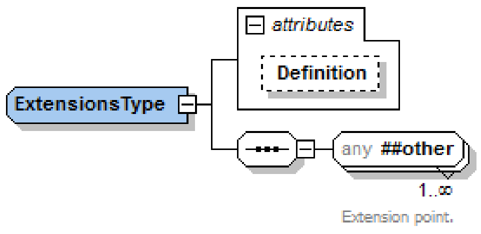


Figure 4-2: `ExtensionsType` type

This element may be included in order to provide extensibility features.

The `NameValuePairType` type also allows a neutral represent of any data (see section 3.1.1.3) based on a "Name:Value pair" representation. Such representation can be used within the extensibility mechanism defined above, but also directly by functions that allows it or a `xsd:anyType` as input or output data.

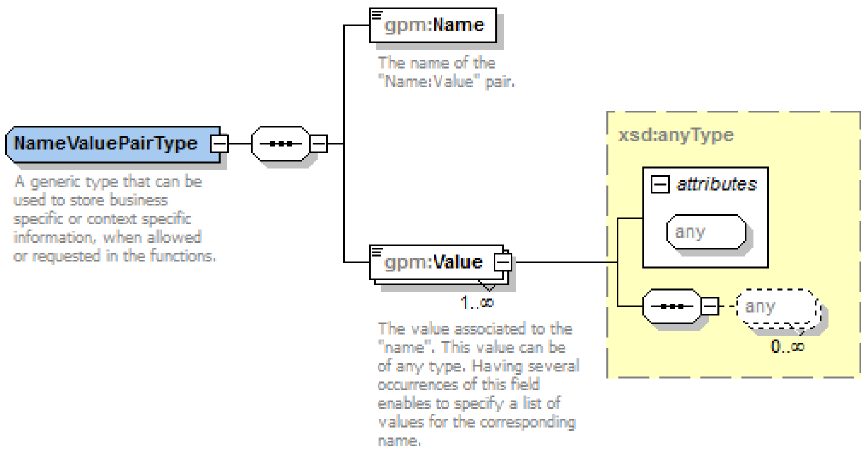

gpm:Name
The name of the "Name:Value" pair.
NameValuePairType
A generic type that can be used to store business specific or context specific information, when allowed or requested in the functions.
xsd:anyType
attributes
any
gpm:Value
1..∞
The value associated to the "name". This value can be of any type. Having several occurrences of this field enables to specify a list of values for the corresponding name.
any
0..∞

Figure 4-3: `NameValuePairType` type

4.5.1.2 Identifiers

Simple types:

The following simple types SHALL be declared. Regular expressions used to force the format of each type SHALL map to the ones defined in section 3.1.1.1.

- `CardUniqueDataType`
- `ICCIDType`

- `MSISDNType`
- `AliasType`
- `AIDType`
- `TARType`
- `ServiceQualifierSimpleType`
- `DeviceApplicationIdentifierType`
- `SCWSPortalIdentifierType`
- `APDUType`
- `APDUResponseType`

SEApplicationIdentifierType:

The `SEApplicationIdentifierType` abstract super type SHALL be defined to identify a SE Application either by an AID or a TAR. The `SEAppId_AIDType` type or the `SEAppId_TARType` SHALL then be used as the concrete SE Application identifier using an AID or the concrete SE Application identifier using a TAR.

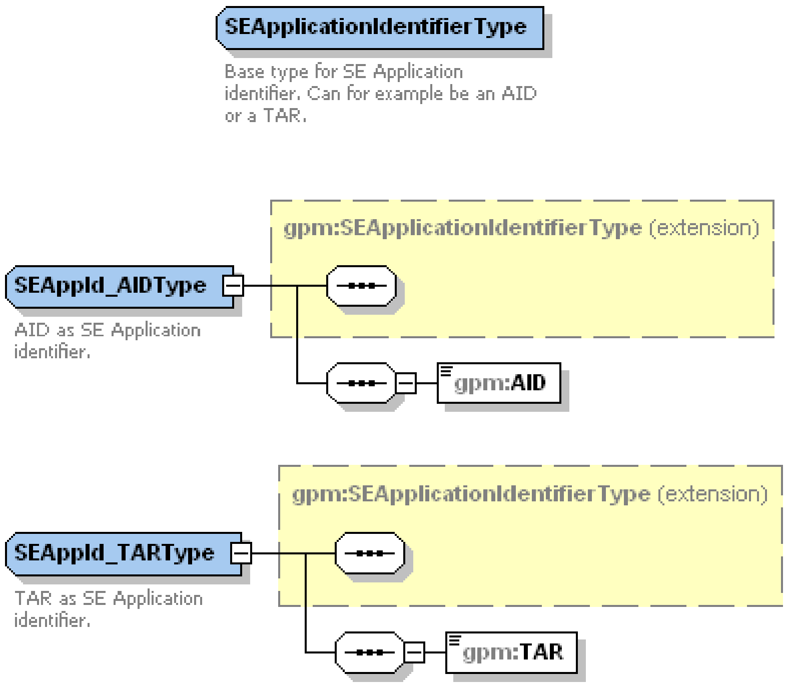

SEApplicationIdentifierType
Base type for SE Application identifier. Can for example be an AID or a TAR.
SEAppId_AIDType
AID as SE Application identifier.
gpm:SEApplicationIdentifierType (extension)
gpm:AID
SEAppId_TARType
TAR as SE Application identifier.
gpm:SEApplicationIdentifierType (extension)
gpm:TAR

Figure 4-4: `SEApplicationIdentifierType`, `SEAppId_AIDType` and `SEAppId_TARType` types

MobileSubscriptionIdentifierType:

The Mobile Subscription Identifier type defined in section 3.1.1 SHALL be mapped to the `MobileSubscriptionIdentifierType` abstract super type described in the following figure:

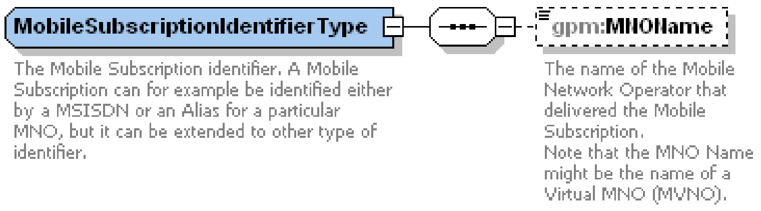

MobileSubscriptionIdentifierType
The Mobile Subscription identifier. A Mobile Subscription can for example be identified either by a MSISDN or an Alias for a particular MNO, but it can be extended to other type of identifier.
gpm:MNOName
The name of the Mobile Network Operator that delivered the Mobile Subscription.
Note that the MNO Name might be the name of a Virtual MNO (MVNO).

Figure 4-5: `MobileSubscriptionIdentifierType` abstract type

The Type and Value data of Table 3-5 are in fact mapped into concrete sub types of `MobileSubscriptionIdentifierType`. Any concrete type of Mobile Subscription identifier SHALL be defined as an extension of the `MobileSubscriptionIdentifierType` abstract type.

When the Mobile Subscription identifier is a MSISDN, the concrete `MSId_MSISDNType` described in the following figure SHALL be used:

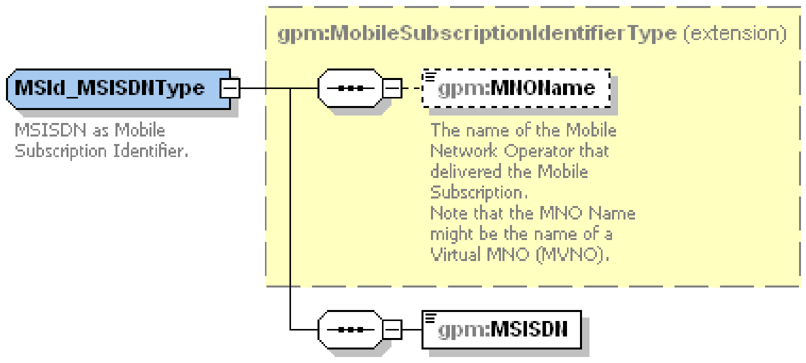

gpm:MobileSubscriptionIdentifierType (extension)
MSId_MSISDNType
MSISDN as Mobile Subscription Identifier.
gpm:MNOName
The name of the Mobile Network Operator that delivered the Mobile Subscription.
Note that the MNO Name might be the name of a Virtual MNO (MVNO).
gpm:MSISDN

Figure 4-6: `MSId_MSISDNType` type

When the Mobile Subscription identifier is an Alias, the concrete `MSId_AliasType` described in the following figure SHALL be used:

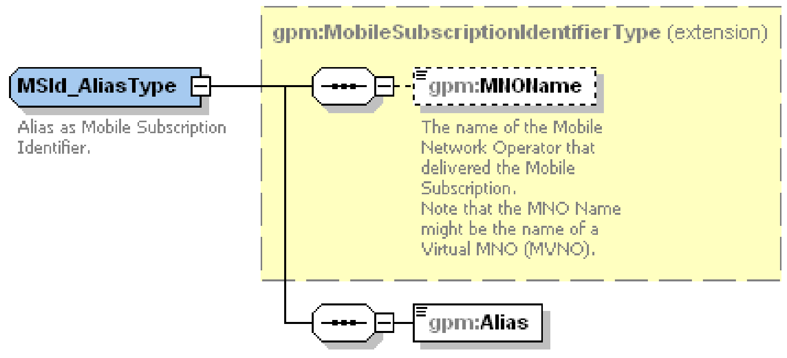

gpm:MobileSubscriptionIdentifierType (extension)
MSId_AliasType
Alias as Mobile Subscription Identifier.
gpm:MNOName
The name of the Mobile Network Operator that delivered the Mobile Subscription. Note that the MNO Name might be the name of a Virtual MNO (MVNO).
gpm:Alias

Figure 4-7: `MSId_AliasType` type

When the Mobile Subscription identifier is defined by a simple string, the concrete `MSId_GenericType` described in the following figure MAY be used:

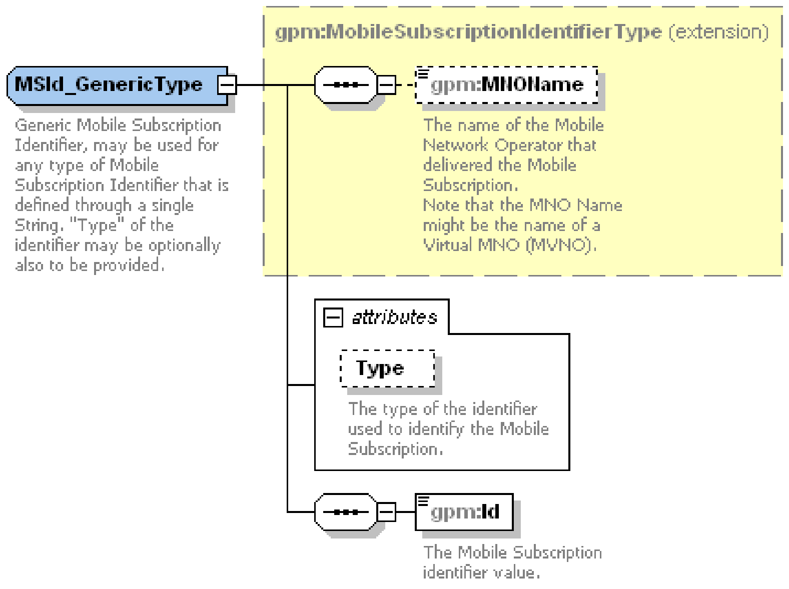

gpm:MobileSubscriptionIdentifierType (extension)
MSId_GenericType
Generic Mobile Subscription Identifier, may be used for any type of Mobile Subscription Identifier that is defined through a single String. "Type" of the identifier may be optionally also to be provided.
gpm:MNOName
The name of the Mobile Network Operator that delivered the Mobile Subscription. Note that the MNO Name might be the name of a Virtual MNO (MVNO).
attributes
Type
The type of the identifier used to identify the Mobile Subscription.
gpm:Id
The Mobile Subscription identifier value.

Figure 4-8: `MSId_GenericType` type

Where the `Type` attribute corresponds to the Type data of Table 3-5.

For all other kinds of Mobile Subscription identifier, the `MSId_ExtendableType` type SHALL be used.

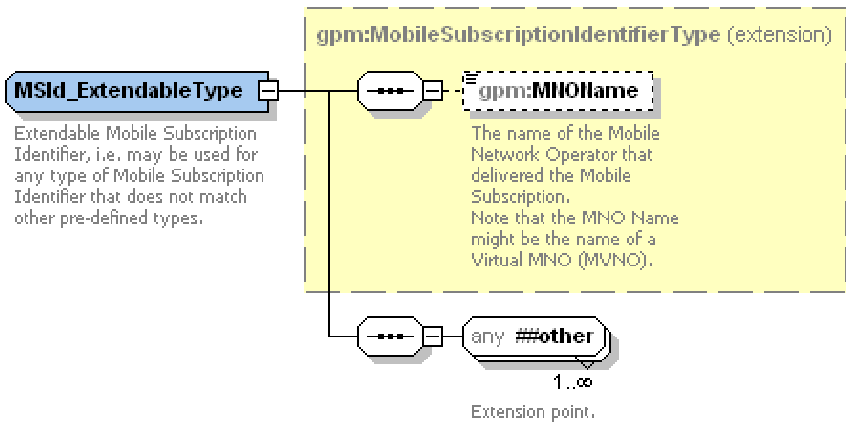

gpm:MobileSubscriptionIdentifierType (extension)
MSId_ExtendableType
Extendable Mobile Subscription Identifier, i.e. may be used for any type of Mobile Subscription Identifier that does not match other pre-defined types.
gpm:MNOName
The name of the Mobile Network Operator that delivered the Mobile Subscription.
Note that the MNO Name might be the name of a Virtual MNO (MVNO).
any ##other
1..∞
Extension point.

Figure 4-9: `MSId_ExtendableType` type

The extensibility mechanism enables to define new elements for defining other kinds of identifiers for the Mobile Subscription.

<u>SEIdentifierType:</u>

The SE Identifier type defined in section 3.1.1 SHALL be mapped to the `SEIdentifierType` abstract super type described in the following figure:

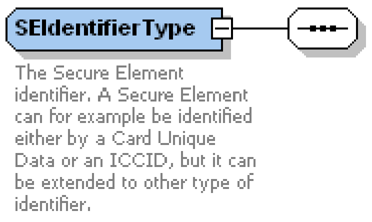

SEIdentifierType
The Secure Element identifier. A Secure Element can for example be identified either by a Card Unique Data or an ICCID, but it can be extended to other type of identifier.

Figure 4-10: `SEIdentifierType` abstract type

The Type and Value data of Table 3-6 are in fact mapped into concrete sub types of `SEIdentifierType`. Any concrete type of Secure Element identifier SHALL be defined as an extension of the `MobileSubscriptionIdentifierType` abstract type.

When the Secure Element identifier is a Card Unique Data, the concrete `SEId_CardUniqueDataType` described in the following figure SHALL be used:

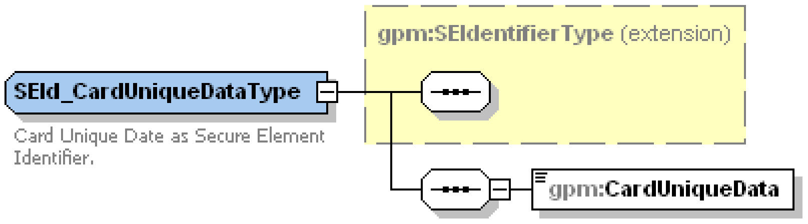

gpm:SEIdentifierType (extension)
SEId_CardUniqueDataType
Card Unique Date as Secure Element Identifier.
gpm:CardUniqueData

Figure 4-11: `SEId_CardUniqueDataType` type

When the Secure Element identifier is a ICCID, the concrete `SEId_ICCIDType` described in the following figure SHALL be used:

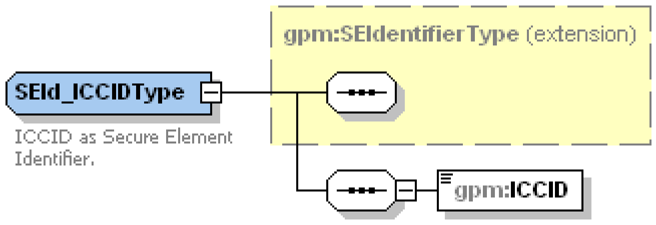

gpm:SEIdentifierType (extension)
SEId_ICCIDType
ICCID as Secure Element Identifier.
gpm:ICCID

Figure 4-12: `SEId_ICCIDType` type

When the Secure Element identifier is defined by a simple string, the concrete `SEId_GenericType` described in the following figure MAY be used:

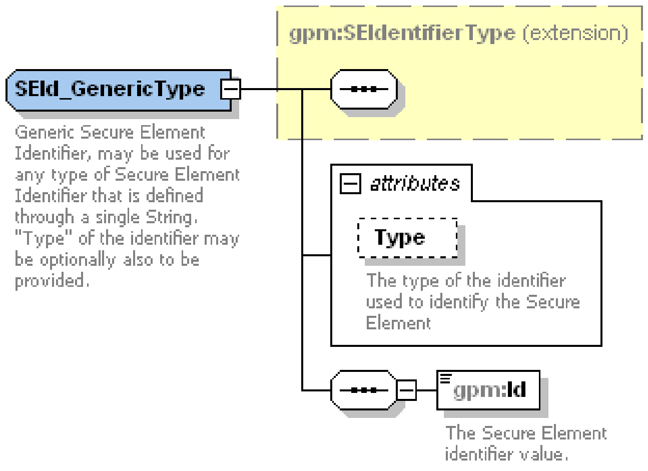

gpm:SEIdentifierType (extension)
SEId_GenericType
Generic Secure Element Identifier, may be used for any type of Secure Element Identifier that is defined through a single String. "Type" of the identifier may be optionally also to be provided.
attributes
Type
The type of the identifier used to identify the Secure Element
gpm:Id
The Secure Element identifier value.

Figure 4-13: `SEId_GenericType` type

Where the Type attribute corresponds to the Type data of Table 3-6.

For all other kinds of Secure Element identifier, the `SEId_ExtendableType` type SHALL be used.

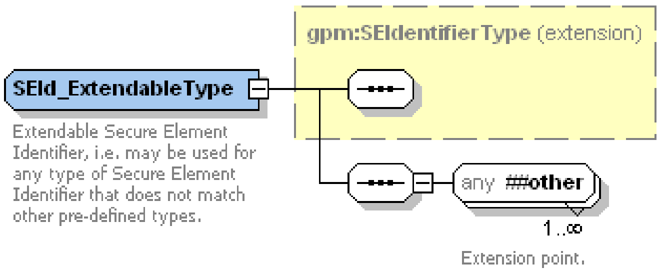

gpm:SEIdentifierType (extension)
SEId_ExtendableType
Extendable Secure Element Identifier, i.e. may be used for any type of Secure Element Identifier that does not match other pre-defined types.
any ##other
1..∞
Extension point.

Figure 4-14: `SEId_ExtendableType` type

The extensibility mechanism enables to define new elements for defining other kinds of identifiers for the Secure Element.

<u>ServiceIdentifierType:</u>

The Service Identifier type defined in section 3.1.1 SHALL be mapped to the `ServiceIdentifierType` type described in the following figure:

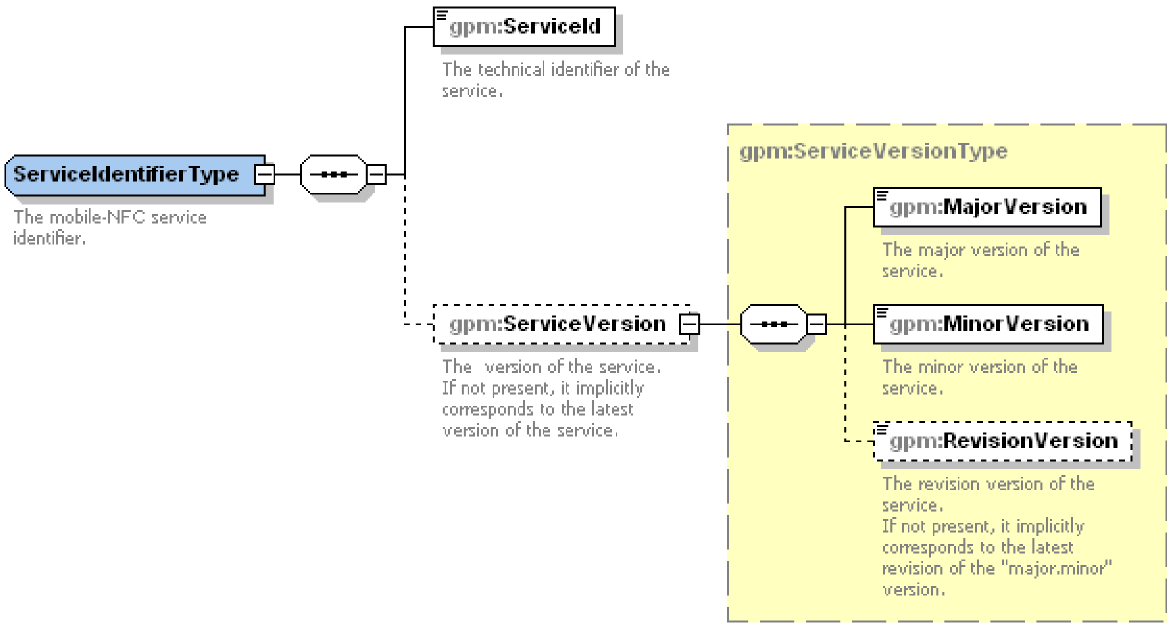

gpm:ServiceId
The technical identifier of the service.
ServiceIdentifierType
The mobile-NFC service identifier.
gpm:ServiceVersionType
gpm:MajorVersion
The major version of the service.
gpm:ServiceVersion
The version of the service. If not present, it implicitly corresponds to the latest version of the service.
gpm:MinorVersion
The minor version of the service.
gpm:RevisionVersion
The revision version of the service. If not present, it implicitly corresponds to the latest revision of the "major.minor" version.

Figure 4-15: `ServiceIdentifierType` type

ServiceQualifierType:

The Service Qualifier simple data defined in section 3.1.1 SHALL be mapped to the `ServiceQualifierType` type described in the following figure:

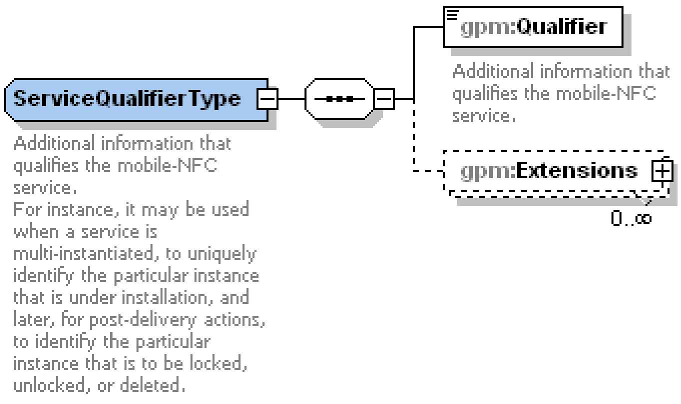

ServiceQualifierType
Additional information that qualifies the mobile-NFC service.
For instance, it may be used when a service is multi-instantiated, to uniquely identify the particular instance that is under installation, and later, for post-delivery actions, to identify the particular instance that is to be locked, unlocked, or deleted.
gpm:Qualifier
Additional information that qualifies the mobile-NFC service.
gpm:Extensions
0..∞

Figure 4-16: `ServiceQualifierType` type

Where the `Extensions` element is of type `ExtensionsType` (see section 4.5.1.1).

The extensibility mechanism enables to define new qualification elements, if required.

ServiceInstanceReferenceType:

The Service Instance Reference simple data defined in section 3.1.1 SHALL be mapped to the `ServiceInstanceReferenceType` type described in the following figure:

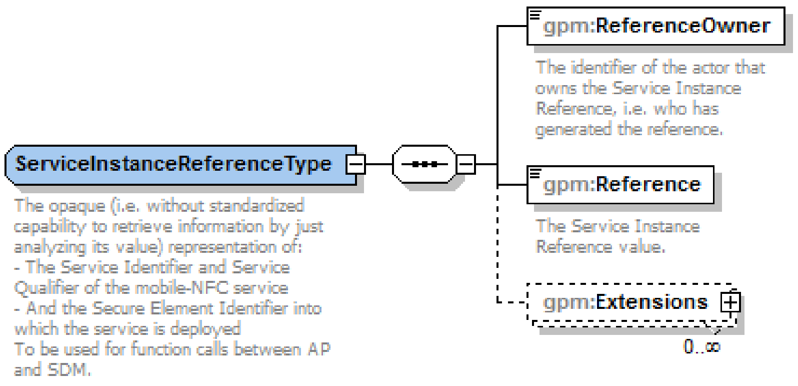

ServiceInstanceReferenceType
The opaque (i.e. without standardized capability to retrieve information by just analyzing its value) representation of:
- The Service Identifier and Service Qualifier of the mobile-NFC service
- And the Secure Element Identifier into which the service is deployed
To be used for function calls between AP and SDM.
gpm:ReferenceOwner
The identifier of the actor that owns the Service Instance Reference, i.e. who has generated the reference.
gpm:Reference
The Service Instance Reference value.
gpm:Extensions
0..∞

Figure 4-17: `ServiceInstanceReferenceType` type

Where:

- The `ReferenceOwner` element is of type `ObjectIdentifierType`.
- The `Extensions` element is of type `ExtensionsType` (see section 4.5.1.1).

The extensibility mechanism enables to define new reference elements, if required.

4.5.1.3 Request Base Type

The input header type defined in section 3.1.2 SHALL be mapped to the `BasicRequestType` type described in the following figure:

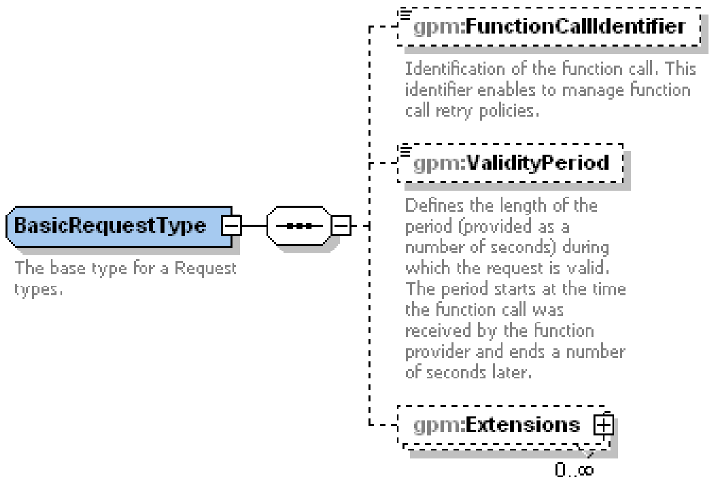

gpm:FunctionCallIdentifier
Identification of the function call. This identifier enables to manage function call retry policies.
gpm:ValidityPeriod
Defines the length of the period (provided as a number of seconds) during which the request is valid. The period starts at the time the function call was received by the function provider and ends a number of seconds later.
BasicRequestType
The base type for a Request types.
gpm:Extensions
0..∞

Figure 4-18: `BasicRequestType` type

Where the `Extensions` element is of type `ExtensionsType` (see section 4.5.1.1).

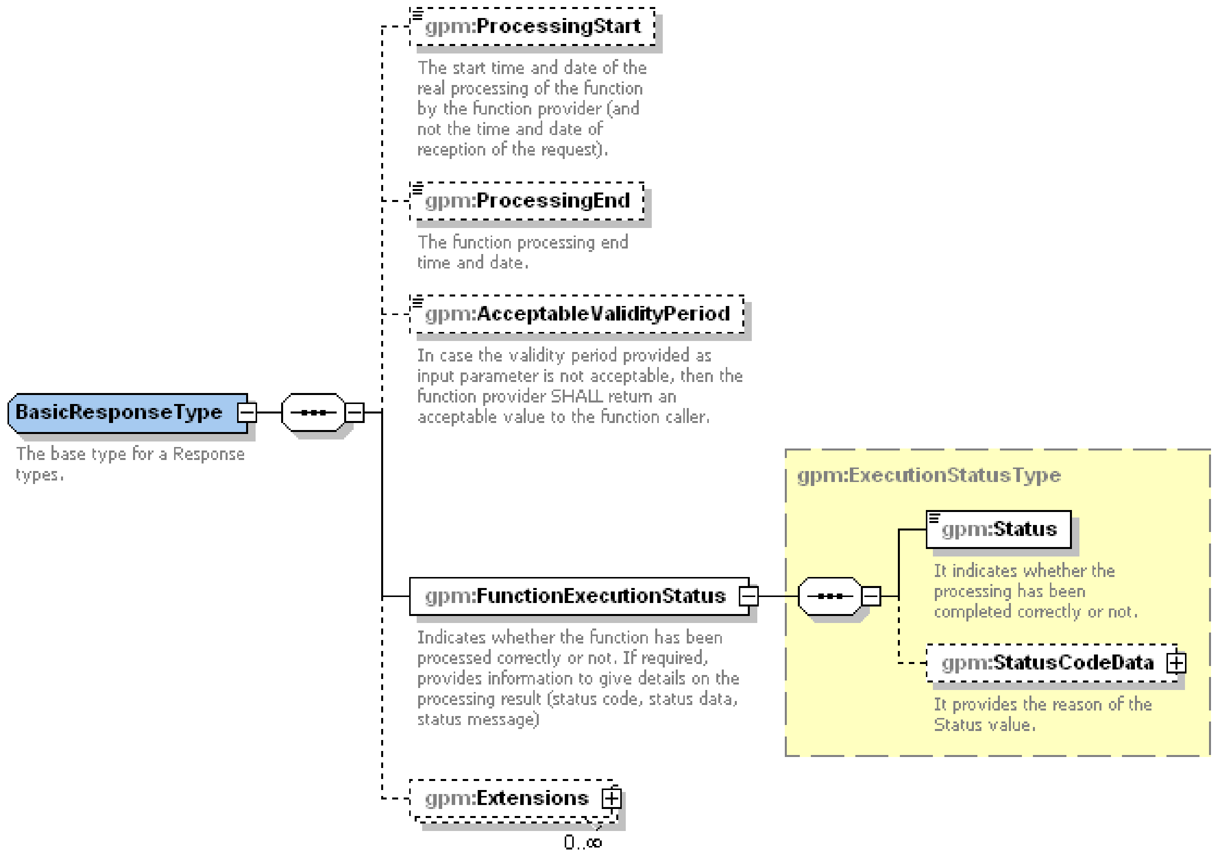
The `BasicRequestType` acts as a base type that each Request SHALL extend.

As the Function Provider data of Table 3-13 is already mapped to the `GPHeader.Sender` element, it is not put in the `BasicRequestType` type.

An extensibility mechanism enables to define new elements common to all Requests, if required.

4.5.1.4 Response Base Type

The output header type defined in section 3.1.2 SHALL be mapped to the `BasicResponseType` type described in the following figure:



gpm:ProcessingStart
The start time and date of the real processing of the function by the function provider (and not the time and date of reception of the request).
gpm:ProcessingEnd
The function processing end time and date.
gpm:AcceptableValidityPeriod
In case the validity period provided as input parameter is not acceptable, then the function provider SHALL return an acceptable value to the function caller.
BasicResponseType
The base type for a Response types.
gpm:ExecutionStatusType
gpm:Status
It indicates whether the processing has been completed correctly or not.
gpm:FunctionExecutionStatus
Indicates whether the function has been processed correctly or not. If required, provides information to give details on the processing result (status code, status data, status message)
gpm:StatusCodeData
It provides the reason of the Status value.
gpm:Extensions
0..∞

Figure 4-19: `BasicResponseType` type

Where:

- The `Extensions` element is of type `ExtensionsType` (see section 4.5.1.1).
- The `Status` element of `FunctionExecutionStatus` is an enumeration of the possible status codes (`FunctionExecutionStatusType`).
- The `StatusCodeData` element is of type `StatusCodeDataType` described below:

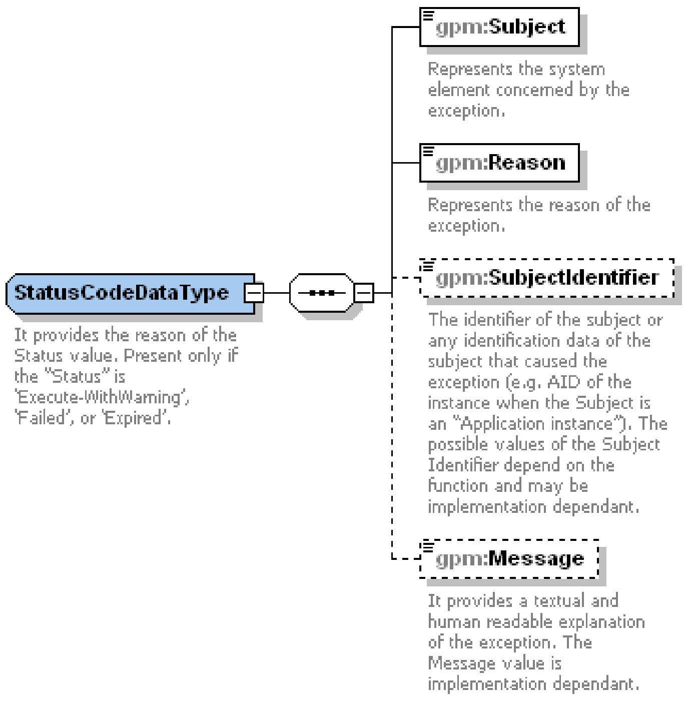

gpm:Subject
Represents the system element concerned by the exception.
gpm:Reason
Represents the reason of the exception.
StatusCodeDataType
It provides the reason of the Status value. Present only if the "Status" is 'Execute-WithWarning', 'Failed', or 'Expired'.
gpm:SubjectIdentifier
The identifier of the subject or any identification data of the subject that caused the exception (e.g. AID of the instance when the Subject is an "Application instance"). The possible values of the Subject Identifier depend on the function and may be implementation dependant.
gpm:Message
It provides a textual and human readable explanation of the exception. The Message value is implementation dependant.

Figure 4-20: `StatusCodeDataType` type

The `BasicResponseType` acts as a base type that each Response SHALL extend.

An extensibility mechanism enables to define new elements common to all Responses, if required.

Note that the `Message` element is a language dependent string. As a consequence, it SHOULD have an attribute `xml:lang="xx"` where `xx` is the language identifier as specified in RFC 4646. If no `xml:lang` attribute is present, implementations MUST assume the language to be English as defined by setting the attribute value to "en" (e.g. `xml:lang="en"`).

4.5.2 Control Functions

4.5.2.1 Eligibility Check

4.5.2.1.1 The "CheckGlobalEligibility" Function

The input data of the `CheckGlobalEligibility` function defined in section 3.2.1.1 SHALL be mapped to the `CheckGlobalEligibilityRequest` element that is of type `CheckGlobalEligibilityRequestType`, described in the following figure:

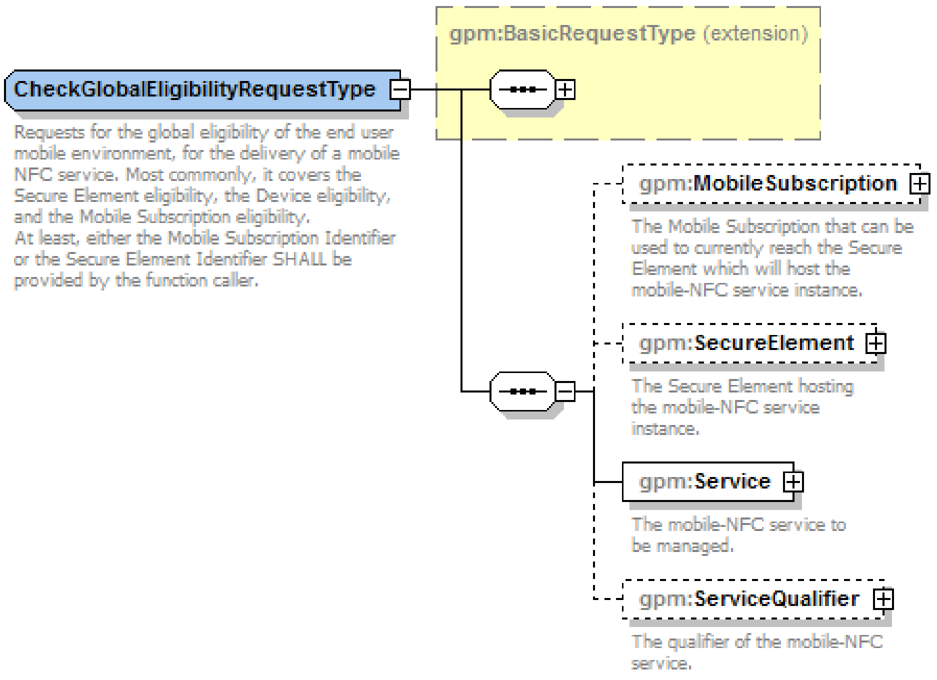

gpm:BasicRequestType (extension)
CheckGlobalEligibilityRequestType
Requests for the global eligibility of the end user mobile environment, for the delivery of a mobile NFC service. Most commonly, it covers the Secure Element eligibility, the Device eligibility, and the Mobile Subscription eligibility.
At least, either the Mobile Subscription Identifier or the Secure Element Identifier SHALL be provided by the function caller.
gpm:MobileSubscription
The Mobile Subscription that can be used to currently reach the Secure Element which will host the mobile-NFC service instance.
gpm:SecureElement
The Secure Element hosting the mobile-NFC service instance.
gpm:Service
The mobile-NFC service to be managed.
gpm:ServiceQualifier
The qualifier of the mobile-NFC service.

Figure 4-21: `CheckGlobalEligibilityRequestType` type

Where:

- The `MobileSubscription` element is of type `MobileSubscriptionIdentifierType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "CheckGlobalEligibilityRequest".

The output data of the `CheckGlobalEligibility` function defined in section 3.2.1.1 SHALL be mapped to the `CheckGlobalEligibilityResponse` element that is of type `CheckGlobalEligibilityResponseType` described in the following figure:

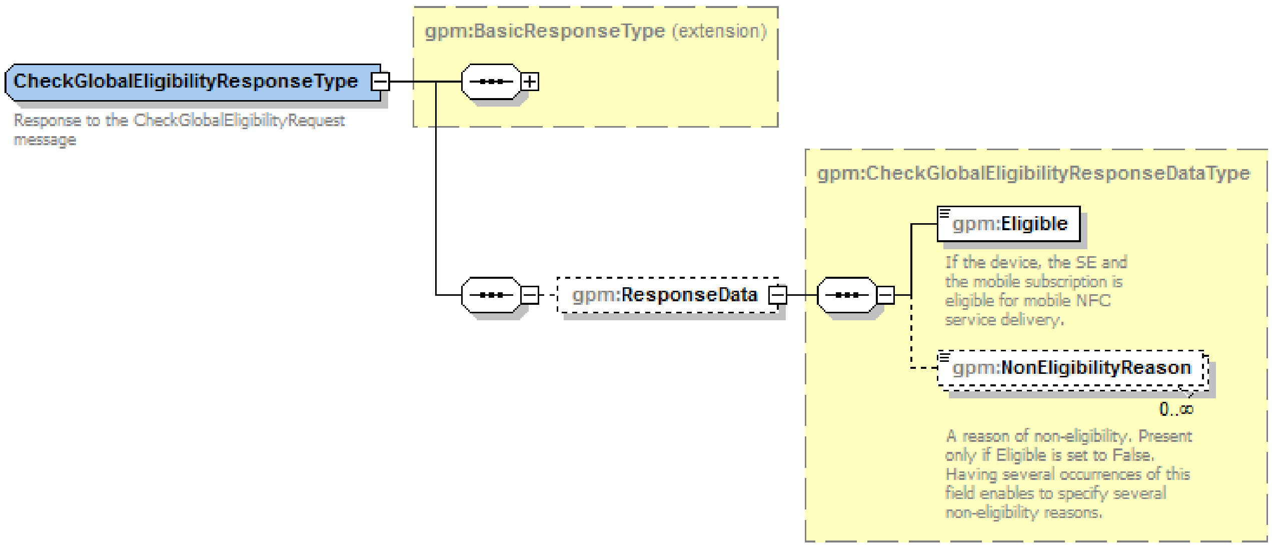

gpm:BasicResponseType (extension)
CheckGlobalEligibilityResponseType
Response to the CheckGlobalEligibilityRequest message
gpm:CheckGlobalEligibilityResponseDataType
gpm:Eligible
If the device, the SE and the mobile subscription is eligible for mobile NFC service delivery.
gpm:ResponseData
gpm:NonEligibilityReason
0..∞
A reason of non-eligibility. Present only if Eligible is set to False. Having several occurrences of this field enables to specify several non-eligibility reasons.

Figure 4-22: `CheckGlobalEligibilityResponseType` type

The value of the `GPHeader.Type` associated to this element SHALL be "CheckGlobalEligibilityResponse".

4.5.2.1.2 The "GetDeviceCapabilityProfileId" Function

The input data of the `GetDeviceCapabilityProfileId` function defined in section 3.2.1.2 SHALL be mapped to the `GetDeviceCapabilityProfileIdRequest` element that is of type `GetDeviceCapabilityProfileIdRequestType`, described in the following figure:

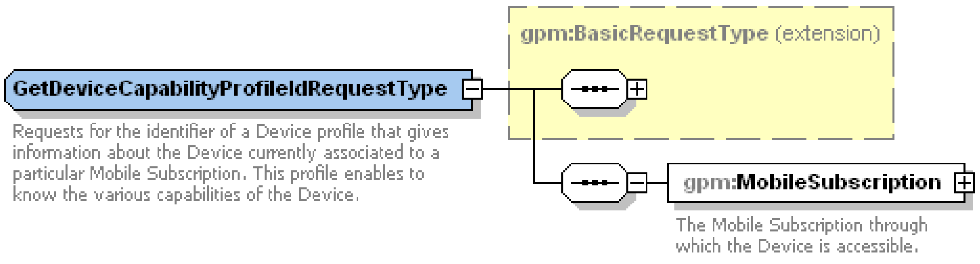

gpm:BasicRequestType (extension)
GetDeviceCapabilityProfileIdRequestType
Requests for the identifier of a Device profile that gives information about the Device currently associated to a particular Mobile Subscription. This profile enables to know the various capabilities of the Device.
gpm:MobileSubscription
The Mobile Subscription through which the Device is accessible.

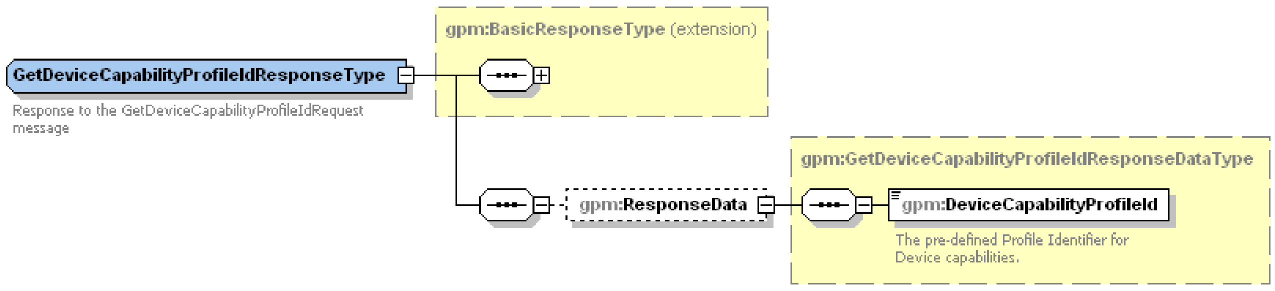
Figure 4-23: `GetDeviceCapabilityProfileIdRequestType` type

Where:

- The `MobileSubscription` element is of type `MobileSubscriptionIdentifierType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "GetDeviceCapabilityProfileIdRequest".

The output data of the `GetDeviceCapabilityProfileId` function defined in section 3.2.1.2 SHALL be mapped to the `GetDeviceCapabilityProfileIdResponse` element that is of type `GetDeviceCapabilityProfileIdResponseType` described in the following figure:



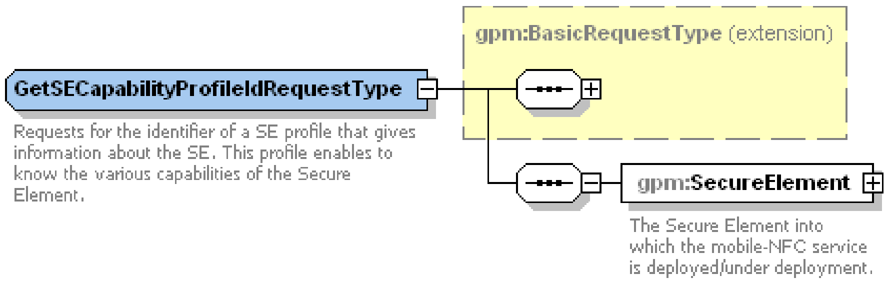
gpm:BasicResponseType (extension)
GetDeviceCapabilityProfileIdResponseType
Response to the GetDeviceCapabilityProfileIdRequest message
gpm:GetDeviceCapabilityProfileIdResponseDataType
gpm:ResponseData
gpm:DeviceCapabilityProfileId
The pre-defined Profile Identifier for Device capabilities.

Figure 4-24: `GetDeviceCapabilityProfileIdResponseType` type

The value of the `GPHeader.Type` associated to this element SHALL be "GetDeviceCapabilityProfileIdResponse".

4.5.2.1.3 The "GetSECapabilityProfileId" Function

The input data for the `GetSECapabilityProfileId` function defined in section 3.2.1.3 SHALL be mapped to the `GetSECapabilityProfileIdRequest` element that is of type `GetSECapabilityProfileIdRequestType` described in the following figure:



gpm:BasicRequestType (extension)
GetSECapabilityProfileIdRequestType
Requests for the identifier of a SE profile that gives information about the SE. This profile enables to know the various capabilities of the Secure Element.
gpm:SecureElement
The Secure Element into which the mobile-NFC service is deployed/under deployment.

Figure 4-25: `GetSECapabilityProfileIdRequestType` type

Where:

- The `SecureElement` element is of type `SEIdentifierType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "GetSECapabilityProfileIdRequest".

The output data of the `GetSECapabilityProfileId` function defined in section 3.2.1.3 SHALL be mapped to the `GetSECapabilityProfileIdResponse` element that is of type `GetSECapabilityProfileIdResponseType` described in the following figure:

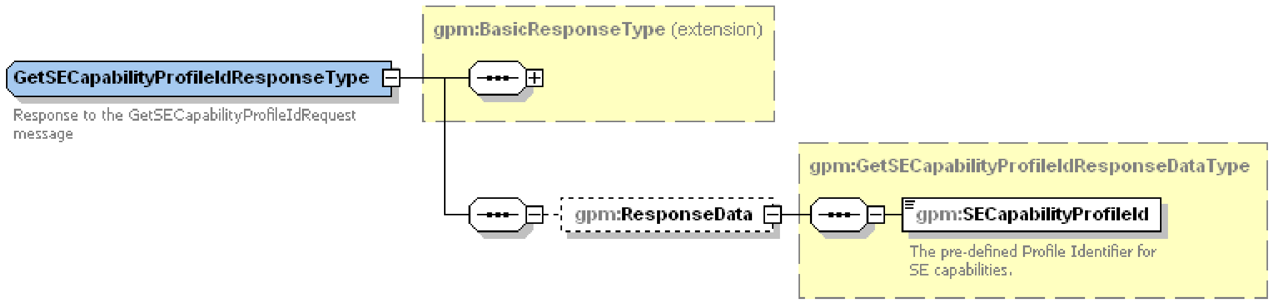

gpm:BasicResponseType (extension)
GetSECapabilityProfileIdResponseType
Response to the GetSECapabilityProfileIdRequest message
gpm:GetSECapabilityProfileIdResponseDataType
gpm:ResponseData
gpm:SECapabilityProfileId
The pre-defined Profile Identifier for SE capabilities.

Figure 4-26: `GetSECapabilityProfileIdResponseType` type

The value of the `GPHeader.Type` associated to this element SHALL be "GetSECapabilityProfileIdResponse".

4.5.2.1.4 The "CheckMobileSubscriptionEligibility" Function

The input data for the `CheckMobileSubscriptionEligibility` function defined in section 3.2.1.4 SHALL be mapped to the `CheckMobileSubscriptionEligibilityRequest` element that is of type `CheckMobileSubscriptionEligibilityRequestType` described in the following figure:

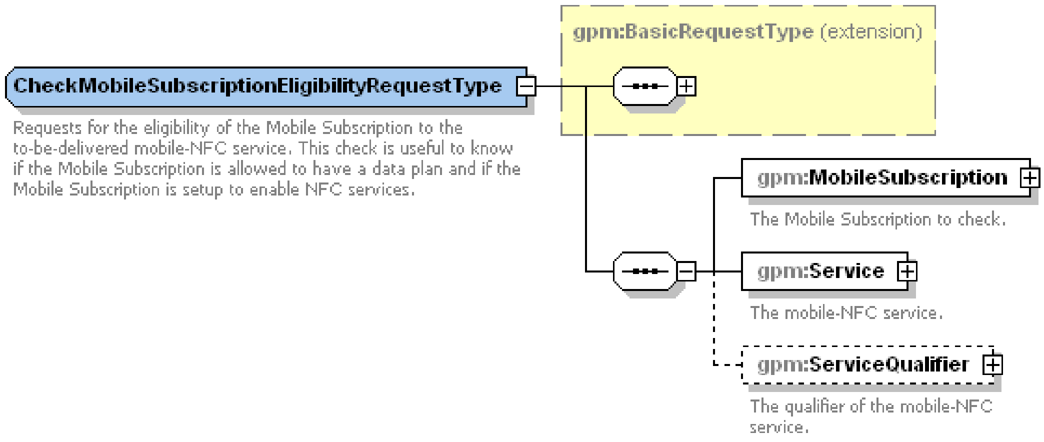

gpm:BasicRequestType (extension)
CheckMobileSubscriptionEligibilityRequestType
Requests for the eligibility of the Mobile Subscription to the to-be-delivered mobile-NFC service. This check is useful to know if the Mobile Subscription is allowed to have a data plan and if the Mobile Subscription is setup to enable NFC services.
gpm:MobileSubscription
The Mobile Subscription to check.
gpm:Service
The mobile-NFC service.
gpm:ServiceQualifier
The qualifier of the mobile-NFC service.

Figure 4-27: `CheckMobileSubscriptionEligibilityRequestType` type

Where:

- The `MobileSubscription` element is of type `MobileSubscriptionIdentifierType` (see section 4.5.1.2).
- The `Service` element is of type `ServiceIdentifierType` (see section 4.5.1.2).
- The `ServiceQualifier` element is of type `ServiceQualifierType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "CheckMobileSubscriptionEligibilityRequest".

The output data of the `CheckMobileSubscriptionEligibility` function defined in section 3.2.1.4 SHALL be mapped to the `CheckMobileSubscriptionEligibilityResponse` element that is of type `CheckMobileSubscriptionEligibilityResponseType` described in the following figure:

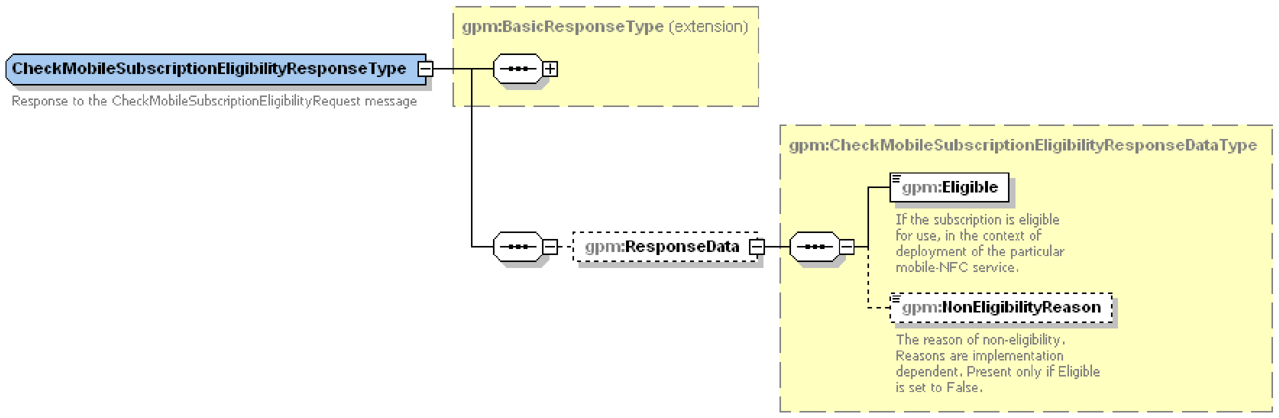

gpm:BasicResponseType (extension)
CheckMobileSubscriptionEligibilityResponseType
Response to the CheckMobileSubscriptionEligibilityRequest message
gpm:CheckMobileSubscriptionEligibilityResponseDataType
gpm:Eligible
If the subscription is eligible for use, in the context of deployment of the particular mobile-NFC service.
gpm:ResponseData
gpm:NonEligibilityReason
The reason of non-eligibility. Reasons are implementation dependent. Present only if Eligible is set to False.

Figure 4-28: `CheckMobileSubscriptionEligibilityResponseType` type

The value of the `GPHeader.Type` associated to this element SHALL be "CheckMobileSubscriptionEligibilityResponse".

4.5.2.2 AID Generation

No message, as no function.

4.5.2.3 Service Management Life Cycle Notifications

4.5.2.3.1 The "HandleStartServiceStateChangeNotification" Notification Function

The input data for the `HandleStartServiceStateChangeNotification` notification function defined in section 3.2.3.1 SHALL be mapped to the `HandleStartServiceStateChangeNotificationRequest` element that is of type `HandleStartServiceStateChangeNotificationRequestType` described in the following figure:

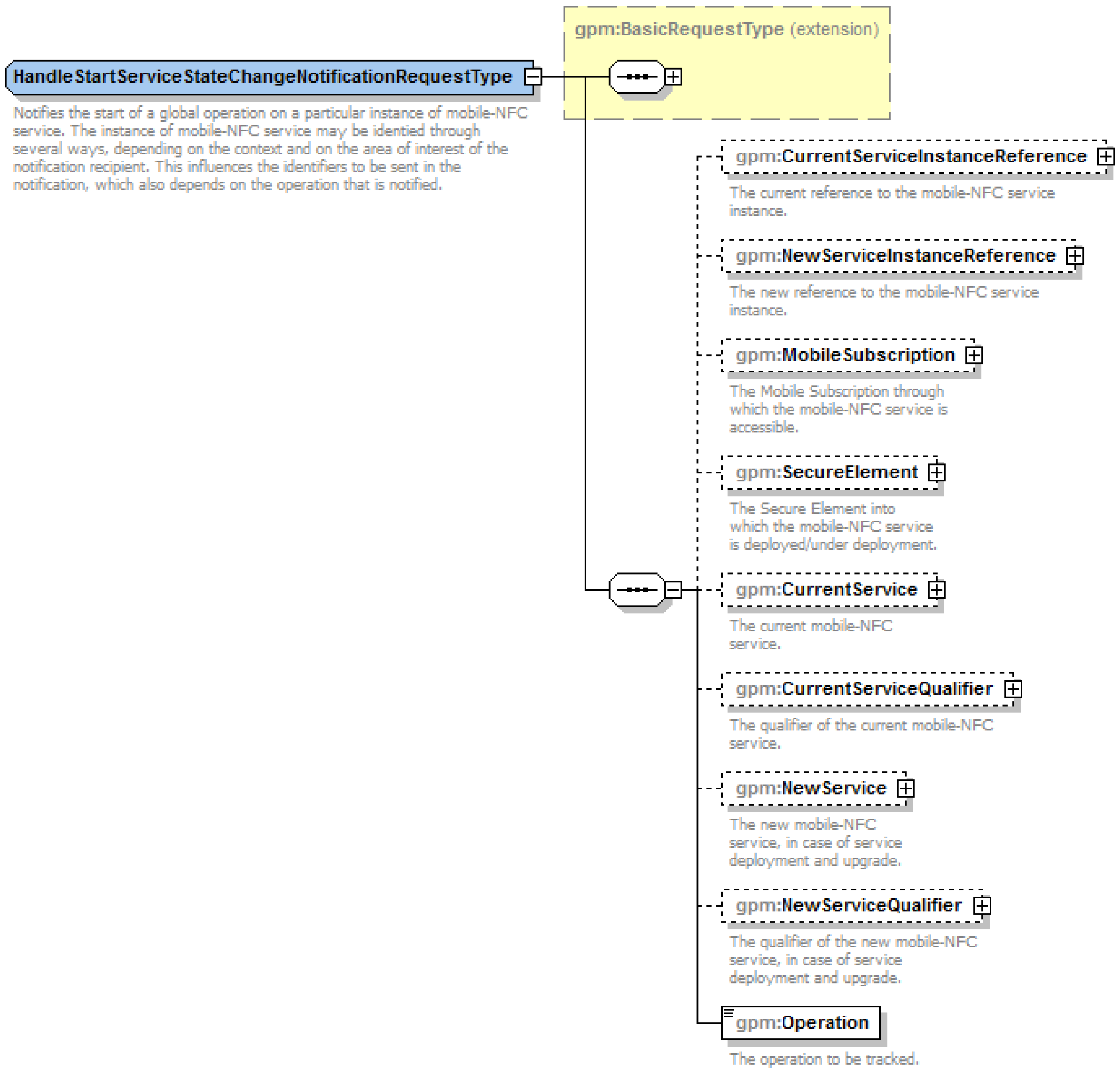

gpm:BasicRequestType (extension)
HandleStartServiceStateChangeNotificationRequestType
Notifies the start of a global operation on a particular instance of mobile-NFC service. The instance of mobile-NFC service may be identied through several ways, depending on the context and on the area of interest of the notification recipient. This influences the identifiers to be sent in the notification, which also depends on the operation that is notified.
gpm:CurrentServiceInstanceReference
The current reference to the mobile-NFC service instance.
gpm:NewServiceInstanceReference
The new reference to the mobile-NFC service instance.
gpm:MobileSubscription
The Mobile Subscription through which the mobile-NFC service is accessible.
gpm:SecureElement
The Secure Element into which the mobile-NFC service is deployed/under deployment.
gpm:CurrentService
The current mobile-NFC service.
gpm:CurrentServiceQualifier
The qualifier of the current mobile-NFC service.
gpm:NewService
The new mobile-NFC service, in case of service deployment and upgrade.
gpm:NewServiceQualifier
The qualifier of the new mobile-NFC service, in case of service deployment and upgrade.
gpm:Operation
The operation to be tracked.

Figure 4-29: `HandleStartServiceStateChangeNotificationRequestType` type

Where:

- The `ServiceInstanceReference` element is of type `ServiceInstanceReferenceType` (see section 4.5.1.2).

- The `MobileSubscription` element is of type `MobileSubscriptionIdentifierType` (see section 4.5.1.2).
- The `SecureElement` element is of type `SEIdentifierType` (see section 4.5.1.2).
- The `CurrentService` and `NewService` elements are of type `ServiceIdentifierType` (see section 4.5.1.2).
- The `CurrentServiceQualifier` and `NewServiceQualifier` elements are of type `ServiceQualifierType` (see section 4.5.1.2).
- The `Operation` is an enumeration of the possible status codes (`OperationType`).

The value of the `GPHeader.Type` associated to this element SHALL be "HandleStartServiceStateChangeNotificationRequest".

4.5.2.3.2 The "HandleEndServiceStateChangeNotification" Notification Function

The input data for the `HandleEndServiceStateChangeNotification` function defined in section 3.2.3.2 SHALL be mapped to the `HandleEndServiceStateChangeNotification` Request element that is of type `HandleEndServiceStateChangeNotificationRequestType` described in the following figure:

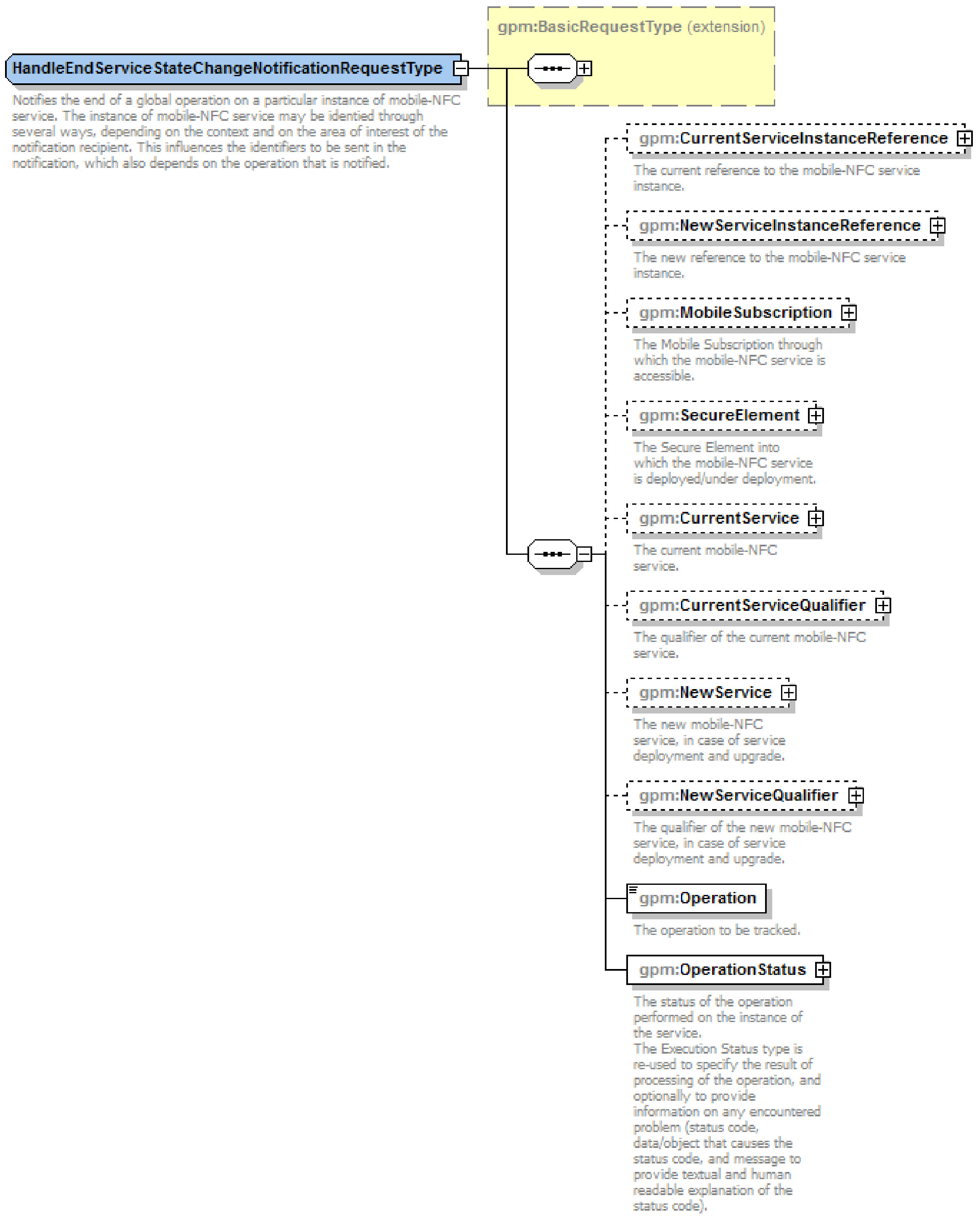

gpm:BasicRequestType (extension)
HandleEndServiceStateChangeNotificationRequestType
Notifies the end of a global operation on a particular instance of mobile-NFC service. The instance of mobile-NFC service may be identied through several ways, depending on the context and on the area of interest of the notification recipient. This influences the identifiers to be sent in the notification, which also depends on the operation that is notified.
gpm:CurrentServiceInstanceReference
The current reference to the mobile-NFC service instance.
gpm:NewServiceInstanceReference
The new reference to the mobile-NFC service instance.
gpm:MobileSubscription
The Mobile Subscription through which the mobile-NFC service is accessible.
gpm:SecureElement
The Secure Element into which the mobile-NFC service is deployed/under deployment.
gpm:CurrentService
The current mobile-NFC service.
gpm:CurrentServiceQualifier
The qualifier of the current mobile-NFC service.
gpm:NewService
The new mobile-NFC service, in case of service deployment and upgrade.
gpm:NewServiceQualifier
The qualifier of the new mobile-NFC service, in case of service deployment and upgrade.
gpm:Operation
The operation to be tracked.
gpm:OperationStatus
The status of the operation performed on the instance of the service. The Execution Status type is re-used to specify the result of processing of the operation, and optionally to provide information on any encountered problem (status code, data/object that causes the status code, and message to provide textual and human readable explanation of the status code).

Figure 4-30: `HandleEndServiceStateChangeNotificationRequestType` type

Where:

- The `ServiceInstanceReference` element is of type `ServiceInstanceReferenceType` (see section 4.5.1.2).
- The `MobileSubscription` element is of type `MobileSubscriptionIdentifierType` (see section 4.5.1.2).
- The `SecureElement` element is of type `SEIdentifierType` (see section 4.5.1.2).
- The `CurrentService` and `NewService` elements are of type `ServiceIdentifierType` (see section 4.5.1.2).
- The `CurrentServiceQualifier` and `NewServiceQualifier` elements are of type `ServiceQualifierType` (see section 4.5.1.2).
- The `Operation` is an enumeration of the possible status codes (`OperationType`).
- The `OperationStatus` element is of type `ExecutionStatusType` (see section 4.5.1.4).

The value of the `GPHeader.Type` associated to this element SHALL be "`HandleEndServiceStateChangeNotification` Request".

4.5.2.4 Mobile Subscription Life Cycle

4.5.2.4.1 The "GetMobileSubscriptionAlternateIdentifier" Function

The input data for the `GetMobileSubscriptionAlternateIdentifier` function defined in section 3.2.4.1 SHALL be mapped to the `GetMobileSubscriptionAlternateIdentifierRequest` element that is of type `GetMobileSubscriptionAlternateIdentifierRequestType` described in the following figure:

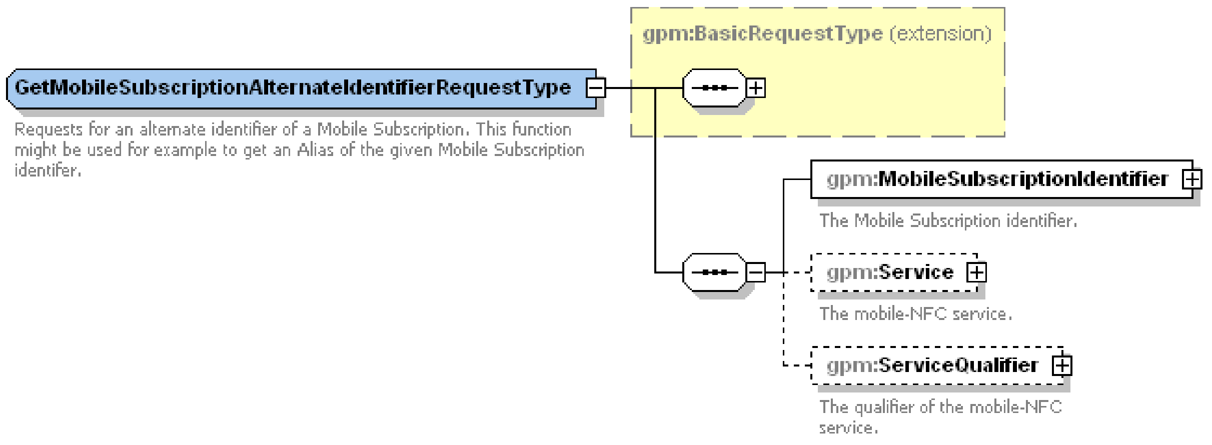

gpm:BasicRequestType (extension)
GetMobileSubscriptionAlternateIdentifierRequestType
Requests for an alternate identifier of a Mobile Subscription. This function might be used for example to get an Alias of the given Mobile Subscription identifer.
gpm:MobileSubscriptionIdentifier
The Mobile Subscription identifier.
gpm:Service
The mobile-NFC service.
gpm:ServiceQualifier
The qualifier of the mobile-NFC service.

Figure 4-31: `GetMobileSubscriptionAlternateIdentifierRequestType` type

Where:

- The `MobileSubscriptionIdentifier` element is of type `MobileSubscriptionIdentifierType` (see section 4.5.1.2).
- The `Service` element is of type `ServiceIdentifierType` (see section 4.5.1.2).

- The `ServiceQualifier` element is of type `ServiceQualifierType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "GetMobileSubscriptionAlternateIdentifierRequest".

The output data of the `GetMobileSubscriptionAlternateIdentifier` function defined in section 3.2.4.1 SHALL be mapped to the `GetMobileSubscriptionAlternateIdentifierResponse` element that is of type `GetMobileSubscriptionAlternateIdentifierResponseType` described in the following figure:

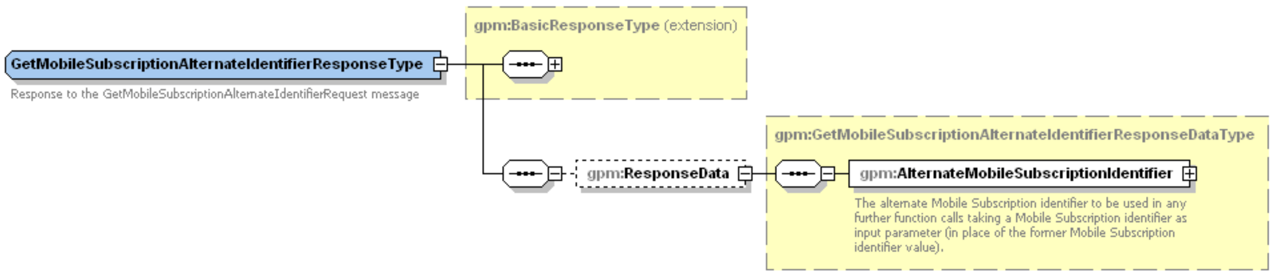

gpm:BasicResponseType (extension)
GetMobileSubscriptionAlternateIdentifierResponseType
Response to the GetMobileSubscriptionAlternateIdentifierRequest message
gpm:GetMobileSubscriptionAlternateIdentifierResponseDataType
gpm:ResponseData
gpm:AlternateMobileSubscriptionIdentifier
The alternate Mobile Subscription identifier to be used in any further function calls taking a Mobile Subscription identifier as input parameter (in place of the former Mobile Subscription identifier value).

Figure 4-32: `GetMobileSubscriptionAlternateIdentifierResponseType` type

Where:

- The `AlternateMobileSubscriptionIdentifier` element is of type `MobileSubscriptionIdentifierType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "GetMobileSubscriptionAlternateIdentifierResponse".

4.5.2.4.2 The "GetSEMobileSubscriptionIdentifier" Function

The input data for the `GetSEMobileSubscriptionIdentifier` function defined in section 3.2.4.2 SHALL be mapped to the `GetSEMobileSubscriptionIdentifierRequest` element that is of type `GetSEMobileSubscriptionIdentifierRequestType` described in the following figure:

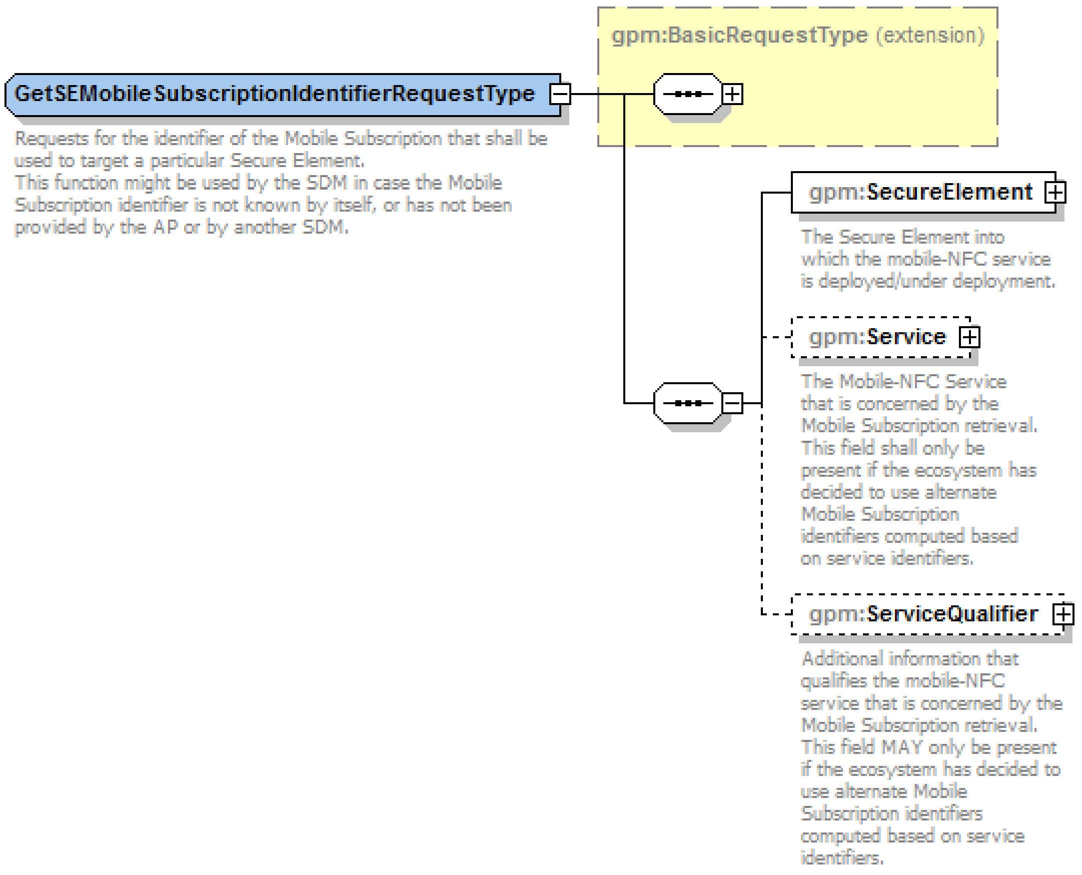

gpm:BasicRequestType (extension)
GetSEMobileSubscriptionIdentifierRequestType
Requests for the identifier of the Mobile Subscription that shall be used to target a particular Secure Element.
This function might be used by the SDM in case the Mobile Subscription identifier is not known by itself, or has not been provided by the AP or by another SDM.
gpm:SecureElement
The Secure Element into which the mobile-NFC service is deployed/under deployment.
gpm:Service
The Mobile-NFC Service that is concerned by the Mobile Subscription retrieval. This field shall only be present if the ecosystem has decided to use alternate Mobile Subscription identifiers computed based on service identifiers.
gpm:ServiceQualifier
Additional information that qualifies the mobile-NFC service that is concerned by the Mobile Subscription retrieval. This field MAY only be present if the ecosystem has decided to use alternate Mobile Subscription identifiers computed based on service identifiers.

Figure 4-33: `GetSEMobileSubscriptionIdentifierRequestType` type

Where:

- The `SecureElement` element is of type `SEIdentifierType` (see section 4.5.1.2).
- The `Service` element is of type `ServiceIdentifierType` (see section 4.5.1.2).
- The `ServiceQualifier` element is of type `ServiceQualifierType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "GetSEMobileSubscriptionIdentifierRequest".

The output data of the `GetSEMobileSubscriptionIdentifier` function defined in section 3.2.4.2 SHALL be mapped to the `GetSEMobileSubscriptionIdentifierResponse` element that is of type `GetSEMobileSubscriptionIdentifierResponseType` described in the following figure:

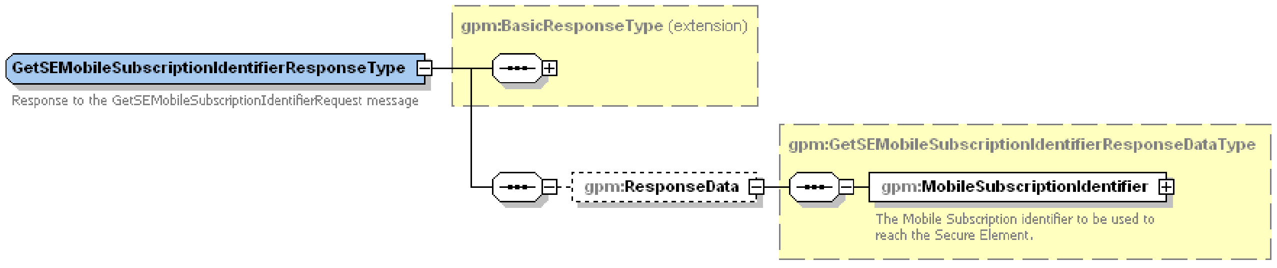

gpm:BasicResponseType (extension)
GetSEMobileSubscriptionIdentifierResponseType
Response to the GetSEMobileSubscriptionIdentifierRequest message
gpm:GetSEMobileSubscriptionIdentifierResponseDataType
gpm:ResponseData
gpm:MobileSubscriptionIdentifier
The Mobile Subscription identifier to be used to reach the Secure Element.

Figure 4-34: `GetSEMobileSubscriptionIdentifierResponseType` type

Where:

- The `MobileSubscriptionIdentifier` element is of type `MobileSubscriptionIdentifierType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "GetSEMobileSubscriptionIdentifierResponse".

4.5.2.4.3 The "GetMobileSubscriptionSEIdentifiers" Function

The input data for the `GetMobileSubscriptionSEIdentifiers` function defined in section 3.2.4.3 SHALL be mapped to the `GetMobileSubscriptionSEIdentifiersRequest` element that is of type `GetMobileSubscriptionSEIdentifiersRequestType` described in the following figure:

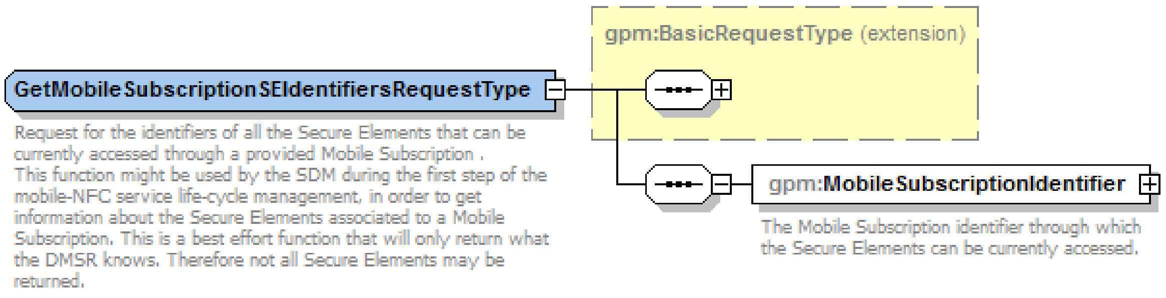

gpm:BasicRequestType (extension)
GetMobileSubscriptionSEIdentifiersRequestType
Request for the identifiers of all the Secure Elements that can be currently accessed through a provided Mobile Subscription . This function might be used by the SDM during the first step of the mobile-NFC service life-cycle management, in order to get information about the Secure Elements associated to a Mobile Subscription. This is a best effort function that will only return what the DMSR knows. Therefore not all Secure Elements may be returned.
gpm:MobileSubscriptionIdentifier
The Mobile Subscription identifier through which the Secure Elements can be currently accessed.

Figure 4-35: `GetMobileSubscriptionSEIdentifiersRequestType` type

Where:

- The `MobileSubscriptionIdentifier` element is of type `MobileSubscriptionIdentifierType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "GetMobileSubscriptionSEIdentifiersRequest".

The output data of the `GetMobileSubscriptionSEIdentifiers` function defined in section 3.2.4.3 SHALL be mapped to the `GetMobileSubscriptionSEIdentifiersResponse` element that is of type `GetMobileSubscriptionSEIdentifiersResponseType` described in the following figure:

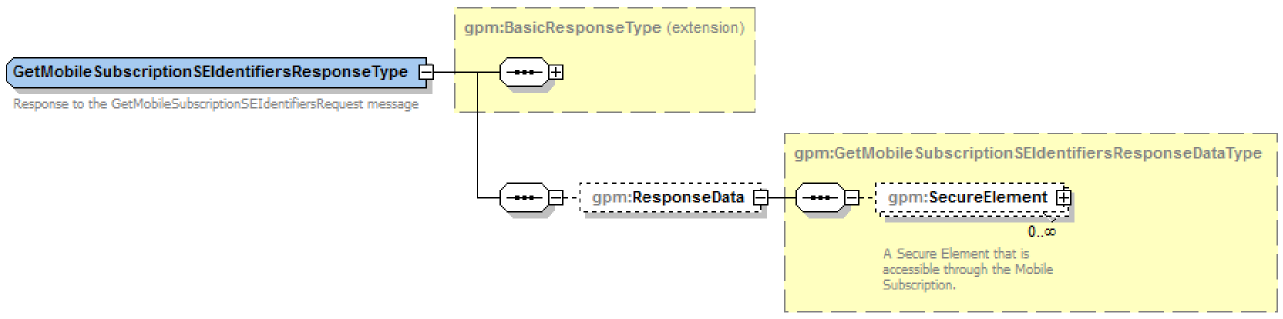

gpm:BasicResponseType (extension)
GetMobileSubscriptionSEIdentifiersResponseType
Response to the GetMobileSubscriptionSEIdentifiersRequest message
gpm:GetMobileSubscriptionSEIdentifiersResponseDataType
gpm:ResponseData
gpm:SecureElement
0..∞
A Secure Element that is accessible through the Mobile Subscription.

Figure 4-36: `GetMobileSubscriptionSEIdentifiersResponseType` type

Where:

- The `SecureElement` element is of type `SEIdentifierType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "GetMobileSubscriptionSEIdentifiersResponse".

4.5.2.4.4 The "HandleMobileSubscriptionIdentifierChangedNotification" Notification Function

The input data for the `HandleMobileSubscriptionIdentifierChangedNotification` function defined in section 3.2.4.4 SHALL be mapped to the `HandleMobileSubscriptionIdentifierChangedNotificationRequest` element that is of type `HandleMobileSubscriptionIdentifierChangedNotificationRequestType` described in the following figure:

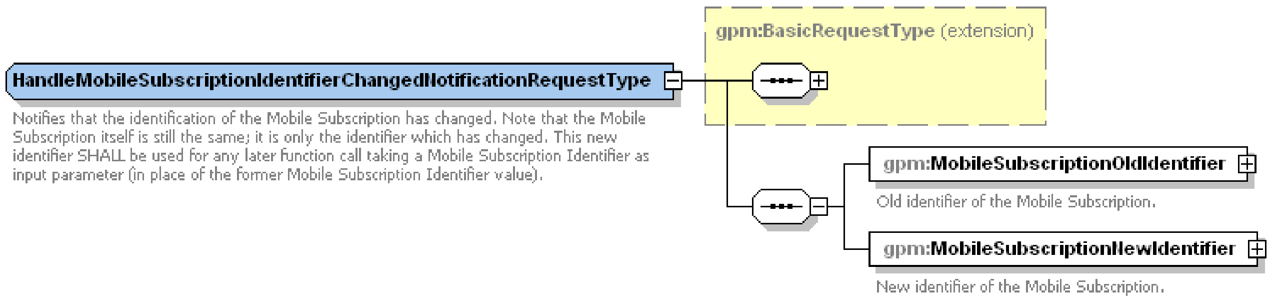

gpm:BasicRequestType (extension)
HandleMobileSubscriptionIdentifierChangedNotificationRequestType
Notifies that the identification of the Mobile Subscription has changed. Note that the Mobile Subscription itself is still the same; it is only the identifier which has changed. This new identifier SHALL be used for any later function call taking a Mobile Subscription Identifier as input parameter (in place of the former Mobile Subscription Identifier value).
gpm:MobileSubscriptionOldIdentifier
Old identifier of the Mobile Subscription.
gpm:MobileSubscriptionNewIdentifier
New identifier of the Mobile Subscription.

Figure 4-37: `HandleMobileSubscriptionIdentifierChangedNotificationRequestType` type

Where:

- The `MobileSubscriptionOldIdentifier` and `MobileSubscriptionNewIdentifier` elements are of type `MobileSubscriptionIdentifierType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "HandleMobileSubscriptionIdentifierChangedNotificationRequest".

4.5.2.4.5 The "HandleMobileSubscriptionStatusChangeNotification" Notification Function

The input data for the `HandleMobileSubscriptionStatusChangeNotification` function defined in section 3.2.4.5 SHALL be mapped to the `HandleMobileSubscriptionStatusChangeNotificationRequest` element that is of type `HandleMobileSubscriptionStatusChangeNotificationRequestType` described in the following figure:

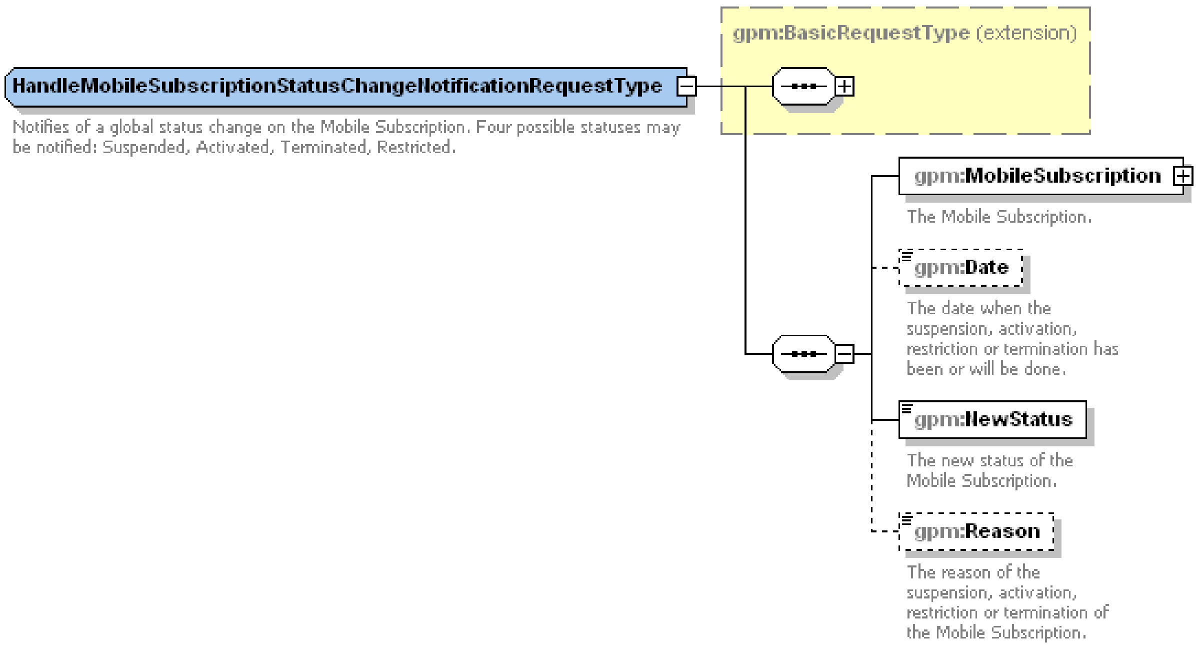

gpm:BasicRequestType (extension)
HandleMobileSubscriptionStatusChangeNotificationRequestType
Notifies of a global status change on the Mobile Subscription. Four possible statuses may be notified: Suspended, Activated, Terminated, Restricted.
gpm:MobileSubscription
The Mobile Subscription.
gpm:Date
The date when the suspension, activation, restriction or termination has been or will be done.
gpm:NewStatus
The new status of the Mobile Subscription.
gpm:Reason
The reason of the suspension, activation, restriction or termination of the Mobile Subscription.

Figure 4-38: `HandleMobileSubscriptionStatusChangeNotificationRequestType` type

Where:

- The `MobileSubscription` element is of type `MobileSubscriptionIdentifierType` (see section 4.5.1.2).
- The `NewStatus` element is an enumeration of the possible Mobile Subscription statuses (`MobileSubscriptionStatusType`).

The value of the `GPHeader.Type` associated to this element SHALL be "HandleMobileSubscriptionStatusChangeNotificationRequest".

4.5.2.5 Secure Element Subscription Life Cycle

4.5.2.5.1 The "HandleSERenewalNotification" Notification Function

The input data for the `HandleSERenewalNotification` function defined in section 3.2.5.1 SHALL be mapped to the `HandleSERenewalNotificationRequest` element that is of type `HandleSERenewalNotificationRequestType` described in the following figure:

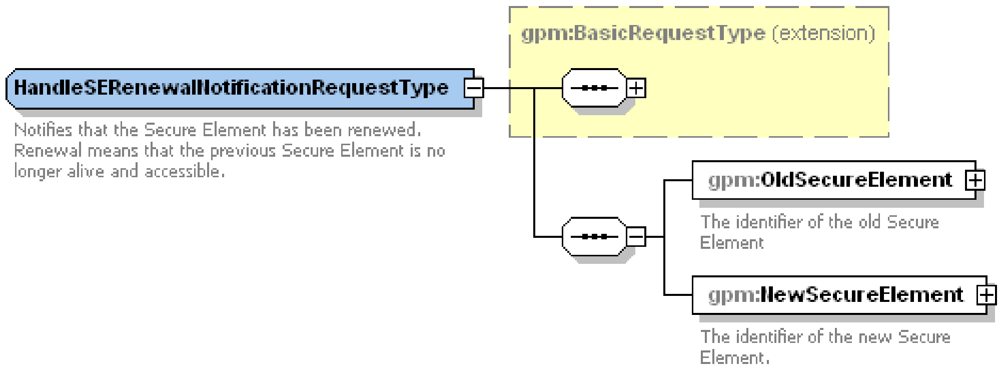

gpm:BasicRequestType (extension)
HandleSERenewalNotificationRequestType
Notifies that the Secure Element has been renewed. Renewal means that the previous Secure Element is no longer alive and accessible.
gpm:OldSecureElement
The identifier of the old Secure Element
gpm:NewSecureElement
The identifier of the new Secure Element.

Figure 4-39: `HandleSERenewalNotificationRequestType` type

Where:

- The `OldSecureElement` and `NewSecureElement` elements are of type `SEIdentifierType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "HandleSERenewalNotificationRequest".

4.5.2.5.2 The "HandleSEDeviceChangedNotification" Notification Function

The input data for the `HandleSEDeviceChangedNotification` function defined in section 3.2.5.2 SHALL be mapped to the `HandleSEDeviceChangedNotificationRequest` element that is of type `HandleSEDeviceChangedNotificationRequestType` described in the following figure:

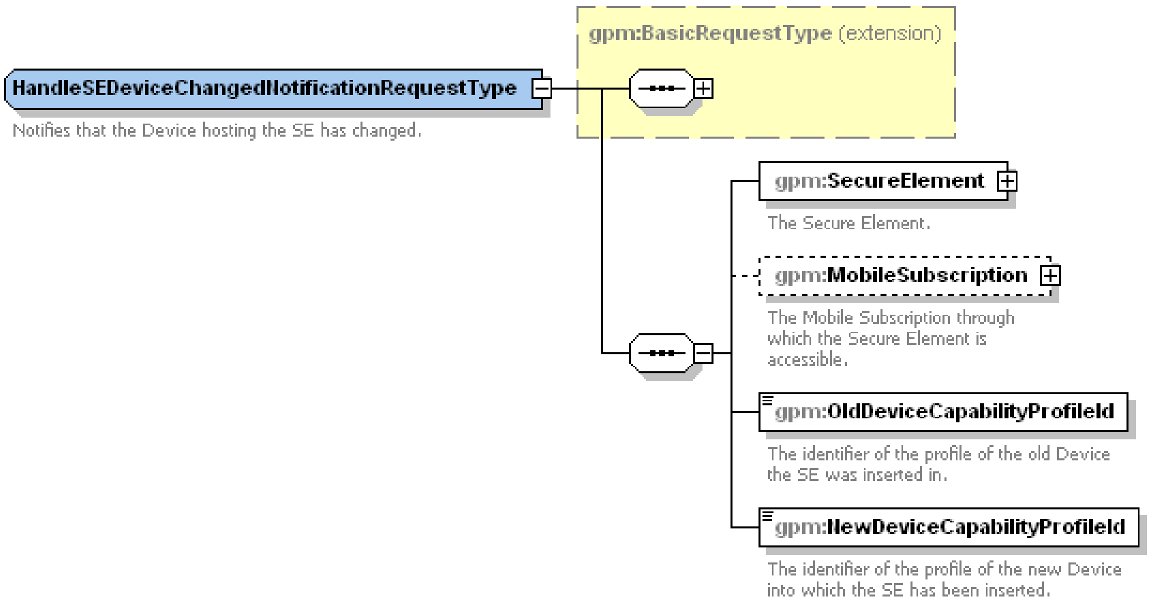

gpm:BasicRequestType (extension)
HandleSEDeviceChangedNotificationRequestType
Notifies that the Device hosting the SE has changed.
gpm:SecureElement
The Secure Element.
gpm:MobileSubscription
The Mobile Subscription through which the Secure Element is accessible.
gpm:OldDeviceCapabilityProfileId
The identifier of the profile of the old Device the SE was inserted in.
gpm:NewDeviceCapabilityProfileId
The identifier of the profile of the new Device into which the SE has been inserted.

Figure 4-40: `HandleSEDeviceChangedNotificationRequestType` type

Where:

- The `MobileSubscription` element is of type `MobileSubscriptionIdentifierType` (see section 4.5.1.2).
- The `SecureElement` element is of type `SEIdentifierType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "HandleSEDeviceChangedNotificationRequest".

4.5.2.5.3 The "HandleSEDeviceStatusChangeNotification" Notification Function

The input data for the `HandleSEDeviceStatusChangeNotification` function defined in section 3.2.5.3 SHALL be mapped to the `HandleSEDeviceStatusChangeNotificationRequest` element that is of type `HandleSEDeviceStatusChangeNotificationRequestType` described in the following figure:

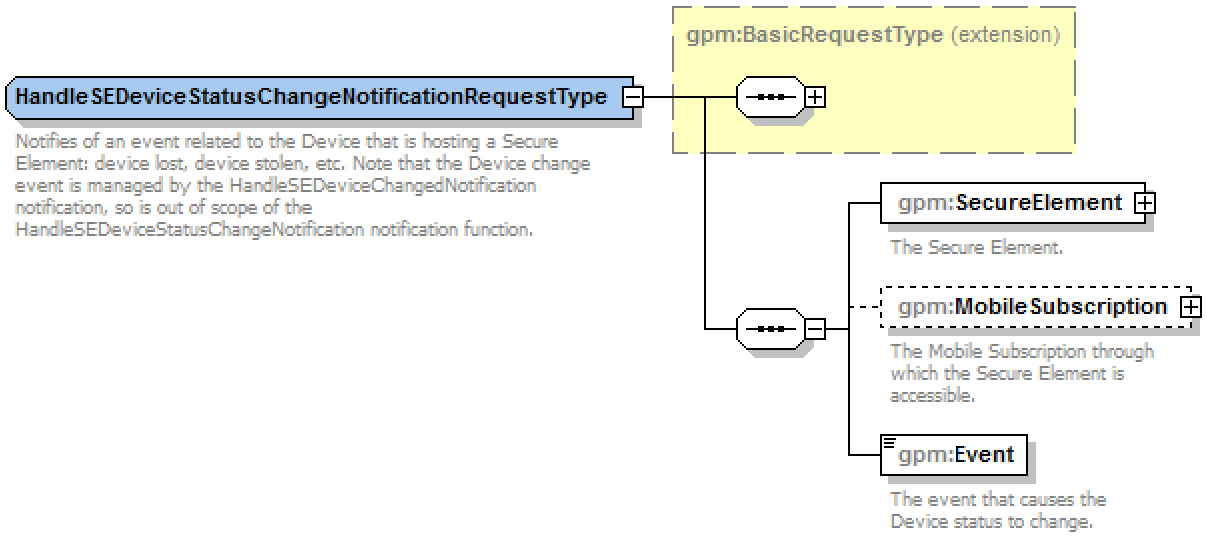

gpm:BasicRequestType (extension)
HandleSEDeviceStatusChangeNotificationRequestType
Notifies of an event related to the Device that is hosting a Secure Element: device lost, device stolen, etc. Note that the Device change event is managed by the HandleSEDeviceChangedNotification notification, so is out of scope of the HandleSEDeviceStatusChangeNotification notification function.
gpm:SecureElement
The Secure Element.
gpm:MobileSubscription
The Mobile Subscription through which the Secure Element is accessible.
gpm:Event
The event that causes the Device status to change.

Figure 4-41: `HandleSEDeviceStatusChangeNotificationRequestType` type

Where:

- The `MobileSubscription` element is of type `MobileSubscriptionIdentifierType` (see section 4.5.1.2).
- The `SecureElement` element is of type `SEIdentifierType` (see section 4.5.1.2).
- The `Event` is an enumeration of the possible events (`DeviceStatusChangeEventType`).

The value of the `GPHeader.Type` associated to this element SHALL be "HandleSEDeviceStatusChangeNotificationRequest".

4.5.2.5.4 The "HandleSEMobileSubscriptionChangedNotification" Notification Function

The input data for the `HandleSEMobileSubscriptionChangedNotification` function defined in section 3.2.5.4 SHALL be mapped to the `HandleSEMobileSubscriptionChangedNotificationRequest` element that is of type `HandleSEMobileSubscriptionChangedNotificationRequestType` described in the following figure:

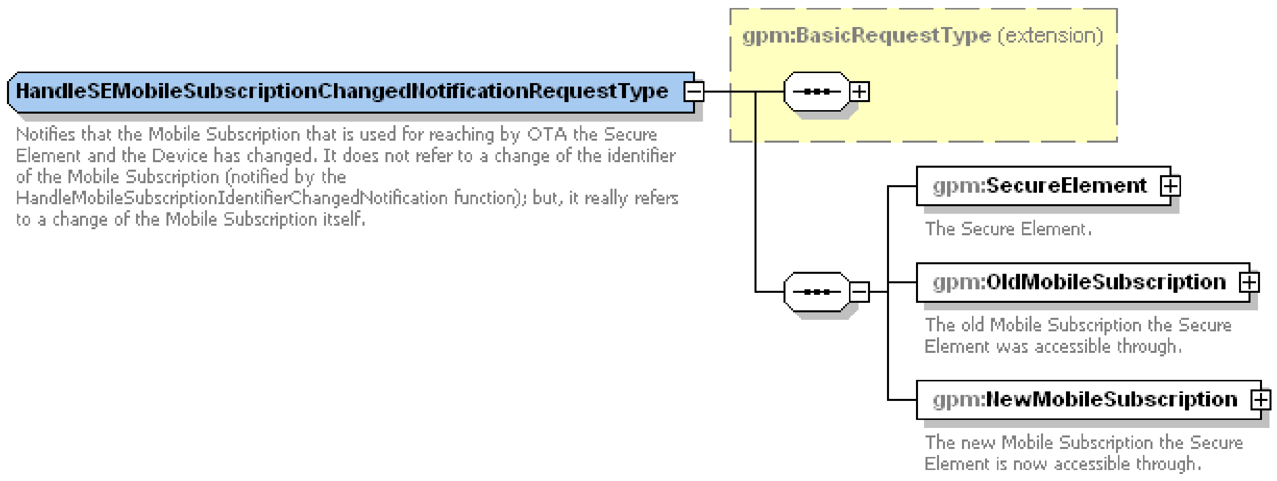

gpm:BasicRequestType (extension)
HandleSEMobileSubscriptionChangedNotificationRequestType
Notifies that the Mobile Subscription that is used for reaching by OTA the Secure Element and the Device has changed. It does not refer to a change of the identifier of the Mobile Subscription (notified by the HandleMobileSubscriptionIdentifierChangedNotification function); but, it really refers to a change of the Mobile Subscription itself.
gpm:SecureElement
The Secure Element.
gpm:OldMobileSubscription
The old Mobile Subscription the Secure Element was accessible through.
gpm:NewMobileSubscription
The new Mobile Subscription the Secure Element is now accessible through.

Figure 4-42: `HandleSEMobileSubscriptionChangedNotificationRequestType` type

Where:

- The `SecureElement` element is of type `SEIdentifierType` (see section 4.5.1.2).
- The `OldMobileSubscription` and `NewMobileSubscription` elements are of type `MobileSubscriptionIdentifierType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "HandleSEMobileSubscriptionChangedNotificationRequest".

4.5.2.5.5 The "HandleSEStatusChangeNotification" Notification Function

The input data for the `HandleSEStatusChangeNotification` function defined in section 3.2.5.5 SHALL be mapped to the `HandleSEStatusChangeNotificationRequest` element that is of type `HandleSEStatusChangeNotificationRequestType` described in the following figure:

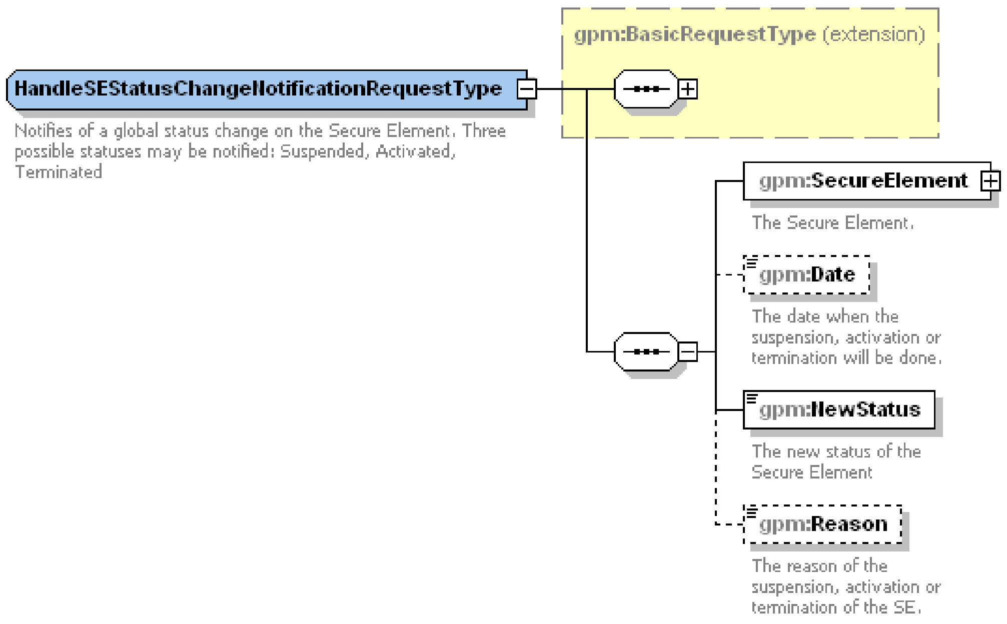

gpm:BasicRequestType (extension)
HandleSEStatusChangeNotificationRequestType
Notifies of a global status change on the Secure Element. Three possible statuses may be notified: Suspended, Activated, Terminated
gpm:SecureElement
The Secure Element.
gpm:Date
The date when the suspension, activation or termination will be done.
gpm:NewStatus
The new status of the Secure Element
gpm:Reason
The reason of the suspension, activation or termination of the SE.

Figure 4-43: `HandleSEStatusChangeNotificationRequestType` type

Where:

- The `SecureElement` element is of type `SEIdentifierType` (see section 4.5.1.2).
- The `NewStatus` element is an enumeration of the possible SE statuses (`SecureElementStatusType`).

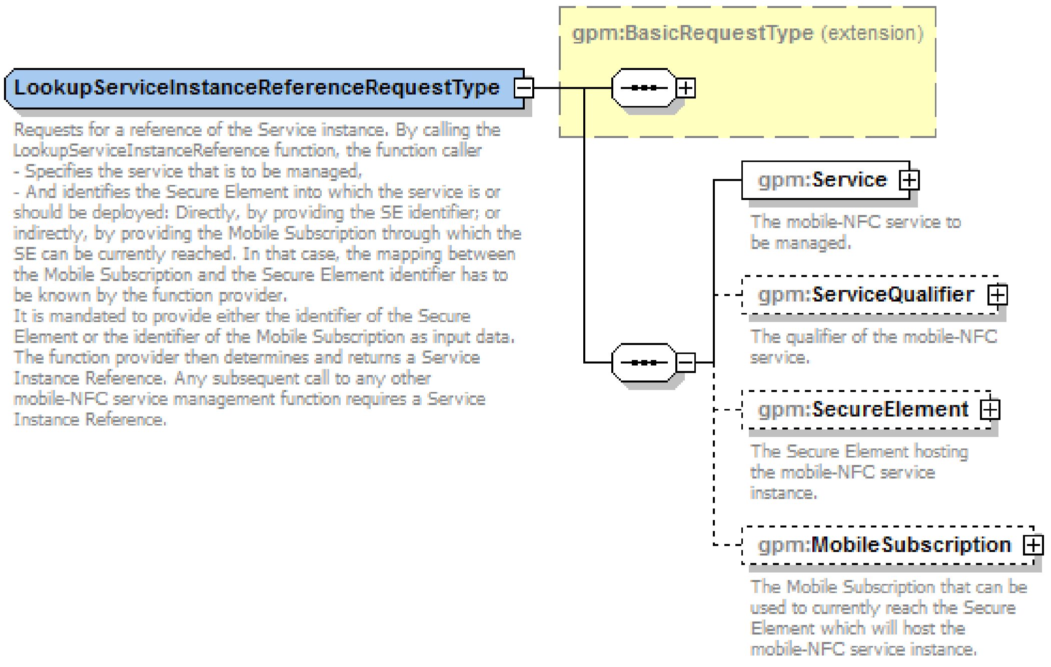
The value of the `GPHeader.Type` associated to this element SHALL be "HandleSEStatusChangeNotificationRequest".

4.5.3 Functions for Global Mobile-NFC Service Management

4.5.3.1 The Service Deployment Function

4.5.3.1.1 The "LookupServiceInstanceReference" Function

The input data for the `LookupServiceInstanceReference` function defined in section 3.3.2.1 SHALL be mapped to the `LookupServiceInstanceReferenceRequest` element that is of type `LookupServiceInstanceReferenceRequestType` described in the following figure:


gpm:BasicRequestType (extension)
LookupServiceInstanceReferenceRequestType
gpm:Service
gpm:ServiceQualifier
gpm:SecureElement
gpm:MobileSubscription

Requests for a reference of the Service instance. By calling the LookupServiceInstanceReference function, the function caller
- Specifies the service that is to be managed.
- And identifies the Secure Element into which the service is or should be deployed: Directly, by providing the SE identifier; or indirectly, by providing the Mobile Subscription through which the SE can be currently reached. In that case, the mapping between the Mobile Subscription and the Secure Element identifier has to be known by the function provider.
It is mandated to provide either the identifier of the Secure Element or the identifier of the Mobile Subscription as input data. The function provider then determines and returns a Service Instance Reference. Any subsequent call to any other mobile-NFC service management function requires a Service Instance Reference.

The mobile-NFC service to be managed.

The qualifier of the mobile-NFC service.

The Secure Element hosting the mobile-NFC service instance.

The Mobile Subscription that can be used to currently reach the Secure Element which will host the mobile-NFC service instance.

Figure 4-44: `LookupServiceInstanceReferenceRequestType` type

Where:

- The `Service` element is of type `ServiceIdentifierType` (see section 4.5.1.2).
- The `ServiceQualifier` element is of type `ServiceQualifierType` (see section 4.5.1.2).
- The `SecureElement` element is of type `SEIdentifierType` (see section 4.5.1.2).
- The `MobileSubscription` element is of type `MobileSubscriptionIdentifierType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "LookupServiceInstanceReferenceRequest".

The output data of the `LookupServiceInstanceReference` function defined in section 3.3.2.1 SHALL be mapped to the `LookupServiceInstanceReferenceResponse` element that is of type `LookupServiceInstanceReferenceResponseType` described in the following figure:

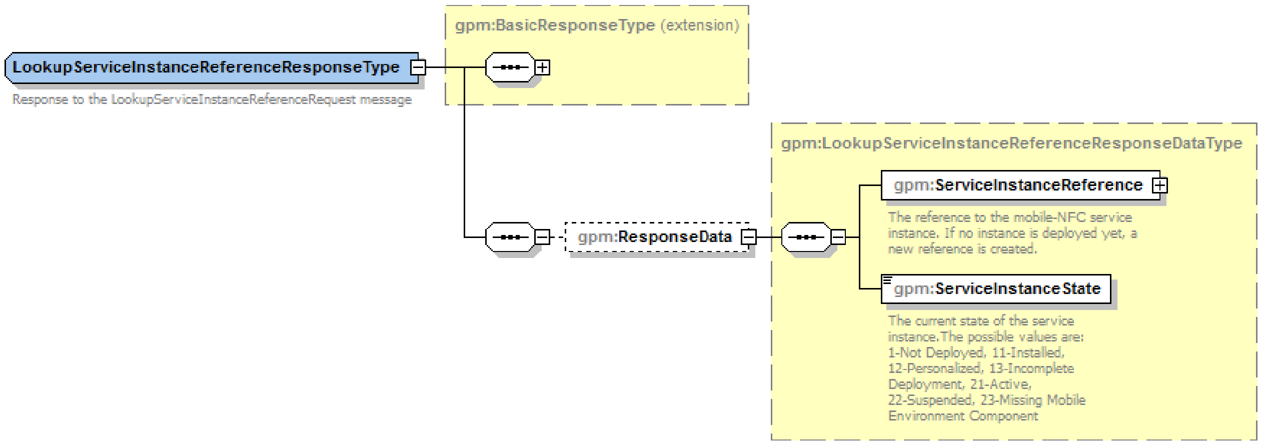

gpm:BasicResponseType (extension)
LookupServiceInstanceReferenceResponseType
Response to the LookupServiceInstanceReferenceRequest message
gpm:LookupServiceInstanceReferenceResponseDataType
gpm:ServiceInstanceReference
The reference to the mobile-NFC service instance. If no instance is deployed yet, a new reference is created.
gpm:ResponseData
gpm:ServiceInstanceState
The current state of the service instance.The possible values are: 1-Not Deployed, 11-Installed, 12-Personalized, 13-Incomplete Deployment, 21-Active, 22-Suspended, 23-Missing Mobile Environment Component

Figure 4-45: `LookupServiceInstanceReferenceResponseType` type

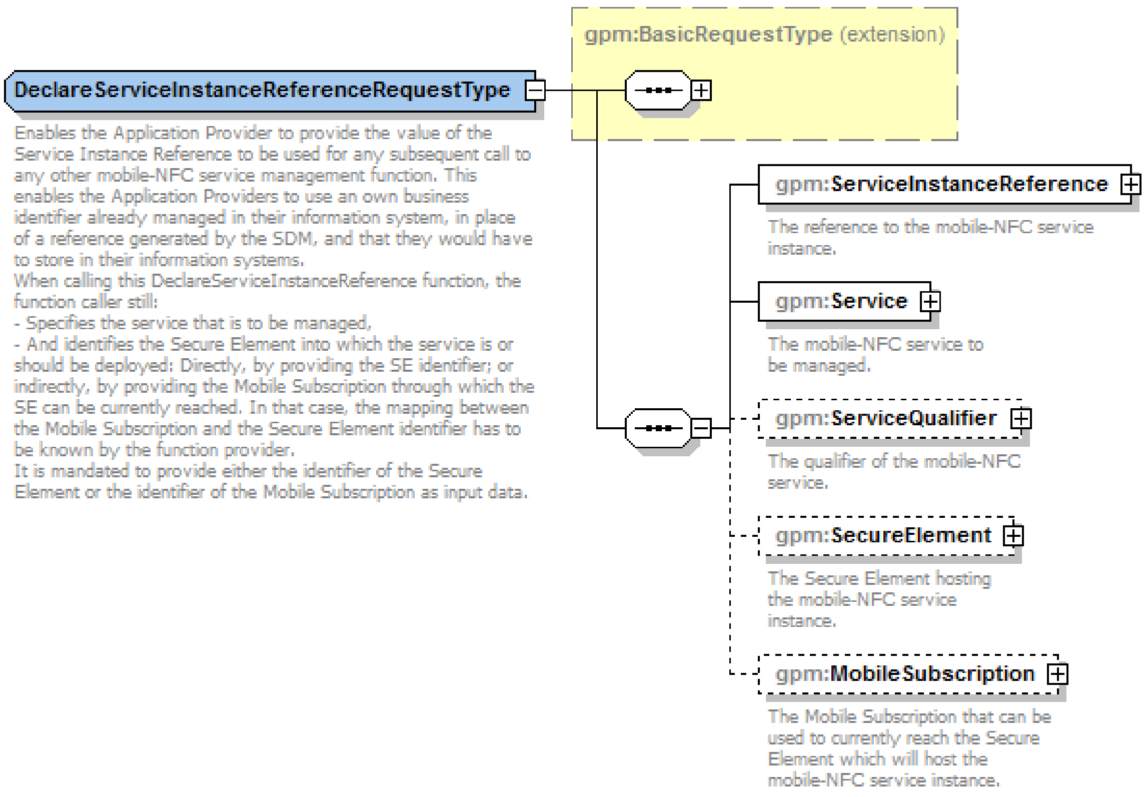
Where:

- The `ServiceInstanceReference` element is of type `ServiceInstanceReferenceType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "LookupServiceInstanceReferenceResponse".

4.5.3.1.2 The "DeclareServiceInstanceReference" Function

The input data for the `DeclareServiceInstanceReference` function defined in section 3.3.2.2 SHALL be mapped to the `DeclareServiceInstanceReferenceRequest` element that is of type `DeclareServiceInstanceReferenceRequestType` described in the following figure:


gpm:BasicRequestType (extension)
DeclareServiceInstanceReferenceRequestType
gpm:ServiceInstanceReference
gpm:Service
gpm:ServiceQualifier
gpm:SecureElement
gpm:MobileSubscription

Enables the Application Provider to provide the value of the Service Instance Reference to be used for any subsequent call to any other mobile-NFC service management function. This enables the Application Providers to use an own business identifier already managed in their information system, in place of a reference generated by the SDM, and that they would have to store in their information systems.
When calling this DeclareServiceInstanceReference function, the function caller still:
- Specifies the service that is to be managed,
- And identifies the Secure Element into which the service is or should be deployed: Directly, by providing the SE identifier; or indirectly, by providing the Mobile Subscription through which the SE can be currently reached. In that case, the mapping between the Mobile Subscription and the Secure Element identifier has to be known by the function provider.
It is mandated to provide either the identifier of the Secure Element or the identifier of the Mobile Subscription as input data.

The reference to the mobile-NFC service instance.

The mobile-NFC service to be managed.

The qualifier of the mobile-NFC service.

The Secure Element hosting the mobile-NFC service instance.

The Mobile Subscription that can be used to currently reach the Secure Element which will host the mobile-NFC service instance.

Figure 4-46: `DeclareServiceInstanceReferenceRequestType` type

Where:

- The `ServiceInstanceReference` element is of type `ServiceInstanceReferenceType` (see section 4.5.1.2).
- The `Service` element is of type `ServiceIdentifierType` (see section 4.5.1.2).
- The `ServiceQualifier` element is of type `ServiceQualifierType` (see section 4.5.1.2).
- The `SecureElement` element is of type `SEIdentifierType` (see section 4.5.1.2).
- The `MobileSubscription` element is of type `MobileSubscriptionIdentifierType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "DeclareServiceInstanceReferenceRequest".

The output data of the `DeclareServiceInstanceReference` function defined in section 3.3.2.2 SHALL be mapped to the `DeclareServiceInstanceReferenceResponse` element that is of type `DeclareServiceInstanceReferenceResponseType` described in the following figure:

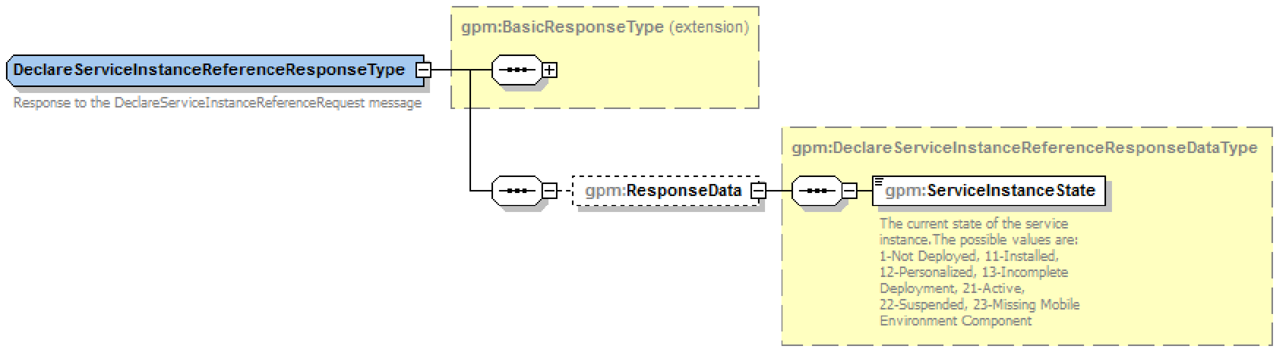

gpm:BasicResponseType (extension)
DeclareServiceInstanceReferenceResponseType
Response to the DeclareServiceInstanceReferenceRequest message
gpm:DeclareServiceInstanceReferenceResponseDataType
gpm:ResponseData
gpm:ServiceInstanceState
The current state of the service instance.The possible values are: 1-Not Deployed, 11-Installed, 12-Personalized, 13-Incomplete Deployment, 21-Active, 22-Suspended, 23-Missing Mobile Environment Component

Figure 4-47: `DeclareServiceInstanceReferenceResponseType` type

The value of the `GPHeader.Type` associated to this element SHALL be "DeclareServiceInstanceReferenceResponse".

4.5.3.1.3 The "GetServiceInstanceReferenceDescriptor" Function

The input data for the `GetServiceInstanceReferenceDescriptor` function defined in section 3.3.2.3 SHALL be mapped to the `GetServiceInstanceReferenceDescriptorRequest` element that is of type `GetServiceInstanceReferenceDescriptorRequestType` described in the following figure:

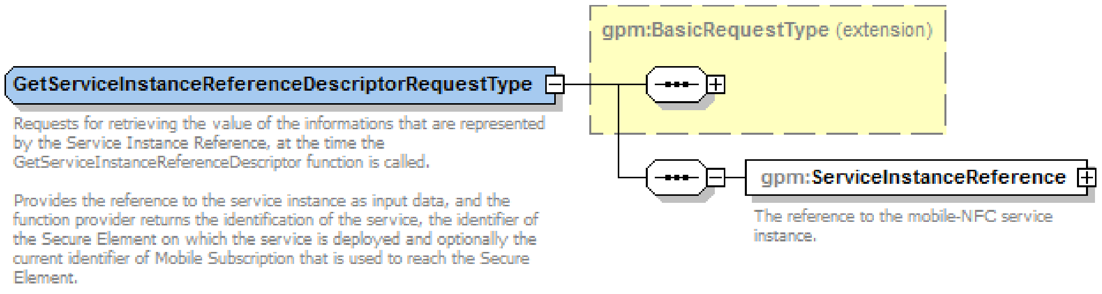

gpm:BasicRequestType (extension)
GetServiceInstanceReferenceDescriptorRequestType
Requests for retrieving the value of the informations that are represented by the Service Instance Reference, at the time the GetServiceInstanceReferenceDescriptor function is called.
Provides the reference to the service instance as input data, and the function provider returns the identification of the service, the identifier of the Secure Element on which the service is deployed and optionally the current identifier of Mobile Subscription that is used to reach the Secure Element.
gpm:ServiceInstanceReference
The reference to the mobile-NFC service instance.

Figure 4-48: `GetServiceInstanceReferenceDescriptorRequestType` type

Where:

- The `ServiceInstanceReference` element is of type `ServiceInstanceReferenceType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "GetServiceInstanceReferenceDescriptorRequest".

The output data of the `GetServiceInstanceReferenceDescriptor` function defined in section 3.3.2.3 SHALL be mapped to the `GetServiceInstanceReferenceDescriptorResponse` element that is of type `GetServiceInstanceReferenceDescriptorResponseType` described in the following figure:

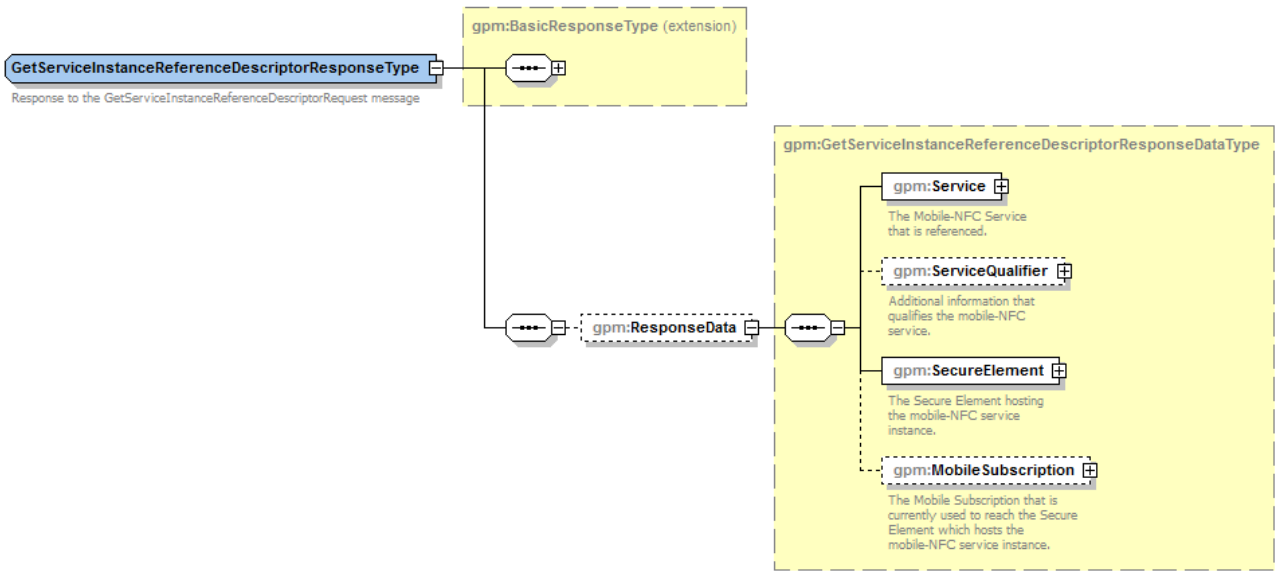

gpm:BasicResponseType (extension)
GetServiceInstanceReferenceDescriptorResponseType
Response to the GetServiceInstanceReferenceDescriptorRequest message
gpm:GetServiceInstanceReferenceDescriptorResponseDataType
gpm:Service
The Mobile-NFC Service that is referenced.
gpm:ServiceQualifier
Additional information that qualifies the mobile-NFC service.
gpm:ResponseData
gpm:SecureElement
The Secure Element hosting the mobile-NFC service instance.
gpm:MobileSubscription
The Mobile Subscription that is currently used to reach the Secure Element which hosts the mobile-NFC service instance.

Figure 4-49: `GetServiceInstanceReferenceDescriptorResponseType` type

Where:

- The `Service` element is of type `ServiceIdentifierType` (see section 4.5.1.2).
- The `ServiceQualifier` element is of type `ServiceQualifierType` (see section 4.5.1.2).
- The `SecureElement` element is of type `SEIdentifierType` (see section 4.5.1.2).
- The `MobileSubscription` element is of type `MobileSubscriptionIdentifierType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "GetServiceInstanceReferenceDescriptorResponse".

4.5.3.1.4 The "GetServiceState" Function

The input data for the `GetServiceState` function defined in section 3.3.2.4 SHALL be mapped to the `GetServiceStateRequest` element that is of type `GetServiceStateRequestType` described in the following figure:

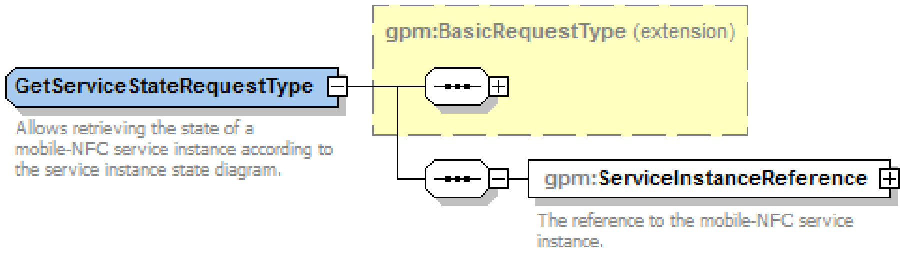

gpm:BasicRequestType (extension)
GetServiceStateRequestType
Allows retrieving the state of a mobile-NFC service instance according to the service instance state diagram.
gpm:ServiceInstanceReference
The reference to the mobile-NFC service instance.

Figure 4-50: `GetServiceStateRequestType` type

Where:

- The `ServiceInstanceReference` element is of type `ServiceInstanceReferenceType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "GetServiceStateRequest".

The output data of the `GetServiceState` function defined in section 3.3.2.1 SHALL be mapped to the `GetServiceStateResponse` element that is of type `GetServiceStateResponseType` described in the following figure:

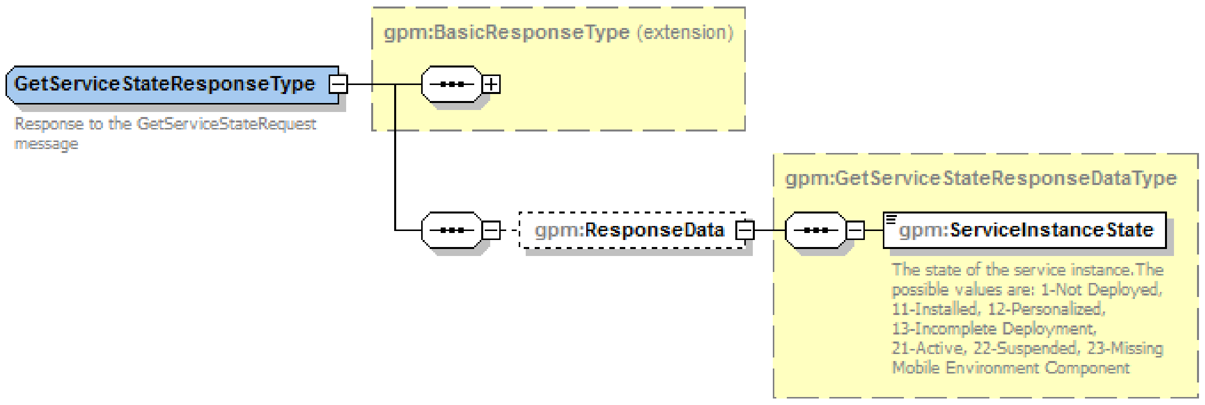

gpm:BasicResponseType (extension)
GetServiceStateResponseType
Response to the GetServiceStateRequest message
gpm:ResponseData
gpm:GetServiceStateResponseDataType
gpm:ServiceInstanceState
The state of the service instance.The possible values are: 1-Not Deployed, 11-Installed, 12-Personalized, 13-Incomplete Deployment, 21-Active, 22-Suspended, 23-Missing Mobile Environment Component

Figure 4-51: `GetServiceStateResponseType` type

Where:

- The `ServiceInstanceReference` element is of type `ServiceInstanceReferenceType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "GetServiceStateResponse".

4.5.3.1.5 The "DeployService" Function

The input data for the `DeployServiceService` function defined in section 3.3.2.5.2 SHALL be mapped to the `DeployServiceRequest` element that is of type `DeployServiceRequestType` described in the following figure:

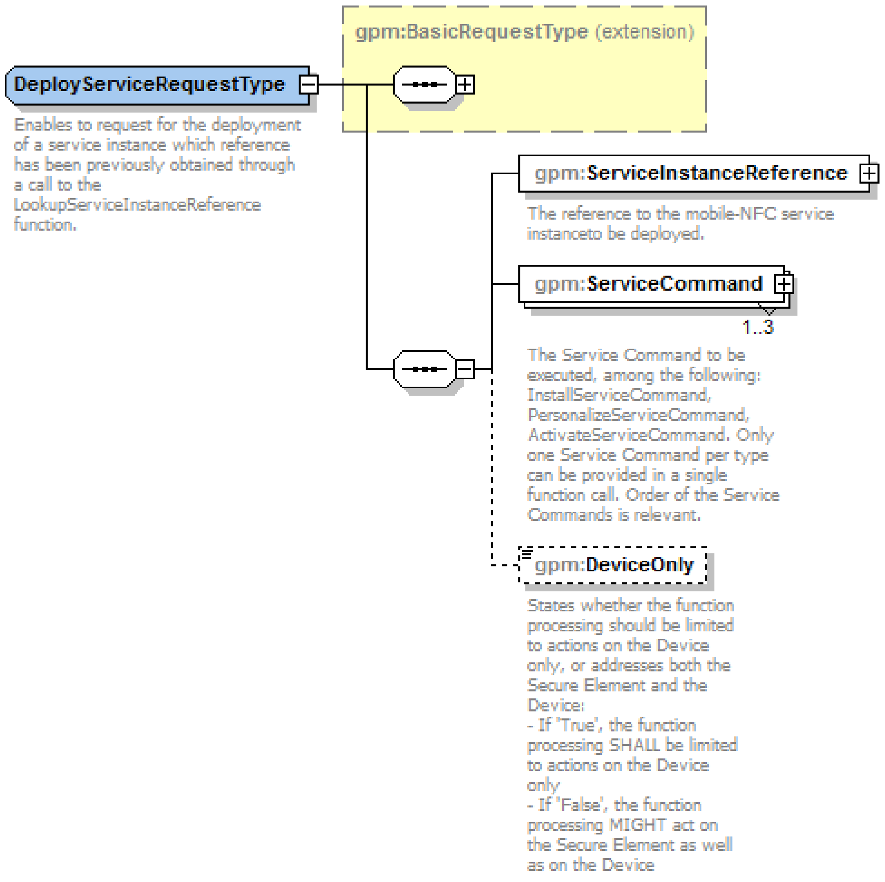

gpm:BasicRequestType (extension)
DeployServiceRequestType
Enables to request for the deployment of a service instance which reference has been previously obtained through a call to the LookupServiceInstanceReference function.
gpm:ServiceInstanceReference
The reference to the mobile-NFC service instanceto be deployed.
gpm:ServiceCommand
1..3
The Service Command to be executed, among the following: InstallServiceCommand, PersonalizeServiceCommand, ActivateServiceCommand. Only one Service Command per type can be provided in a single function call. Order of the Service Commands is relevant.
gpm:DeviceOnly
States whether the function processing should be limited to actions on the Device only, or addresses both the Secure Element and the Device:
- If 'True', the function processing SHALL be limited to actions on the Device only
- If 'False', the function processing MIGHT act on the Secure Element as well as on the Device

Figure 4-52: `DeployServiceRequestType` type

Where:

- The `ServiceInstanceReference` element is of type `ServiceInstanceReferenceType` (see section 4.5.1.2).
- The `ServiceCommand` element is of type `ServiceCommandType` (see section 4.5.3.1.5.1).

The value of the `GPHeader.Type` associated to this element SHALL be "DeployServiceRequest".

The output data of the `DeployServiceReference` function defined in section 3.3.2.5.2 SHALL be mapped to the `DeployServiceResponse` element that is of type `DeployServiceResponseType` described in the following figure:

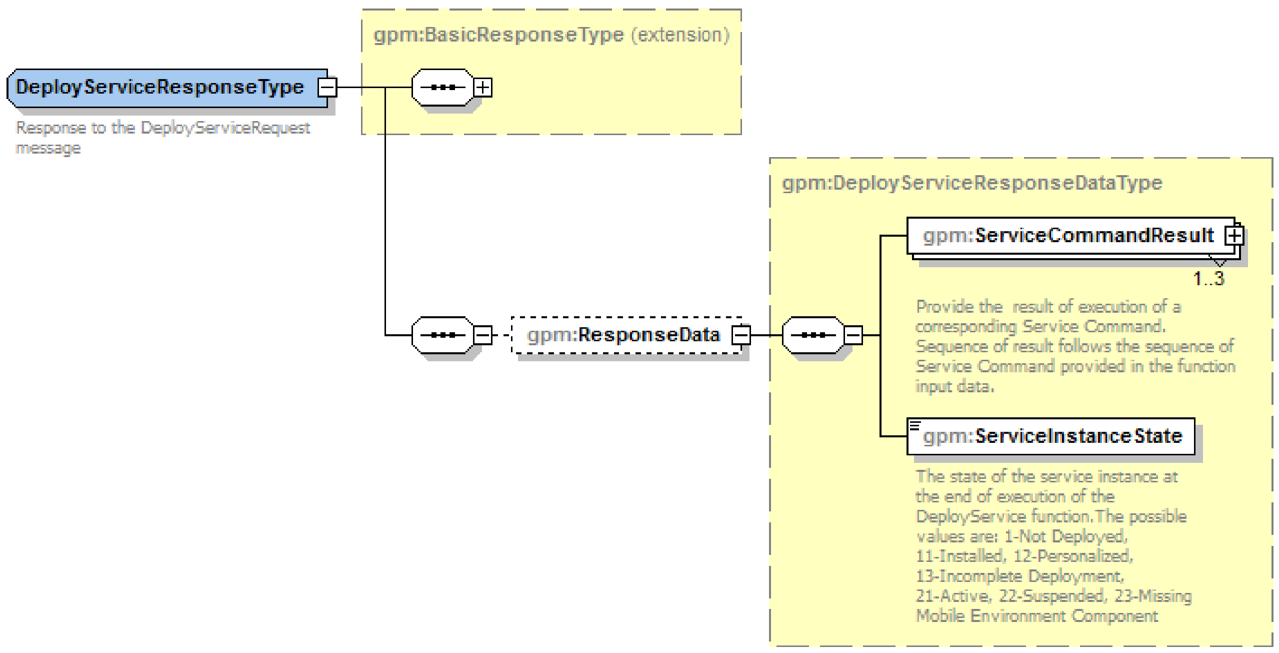

DeployServiceResponseType
Response to the DeployServiceRequest message
gpm:BasicResponseType (extension)
gpm:ResponseData
gpm:DeployServiceResponseDataType
gpm:ServiceCommandResult
1..3
Provide the result of execution of a corresponding Service Command. Sequence of result follows the sequence of Service Command provided in the function input data.
gpm:ServiceInstanceState
The state of the service instance at the end of execution of the DeployService function.The possible values are: 1-Not Deployed, 11-Installed, 12-Personalized, 13-Incomplete Deployment, 21-Active, 22-Suspended, 23-Missing Mobile Environment Component

Figure 4-53: `DeployServiceResponseType` type

Where:

- The `ServiceCommandResult` element is of type `ServiceCommandResultType` (see section 4.5.3.1.5.2).

The value of the `GPHeader.Type` associated to this element SHALL be "DeployServiceResponse".

4.5.3.1.5.1 Service Commands

The `ServiceCommandType` type acts as a base abstract type for the possible Service Commands defined in section 3.3.2.5.3. Each concrete Service Command extends this `ServiceCommandType` type.

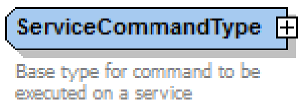


Figure 4-54: `ServiceCommandType` type

The `InstallServiceCommand` Service Command defined in section 3.3.2.5.3.1 SHALL be mapped to the `InstallServiceCommandType` type described in the following figure. This `InstallServiceCommandType` type extends the `ServiceCommandType` type.

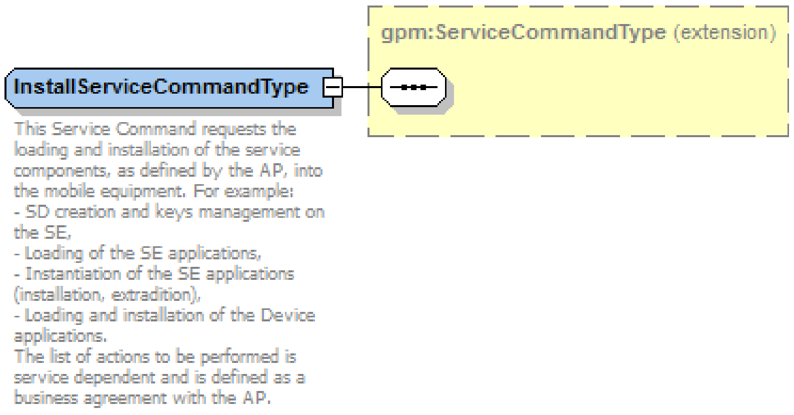

gpm:ServiceCommandType (extension)
InstallServiceCommandType
This Service Command requests the loading and installation of the service components, as defined by the AP, into the mobile equipment. For example:
- SD creation and keys management on the SE,
- Loading of the SE applications,
- Instantiation of the SE applications (installation, extradition),
- Loading and installation of the Device applications.
The list of actions to be performed is service dependent and is defined as a business agreement with the AP.

Figure 4-55: `InstallServiceCommandType` type

The `PersonalizeServiceCommand` Service Command defined in section 3.3.2.5.3.2 SHALL be mapped to the `PersonalizeServiceCommandType` type described in the following figure. This `PersonalizeServiceCommandType` type extends the `ServiceCommandType` type.

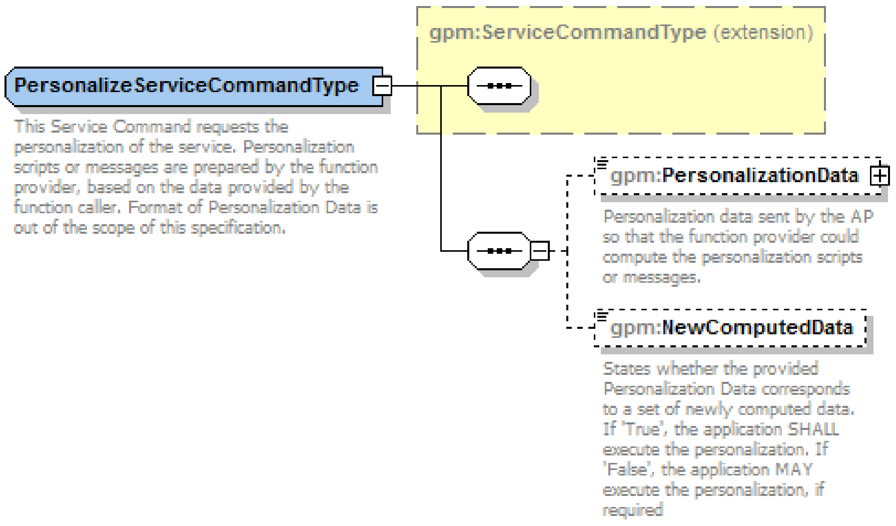

gpm:ServiceCommandType (extension)
PersonalizeServiceCommandType
This Service Command requests the personalization of the service. Personalization scripts or messages are prepared by the function provider, based on the data provided by the function caller. Format of Personalization Data is out of the scope of this specification.
gpm:PersonalizationData
Personalization data sent by the AP so that the function provider could compute the personalization scripts or messages.
gpm:NewComputedData
States whether the provided Personalization Data corresponds to a set of newly computed data. If 'True', the application SHALL execute the personalization. If 'False', the application MAY execute the personalization, if required

Figure 4-56: `PersonalizeServiceCommandType` type

Where:

- The `PersonalizationData` element is of type `xsd:anyType`.

The `ActivateServiceCommand` Service Command defined in section 3.3.2.5.3.3 SHALL be mapped to the `ActivateServiceCommandType` type described in the following figure. This `ActivateServiceCommandType` type extends the `ServiceCommandType` type.

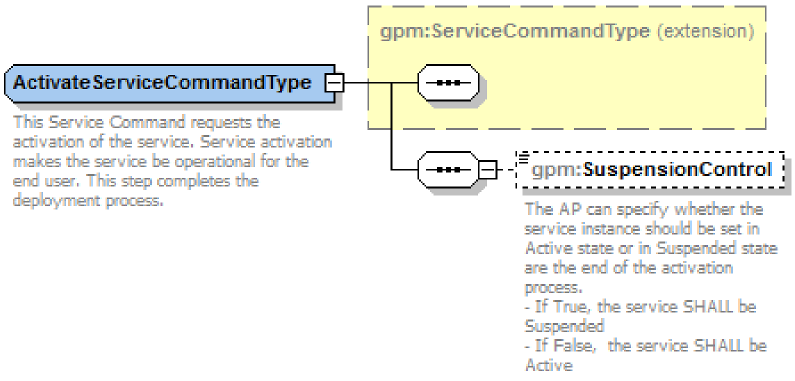

gpm:ServiceCommandType (extension)
ActivateServiceCommandType
This Service Command requests the activation of the service. Service activation makes the service be operational for the end user. This step completes the deployment process.
gpm:SuspensionControl
The AP can specify whether the service instance should be set in Active state or in Suspended state are the end of the activation process.
- If True, the service SHALL be Suspended
- If False, the service SHALL be Active

Figure 4-57: `ActivateServiceCommandType` type

4.5.3.1.5.2 Service Commands Results

The `ServiceCommandResult` defined in section 3.3.2.5.2 SHALL be mapped to the `ServiceCommandResultType` type described in the following figure:

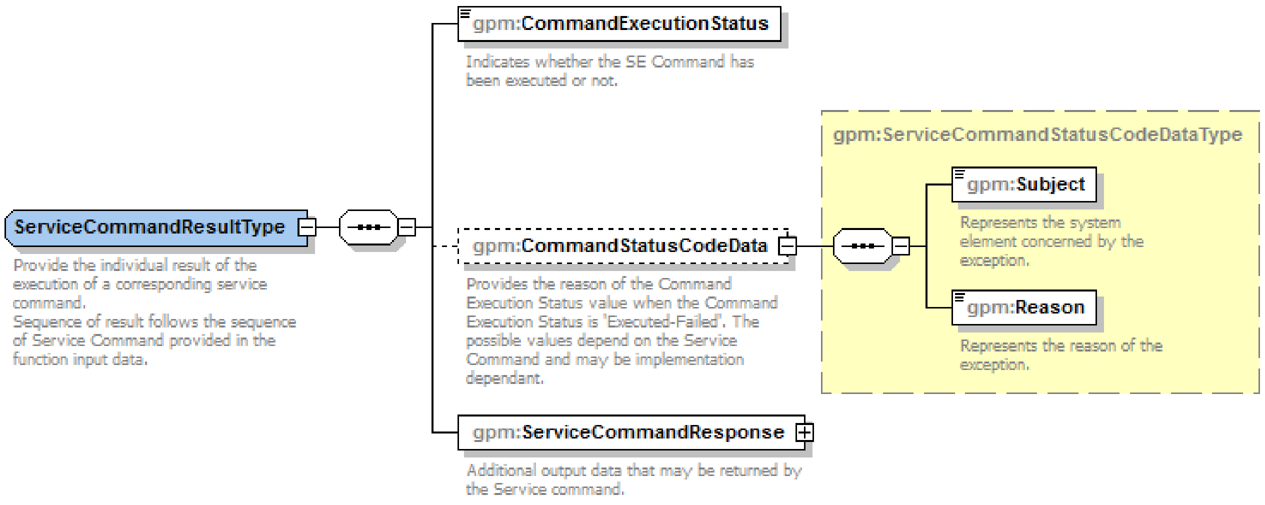

gpm:CommandExecutionStatus
Indicates whether the SE Command has been executed or not.
gpm:ServiceCommandStatusCodeDataType
gpm:Subject
Represents the system element concerned by the exception.
ServiceCommandResultType
Provide the individual result of the execution of a corresponding service command.
Sequence of result follows the sequence of Service Command provided in the function input data.
gpm:CommandStatusCodeData
Provides the reason of the Command Execution Status value when the Command Execution Status is 'Executed-Failed'. The possible values depend on the Service Command and may be implementation dependant.
gpm:Reason
Represents the reason of the exception.
gpm:ServiceCommandResponse
Additional output data that may be returned by the Service command.

Figure 4-58: `ServiceCommandResultType` type

Where:

- The `CommandStatusCodeData` is an element of type `ServiceCommandStatusCodeDataType`.
- The `ServiceCommandResponse` is an element of type `ServiceCommandResponseType`. This `ServiceCommandResponseType` type acts as a base type for the possible Service Command responses defined in section 3.3.2.5.3. Each concrete Service Command response extends this `ServiceCommandResponseType` type.

The `InstallServiceCommandResponse` Service Command defined in section 3.3.2.5.3.1 SHALL be mapped to the `InstallServiceCommandResponseType` type described in the following figure. This `InstallServiceCommandResponseType` type extends the `ServiceCommandResponseType` type.

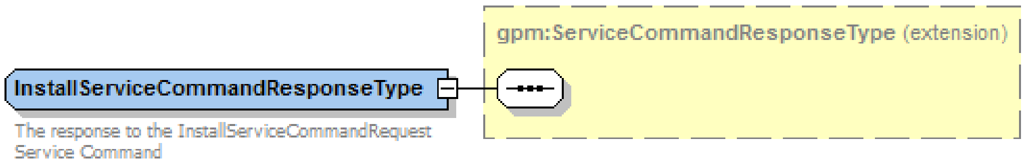

gpm:ServiceCommandResponseType (extension)
InstallServiceCommandResponseType
The response to the InstallServiceCommandRequest Service Command

Figure 4-59: `InstallServiceCommandResponseType` type

The `PersonalizeServiceCommandResponse` Service Command defined in section 3.3.2.5.3.2 SHALL be mapped to the `PersonalizeServiceCommandResponseType` type described in the following figure. This `PersonalizeServiceCommandResponseType` type extends the `ServiceCommandResponseType` type.

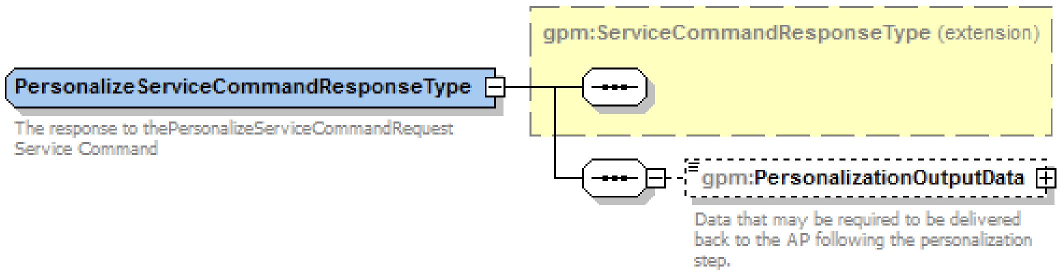

gpm:ServiceCommandResponseType (extension)
PersonalizeServiceCommandResponseType
The response to thePersonalizeServiceCommandRequest Service Command
gpm:PersonalizationOutputData
Data that may be required to be delivered back to the AP following the personalization step.

Figure 4-60: `PersonalizeServiceCommandResponseType` type

Where:

- The `PersonalizationOutputData` element is of type `xsd:anyType`.

The `ActivateServiceCommandResponse` Service Command defined in section 3.3.2.5.3.3 SHALL be mapped to the `ActivateServiceCommandResponseType` type described in the following figure. This `ActivateServiceCommandResponseType` type extends the `ServiceCommandResponseType` type.

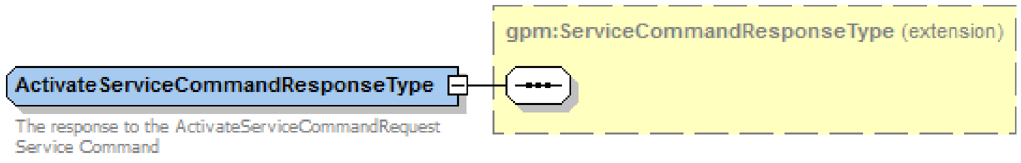

gpm:ServiceCommandResponseType (extension)
ActivateServiceCommandResponseType
The response to the ActivateServiceCommandRequest Service Command

Figure 4-61: `ActivateServiceCommandResponseType` type

4.5.3.1.6 The "UpgradeService" Function

The input data for the UpgradeService function defined in section 3.3.2.6 SHALL be mapped to the UpgradeServiceRequest element that is of type UpgradeRequestType described in the following figure:

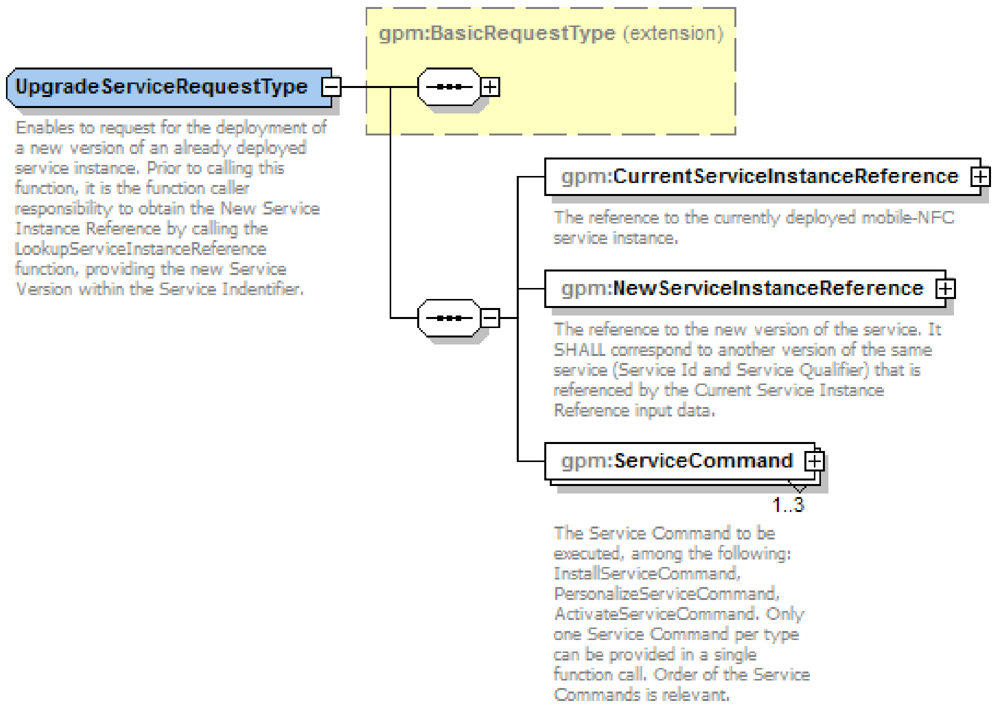

gpm:BasicRequestType (extension)
UpgradeServiceRequestType
Enables to request for the deployment of a new version of an already deployed service instance. Prior to calling this function, it is the function caller responsibility to obtain the New Service Instance Reference by calling the LookupServiceInstanceReference function, providing the new Service Version within the Service Indentifier.
gpm:CurrentServiceInstanceReference
The reference to the currently deployed mobile-NFC service instance.
gpm:NewServiceInstanceReference
The reference to the new version of the service. It SHALL correspond to another version of the same service (Service Id and Service Qualifier) that is referenced by the Current Service Instance Reference input data.
gpm:ServiceCommand
1..3
The Service Command to be executed, among the following: InstallServiceCommand, PersonalizeServiceCommand, ActivateServiceCommand. Only one Service Command per type can be provided in a single function call. Order of the Service Commands is relevant.

Figure 4-62: `UpgradeServiceRequestType` type

Where:

- The `CurrentServiceInstanceReference` and the `NewServiceInstanceReference` elements are of type `ServiceInstanceReferenceType` (see section 4.5.1.2).
- The `ServiceCommand` element is of type `ServiceCommandType` (see section 4.5.3.1.5.1).

The value of the `GPHeader.Type` associated to this element SHALL be "UpgradeServiceRequest".

The output data of the `UpgradeServiceReference` function defined in section 3.3.2.6 SHALL be mapped to the `UpgradeServiceResponse` element that is of type `UpgradeServiceResponseType` described in the following figure:

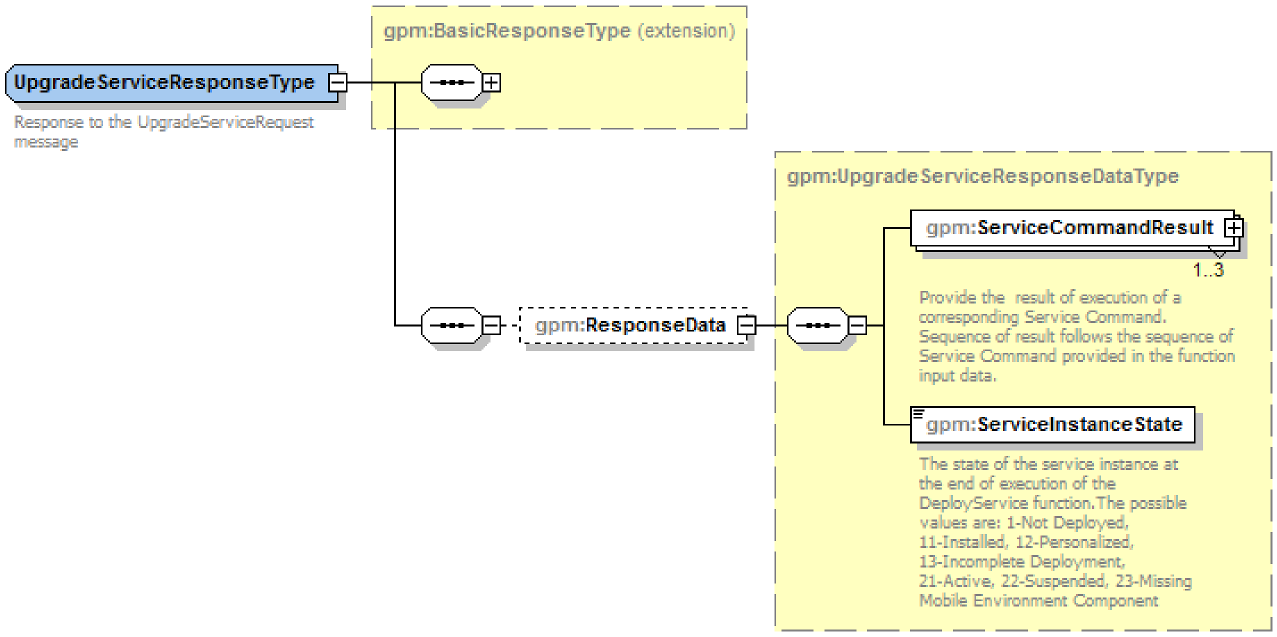

gpm:BasicResponseType (extension)
UpgradeServiceResponseType
Response to the UpgradeServiceRequest message
gpm:UpgradeServiceResponseDataType
gpm:ServiceCommandResult
1..3
Provide the result of execution of a corresponding Service Command. Sequence of result follows the sequence of Service Command provided in the function input data.
gpm:ResponseData
gpm:ServiceInstanceState
The state of the service instance at the end of execution of the DeployService function.The possible values are: 1-Not Deployed, 11-Installed, 12-Personalized, 13-Incomplete Deployment, 21-Active, 22-Suspended, 23-Missing Mobile Environment Component

Figure 4-63: `UpgradeServiceResponseType` type

Where:

- The `ServiceCommandResult` element is of type `ServiceCommandResultType` (see section 4.5.3.1.5.2).

The value of the `GPHeader.Type` associated to this element SHALL be "UpgradeServiceResponse".

4.5.3.1.7 The "ExchangeServiceData" Function

The input data for the `ExchangeServiceData` function defined in section 3.3.2.7 SHALL be mapped to the `ExchangeServiceDataRequest` element that is of type `ExchangeServiceDataRequestType` described in the following figure:

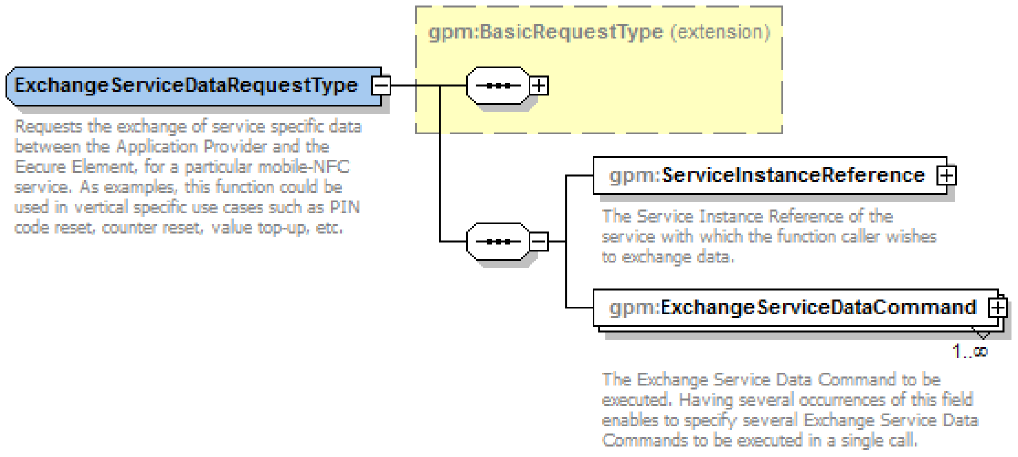

gpm:BasicRequestType (extension)
ExchangeServiceDataRequestType
Requests the exchange of service specific data between the Application Provider and the Secure Element, for a particular mobile-NFC service. As examples, this function could be used in vertical specific use cases such as PIN code reset, counter reset, value top-up, etc.
gpm:ServiceInstanceReference
The Service Instance Reference of the service with which the function caller wishes to exchange data.
gpm:ExchangeServiceDataCommand
1..∞
The Exchange Service Data Command to be executed. Having several occurrences of this field enables to specify several Exchange Service Data Commands to be executed in a single call.

Figure 4-64: `ExchangeServiceDataRequestType` type

Where:

- The `ServiceInstanceReference` element is of type `ServiceInstanceReferenceType` (see section 4.5.1.2).
- The `ExchangeServiceDataCommand` element is of type `ExchangeServiceDataCommandType`:

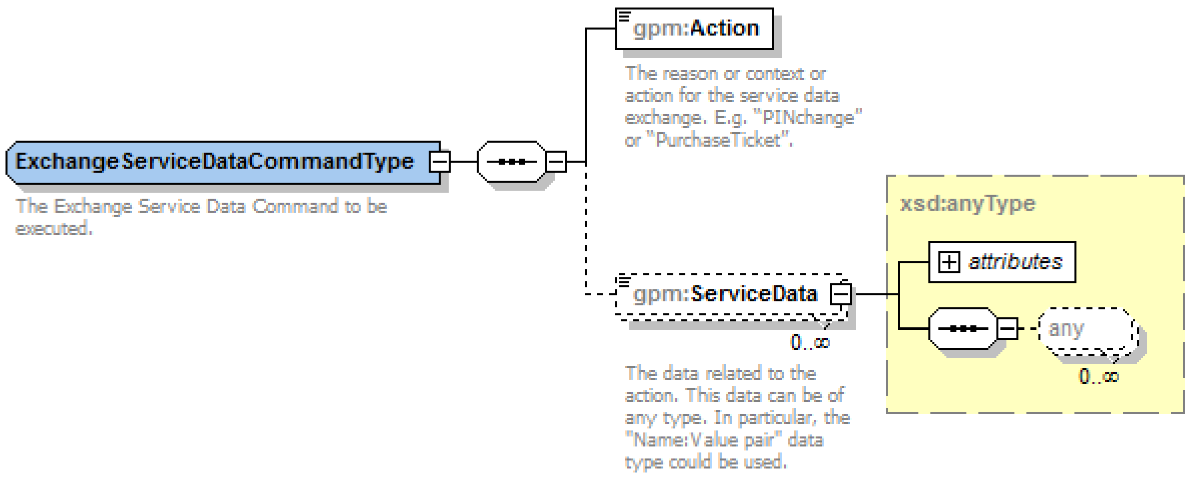

gpm:Action
The reason or context or action for the service data exchange. E.g. "PINchange" or "PurchaseTicket".
ExchangeServiceDataCommandType
The Exchange Service Data Command to be executed.
xsd:anyType
attributes
gpm:ServiceData
0..∞
any
0..∞
The data related to the action. This data can be of any type. In particular, the "Name:Value pair" data type could be used.

Figure 4-65: `ExchangeServiceDataCommandType` type

Where:

  o The `ServiceData` element is of type `xsd:anyType`.

The value of the `GPHeader.Type` associated to this element SHALL be "ExchangeServiceDataRequest".

The output data of the `ExchangeServiceData` function defined in section 3.3.2.7 SHALL be mapped to the `ExchangeServiceDataResponse` element that is of type `ExchangeServiceDataResponseType` described in the following figure:

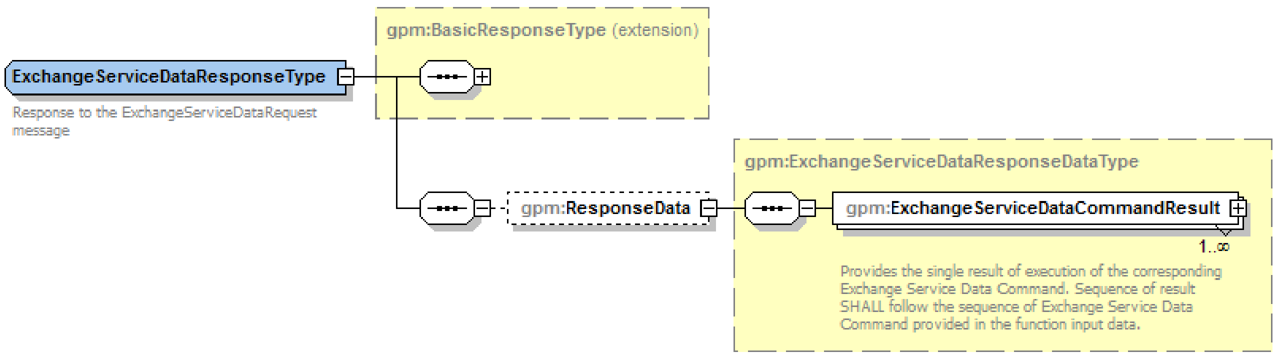

gpm:BasicResponseType (extension)
ExchangeServiceDataResponseType
Response to the ExchangeServiceDataRequest message
gpm:ResponseData
gpm:ExchangeServiceDataResponseDataType
gpm:ExchangeServiceDataCommandResult
1..∞
Provides the single result of execution of the corresponding Exchange Service Data Command. Sequence of result SHALL follow the sequence of Exchange Service Data Command provided in the function input data.

Figure 4-66: `ExchangeServiceDataResponseType` type

Where:

- The `ExchangeServiceDataCommandResult` element is of type `ExchangeServiceDataCommandResultType`:

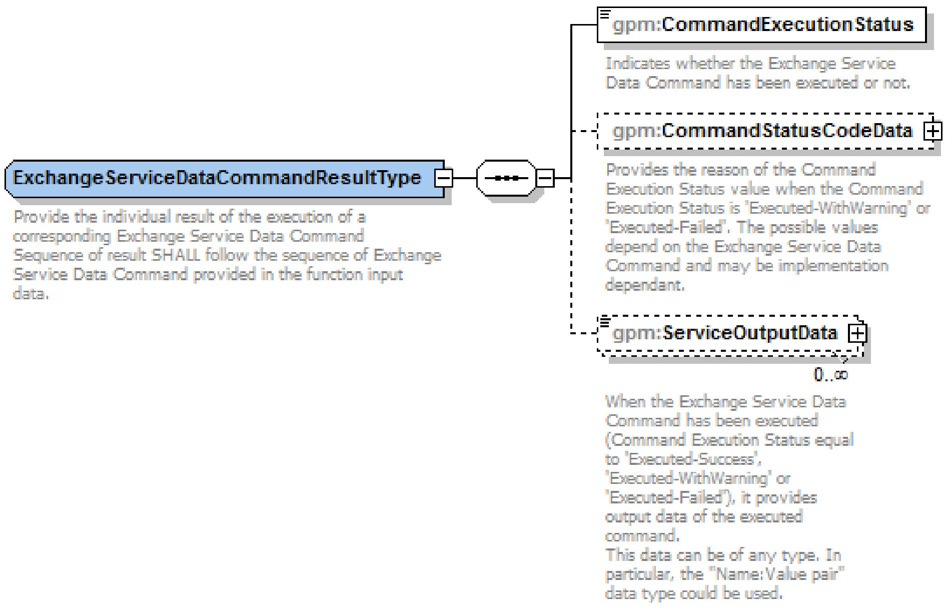

gpm:CommandExecutionStatus
Indicates whether the Exchange Service Data Command has been executed or not.
gpm:CommandStatusCodeData
Provides the reason of the Command Execution Status value when the Command Execution Status is 'Executed-WithWarning' or 'Executed-Failed'. The possible values depend on the Exchange Service Data Command and may be implementation dependant.
ExchangeServiceDataCommandResultType
Provide the individual result of the execution of a corresponding Exchange Service Data Command Sequence of result SHALL follow the sequence of Exchange Service Data Command provided in the function input data.
gpm:ServiceOutputData
0..∞
When the Exchange Service Data Command has been executed (Command Execution Status equal to 'Executed-Success', 'Executed-WithWarning' or 'Executed-Failed'), it provides output data of the executed command.
This data can be of any type. In particular, the "Name:Value pair" data type could be used.

Figure 4-67: `ExchangeServiceDataCommandResultType` type

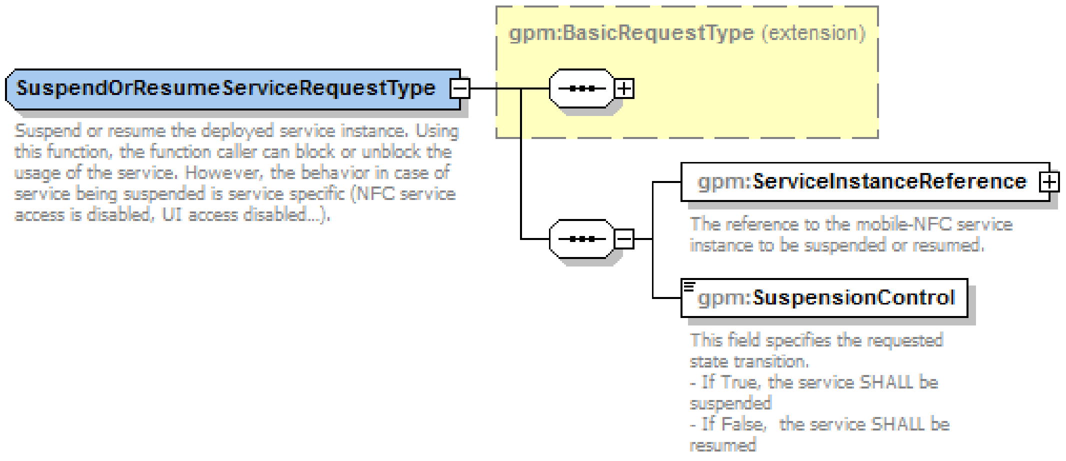
Where:

- o The `CommandStatusCodeData` is an element of type `ExchangeServiceDataCommandStatusCodeDataType`.
- o The `ServiceOutputData` element is of type `xsd:anyType`.

The value of the `GPHeader.Type` associated to this element SHALL be "ExchangeServiceDataResponse".

4.5.3.1.8 The "SuspendOrResumeService" Function

The input data for the `SuspendOrResumeService` function defined in section 3.3.2.8 SHALL be mapped to the `SuspendOrResumeServiceRequest` element that is of type `SuspendOrResumeServiceRequestType` described in the following figure:



gpm:BasicRequestType (extension)
SuspendOrResumeServiceRequestType
Suspend or resume the deployed service instance. Using this function, the function caller can block or unblock the usage of the service. However, the behavior in case of service being suspended is service specific (NFC service access is disabled, UI access disabled...).
gpm:ServiceInstanceReference
The reference to the mobile-NFC service instance to be suspended or resumed.
gpm:SuspensionControl
This field specifies the requested state transition.
- If True, the service SHALL be suspended
- If False, the service SHALL be resumed

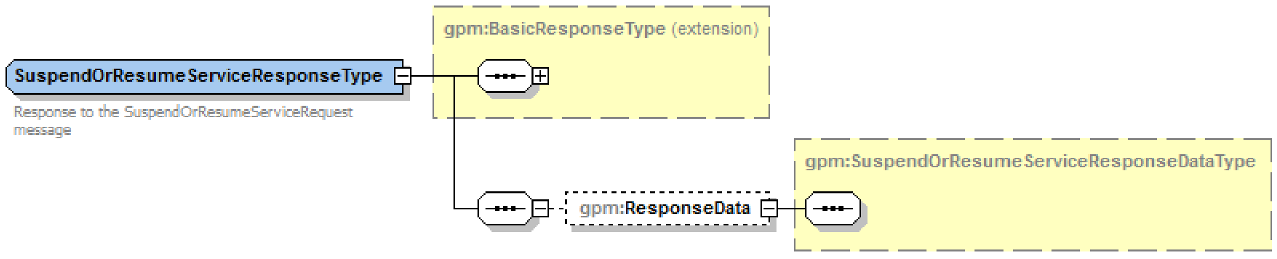
Figure 4-68: `SuspendOrResumeServiceRequestType` type

Where:

- The `ServiceInstanceReference` element is of type `ServiceInstanceReferenceType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "SuspendOrResumeServiceRequest".

The output data of the `SuspendOrResumeServiceReference` function defined in section 3.3.2.8 SHALL be mapped to the `SuspendOrResumeServiceResponse` element that is of type `SuspendOrResumeServiceResponseType` described in the following figure:



gpm:BasicResponseType (extension)
SuspendOrResumeServiceResponseType
Response to the SuspendOrResumeServiceRequest message
gpm:SuspendOrResumeServiceResponseDataType
gpm:ResponseData

Figure 4-69: `SuspendOrResumeServiceResponseType` type

The value of the `GPHeader.Type` associated to this element SHALL be "SuspendOrResumeServiceResponse".

4.5.3.1.9 The "TerminateService" Function

The input data for the `TerminateService` function defined in section 3.3.2.8 SHALL be mapped to the `TerminateServiceRequest` element that is of type `TerminateServiceRequestType` described in the following figure:

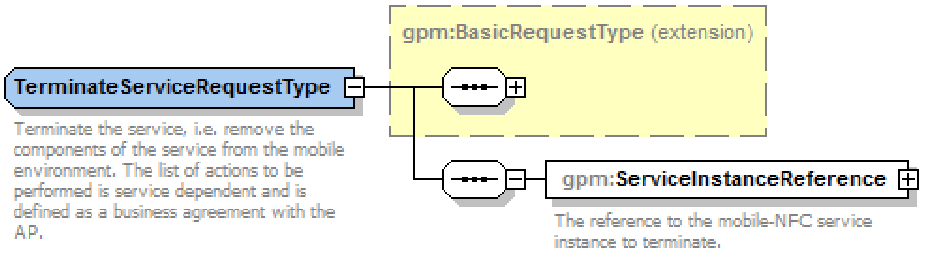

gpm:BasicRequestType (extension)
TerminateServiceRequestType
Terminate the service, i.e. remove the components of the service from the mobile environment. The list of actions to be performed is service dependent and is defined as a business agreement with the AP.
gpm:ServiceInstanceReference
The reference to the mobile-NFC service instance to terminate.

Figure 4-70: `TerminateServiceRequestType` type

Where:

- The `ServiceInstanceReference` element is of type `ServiceInstanceReferenceType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "TerminateServiceRequest".

The output data of the `TerminateServiceReference` function defined in section 3.3.2.8 SHALL be mapped to the `TerminateServiceResponse` element that is of type `TerminateServiceResponseType` described in the following figure:

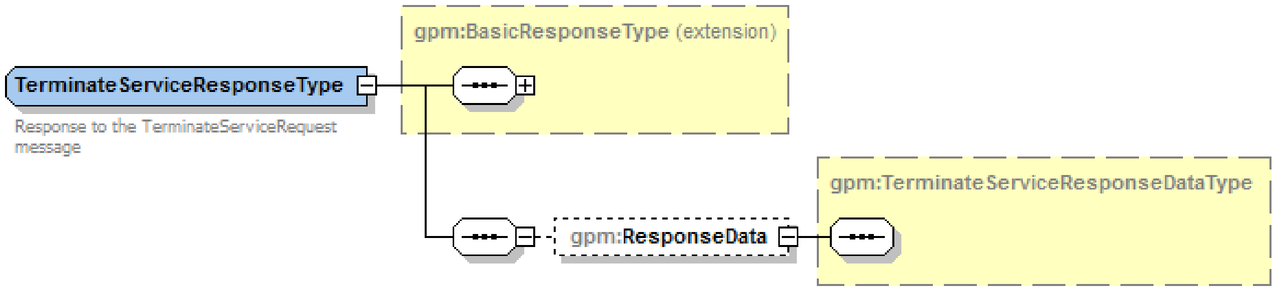

gpm:BasicResponseType (extension)
TerminateServiceResponseType
Response to the TerminateServiceRequest message
gpm:TerminateServiceResponseDataType
gpm:ResponseData

Figure 4-71: `TerminateServiceResponseType` type

The value of the `GPHeader.Type` associated to this element SHALL be "TerminateServiceResponse".

4.5.3.2 Service Environment Change

4.5.3.2.1 The "HandleServiceEnvironmentChangeNotification" Notification Function

The input data for the `HandleServiceEnvironmentChangeNotification` notification function defined in section 3.3.3.1 SHALL be mapped to the `HandleServiceEnvironmentChangeNotificationRequest` element that is of type `HandleServiceEnvironmentChangeNotificationRequestType` described in the following figure:

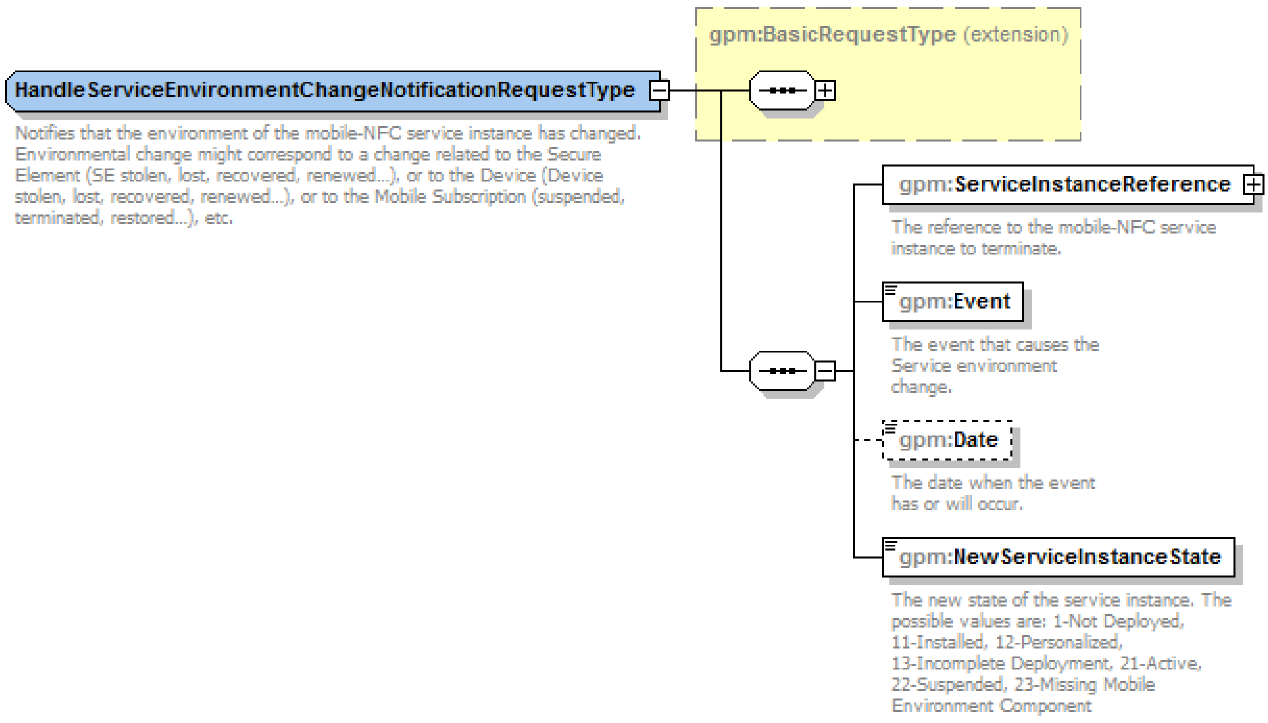

gpm:BasicRequestType (extension)
HandleServiceEnvironmentChangeNotificationRequestType
Notifies that the environment of the mobile-NFC service instance has changed. Environmental change might correspond to a change related to the Secure Element (SE stolen, lost, recovered, renewed...), or to the Device (Device stolen, lost, recovered, renewed...), or to the Mobile Subscription (suspended, terminated, restored...), etc.
gpm:ServiceInstanceReference
The reference to the mobile-NFC service instance to terminate.
gpm:Event
The event that causes the Service environment change.
gpm:Date
The date when the event has or will occur.
gpm:NewServiceInstanceState
The new state of the service instance. The possible values are: 1-Not Deployed, 11-Installed, 12-Personalized, 13-Incomplete Deployment, 21-Active, 22-Suspended, 23-Missing Mobile Environment Component

Figure 4-72: `HandleServiceEnvironmentChangeNotificationRequestType` type

Where:

- The `ServiceInstanceReference` element is of type `ServiceEnvironmentChangeEventType` (see section 4.5.1.2).
- The `Event` is an enumeration of the possible events (`ServiceEnvironmentChangeEventType`).

The value of the `GPHeader.Type` associated to this element SHALL be "HandleServiceEnvironmentChangeNotificationRequest".

4.5.3.2.2 The "HandleActionDoneOnServiceNotification" Notification Function

The input data for the `HandleActionDoneOnServiceNotification` notification function defined in section 3.3.3.2 SHALL be mapped to the `HandleActionDoneOnServiceNotificationRequest` element that is of type `HandleActionDoneOnServiceNotificationRequestType` described in the following figure:

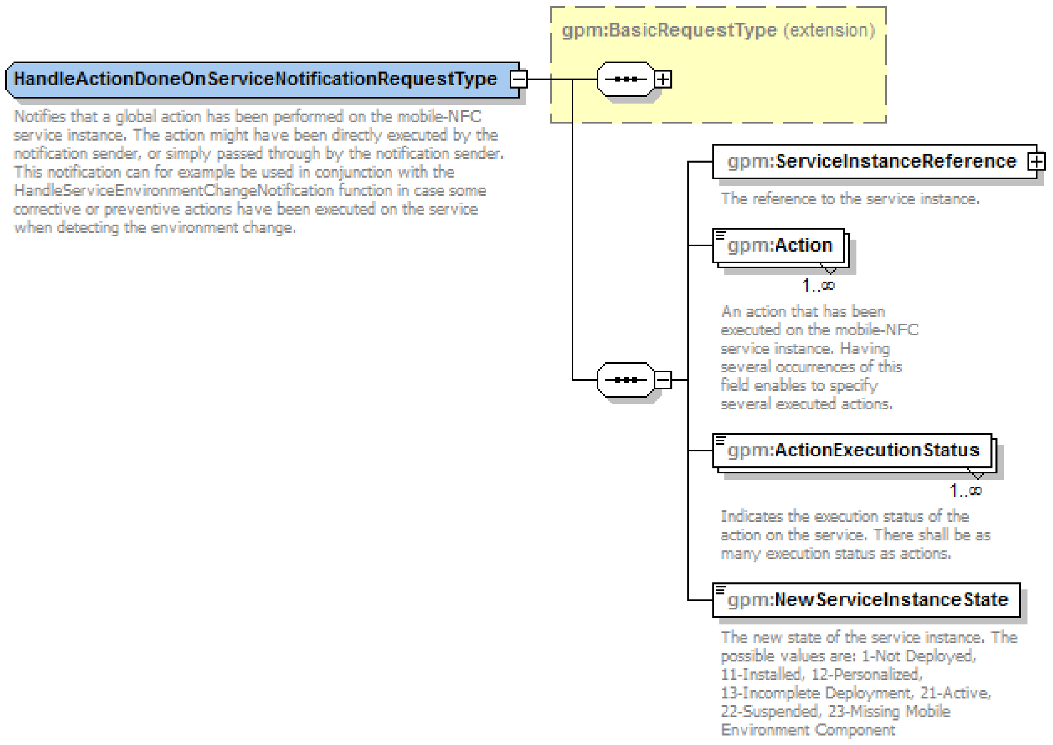

gpm:BasicRequestType (extension)
HandleActionDoneOnServiceNotificationRequestType
Notifies that a global action has been performed on the mobile-NFC service instance. The action might have been directly executed by the notification sender, or simply passed through by the notification sender. This notification can for example be used in conjunction with the HandleServiceEnvironmentChangeNotification function in case some corrective or preventive actions have been executed on the service when detecting the environment change.
gpm:ServiceInstanceReference
The reference to the service instance.
gpm:Action
1..∞
An action that has been executed on the mobile-NFC service instance. Having several occurrences of this field enables to specify several executed actions.
gpm:ActionExecutionStatus
1..∞
Indicates the execution status of the action on the service. There shall be as many execution status as actions.
gpm:NewServiceInstanceState
The new state of the service instance. The possible values are: 1-Not Deployed, 11-Installed, 12-Personalized, 13-Incomplete Deployment, 21-Active, 22-Suspended, 23-Missing Mobile Environment Component

Figure 4-73: `HandleActionDoneOnServiceNotificationRequestType` type

Where:

- The `ServiceInstanceReference` element is of type `ServiceInstanceReferenceType` (see section 4.5.1.2).
- The `Action` is an enumeration of the possible actions (`ActionOnServiceType`).
- The `ActionExecutionStatus` is an enumeration of the possible actions (`ActionOnServiceExecutionStatusType`).

The value of the `GPHeader.Type` associated to this element SHALL be "HandleActionDoneOnServiceNotificationRequest".

4.5.4 Functions for CCCM Certificate Management

4.5.4.1 The "EnrollSSDOwnerCertificate" Function

The input data for the `EnrollSSDOwnerCertificate` function defined in section 3.4.2 SHALL be mapped to the `EnrollSSDOwnerCertificateRequest` element that is of type `EnrollSSDOwnerCertificateRequestType` described in the following figure:

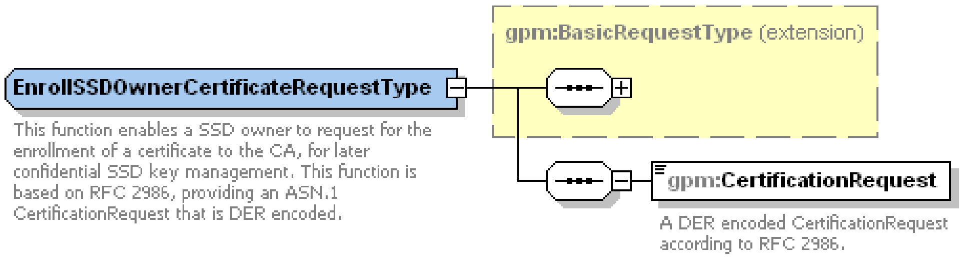

gpm:BasicRequestType (extension)
EnrollSSDOwnerCertificateRequestType
This function enables a SSD owner to request for the enrollment of a certificate to the CA, for later confidential SSD key management. This function is based on RFC 2986, providing an ASN.1 CertificationRequest that is DER encoded.
gpm:CertificationRequest
A DER encoded CertificationRequest according to RFC 2986.

Figure 4-74: `EnrollSSDOwnerCertificateRequestType` type

The value of the `GPHeader.Type` associated to this element SHALL be "EnrollSSDOwnerCertificateRequest".

The output data of the `EnrollSSDOwnerCertificate` function defined in section 3.4.2 SHALL be mapped to the `EnrollSSDOwnerCertificateResponse` element that is of type `EnrollSSDOwnerCertificateResponseType` described in the following figure:

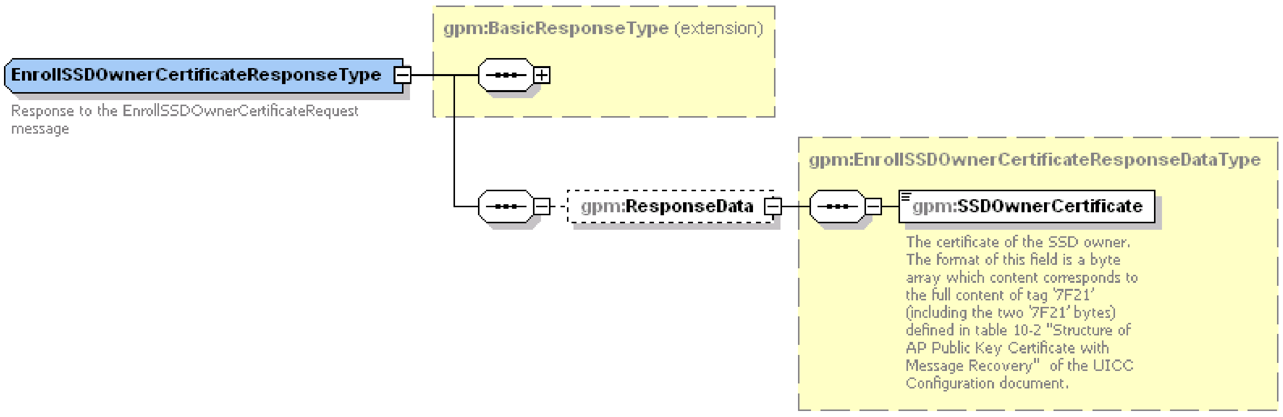

gpm:BasicResponseType (extension)
EnrollSSDOwnerCertificateResponseType
Response to the EnrollSSDOwnerCertificateRequest message
gpm:ResponseData
gpm:EnrollSSDOwnerCertificateResponseDataType
gpm:SSDOwnerCertificate
The certificate of the SSD owner. The format of this field is a byte array which content corresponds to the full content of tag '7F21' (including the two '7F21' bytes) defined in table 10-2 "Structure of AP Public Key Certificate with Message Recovery" of the UICC Configuration document.

Figure 4-75: `EnrollSSDOwnerCertificateResponseType` type

The value of the `GPHeader.Type` associated to this element SHALL be "EnrollSSDOwnerCertificateResponse".

4.5.4.2 The "AuditCAInformation" Function

Note that the `GetCAInformation` function was initially used for retrieving the CA information. It is now replaced by the `AuditCAInformation` function that provides the same feature but using an asynchronous Message Exchange Pattern (see section 5).

However, due to backward compatibility, the `GetCAInformationRequestType` (and not *`Audit`*`CAInformationRequestType`) and the `GetCAInformationResponseType` (and not *`Audit`*`CAInformationResponseType`) remains the message types to be used.

The input data for the `AuditCAInformation` function defined in section 3.4.1 SHALL be mapped to the `AuditCAInformationRequest` element that is of type `GetCAInformationRequestType` described in the following figure:

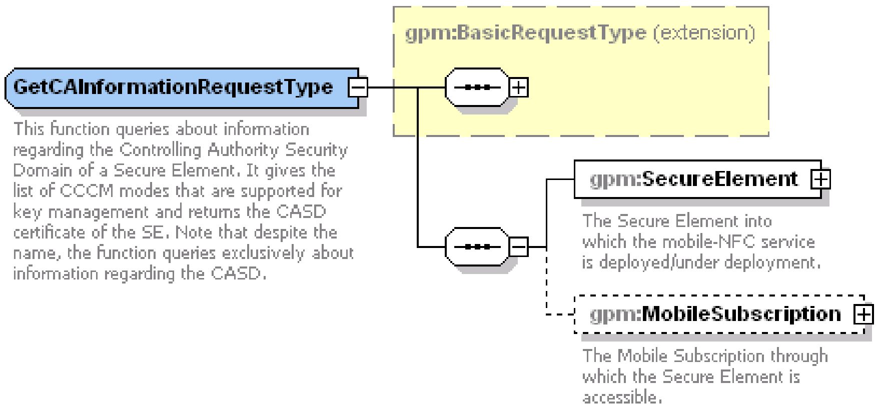

gpm:BasicRequestType (extension)
GetCAInformationRequestType
This function queries about information regarding the Controlling Authority Security Domain of a Secure Element. It gives the list of CCCM modes that are supported for key management and returns the CASD certificate of the SE. Note that despite the name, the function queries exclusively about information regarding the CASD.
gpm:SecureElement
The Secure Element into which the mobile-NFC service is deployed/under deployment.
gpm:MobileSubscription
The Mobile Subscription through which the Secure Element is accessible.

Figure 4-76: `GetCAInformationRequestType` type

Where:

- The `SecureElement` element is of type `SEIdentifierType` (see section 4.5.1.2).
- The `MobileSubscription` element is of type `MobileSubscriptionIdentifierType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "GetCAInformationRequest".

The output data of the `AuditCAInformation` function defined in section 3.4.1 SHALL be mapped to the `AuditCAInformationResponse` element that is of type `GetCAInformationResponseType` described in the following figure:

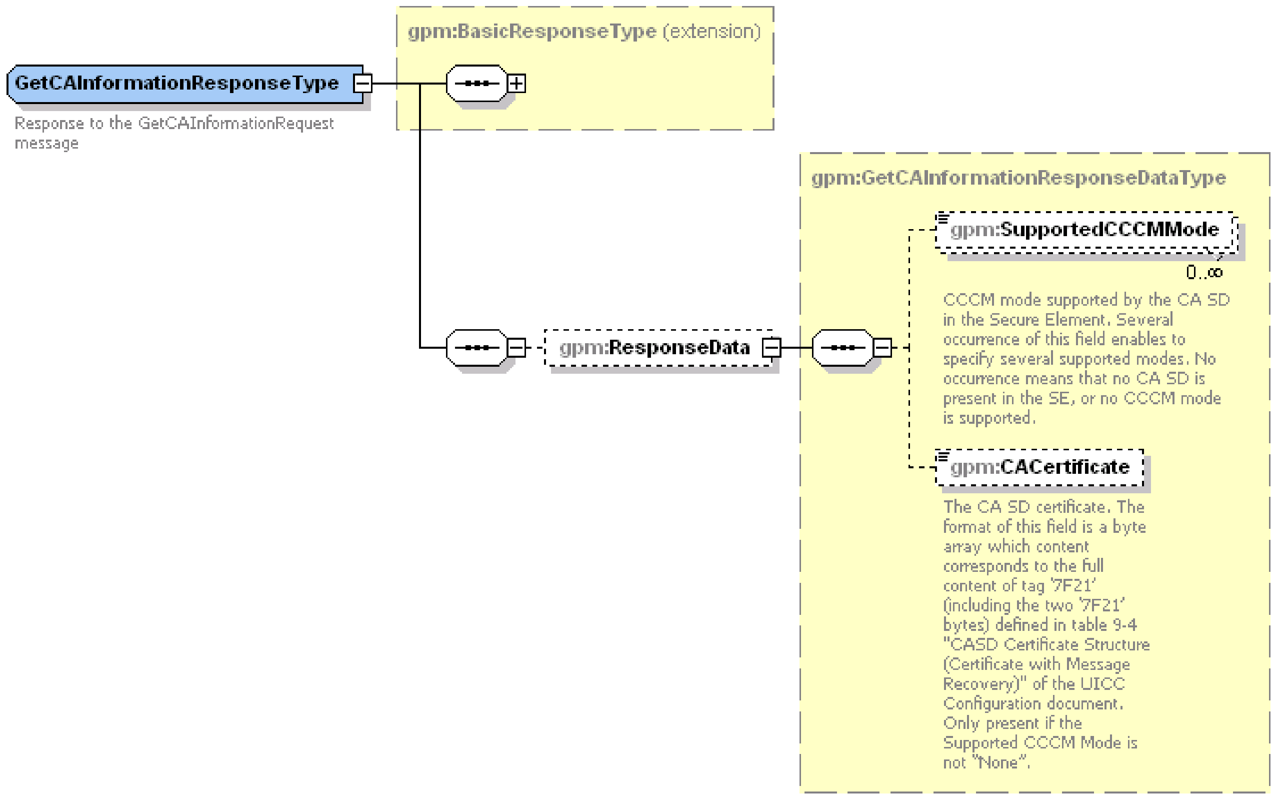

gpm:BasicResponseType (extension)
GetCAInformationResponseType
Response to the GetCAInformationRequest message
gpm:GetCAInformationResponseDataType
gpm:SupportedCCCMMode
0..∞
CCCM mode supported by the CA SD in the Secure Element. Several occurrence of this field enables to specify several supported modes. No occurrence means that no CA SD is present in the SE, or no CCCM mode is supported.
gpm:ResponseData
gpm:CACertificate
The CA SD certificate. The format of this field is a byte array which content corresponds to the full content of tag '7F21' (including the two '7F21' bytes) defined in table 9-4 "CASD Certificate Structure (Certificate with Message Recovery)" of the UICC Configuration document. Only present if the Supported CCCM Mode is not "None".

Figure 4-77: `GetCAInformationResponseType` type

Where:

- The `SupportedCCCMMode` is an enumeration of the possible Confidential Card Content Management scenario defined by [3] and [6] (`SupportedCCCMModeType`).

The value of the `GPHeader.Type` associated to this element SHALL be "GetCAInformationResponse".

Note that for backward compatibility, the `GetCAInformationRequest` element (of type `GetCAInformationRequestType`) and the `GetCAInformationResponse` element (of type `GetCAInformationResponseType`) SHALL also be defined.

4.5.5 Functions for the Secure Element Content Management

4.5.5.1 SE Card Content Management Functions for Simple Mode

The input data for the `SECommandsGenerationAndRemoteExecution` function defined in section 3.5.1.2 SHALL be mapped to the `SECommandsGenerationAndRemoteExecutionRequest` element that is of type `SECommandsGenerationAndRemoteExecutionRequestType` described in the following figure:

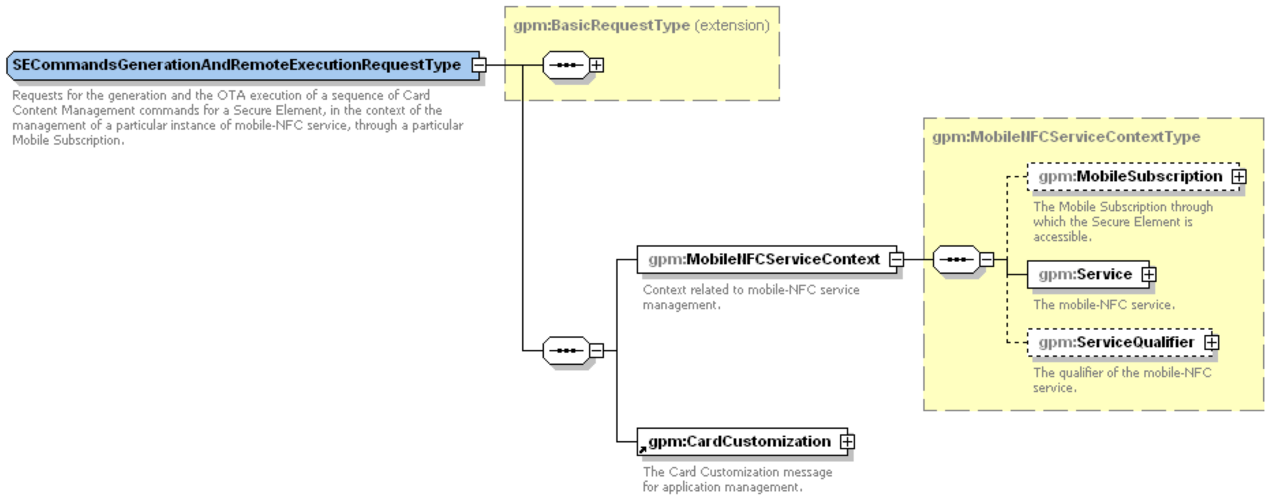

gpm:BasicRequestType (extension)
SECommandsGenerationAndRemoteExecutionRequestType
Requests for the generation and the OTA execution of a sequence of Card Content Management commands for a Secure Element, in the context of the management of a particular instance of mobile-NFC service, through a particular Mobile Subscription.
gpm:MobileNFCServiceContextType
gpm:MobileSubscription
The Mobile Subscription through which the Secure Element is accessible.
gpm:MobileNFCServiceContext
Context related to mobile-NFC service management.
gpm:Service
The mobile-NFC service.
gpm:ServiceQualifier
The qualifier of the mobile-NFC service.
gpm:CardCustomization
The Card Customization message for application management.

Figure 4-78: `SECommandsGenerationAndRemoteExecutionRequestType` type

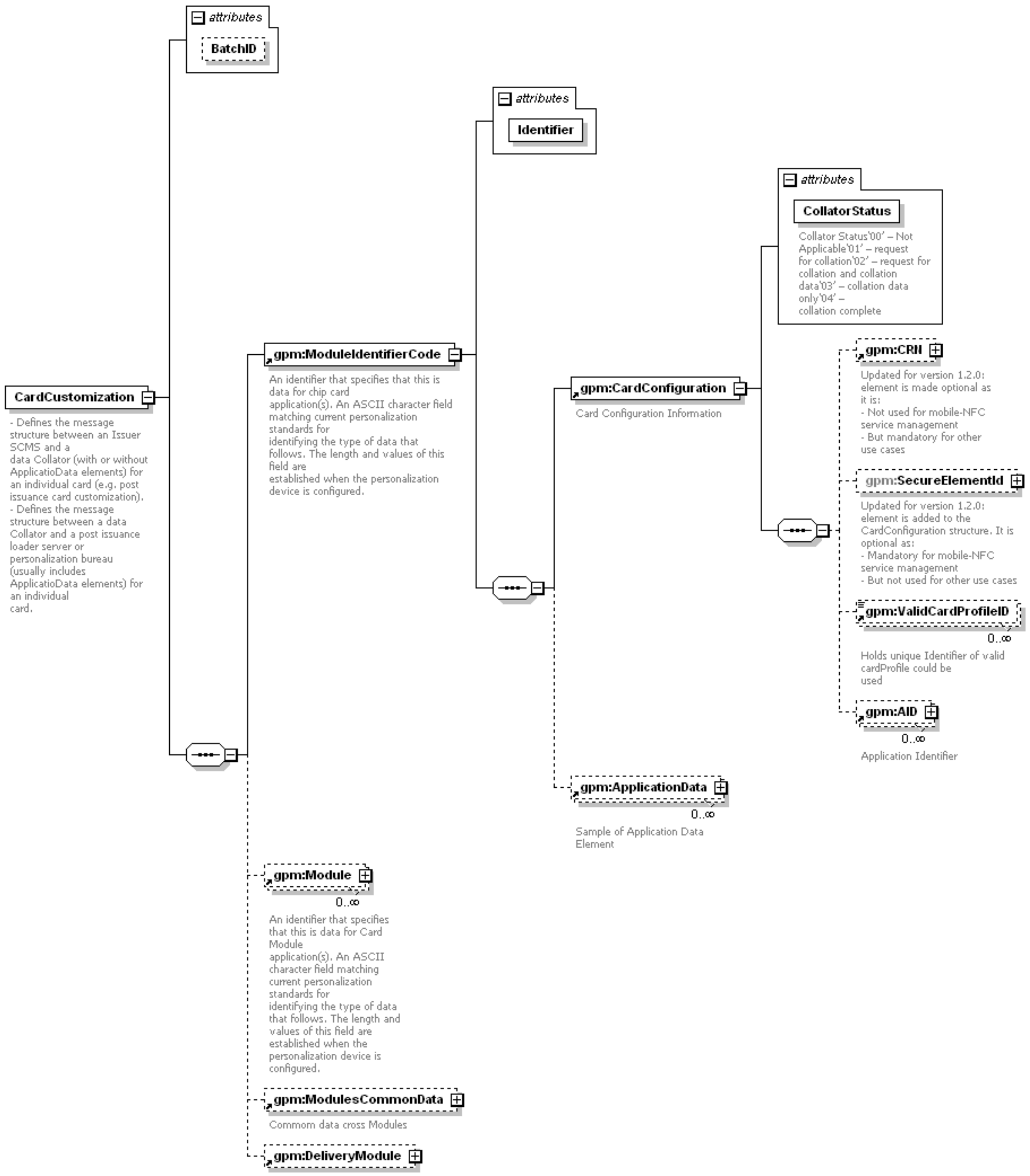
Where:

- The `MobileSubscription` element is of type `MobileSubscriptionIdentifierType` (see section 4.5.1.2).
- The `Service` element is of type `ServiceIdentifierType` (see section 4.5.1.2).
- The `ServiceQualifier` element is of type `ServiceQualifierType` (see section 4.5.1.2).
- The `CardCustomization` element is a message already existing in the GP System Messaging [2].

The value of the `GPHeader.Type` associated to this element SHALL be "SECommandsGenerationAndRemoteExecutionRequest".

The `CardCustomization` message defined in the version 2.0.0 of the XML schema is updated compared to version 1.1.0. This structure is used to send CCM command (in SE Command structure) as explained below.

Here is the new `CardCustomization` element structure:



attributes
BatchID
CardCustomization
- Defines the message structure between an Issuer SCMS and a data Collator (with or without ApplicatioData elements) for an individual card (e.g. post issuance card customization). - Defines the message structure between a data Collator and a post issuance loader server or personalization bureau (usually includes ApplicatioData elements) for an individual card.
gpm:ModuleIdentifierCode
An identifier that specifies that this is data for chip card application(s). An ASCII character field matching current personalization standards for identifying the type of data that follows. The length and values of this field are established when the personalization device is configured.
attributes
Identifier
gpm:CardConfiguration
Card Configuration Information
attributes
CollatorStatus
Collator Status'00' – Not Applicable'01' – request for collation'02' – request for collation and collation data'03' – collation data only'04' – collation complete
gpm:CRN
Updated for version 1.2.0: element is made optional as it is: - Not used for mobile-NFC service management - But mandatory for other use cases
gpm:SecureElementId
Updated for version 1.2.0: element is added to the CardConfiguration structure. It is optional as: - Mandatory for mobile-NFC service management - But not used for other use cases
gpm:ValidCardProfileID
0..∞
Holds unique Identifier of valid cardProfile could be used
gpm:AID
0..∞
Application Identifier
gpm:ApplicationData
0..∞
Sample of Application Data Element
gpm:Module
0..∞
An identifier that specifies that this is data for Card Module application(s). An ASCII character field matching current personalization standards for identifying the type of data that follows. The length and values of this field are established when the personalization device is configured.
gpm:ModulesCommonData
Commom data cross Modules
gpm:DeliveryModule

Figure 4-79: `CardCustomization` element

Where:

- Compared to version 1.1.0, the CRN element of `CardConfiguration` is made optional

- Compared to version 1.1.0, the `SecureElementId` element is added (as optional) under `CardConfiguration`. This element is of type `SEIdentifier` (see section 4.5.1.2), and is only supported in the context of this specification.
- Compared to version 1.1.0, the `Module` element is renamed as `GenericModule`

In the context of this specification, the following configuration SHALL be respected:

- No batch processing is supported; the `BatchID` attribute of the `CardCustomization` element SHALL NOT be filled.
- Only the `ModuleIdentifierCode` element SHALL be present under `CardCustomization`; `GenericModule`, `ModulesCommonData` and `DeliveryModule` elements are not supported, and thus SHALL NOT be present.
- The Identifier attribute of the `ModuleIdentifierCode` element SHALL be set to "COMMON_CHIP_ENCODING".
- The `CollatorStatus` attribute of the `CardConfiguration` element SHALL be set to `'0'` – Not applicable.
- The `CRN`, `ValidCardProfileID` and `AID` elements of `CardConfiguration` SHALL NOT be supported, and thus SHALL NOT be present.
- Exactly one `ApplicationData` element SHALL be present.

Looking into details of the `ApplicationData` element:

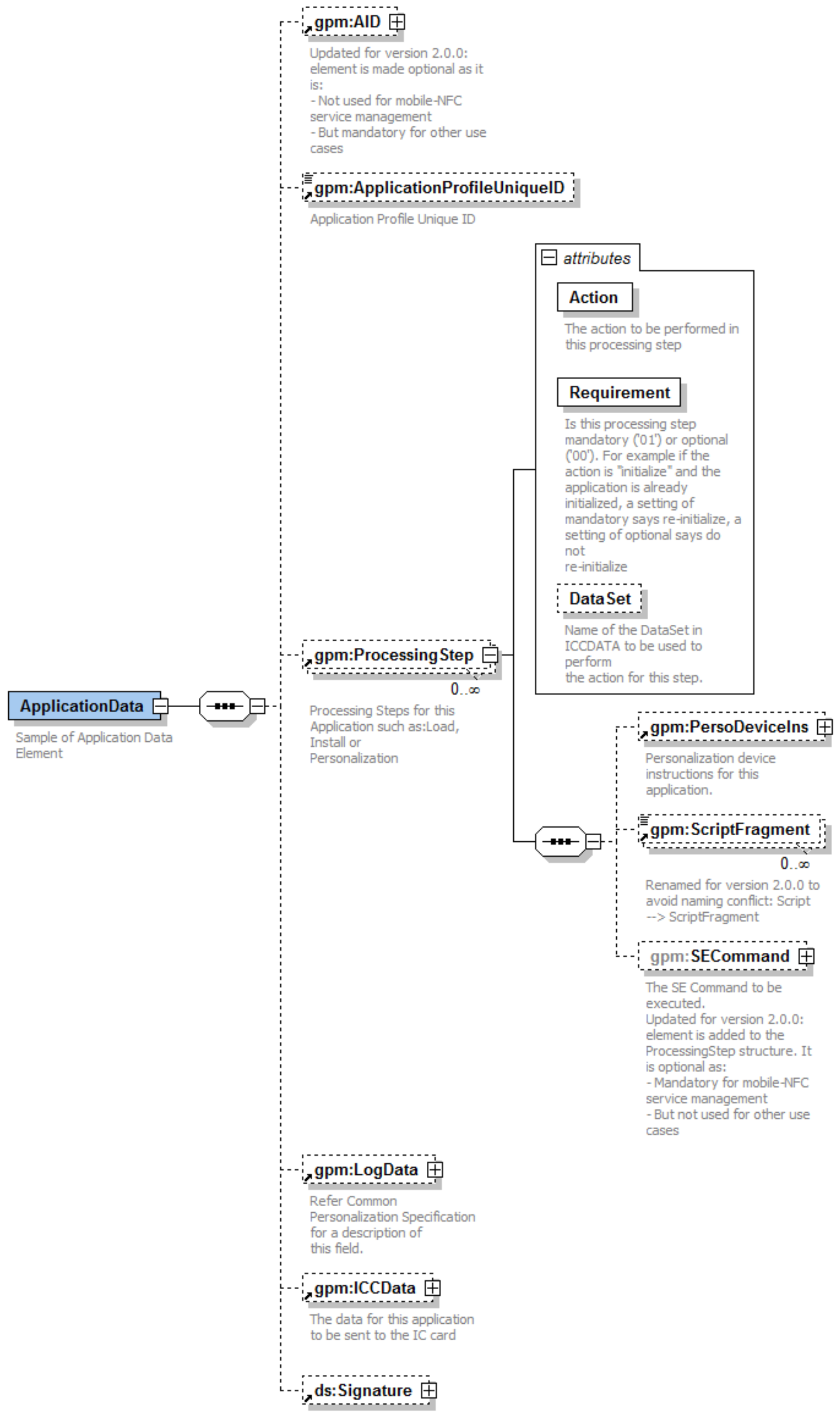

gpm:AID
Updated for version 2.0.0: element is made optional as it is: - Not used for mobile-NFC service management - But mandatory for other use cases
gpm:ApplicationProfileUniqueID
Application Profile Unique ID
attributes
Action
The action to be performed in this processing step
Requirement
Is this processing step mandatory ('01') or optional ('00'). For example if the action is "initialize" and the application is already initialized, a setting of mandatory says re-initialize, a setting of optional says do not re-initialize
DataSet
Name of the DataSet in ICCDATA to be used to perform the action for this step.
gpm:ProcessingStep
0..∞
Processing Steps for this Application such as:Load, Install or Personalization
ApplicationData
Sample of Application Data Element
gpm:PersoDeviceIns
Personalization device instructions for this application.
gpm:ScriptFragment
0..∞
Renamed for version 2.0.0 to avoid naming conflict: Script --> ScriptFragment
gpm:SECommand
The SE Command to be executed. Updated for version 2.0.0: element is added to the ProcessingStep structure. It is optional as: - Mandatory for mobile-NFC service management - But not used for other use cases
gpm:LogData
Refer Common Personalization Specification for a description of this field.
gpm:ICCData
The data for this application to be sent to the IC card
ds:Signature

Figure 4-80: `ApplicationData` element

Where:

- Compared to version 1.1.0, the AID element of `CardConfiguration` is made optional
- Compared to version 1.1.0, the `SECommand` element is added (as optional) under `ProcessingStep`. This element is of type `SECommandType` (see section 4.5.5.1.1), and is only supported in the context of this specification.
- Compared to version 1.1.0, the `Script` element is renamed as `ScriptFragment`

In the context of this specification, the following configuration SHALL be respected:

- The `AID`, `ApplicationProfileUniqueID`, `LogData`, `ICCData` and `Signature` elements of `ApplicationData` SHALL NOT be present.
- There SHALL be as many `ProcessingStep` elements as SE Commands in the `SECommandsGenerationAndRemoteExecution` function.
- The `Action` attribute of each of the `ProcessingStep` elements is an informative string that SHALL be filled with the name of the corresponding `SECommand` element of the `ProcessingStep`. E.g.: "ExtraditeCommand" if the `SECommand` element contains an `ExtraditeCommand` element.
- The `Requirement` attribute of each of the `ProcessingStep` elements SHALL be filled according to the `Stop On Warning` input parameter of the `SECommandsGenerationAndRemoteExecution` function:
  - o If `Stop On Warning` is set to `'False'` or not present, the `Requirement` string SHALL be set to `"00"`
  - o If `Stop On Warning` is set to `'True'`, the `Requirement` string SHALL be set to `"01"`
- The `DataSet` attribute of the `ProcessingStep` elements SHALL NOT be filled.
- The `PersoDeviceIns` and Script elements of the `ProcessingStep` elements SHALL NOT be present.

The output data of the `SECommandsGenerationAndRemoteExecution` function defined in section 3.5.1.2 SHALL be mapped to the `SECommandsGenerationAndRemoteExecutionResponse` element that is of type `SECommandsGenerationAndRemoteExecutionResponseType` described in the following figure:

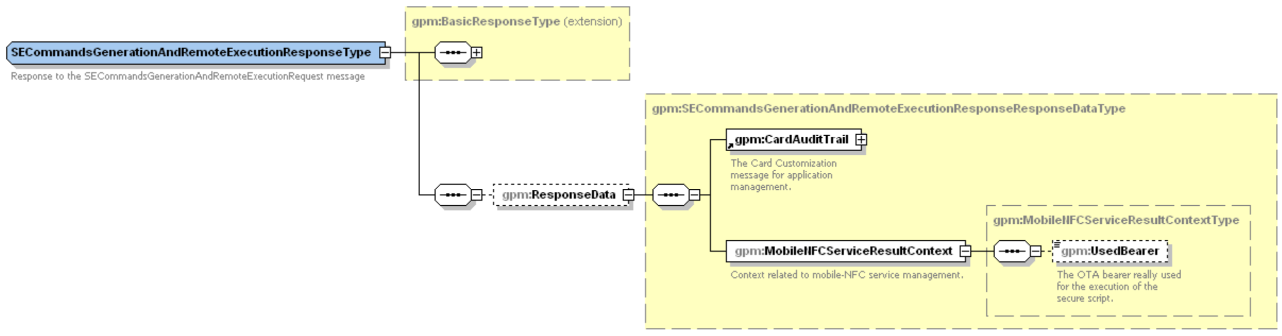

gpm:BasicResponseType (extension)
SECommandsGenerationAndRemoteExecutionResponseType
Response to the SECommandsGenerationAndRemoteExecutionRequest message
gpm:SECommandsGenerationAndRemoteExecutionResponseResponseDataType
gpm:CardAuditTrail
The Card Customization message for application management.
gpm:ResponseData
gpm:MobileNFCServiceResultContextType
gpm:MobileNFCServiceResultContext
Context related to mobile-NFC service management.
gpm:UsedBearer
The OTA bearer really used for the execution of the secure script.

Figure 4-81: `SECommandsGenerationAndRemoteExecutionResponseType` type

Where:

- The `CardAuditTrail` element is a message already existing in the GP System Messaging [2].

The value of the `GPHeader.Type` associated to this element SHALL be "SECommandsGenerationAndRemoteExecutionResponse".

The `CardAuditTrail` message defined in the version 2.0.0 of the XML schema is updated compared to version 1.1.0. Here is the new `CardAuditTrail` element structure;

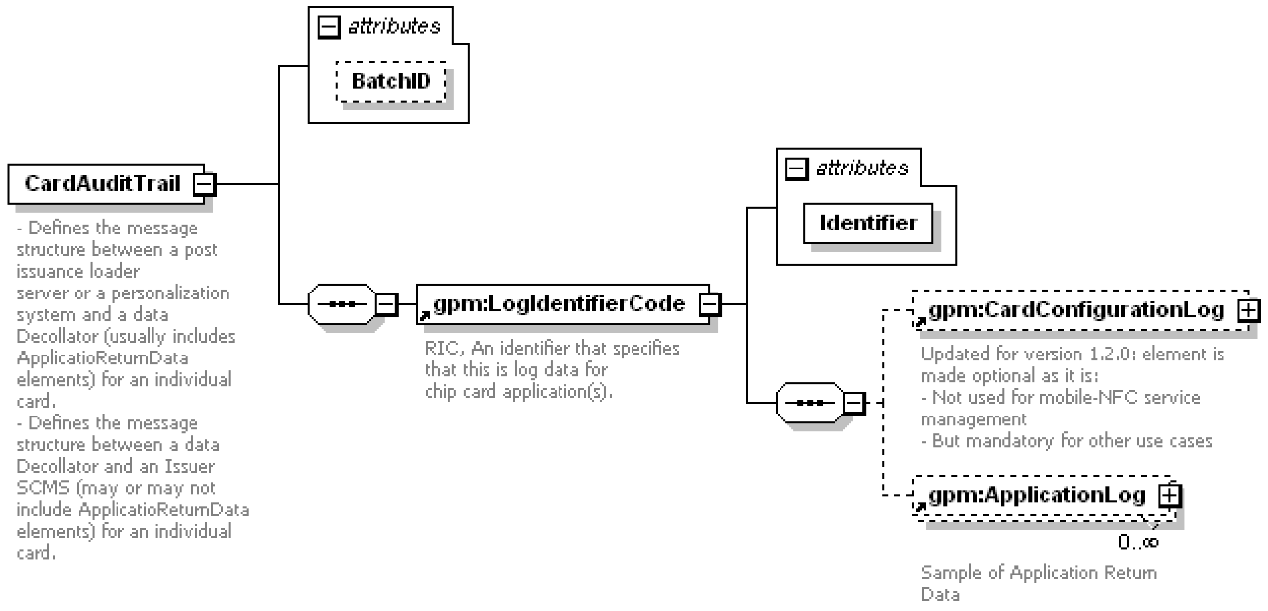

attributes
BatchID
CardAuditTrail
- Defines the message structure between a post issuance loader server or a personalization system and a data Decollator (usually includes ApplicatioReturnData elements) for an individual card.
- Defines the message structure between a data Decollator and an Issuer SCMS (may or may not include ApplicatioReturnData elements) for an individual card.
gpm:LogIdentifierCode
RIC, An identifier that specifies that this is log data for chip card application(s).
attributes
Identifier
gpm:CardConfigurationLog
Updated for version 1.2.0: element is made optional as it is:
- Not used for mobile-NFC service management
- But mandatory for other use cases
gpm:ApplicationLog
0..∞
Sample of Application Return Data

Figure 4-82: `CardAuditTrail` element

Where:

- Compared to version 1.1.0, the `CardConfigurationLog` elements of `CardAuditTrail` is made optional

In the context of this specification, the following configuration SHALL be respected:

- No batch processing is supported; the `BatchID` attribute of the `CardAuditTrail` element SHALL NOT be filled.
- The Identifier attribute of the `LogIdentifierCode` element SHALL be set to "COMMON_CHIP_ENCODING".
- The `CardConfigurationLog` element of `CardAuditTrail` SHALL NOT be supported, and thus SHALL NOT be present.
- Exactly one `ApplicationLog` element SHALL be present.

Looking into details of the `ApplicationLog` element:

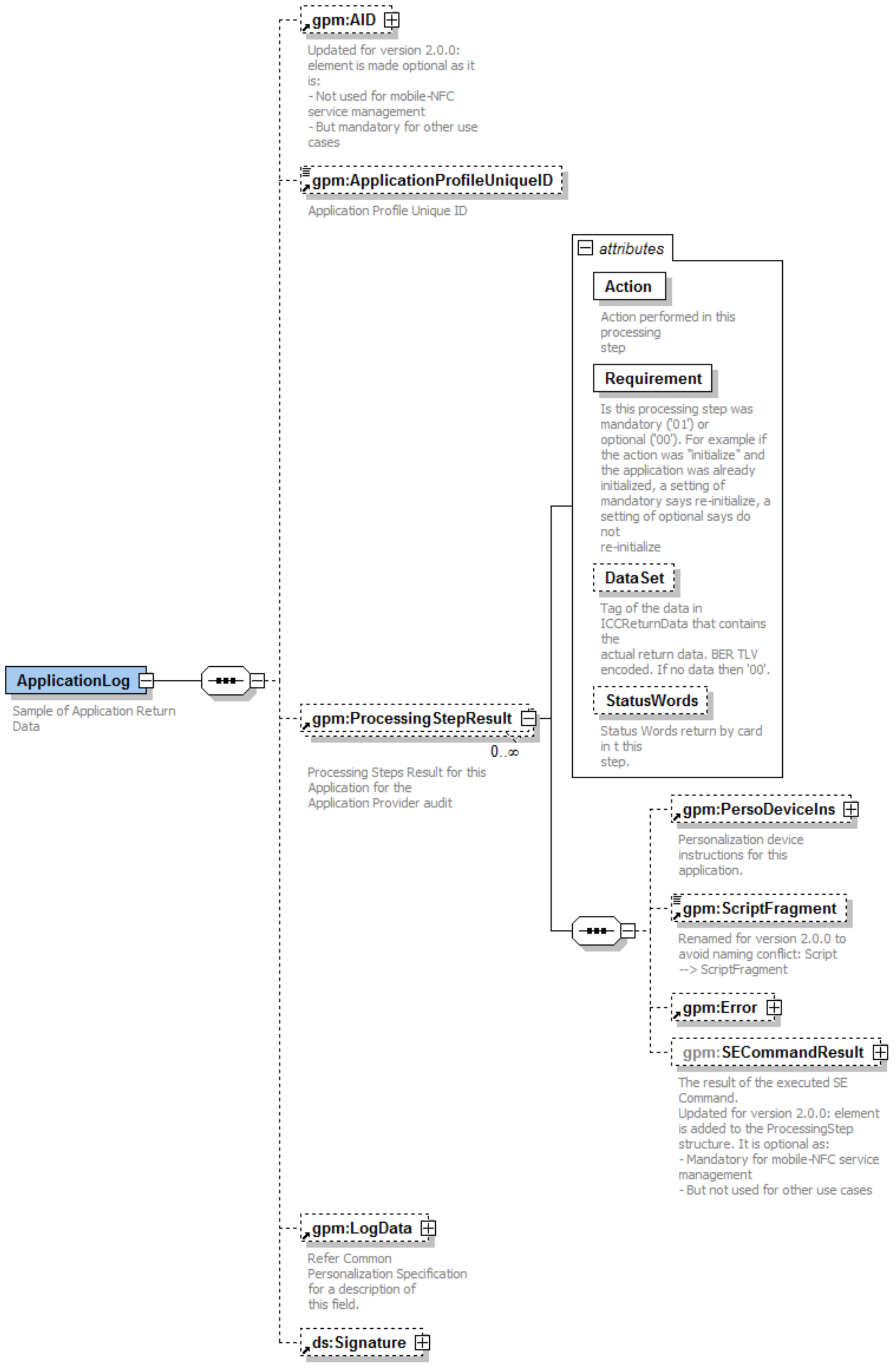

gpm:AID
Updated for version 2.0.0: element is made optional as it is:
- Not used for mobile-NFC service management
- But mandatory for other use cases
gpm:ApplicationProfileUniqueID
Application Profile Unique ID
attributes
Action
Action performed in this processing step
Requirement
Is this processing step was mandatory ('01') or optional ('00'). For example if the action was "initialize" and the application was already initialized, a setting of mandatory says re-initialize, a setting of optional says do not re-initialize
DataSet
Tag of the data in ICCReturnData that contains the actual return data. BER TLV encoded. If no data then '00'.
StatusWords
Status Words return by card in t this step.
ApplicationLog
Sample of Application Return Data
gpm:ProcessingStepResult
0..∞
Processing Steps Result for this Application for the Application Provider audit
gpm:PersoDeviceIns
Personalization device instructions for this application.
gpm:ScriptFragment
Renamed for version 2.0.0 to avoid naming conflict: Script --> ScriptFragment
gpm:Error
gpm:SECommandResult
The result of the executed SE Command.
Updated for version 2.0.0: element is added to the ProcessingStep structure. It is optional as:
- Mandatory for mobile-NFC service management
- But not used for other use cases
gpm:LogData
Refer Common Personalization Specification for a description of this field.
ds:Signature

Figure 4-83: `ApplicationLog` element

Where:

- Compared to version 1.1.0, the AID element of `CardConfigurationLog` is made optional
- Compared to version 1.1.0, the `SECommandResult` element is added (as optional) under `ProcessingStepResult`. This element is of type `SECommandResultType` (see section 4.5.5.1.2), and is only supported in the context of this specification.
- Compared to version 1.1.0, the `Script` element is renamed as `ScriptFragment`

In the context of this specification, the following configuration SHALL be respected:

- The `AID`, `ApplicationProfileUniqueID`, `LogData` and `Signature` elements of `ApplicationLog` SHALL NOT be present.
- There SHALL be as many `ProcessingStepResult` elements as SE Commands Results in the `SECommandsGenerationAndRemoteExecution` function output data.
- The `Action` attribute of each of the `ProcessingStepResult` elements is an informative binary data that SHALL be filled with the binary representation in UTF-8 of the name of the corresponding `SECommand` element that initiated the `ProcessingStepResult`. E.g.: 0x45 0x78 0x74 0x72 0x61 0x64 0x69 0x74 0x65 0x43 0x6F 0x6D 0x6D 0x61 0x6E 0x64 (binary representation of "ExtraditeCommand") if the `SECommand` that initiated the `ProcessingStepResult` was an `ExtraditeCommand` element.
- The `Requirement` attribute of each of the `ProcessingStepResult` elements SHALL be filled with the binary value of the `Requirement` attribute of the corresponding `ProcessingStep` element that initiated the `ProcessingStepResult`.
  - If the `Requirement` string attribute of the corresponding `ProcessingStep` was "00", the `Requirement` binary attribute of `ProcessingStepResult` SHALL be set to 0x00
  - If the `Requirement` string attribute of the corresponding `ProcessingStep` was "01", the `Requirement` binary attribute of `ProcessingStepResult` SHALL be set to 0x01
- The `DataSet` and `StatusWords` attributes of the `ProcessingStepResult` elements SHALL NOT be filled.
- The `PersoDeviceIns`, `Script`, and `Error` elements of the `ProcessingStepResult` elements SHALL NOT be present.

Note that the `CardCustomization` and `CardAuditTrail` messages are reused in order the `SECommandsGenerationAndRemoteExecution` to be *compatible with* the batch and the ECMA scripting mechanisms (with usage of the `Script` and `PersoDeviceIns` elements), knowing however that both batch and scripting are not supported by this version of specification.

4.5.5.1.1 SE Commands

The `SECommandType` type acts as a base abstract type for the possible SE Commands defined in section 3.5.1.3. Each concrete SE Command extends this `SECommandType` type.

SECommandType
Base type for command to be executed on the SE.

Figure 4-84: SECommandType type

4.5.5.1.1.1 The "CreateFirstSSDKeysetCommand" SE Commands

The `CreateFirstSSDKeysetCommand` SE Command defined in section 3.5.1.3.1.2 SHALL be mapped to the `CreateFirstSSDKeysetCommandType` abstract type described in the following figure. This `CreateFirstSSDKeysetCommandType` type extends the `SECommandType` type.

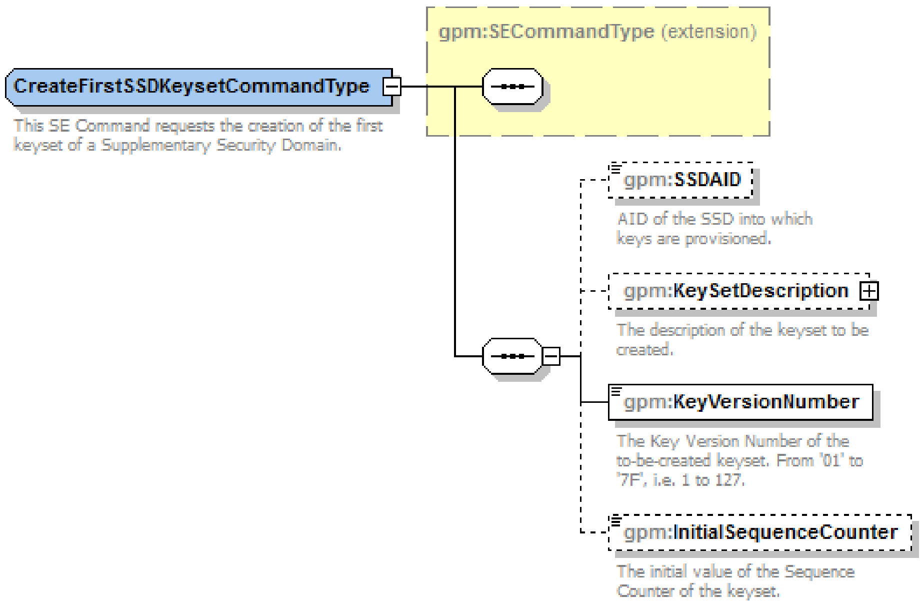

gpm:SECommandType (extension)
CreateFirstSSDKeysetCommandType
This SE Command requests the creation of the first keyset of a Supplementary Security Domain.
gpm:SSDAID
AID of the SSD into which keys are provisioned.
gpm:KeySetDescription
The description of the keyset to be created.
gpm:KeyVersionNumber
The Key Version Number of the to-be-created keyset. From '01' to '7F', i.e. 1 to 127.
gpm:InitialSequenceCounter
The initial value of the Sequence Counter of the keyset.

Figure 4-85: `CreateFirstSSDKeysetCommandType` type

Where:

- The `KeySetDescription` element is of type `KeySetDescriptionType`. This `KeySetDescriptionType` type is indeed a base type for various possible ways to describe a KeySet type.

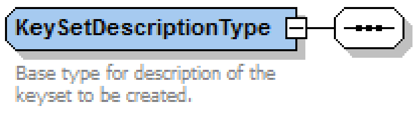


Figure 4-86: `KeySetDescriptionType` type

The concrete `Simple KeySet Profile` type defined in section 3.5.1.3.1.2 SHALL be mapped to the `SimpleKeySetProfile_KeySetDescriptionType` concrete type described in the following figure:

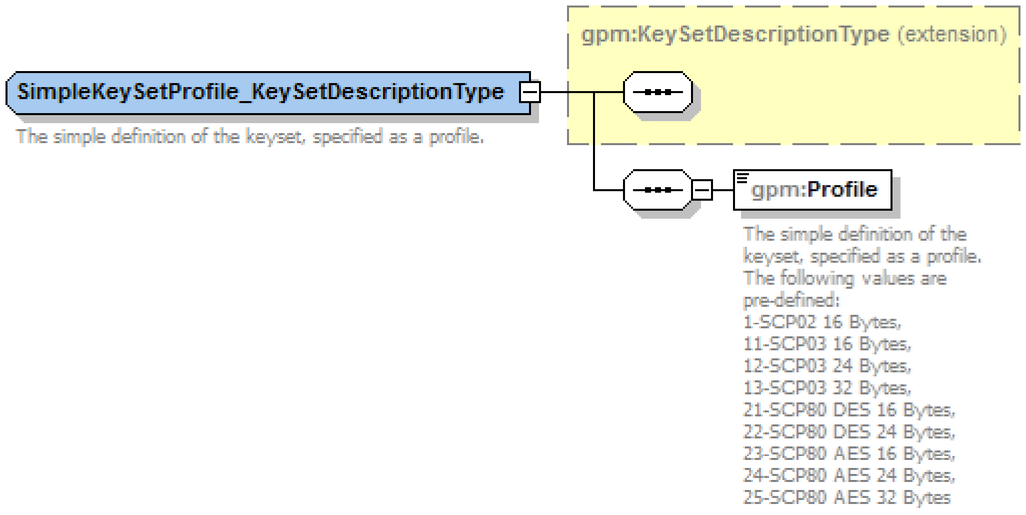

gpm:KeySetDescriptionType (extension)
SimpleKeySetProfile_KeySetDescriptionType
The simple definition of the keyset, specified as a profile.
gpm:Profile
The simple definition of the keyset, specified as a profile. The following values are pre-defined: 1-SCP02 16 Bytes, 11-SCP03 16 Bytes, 12-SCP03 24 Bytes, 13-SCP03 32 Bytes, 21-SCP80 DES 16 Bytes, 22-SCP80 DES 24 Bytes, 23-SCP80 AES 16 Bytes, 24-SCP80 AES 24 Bytes, 25-SCP80 AES 32 Bytes

Figure 4-87: `SimpleKeySetProfile_KeySetDescriptionType` type

The concrete `SCP81 KeySet Profile` type defined in section 3.5.1.3.1.2 SHALL be mapped to the `SCP81KeySetProfile_KeySetDescriptionType` concrete type described in the following figure:

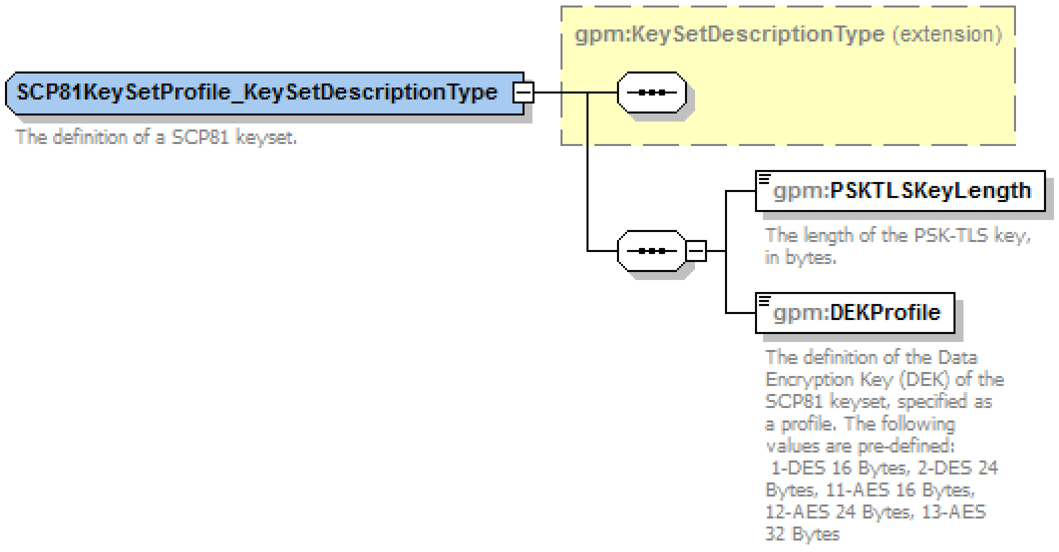
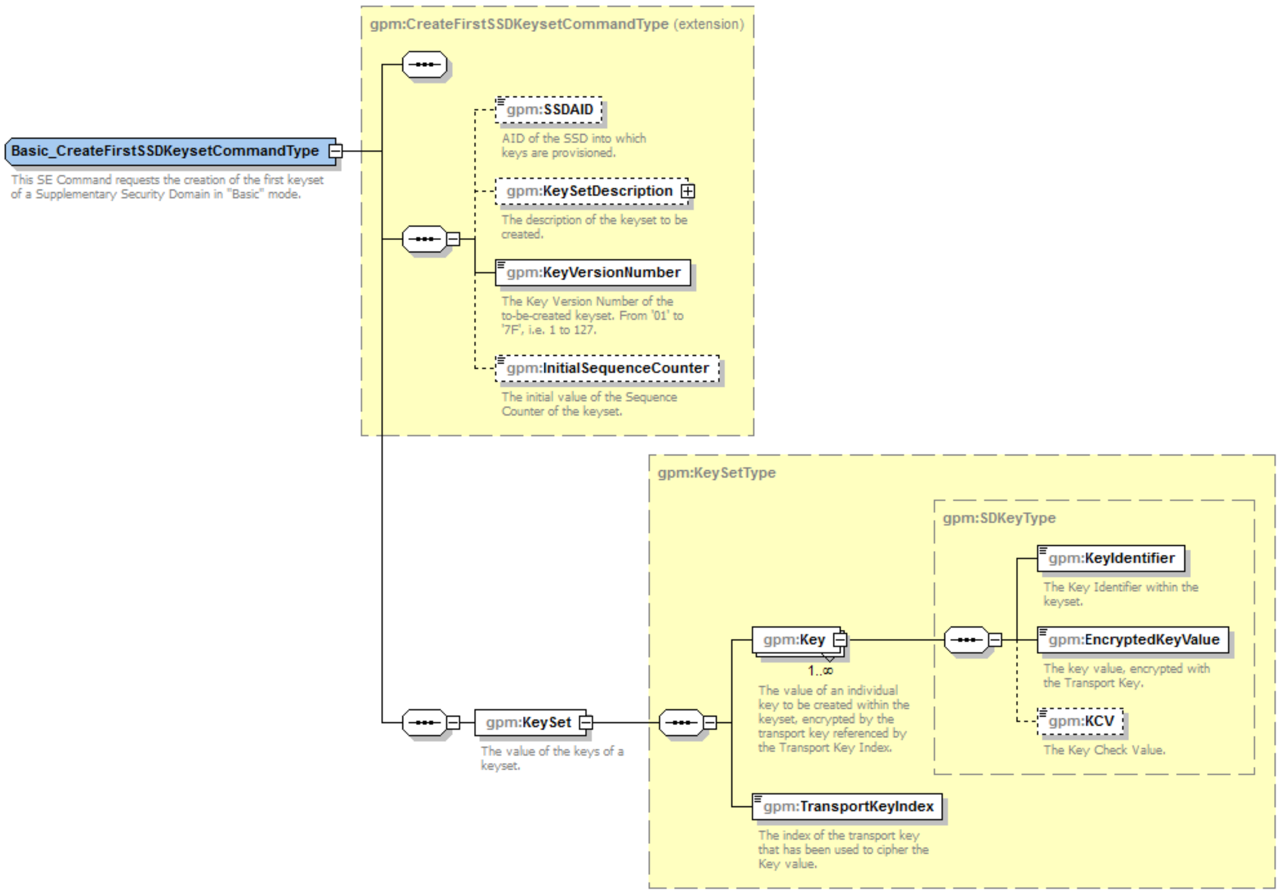

gpm:KeySetDescriptionType (extension)
SCP81KeySetProfile_KeySetDescriptionType
The definition of a SCP81 keyset.
gpm:PSKTLSKeyLength
The length of the PSK-TLS key, in bytes.
gpm:DEKProfile
The definition of the Data Encryption Key (DEK) of the SCP81 keyset, specified as a profile. The following values are pre-defined: 1-DES 16 Bytes, 2-DES 24 Bytes, 11-AES 16 Bytes, 12-AES 24 Bytes, 13-AES 32 Bytes

Figure 4-88: `SCP81KeySetProfile_KeySetDescriptionType` type

The key provisioning mode and key provisioning data (KPD) are mapped through an inheritance mechanism: instead of having dedicated Key Provisioning Mode and Key Provisioning Data elements for a given `Xxx` key provisioning mode, a `Xxx_CreateFirstSSDKeysetCommandType` type is created instead, by extension of the `CreateFirstSSDKeysetCommandType` abstract type.

Then, in case of Basic Create mode, the concrete `Basic_CreateFirstSSDKeysetCommandType` type SHALL be used (see section 3.5.1.3.1.2).



gpm:CreateFirstSSDKeysetCommandType (extension)
Basic_CreateFirstSSDKeysetCommandType
This SE Command requests the creation of the first keyset of a Supplementary Security Domain in "Basic" mode.
gpm:SSDAID
AID of the SSD into which keys are provisioned.
gpm:KeySetDescription
The description of the keyset to be created.
gpm:KeyVersionNumber
The Key Version Number of the to-be-created keyset. From '01' to '7F', i.e. 1 to 127.
gpm:InitialSequenceCounter
The initial value of the Sequence Counter of the keyset.
gpm:KeySetType
gpm:SDKeyType
gpm:KeyIdentifier
The Key Identifier within the keyset.
gpm:EncryptedKeyValue
The key value, encrypted with the Transport Key.
gpm:KCV
The Key Check Value.
gpm:Key
1..∞
The value of an individual key to be created within the keyset, encrypted by the transport key referenced by the Transport Key Index.
gpm:KeySet
The value of the keys of a keyset.
gpm:TransportKeyIndex
The index of the transport key that has been used to cipher the Key value.

Figure 4-89: `Basic_CreateFirstSSDKeysetCommandType` type

In case of Basic Random Create mode, the concrete `BasicRandom_CreateFirstSSDKeysetCommandType` type SHALL be used (see section 3.5.1.3.1.2).

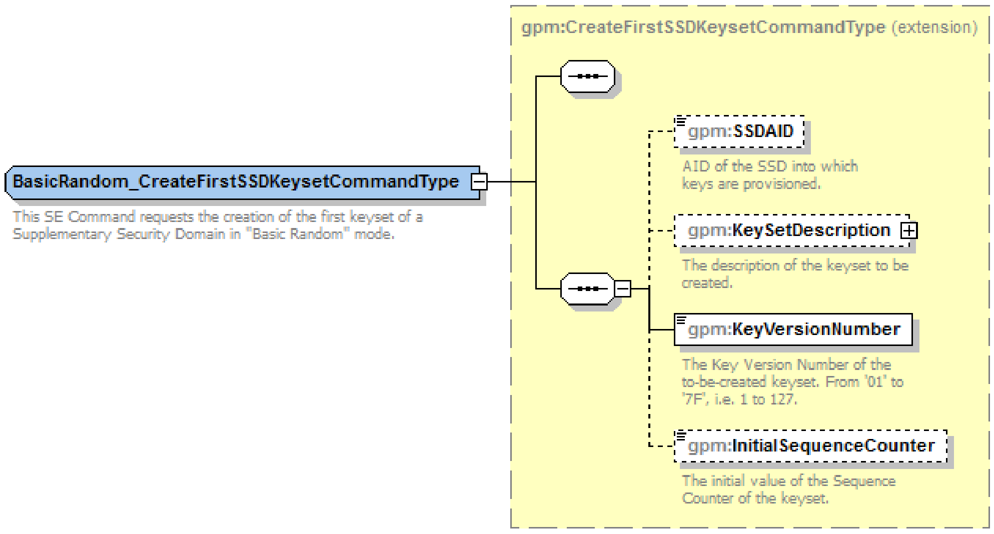

gpm:CreateFirstSSDKeysetCommandType (extension)
BasicRandom_CreateFirstSSDKeysetCommandType
This SE Command requests the creation of the first keyset of a Supplementary Security Domain in "Basic Random" mode.
gpm:SSDAID
AID of the SSD into which keys are provisioned.
gpm:KeySetDescription
The description of the keyset to be created.
gpm:KeyVersionNumber
The Key Version Number of the to-be-created keyset. From '01' to '7F', i.e. 1 to 127.
gpm:InitialSequenceCounter
The initial value of the Sequence Counter of the keyset.

Figure 4-90: `BasicRandom_CreateFirstSSDKeysetCommandType` type

In case of Basic Diversified Create mode, the concrete `BasicDiversified_CreateFirstSSDKeysetCommandType` type SHALL be used (see section 3.5.1.3.1.2).

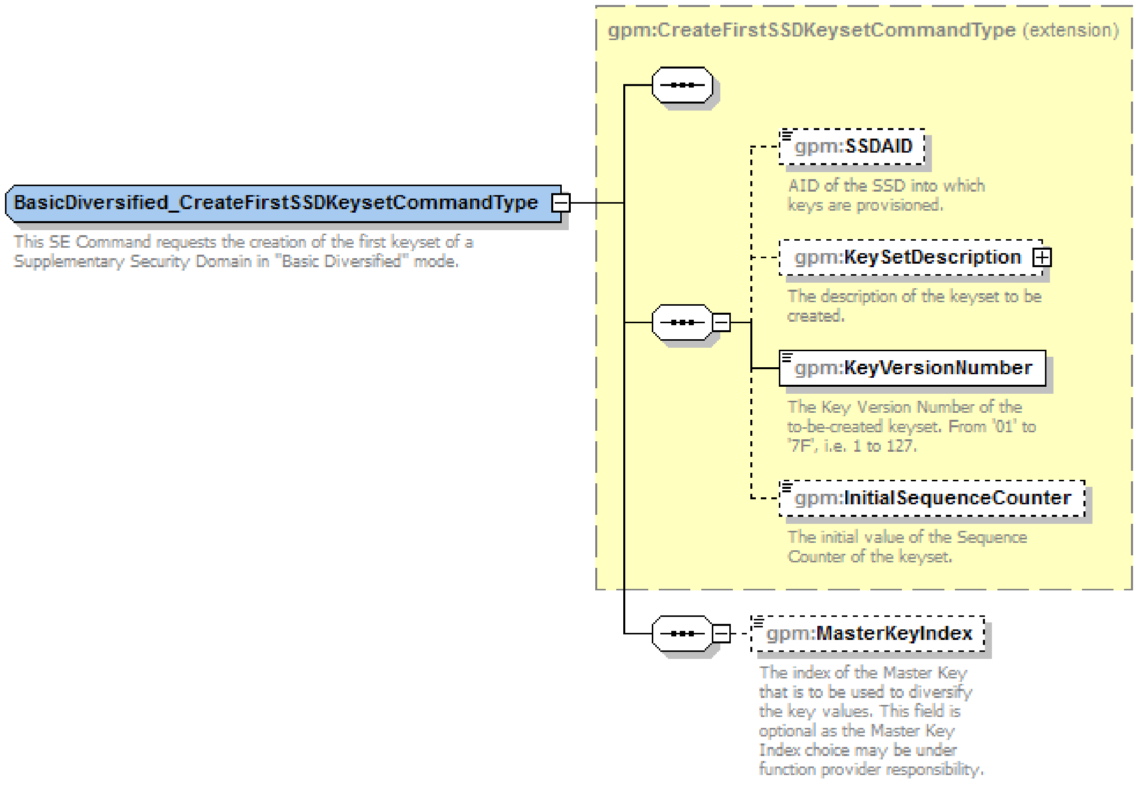

gpm:CreateFirstSSDKeysetCommandType (extension)
BasicDiversified_CreateFirstSSDKeysetCommandType
This SE Command requests the creation of the first keyset of a Supplementary Security Domain in "Basic Diversified" mode.
gpm:SSDAID
AID of the SSD into which keys are provisioned.
gpm:KeySetDescription
The description of the keyset to be created.
gpm:KeyVersionNumber
The Key Version Number of the to-be-created keyset. From '01' to '7F', i.e. 1 to 127.
gpm:InitialSequenceCounter
The initial value of the Sequence Counter of the keyset.
gpm:MasterKeyIndex
The index of the Master Key that is to be used to diversify the key values. This field is optional as the Master Key Index choice may be under function provider responsibility.

Figure 4-91: `BasicDiversified_CreateFirstSSDKeysetCommandType` type

In case of CCCM Scenario #1, using PK scheme mode, the concrete `CCCMScenario1UsingPK_CreateFirstSSDKeysetCommandType` type SHALL be used (see section 3.5.1.3.1.2).

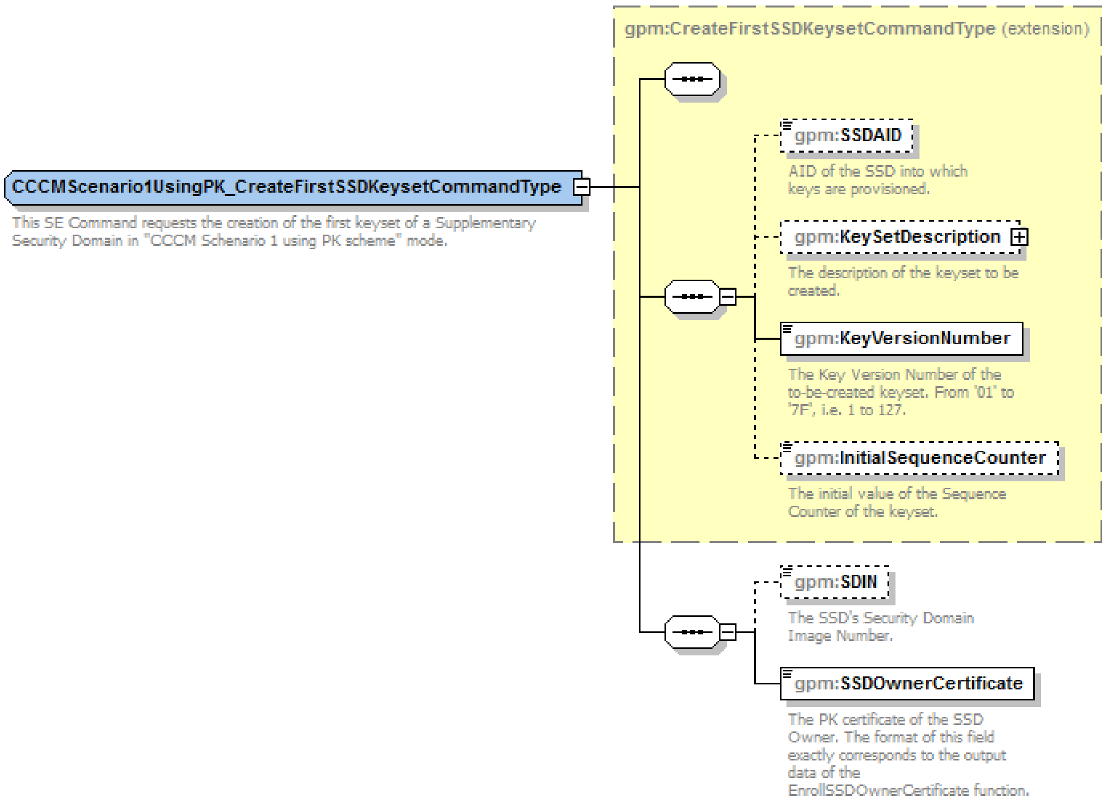

gpm:CreateFirstSSDKeysetCommandType (extension)
CCCMScenario1UsingPK_CreateFirstSSDKeysetCommandType
This SE Command requests the creation of the first keyset of a Supplementary Security Domain in "CCCM Schenario 1 using PK scheme" mode.
gpm:SSDAID
AID of the SSD into which keys are provisioned.
gpm:KeySetDescription
The description of the keyset to be created.
gpm:KeyVersionNumber
The Key Version Number of the to-be-created keyset. From '01' to '7F', i.e. 1 to 127.
gpm:InitialSequenceCounter
The initial value of the Sequence Counter of the keyset.
gpm:SDIN
The SSD's Security Domain Image Number.
gpm:SSDOwnerCertificate
The PK certificate of the SSD Owner. The format of this field exactly corresponds to the output data of the EnrollSSDOwnerCertificate function.

Figure 4-92: `CCCMScenario1UsingPK_CreateFirstSSDKeysetCommandType` type

In case of CCCM Scenario #2A mode, the concrete `CCCMScenario2A_CreateFirstSSDKeysetCommandType` type SHALL be used (see section 3.5.1.3.1.2).

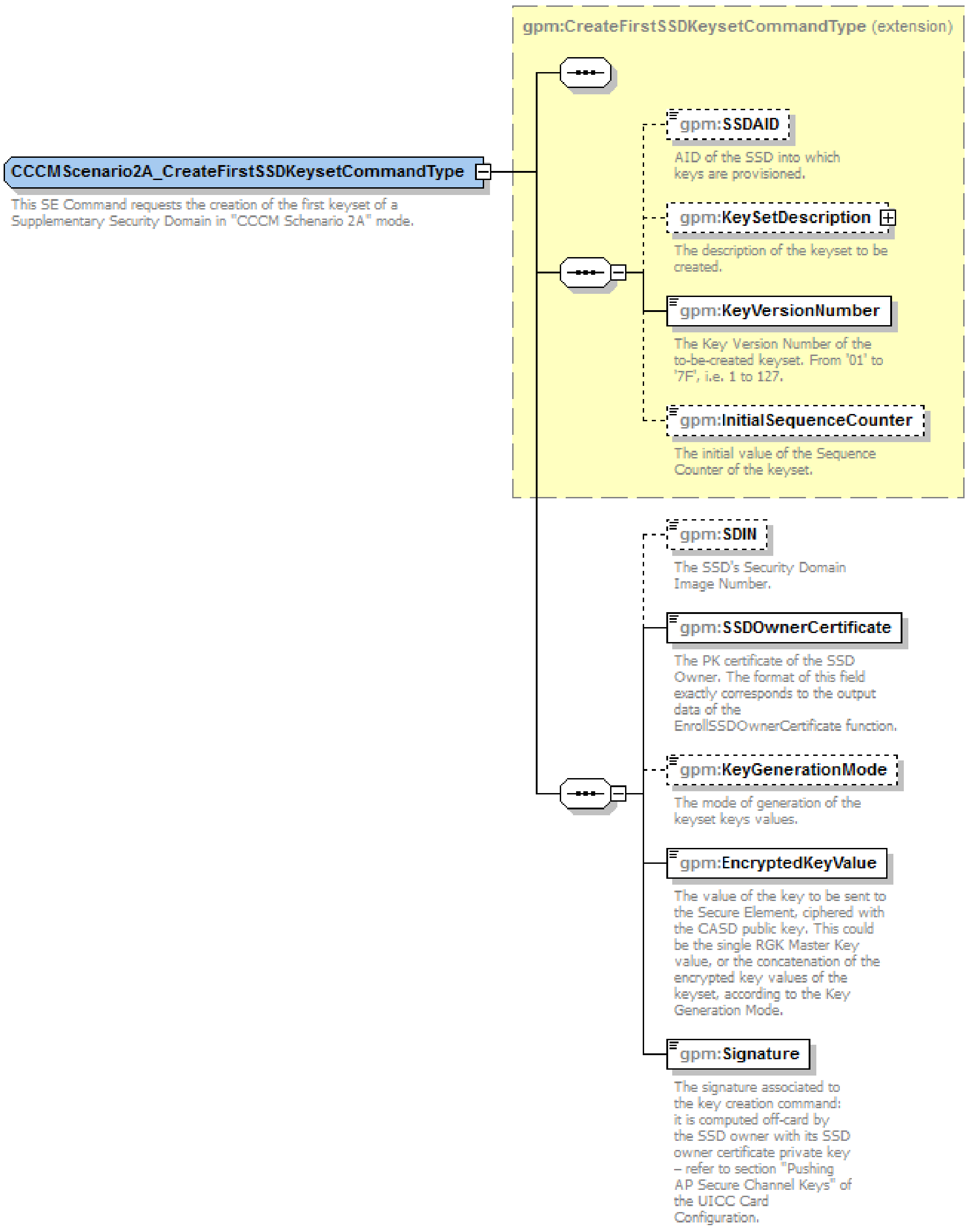

gpm:CreateFirstSSDKeysetCommandType (extension)
CCCMScenario2A_CreateFirstSSDKeysetCommandType
This SE Command requests the creation of the first keyset of a Supplementary Security Domain in "CCCM Schenario 2A" mode.
gpm:SSDAID
AID of the SSD into which keys are provisioned.
gpm:KeySetDescription
The description of the keyset to be created.
gpm:KeyVersionNumber
The Key Version Number of the to-be-created keyset. From '01' to '7F', i.e. 1 to 127.
gpm:InitialSequenceCounter
The initial value of the Sequence Counter of the keyset.
gpm:SDIN
The SSD's Security Domain Image Number.
gpm:SSDOwnerCertificate
The PK certificate of the SSD Owner. The format of this field exactly corresponds to the output data of the EnrollSSDOwnerCertificate function.
gpm:KeyGenerationMode
The mode of generation of the keyset keys values.
gpm:EncryptedKeyValue
The value of the key to be sent to the Secure Element, ciphered with the CASD public key. This could be the single RGK Master Key value, or the concatenation of the encrypted key values of the keyset, according to the Key Generation Mode.
gpm:Signature
The signature associated to the key creation command: it is computed off-card by the SSD owner with its SSD owner certificate private key – refer to section "Pushing AP Secure Channel Keys" of the UICC Card Configuration.

Figure 4-93: `CCCMScenario2A_CreateFirstSSDKeysetCommandType` type

Where:

- The `KeyGenerationMode` is an enumeration of the possible key generation modes (`KeyGenerationModeType`).

In case of CCCM Scenario #2B mode, the concrete `CCCMScenario2B_CreateFirstSSDKeysetCommandType` type SHALL be used (see section 3.5.1.3.1.2).

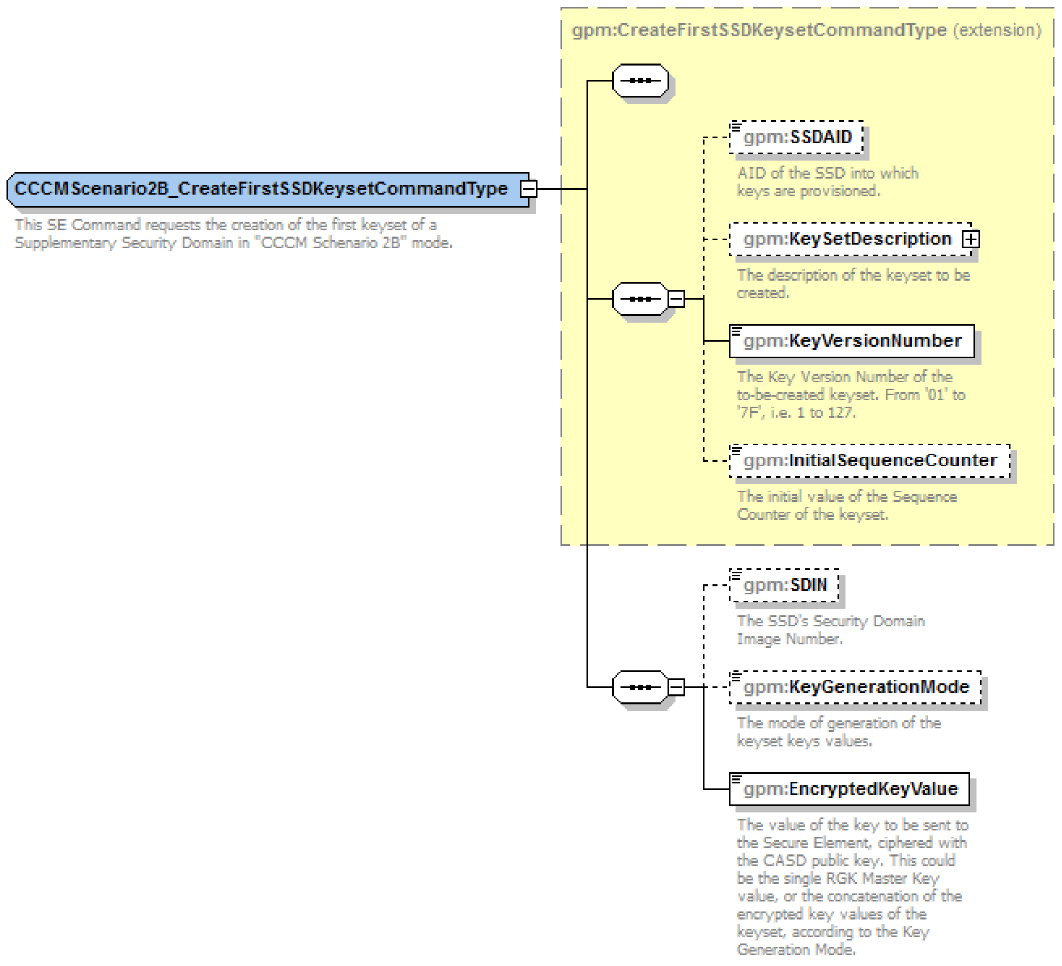

gpm:CreateFirstSSDKeysetCommandType (extension)
CCCMScenario2B_CreateFirstSSDKeysetCommandType
This SE Command requests the creation of the first keyset of a Supplementary Security Domain in "CCCM Schenario 2B" mode.
gpm:SSDAID
AID of the SSD into which keys are provisioned.
gpm:KeySetDescription
The description of the keyset to be created.
gpm:KeyVersionNumber
The Key Version Number of the to-be-created keyset. From '01' to '7F', i.e. 1 to 127.
gpm:InitialSequenceCounter
The initial value of the Sequence Counter of the keyset.
gpm:SDIN
The SSD's Security Domain Image Number.
gpm:KeyGenerationMode
The mode of generation of the keyset keys values.
gpm:EncryptedKeyValue
The value of the key to be sent to the Secure Element, ciphered with the CASD public key. This could be the single RGK Master Key value, or the concatenation of the encrypted key values of the keyset, according to the Key Generation Mode.

Figure 4-94: `CCCMScenario2B_CreateFirstSSDKeysetCommandType` type

Where:

- The `KeyGenerationMode` is an enumeration of the possible key generation modes (`KeyGenerationModeType`).

In case of CCCM Scenario #3 mode, the concrete `CCCMScenario3_CreateFirstSSDKeysetCommandType` type SHALL be used (see section 3.5.1.3.1.2).

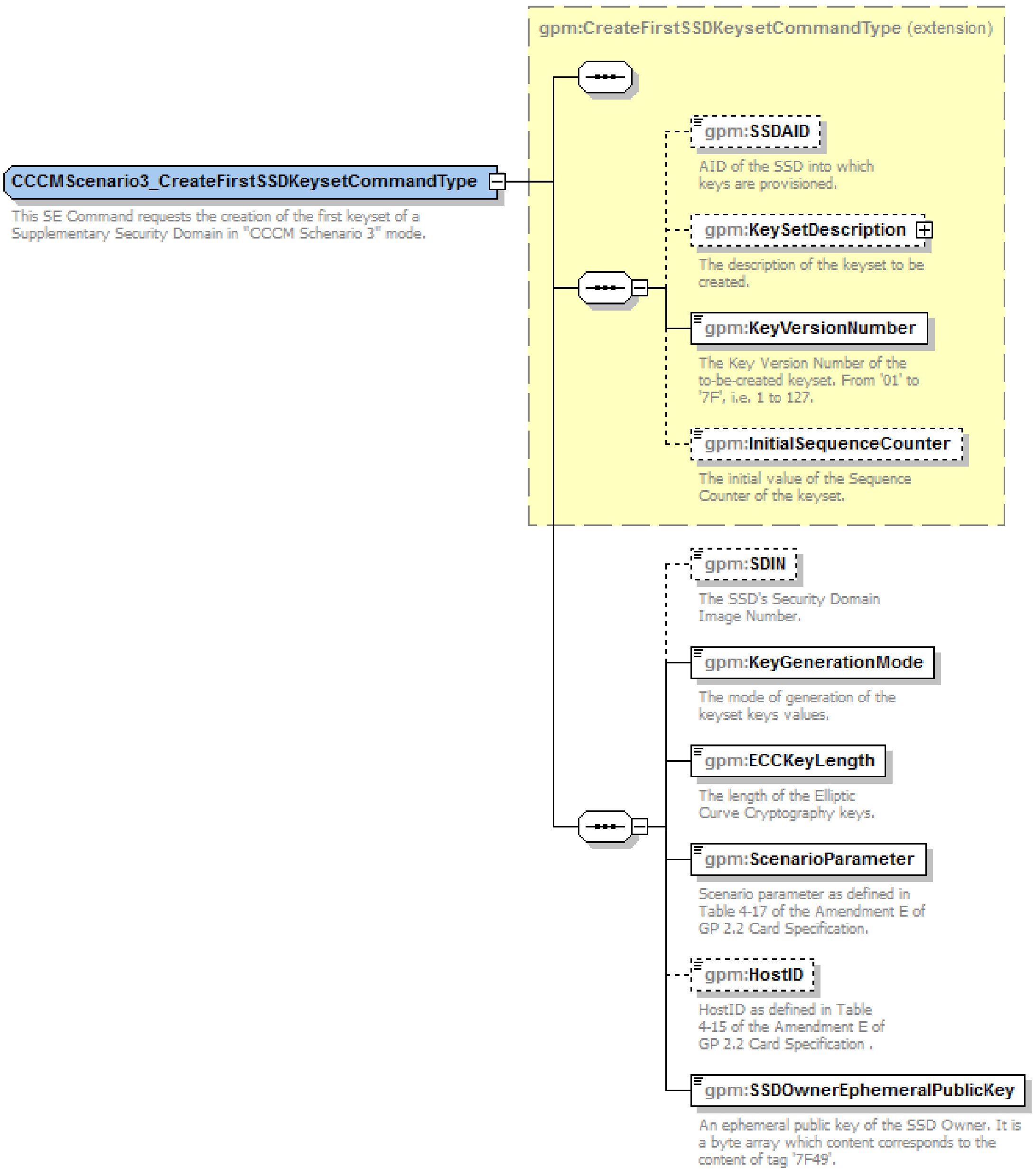

gpm:CreateFirstSSDKeysetCommandType (extension)
CCCMScenario3_CreateFirstSSDKeysetCommandType
This SE Command requests the creation of the first keyset of a Supplementary Security Domain in "CCCM Schenario 3" mode.
gpm:SSDAID
AID of the SSD into which keys are provisioned.
gpm:KeySetDescription
The description of the keyset to be created.
gpm:KeyVersionNumber
The Key Version Number of the to-be-created keyset. From '01' to '7F', i.e. 1 to 127.
gpm:InitialSequenceCounter
The initial value of the Sequence Counter of the keyset.
gpm:SDIN
The SSD's Security Domain Image Number.
gpm:KeyGenerationMode
The mode of generation of the keyset keys values.
gpm:ECCKeyLength
The length of the Elliptic Curve Cryptography keys.
gpm:ScenarioParameter
Scenario parameter as defined in Table 4-17 of the Amendment E of GP 2.2 Card Specification.
gpm:HostID
HostID as defined in Table 4-15 of the Amendment E of GP 2.2 Card Specification .
gpm:SSDOwnerEphemeralPublicKey
An ephemeral public key of the SSD Owner. It is a byte array which content corresponds to the content of tag '7F49'.

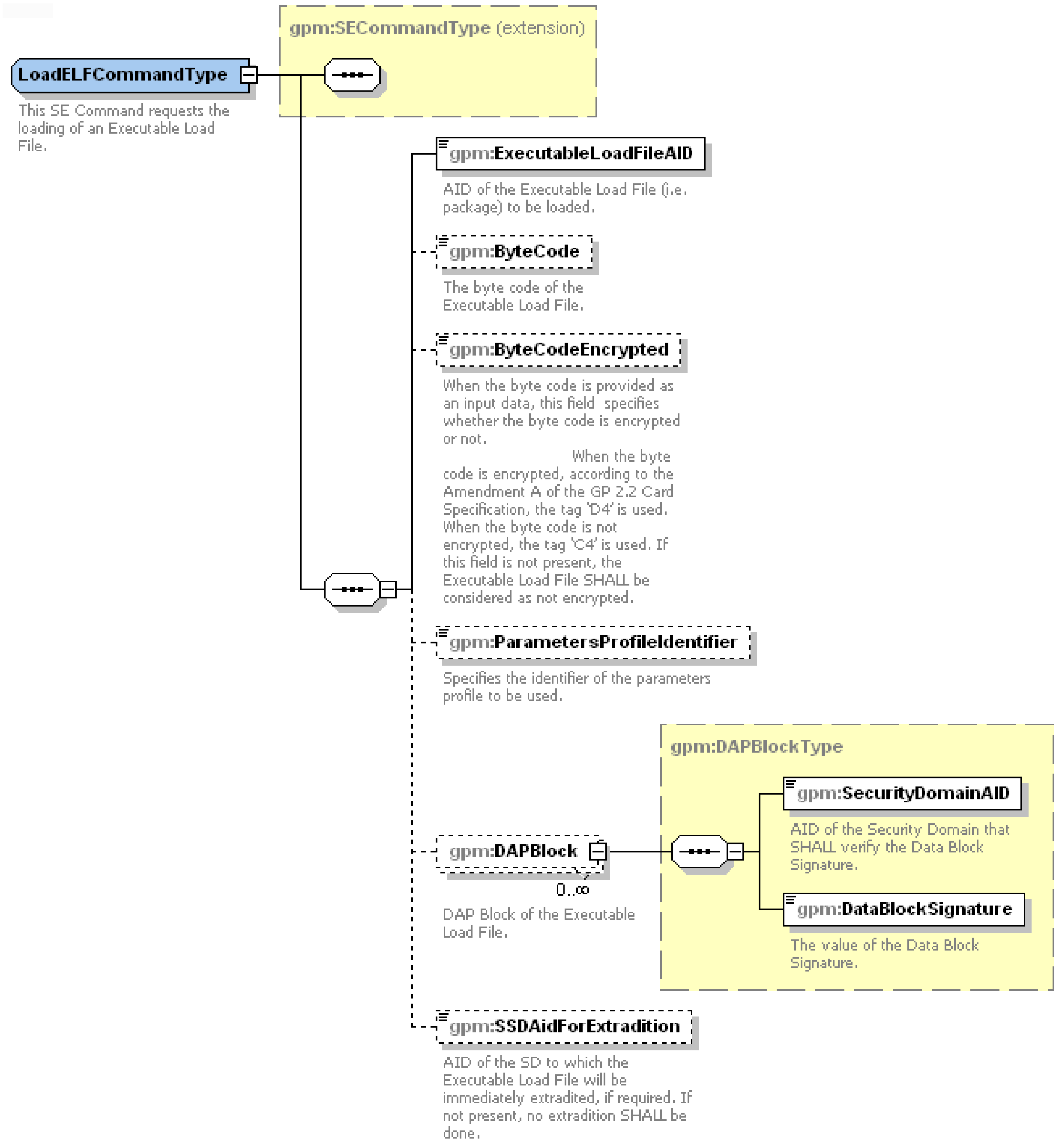
Figure 4-95: `CCCMScenario3_CreateFirstSSDKeysetCommandType` type

Where:

- The `KeyGenerationMode` is an enumeration of the possible key generation modes (`KeyGenerationModeType`).
- The `ECCKeyLength` is an enumeration of the possible Elliptic Curve Cryptography key length (`ECCKeyLengthType`).

4.5.5.1.1.2 The "LoadELFCommand" SE Commands

The `LoadELFCommand` SE Command defined in section 3.5.1.3.2.1 SHALL be mapped to the `LoadELFCommandType` type described in the following figure. This `LoadELFCommandType` type extends the `SECommandType` type.



gpm:SECommandType (extension)
LoadELFCommandType
This SE Command requests the loading of an Executable Load File.
gpm:ExecutableLoadFileAID
AID of the Executable Load File (i.e. package) to be loaded.
gpm:ByteCode
The byte code of the Executable Load File.
gpm:ByteCodeEncrypted
When the byte code is provided as an input data, this field specifies whether the byte code is encrypted or not. When the byte code is encrypted, according to the Amendment A of the GP 2.2 Card Specification, the tag 'D4' is used. When the byte code is not encrypted, the tag 'C4' is used. If this field is not present, the Executable Load File SHALL be considered as not encrypted.
gpm:ParametersProfileIdentifier
Specifies the identifier of the parameters profile to be used.
gpm:DAPBlockType
gpm:SecurityDomainAID
AID of the Security Domain that SHALL verify the Data Block Signature.
gpm:DAPBlock
0..∞
DAP Block of the Executable Load File.
gpm:DataBlockSignature
The value of the Data Block Signature.
gpm:SSDAidForExtradition
AID of the SD to which the Executable Load File will be immediately extradited, if required. If not present, no extradition SHALL be done.

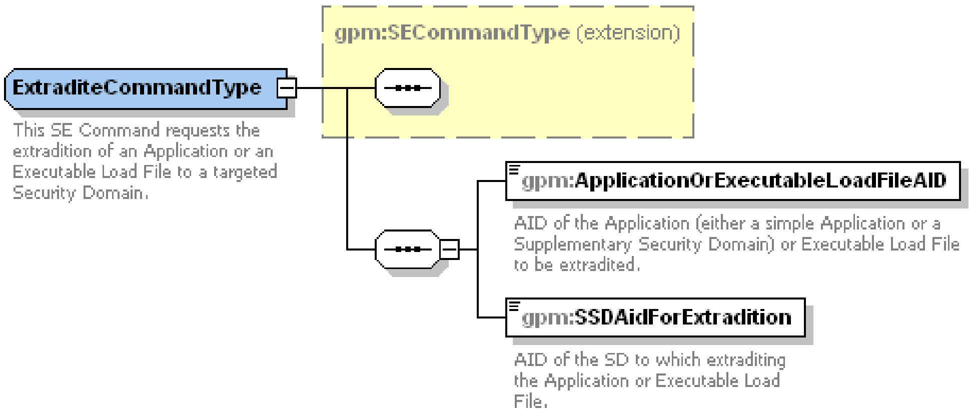
Figure 4-96: LoadELFCommandType type

Where:

- The `ExecutableLoadFileAID` and `SSDAidForExtradition` elements are of type `AIDType` (see section 4.5.1.2).

4.5.5.1.1.3 The "ExtraditeCommand" SE Commands

The ExtraditeCommand SE Command defined in section 3.5.1.3.2.2 SHALL be mapped to the `ExtraditeCommandType` type described in the following figure. This ExtraditeCommandType type extends the `SECommandType` type.



gpm:SECommandType (extension)
ExtraditeCommandType
This SE Command requests the extradition of an Application or an Executable Load File to a targeted Security Domain.
gpm:ApplicationOrExecutableLoadFileAID
AID of the Application (either a simple Application or a Supplementary Security Domain) or Executable Load File to be extradited.
gpm:SSDAidForExtradition
AID of the SD to which extraditing the Application or Executable Load File.

Figure 4-97: `ExtraditeCommandType` type

Where:

- The `ApplicationOrExecutableLoadFileAID` and `SSDAidForExtradition` elements are of type `AIDType` (see section 4.5.1.2).

4.5.5.1.1.4 The "InstantiateApplicationCommand" SE Commands

The `InstantiateApplicationCommand` SE Command defined in section 3.5.1.3.2.3 SHALL be mapped to the `InstantiateApplicationCommandType` type described in the following figure. This `InstantiateApplicationCommandType` type extends the `SECommandType` type.

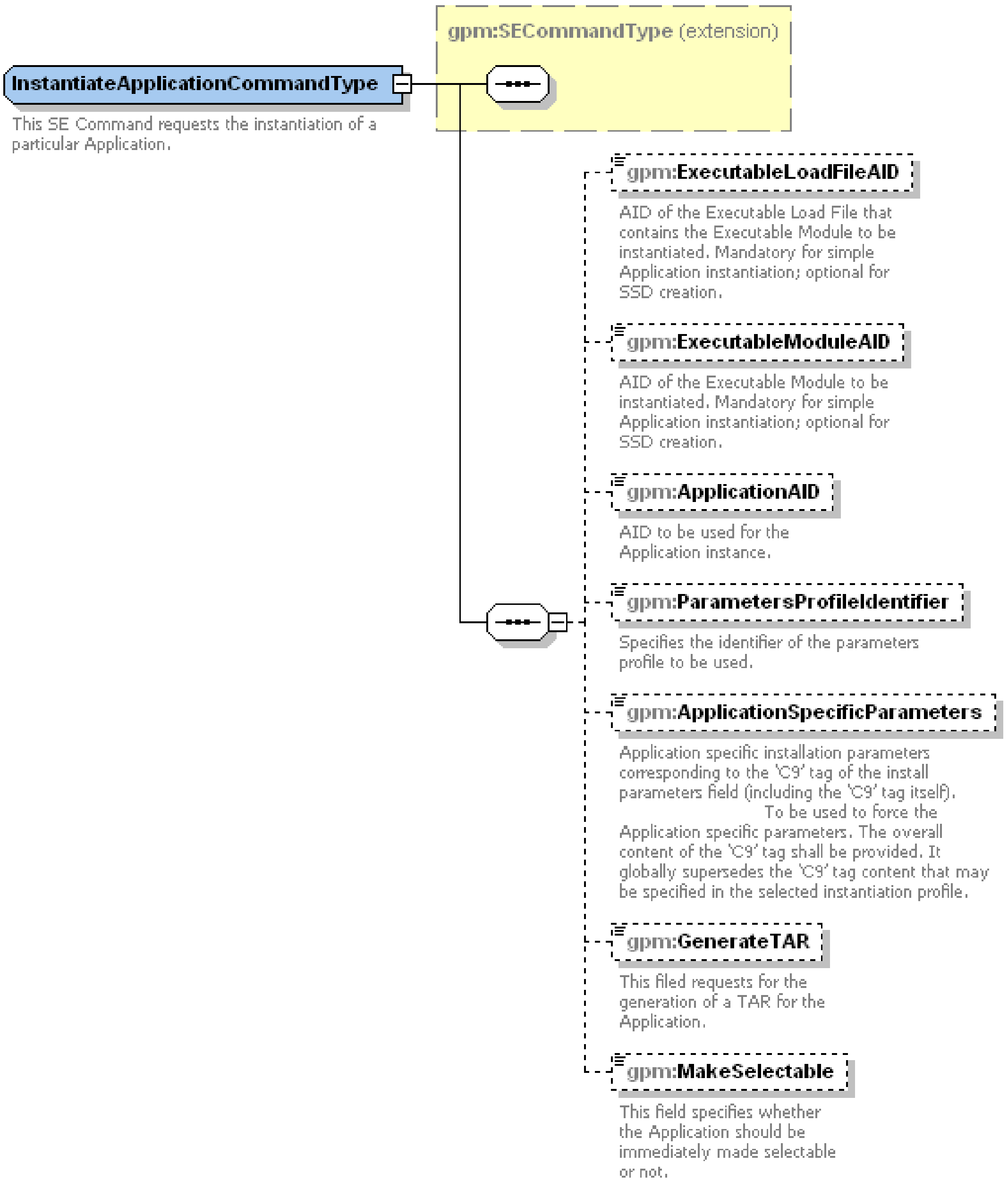

gpm:SECommandType (extension)
InstantiateApplicationCommandType
This SE Command requests the instantiation of a particular Application.
gpm:ExecutableLoadFileAID
AID of the Executable Load File that contains the Executable Module to be instantiated. Mandatory for simple Application instantiation; optional for SSD creation.
gpm:ExecutableModuleAID
AID of the Executable Module to be instantiated. Mandatory for simple Application instantiation; optional for SSD creation.
gpm:ApplicationAID
AID to be used for the Application instance.
gpm:ParametersProfileIdentifier
Specifies the identifier of the parameters profile to be used.
gpm:ApplicationSpecificParameters
Application specific installation parameters corresponding to the 'C9' tag of the install parameters field (including the 'C9' tag itself). To be used to force the Application specific parameters. The overall content of the 'C9' tag shall be provided. It globally supersedes the 'C9' tag content that may be specified in the selected instantiation profile.
gpm:GenerateTAR
This filed requests for the generation of a TAR for the Application.
gpm:MakeSelectable
This field specifies whether the Application should be immediately made selectable or not.

Figure 4-98: `InstantiateApplicationCommandType` type

Where:

- The `ExecutableLoadFileAID`, `SSDAidForExtradition` and `ApplicationAID` elements are of type `AIDType` (see section 4.5.1.2).

4.5.5.1.1.5 The "MakeSelectableApplicationCommand" SE Commands

The `MakeSelectableApplicationCommand` SE Command defined in section 3.5.1.3.2.4 SHALL be mapped to the `MakeSelectableApplicationCommandType` type described in the following figure. This `MakeSelectableApplicationCommandType` type extends the `SECommandType` type.

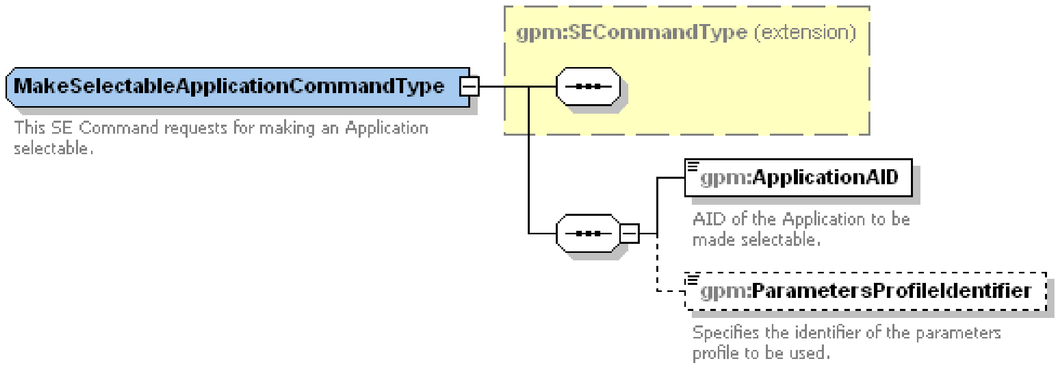

gpm:SECommandType (extension)
MakeSelectableApplicationCommandType
This SE Command requests for making an Application selectable.
gpm:ApplicationAID
AID of the Application to be made selectable.
gpm:ParametersProfileIdentifier
Specifies the identifier of the parameters profile to be used.

Figure 4-99: `MakeSelectableApplicationCommandType` type

Where:

- The `ApplicationAID` element is of type `AIDType` (see section 4.5.1.2).

4.5.5.1.1.6 The "ApplicationRegistryUpdateCommand" SE Commands

The `ApplicationRegistryUpdateCommand` SE Command defined in section 3.5.1.3.2.5 SHALL be mapped to the `ApplicationRegistryUpdateCommandType` type described in the following figure. This `ApplicationRegistryUpdateCommandType` type extends the `SECommandType` type.

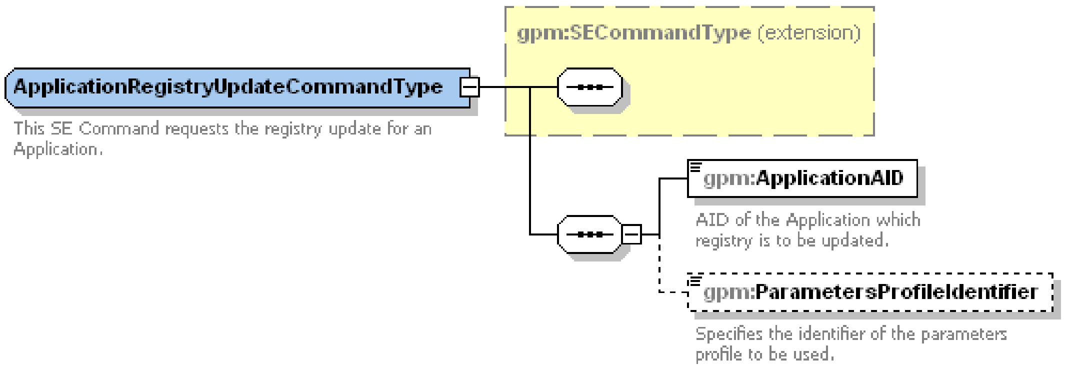

gpm:SECommandType (extension)
ApplicationRegistryUpdateCommandType
This SE Command requests the registry update for an Application.
gpm:ApplicationAID
AID of the Application which registry is to be updated.
gpm:ParametersProfileIdentifier
Specifies the identifier of the parameters profile to be used.

Figure 4-100: `ApplicationRegistryUpdateCommandType` type

Where:

- The `ApplicationAID` element is of type `AIDType` (see section 4.5.1.2).

4.5.5.1.1.7 The "SetStatusCommand" SE Commands

The `SetStatusCommand` SE Command defined in section 3.5.1.3.2.6 SHALL be mapped to the `SetStatusCommandType` type described in the following figure. This `SetStatusCommandType` type extends the `SECommandType` type.

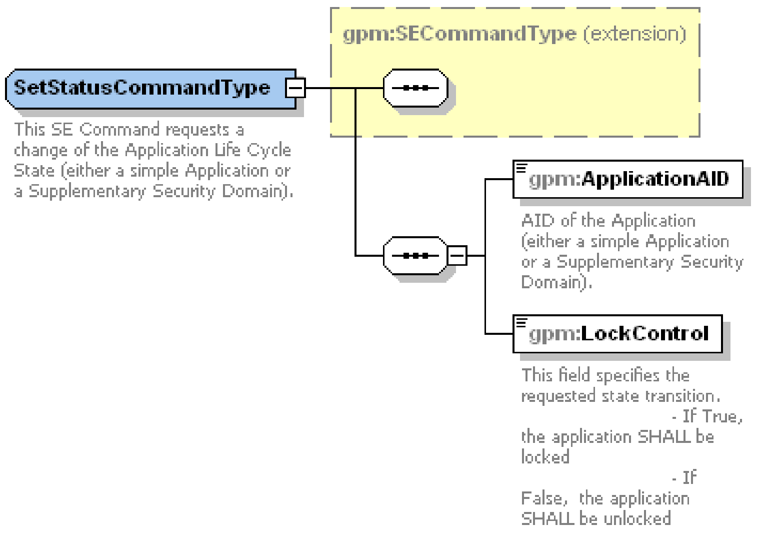

gpm:SECommandType (extension)
SetStatusCommandType
This SE Command requests a change of the Application Life Cycle State (either a simple Application or a Supplementary Security Domain).
gpm:ApplicationAID
AID of the Application (either a simple Application or a Supplementary Security Domain).
gpm:LockControl
This field specifies the requested state transition. - If True, the application SHALL be locked - If False, the application SHALL be unlocked

Figure 4-101: `SetStatusCommandType` type

Where:

- The `ApplicationAID` element is of type `AIDType` (see section 4.5.1.2).

4.5.5.1.1.8 The "DeleteCommand" SE Commands

The `DeleteCommand` SE Command defined in section 3.5.1.3.2.7 SHALL be mapped to the `DeleteCommandType` type described in the following figure. This `DeleteCommandType` type extends the `SECommandType` type.

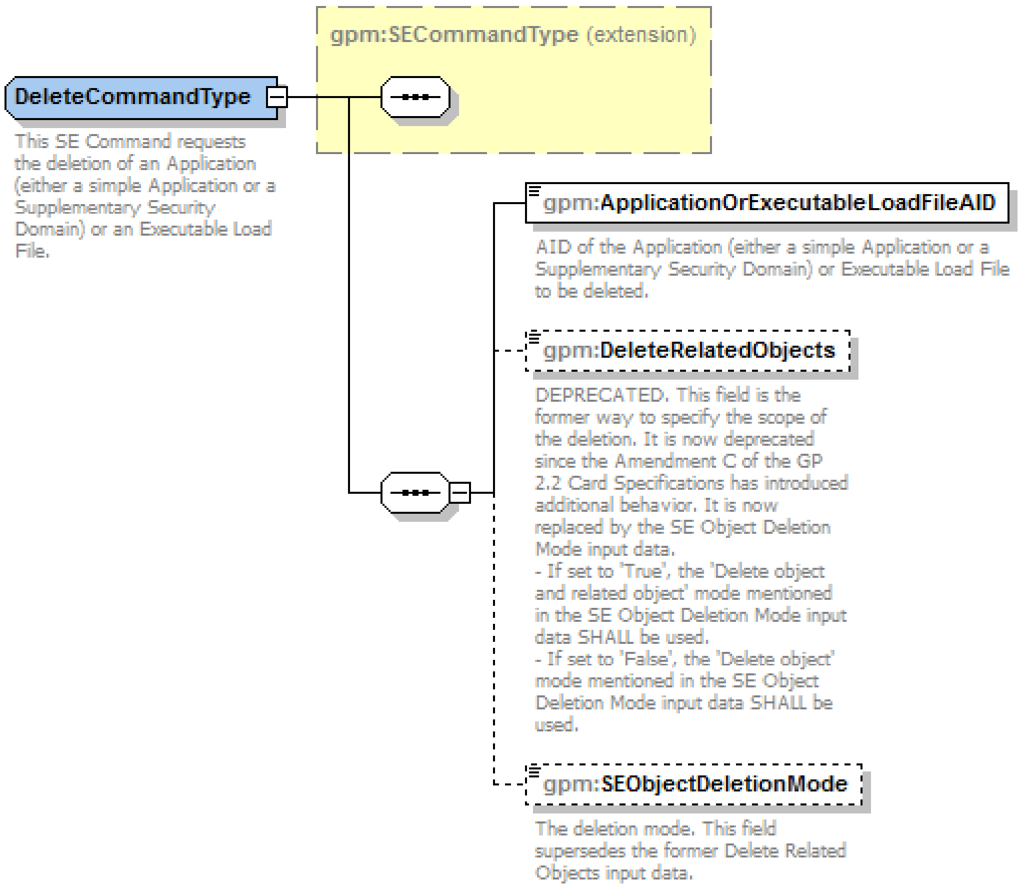

gpm:SECommandType (extension)
DeleteCommandType
This SE Command requests the deletion of an Application (either a simple Application or a Supplementary Security Domain) or an Executable Load File.
gpm:ApplicationOrExecutableLoadFileAID
AID of the Application (either a simple Application or a Supplementary Security Domain) or Executable Load File to be deleted.
gpm:DeleteRelatedObjects
DEPRECATED. This field is the former way to specify the scope of the deletion. It is now deprecated since the Amendment C of the GP 2.2 Card Specifications has introduced additional behavior. It is now replaced by the SE Object Deletion Mode input data.
- If set to 'True', the 'Delete object and related object' mode mentioned in the SE Object Deletion Mode input data SHALL be used.
- If set to 'False', the 'Delete object' mode mentioned in the SE Object Deletion Mode input data SHALL be used.
gpm:SEObjectDeletionMode
The deletion mode. This field supersedes the former Delete Related Objects input data.

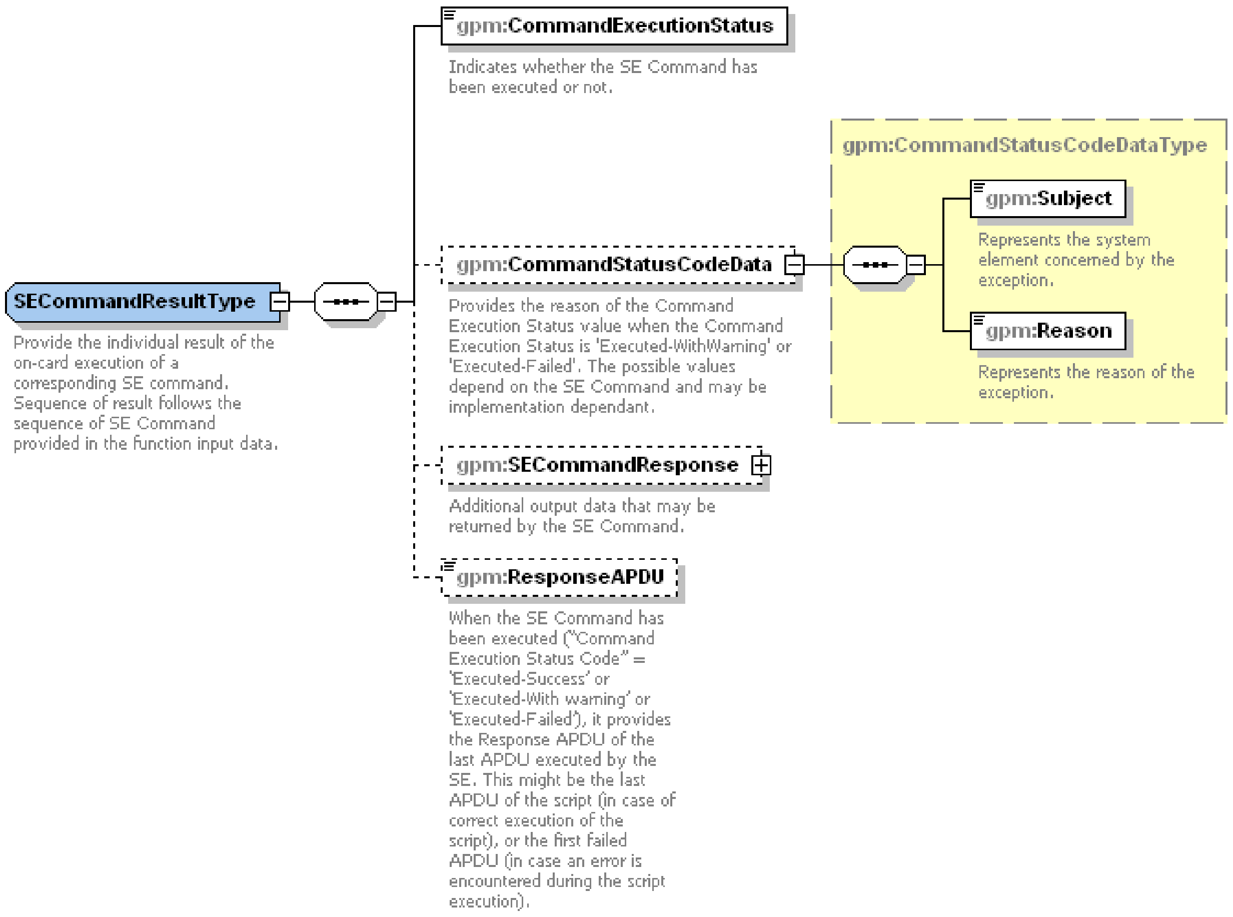
Figure 4-102: `DeleteCommandType` type

Where:

- The `ApplicationOrExecutableLoadFileAID` element is of type `AIDType` (see section 4.5.1.2).
- The `SEObjectDeletionMode` is an enumeration of the possible deletion modes (`SEObjectDeletionModeType`).

4.5.5.1.2 SE Commands Results

The `SECommandResult` defined in section 3.5.1.2 SHALL be mapped to the `SECommandResultType` type described in the following figure:



gpm:CommandExecutionStatus
Indicates whether the SE Command has been executed or not.
gpm:CommandStatusCodeDataType
gpm:Subject
Represents the system element concerned by the exception.
gpm:CommandStatusCodeData
Provides the reason of the Command Execution Status value when the Command Execution Status is 'Executed-WithWarning' or 'Executed-Failed'. The possible values depend on the SE Command and may be implementation dependant.
gpm:Reason
Represents the reason of the exception.
SECommandResultType
Provide the individual result of the on-card execution of a corresponding SE command. Sequence of result follows the sequence of SE Command provided in the function input data.
gpm:SECommandResponse
Additional output data that may be returned by the SE Command.
gpm:ResponseAPDU
When the SE Command has been executed ("Command Execution Status Code" = 'Executed-Success' or 'Executed-With warning' or 'Executed-Failed'), it provides the Response APDU of the last APDU executed by the SE. This might be the last APDU of the script (in case of correct execution of the script), or the first failed APDU (in case an error is encountered during the script execution).

Figure 4-103: `SECommandResultType` type

Where:

- The `CommandStatusCodeData` is an element of type `CommandStatusCodeDataType`.
- The `SECommandResponse` is an element of type `SECommandResponseType`. This `SECommandResponseType` type acts as a base type for the possible SE Command responses defined in section 3.5.1.3. Each concrete SE Command response extends this `SECommandResponseType` type.
- The `ResponseAPDU` element is of type `APDUResponseType` (see section 4.5.1.2).

4.5.5.1.2.1 The "CreateFirstSSDKeysetCommand" SE Commands Response

The `CreateFirstSSDKeysetCommandResponse` SE Command defined in section 3.5.1.3.1.2 SHALL be mapped to the `CreateFirstSSDKeysetCommandResponseType` abstract type described in the following figure. This `CreateFirstSSDKeysetCommandResponseType` type extends the `SECommandResponseType` type.

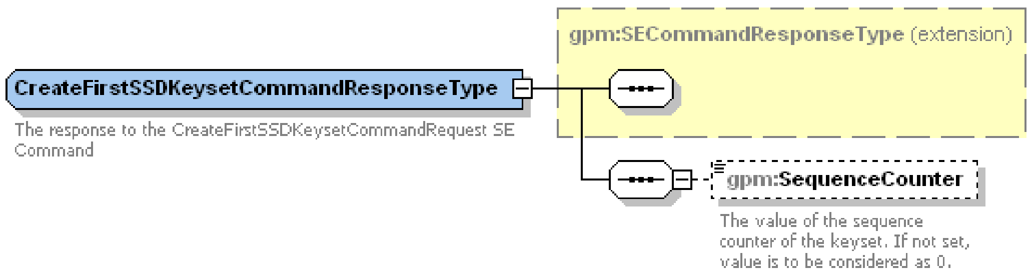

gpm:SECommandResponseType (extension)
CreateFirstSSDKeysetCommandResponseType
The response to the CreateFirstSSDKeysetCommandRequest SE Command
gpm:SequenceCounter
The value of the sequence counter of the keyset. If not set, value is to be considered as 0.

Figure 4-104: `CreateFirstSSDKeysetCommandResponseType` type

The key provisioning output data (KPOD) is mapped through an inheritance mechanism: instead of having a dedicated Key Provisioning Output Data element for a given Xxx key provisioning mode, a `Xxx_CreateFirstSSDKeysetCommandResponseType` type is created instead, by extension of the `CreateFirstSSDKeysetCommandResponseType` abstract type.

Then, in case of Basic Create mode, the concrete `Basic_CreateFirstSSDKeysetCommandResponseType` type SHALL be used (see section 3.5.1.3.1.2).

This `Basic_CreateFirstSSDKeysetCommandResponseType` type extends the `CreateFirstSSDKeysetCommandResponseType` type.

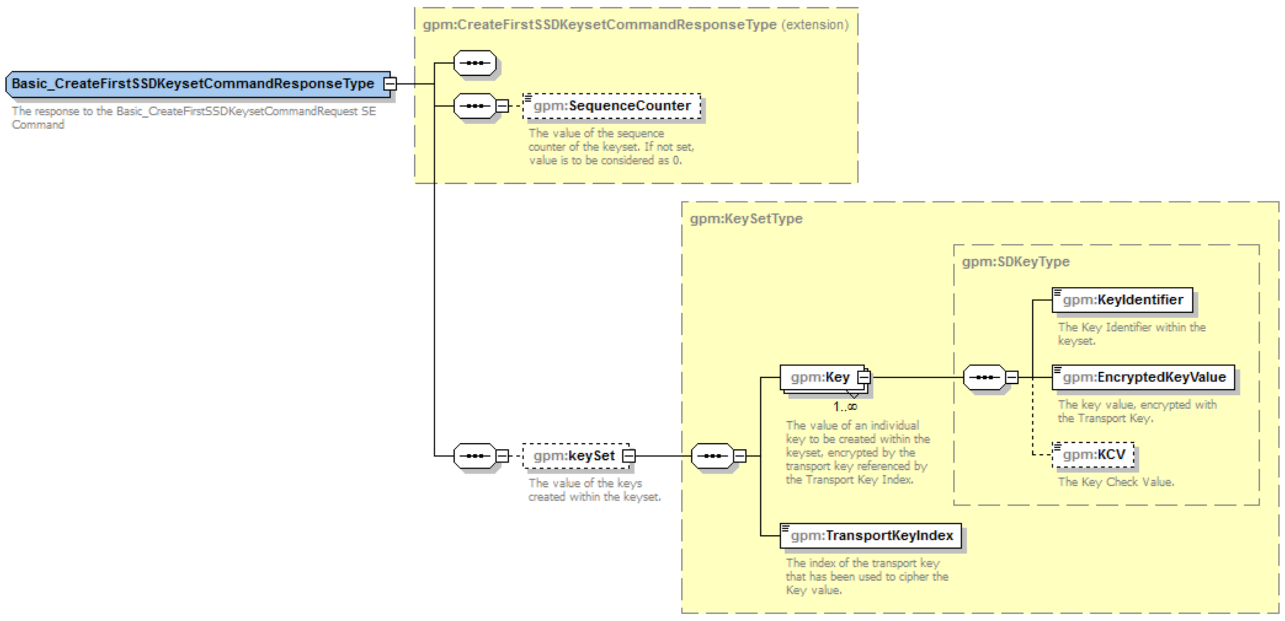

gpm:CreateFirstSSDKeysetCommandResponseType (extension)
Basic_CreateFirstSSDKeysetCommandResponseType
The response to the Basic_CreateFirstSSDKeysetCommandRequest SE Command
gpm:SequenceCounter
The value of the sequence counter of the keyset. If not set, value is to be considered as 0.
gpm:KeySetType
gpm:SDKeyType
gpm:KeyIdentifier
The Key Identifier within the keyset.
gpm:EncryptedKeyValue
The key value, encrypted with the Transport Key.
gpm:KCV
The Key Check Value.
gpm:Key
1..∞
The value of an individual key to be created within the keyset, encrypted by the transport key referenced by the Transport Key Index.
gpm:keySet
The value of the keys created within the keyset.
gpm:TransportKeyIndex
The index of the transport key that has been used to cipher the Key value.

Figure 4-105: `Basic_CreateFirstSSDKeysetCommandResponseType` type

In case of Basic Random Create mode, the concrete `BasicRandom_CreateFirstSSDKeysetCommandResponseType` type SHALL be used (see section 3.5.1.3.1.2).

This `BasicRandom_CreateFirstSSDKeysetCommandResponseType` type extends the `CreateFirstSSDKeysetCommandResponseType` type.

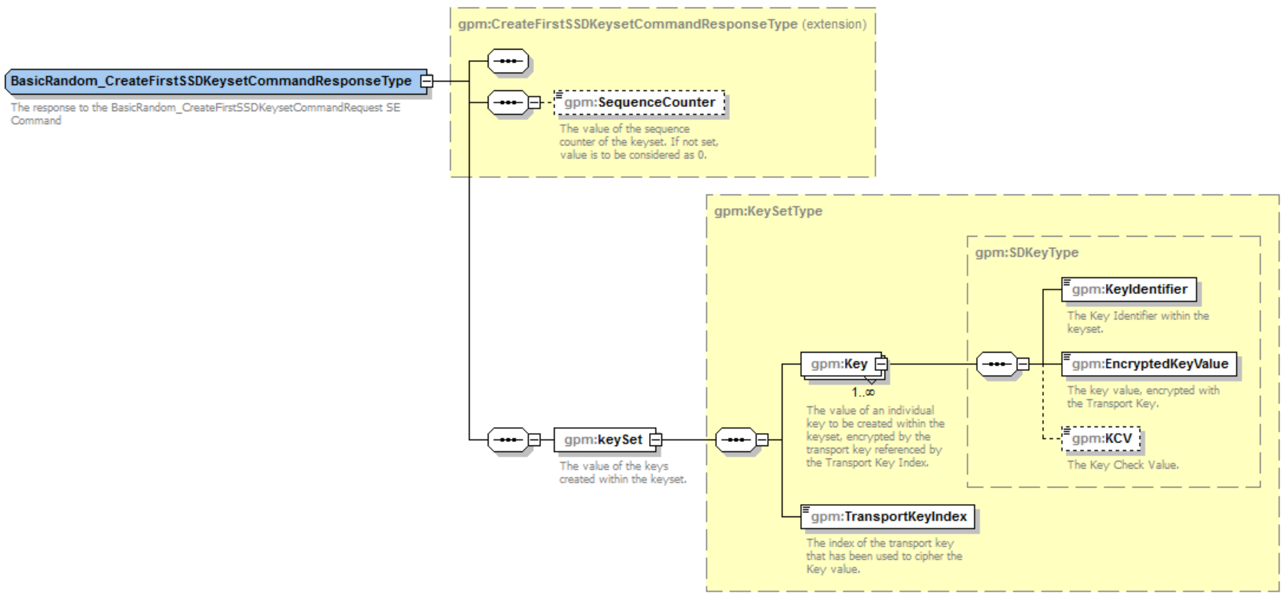

gpm:CreateFirstSSDKeysetCommandResponseType (extension)
BasicRandom_CreateFirstSSDKeysetCommandResponseType
The response to the BasicRandom_CreateFirstSSDKeysetCommandRequest SE Command
gpm:SequenceCounter
The value of the sequence counter of the keyset. If not set, value is to be considered as 0.
gpm:KeySetType
gpm:SDKeyType
gpm:KeyIdentifier
The Key Identifier within the keyset.
gpm:Key
1..∞
The value of an individual key to be created within the keyset, encrypted by the transport key referenced by the Transport Key Index.
gpm:EncryptedKeyValue
The key value, encrypted with the Transport Key.
gpm:keySet
The value of the keys created within the keyset.
gpm:KCV
The Key Check Value.
gpm:TransportKeyIndex
The index of the transport key that has been used to cipher the Key value.

Figure 4-106: `BasicRandom_CreateFirstSSDKeysetCommandResponseType` type

In case of Basic Diversified Create mode, the concrete `BasicDiversified_CreateFirstSSDKeysetCommandResponseType` type SHALL be used (see section 3.5.1.3.1.2).

This `BasicDiversified_CreateFirstSSDKeysetCommandResponseType` type extends the `CreateFirstSSDKeysetCommandResponseType` type.

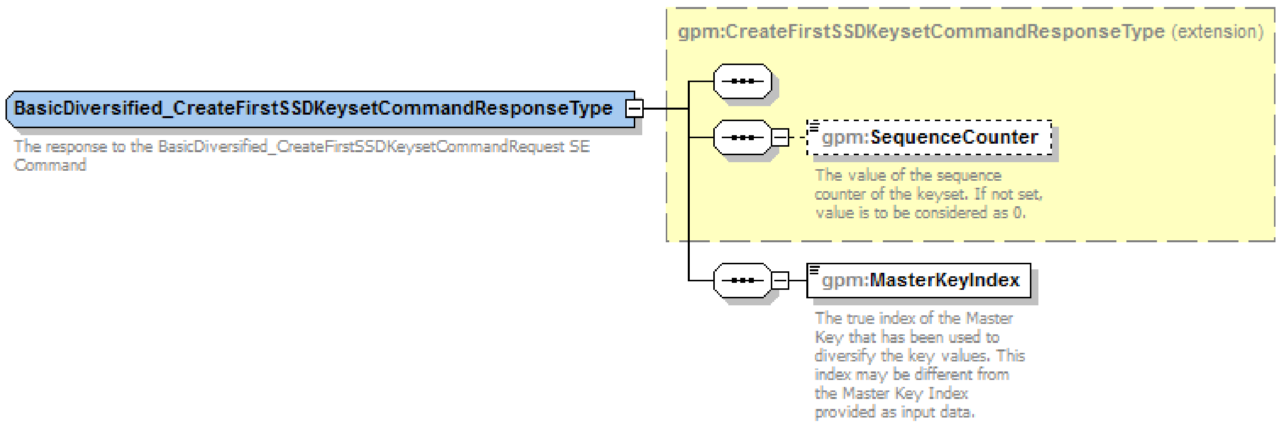

gpm:CreateFirstSSDKeysetCommandResponseType (extension)
BasicDiversified_CreateFirstSSDKeysetCommandResponseType
The response to the BasicDiversified_CreateFirstSSDKeysetCommandRequest SE Command
gpm:SequenceCounter
The value of the sequence counter of the keyset. If not set, value is to be considered as 0.
gpm:MasterKeyIndex
The true index of the Master Key that has been used to diversify the key values. This index may be different from the Master Key Index provided as input data.

Figure 4-107: `BasicDiversified_CreateFirstSSDKeysetCommandResponseType` type

In case of CCCM Scenario #1, using PK scheme mode, the concrete `CCCMScenario1UsingPK_CreateFirstSSDKeysetCommandResponseType` type SHALL be used (see section 3.5.1.3.1.2).

This `CCCMScenario1UsingPK_CreateFirstSSDKeysetCommandResponseType` type extends the `CreateFirstSSDKeysetCommandResponseType` type.

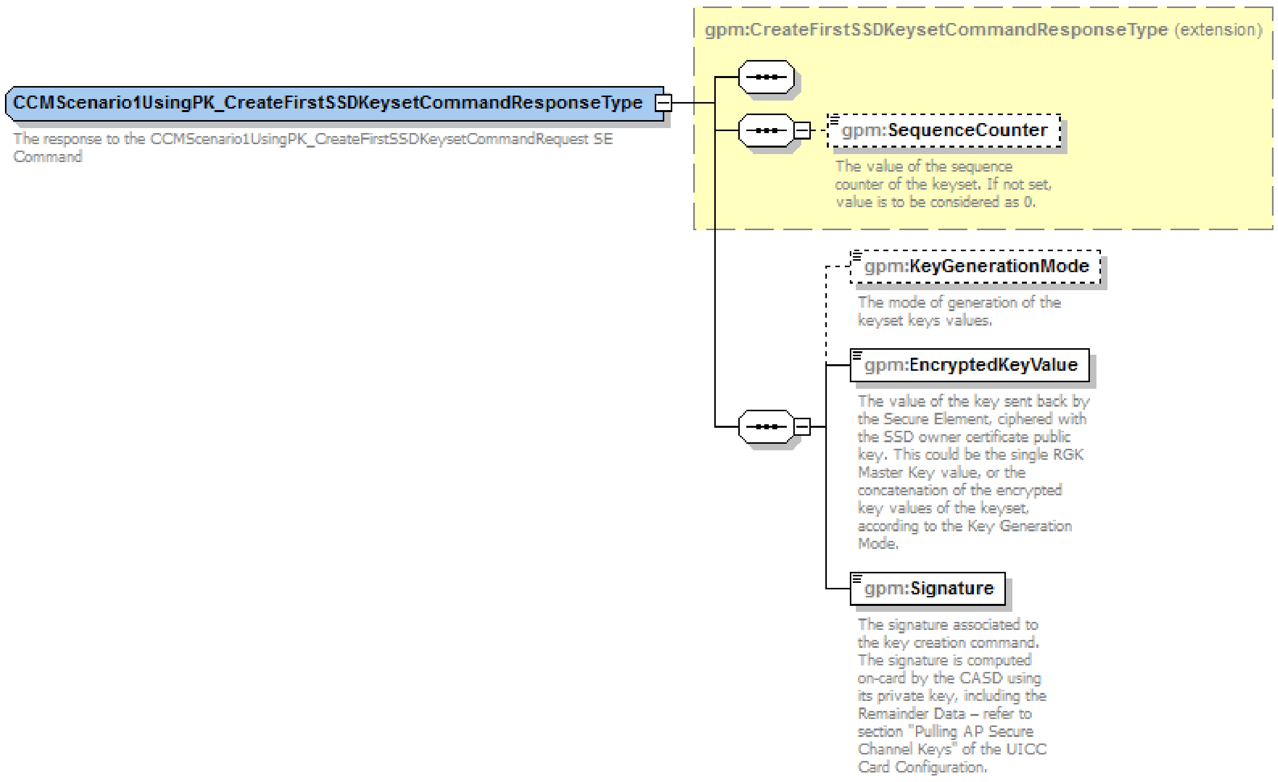

gpm:CreateFirstSSDKeysetCommandResponseType (extension)
CCMScenario1UsingPK_CreateFirstSSDKeysetCommandResponseType
The response to the CCMScenario1UsingPK_CreateFirstSSDKeysetCommandRequest SE Command
gpm:SequenceCounter
The value of the sequence counter of the keyset. If not set, value is to be considered as 0.
gpm:KeyGenerationMode
The mode of generation of the keyset keys values.
gpm:EncryptedKeyValue
The value of the key sent back by the Secure Element, ciphered with the SSD owner certificate public key. This could be the single RGK Master Key value, or the concatenation of the encrypted key values of the keyset, according to the Key Generation Mode.
gpm:Signature
The signature associated to the key creation command. The signature is computed on-card by the CASD using its private key, including the Remainder Data – refer to section "Pulling AP Secure Channel Keys" of the UICC Card Configuration.

Figure 4-108: `CCCMScenario1UsingPK_CreateFirstSSDKeysetCommandResponseType` type

Where:

- The `KeyGenerationMode` is an enumeration of the possible key generation modes (`KeyGenerationModeType`).

In case of CCCM Scenario #2A mode, the concrete `CCCMScenario2A_CreateFirstSSDKeysetCommandResponseType` type SHALL be used (see section 3.5.1.3.1.2).

This `CCCMScenario2A_CreateFirstSSDKeysetCommandResponseType` type extends the `CreateFirstSSDKeysetCommandResponseType` type.

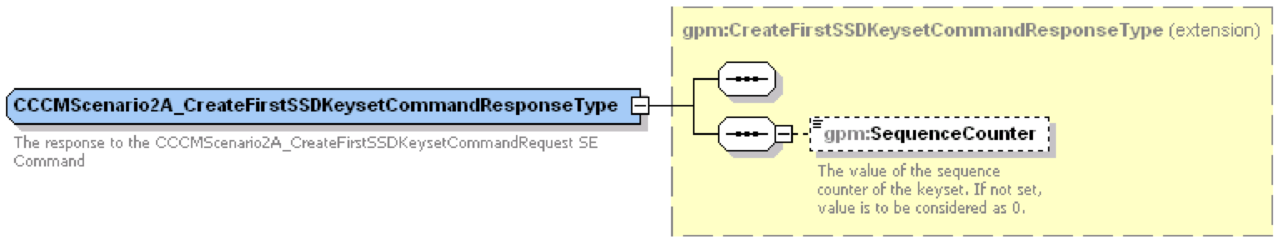

gpm:CreateFirstSSDKeysetCommandResponseType (extension)
CCCMScenario2A_CreateFirstSSDKeysetCommandResponseType
The response to the CCCMScenario2A_CreateFirstSSDKeysetCommandRequest SE Command
gpm:SequenceCounter
The value of the sequence counter of the keyset. If not set, value is to be considered as 0.

Figure 4-109: `CCCMScenario2A_CreateFirstSSDKeysetCommandResponseType` type

In case of CCCM Scenario #2B mode, the concrete `CCCMScenario2B_CreateFirstSSDKeysetCommandType` type SHALL be used (see section 3.5.1.3.1.2).

This `CCCMScenario2B_CreateFirstSSDKeysetCommandResponseType` type extends the `CreateFirstSSDKeysetCommandResponseType` type.

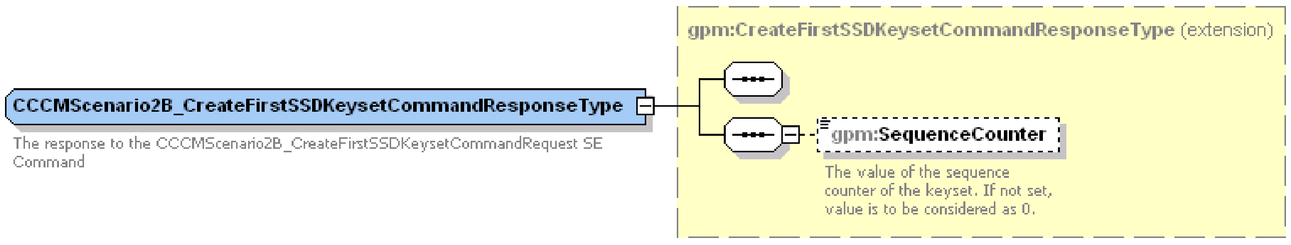

gpm:CreateFirstSSDKeysetCommandResponseType (extension)
CCCMScenario2B_CreateFirstSSDKeysetCommandResponseType
The response to the CCCMScenario2B_CreateFirstSSDKeysetCommandRequest SE Command
gpm:SequenceCounter
The value of the sequence counter of the keyset. If not set, value is to be considered as 0.

Figure 4-110: `CCCMScenario2B_CreateFirstSSDKeysetCommandType` type

In case of CCCM Scenario #3, the concrete `CCCMScenario3_CreateFirstSSDKeysetCommandResponseType` type SHALL be used (see section 3.5.1.3.1.2).

This `CCCMScenario3_CreateFirstSSDKeysetCommandResponseType` type extends the `CreateFirstSSDKeysetCommandResponseType` type.

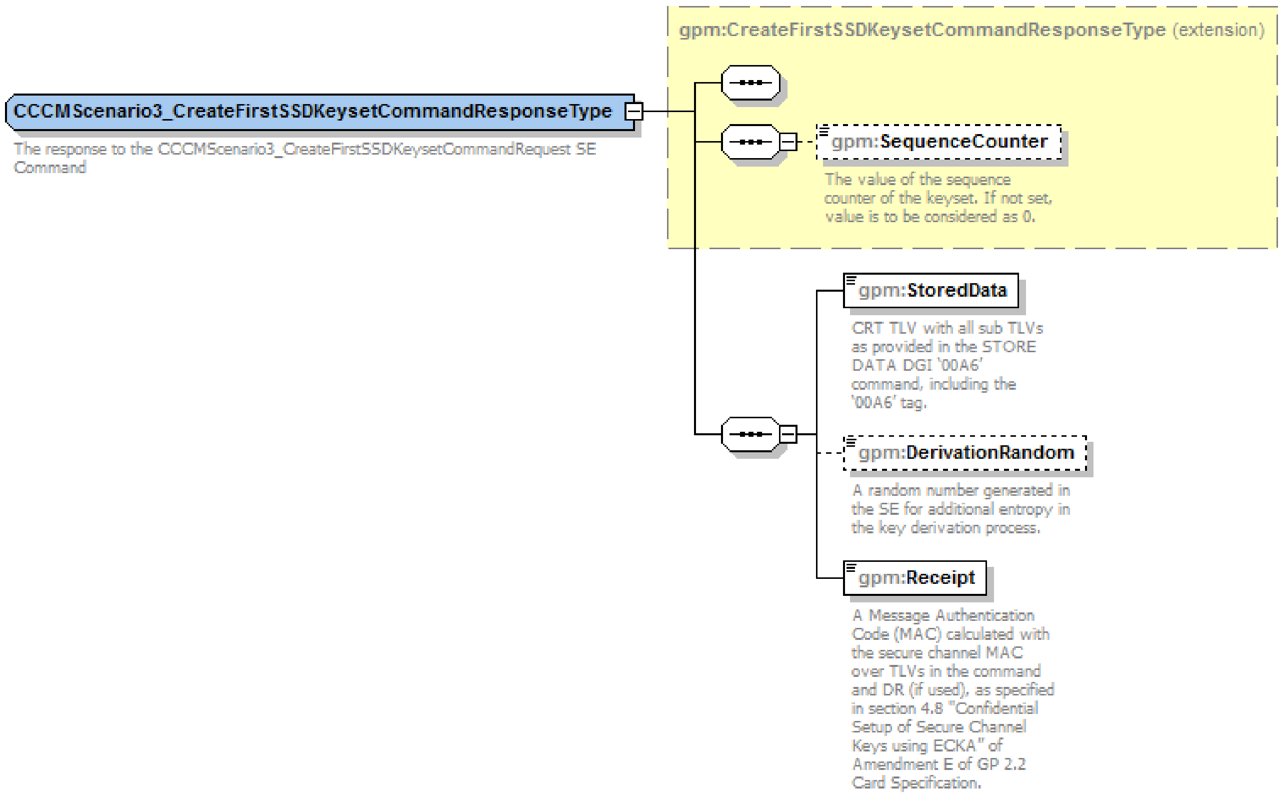

gpm:CreateFirstSSDKeysetCommandResponseType (extension)
CCCMScenario3_CreateFirstSSDKeysetCommandResponseType
The response to the CCCMScenario3_CreateFirstSSDKeysetCommandRequest SE Command
gpm:SequenceCounter
The value of the sequence counter of the keyset. If not set, value is to be considered as 0.
gpm:StoredData
CRT TLV with all sub TLVs as provided in the STORE DATA DGI '00A6' command, including the '00A6' tag.
gpm:DerivationRandom
A random number generated in the SE for additional entropy in the key derivation process.
gpm:Receipt
A Message Authentication Code (MAC) calculated with the secure channel MAC over TLVs in the command and DR (if used), as specified in section 4.8 "Confidential Setup of Secure Channel Keys using ECKA" of Amendment E of GP 2.2 Card Specification.

Figure 4-111: `CCCMScenario3_CreateFirstSSDKeysetCommandResponseType` type

4.5.5.1.2.2 The "InstantiateApplicationCommand" SE Commands Response

The `InstantiateApplicationCommandResponse` SE Command defined in section 3.5.1.3.2.3 SHALL be mapped to the `InstantiateApplicationCommandResponseType` type described in the following figure. This `InstantiateApplicationCommandResponseType` type extends the `SECommandResponseType` type.

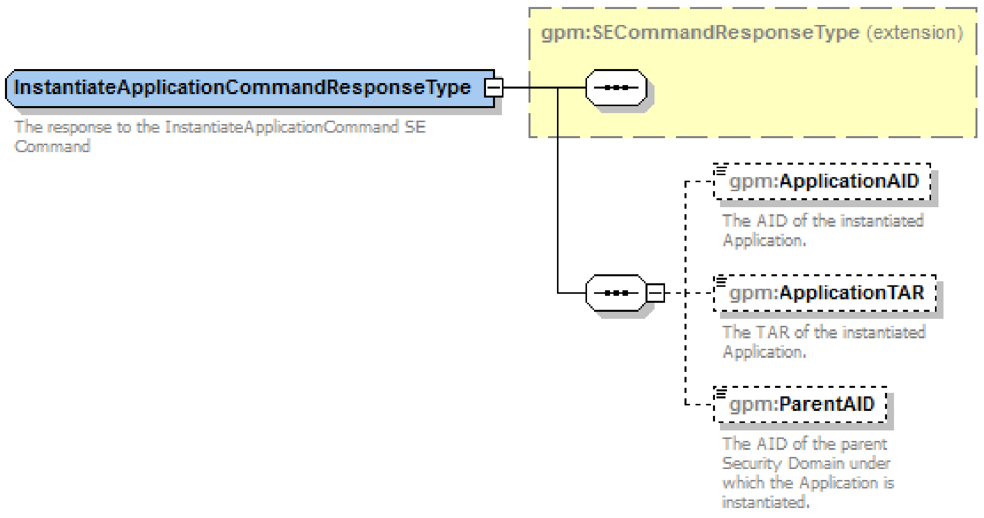

gpm:SECommandResponseType (extension)
InstantiateApplicationCommandResponseType
The response to the InstantiateApplicationCommand SE Command
gpm:ApplicationAID
The AID of the instantiated Application.
gpm:ApplicationTAR
The TAR of the instantiated Application.
gpm:ParentAID
The AID of the parent Security Domain under which the Application is instantiated.

Figure 4-112: `InstantiateApplicationCommandResponseType` type

Where:

- The `ApplicationAID` element is of type `AIDType` (see section 4.5.1.2).
- The `TAR` element is of type `TARType` (see section 4.5.1.2).
- The `ParentAID` element is of type `AIDType` (see section 4.5.1.2).

4.5.5.2 Delegated Management Functions

4.5.5.2.1 The "GenerateDMToken" Function

The input data for the `GenerateDMToken` function defined in section 3.5.2.1 SHALL be mapped to the `GenerateDMTokenRequest` element that is of type `GenerateDMTokenRequestType` described in the following figure:

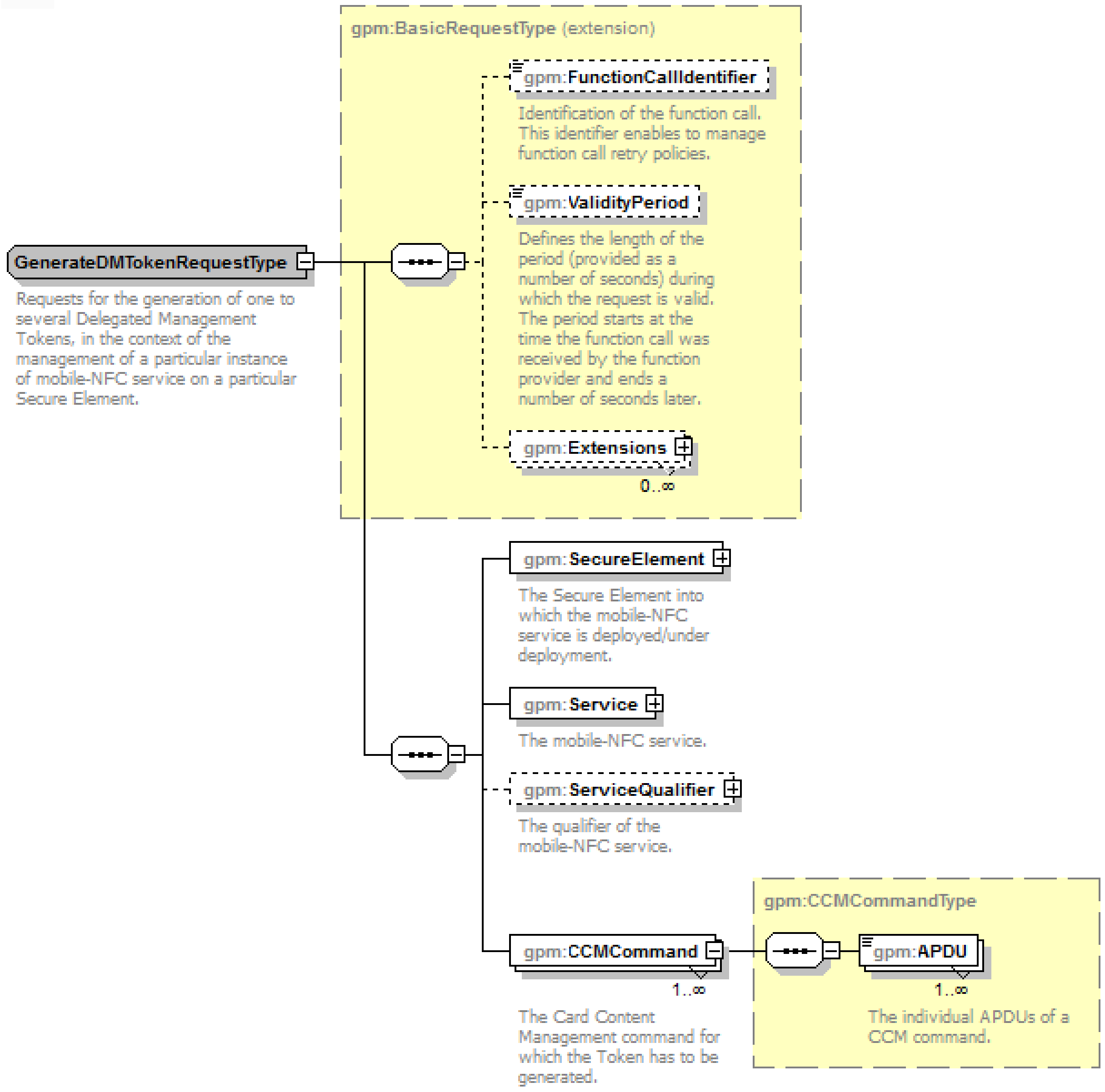

gpm:BasicRequestType (extension)
gpm:FunctionCallIdentifier
Identification of the function call. This identifier enables to manage function call retry policies.
gpm:ValidityPeriod
Defines the length of the period (provided as a number of seconds) during which the request is valid. The period starts at the time the function call was received by the function provider and ends a number of seconds later.
gpm:Extensions
0..∞
GenerateDMTokenRequestType
Requests for the generation of one to several Delegated Management Tokens, in the context of the management of a particular instance of mobile-NFC service on a particular Secure Element.
gpm:SecureElement
The Secure Element into which the mobile-NFC service is deployed/under deployment.
gpm:Service
The mobile-NFC service.
gpm:ServiceQualifier
The qualifier of the mobile-NFC service.
gpm:CCMCommandType
gpm:CCMCommand
1..∞
The Card Content Management command for which the Token has to be generated.
gpm:APDU
1..∞
The individual APDUs of a CCM command.

Figure 4-113: `GenerateDMTokenRequestType` type

Where:

- The `SecureElement` element is of type `SEIdentifierType` (see section 4.5.1.2).
- The `Service` element is of type `ServiceIdentifierType` (see section 4.5.1.2).
- The `ServiceQualifier` element is of type `ServiceQualifierType` (see section 4.5.1.2).
- The `APDU` element is of type `APDUType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "GenerateDMTokenRequest".

The output data of the `GenerateDMToken` function defined in section 3.5.2.1 SHALL be mapped to the `GenerateDMTokenResponse` element that is of type `GenerateDMTokenResponseType` described in the following figure:

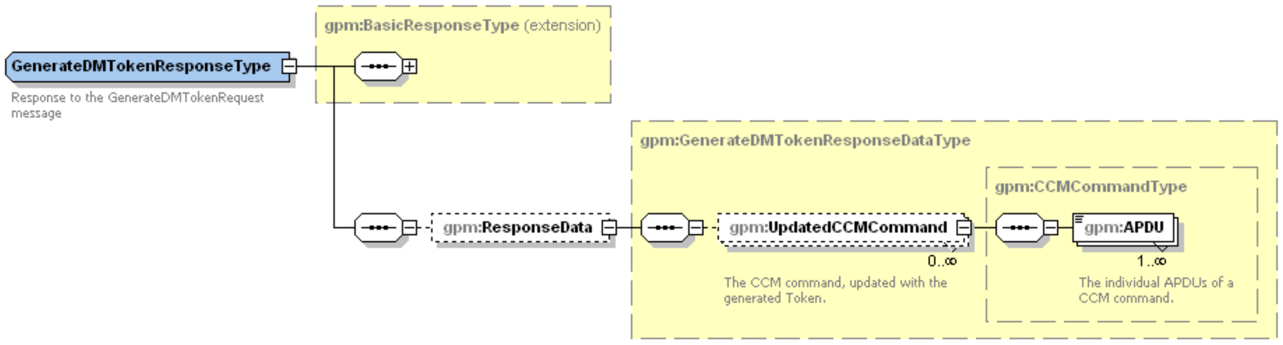

gpm:BasicResponseType (extension)
GenerateDMTokenResponseType
Response to the GenerateDMTokenRequest message
gpm:GenerateDMTokenResponseDataType
gpm:CCMCommandType
gpm:ResponseData
gpm:UpdatedCCMCommand
0..∞
The CCM command, updated with the generated Token.
gpm:APDU
1..∞
The individual APDUs of a CCM command.

Figure 4-114: `GenerateDMTokenResponseType` type

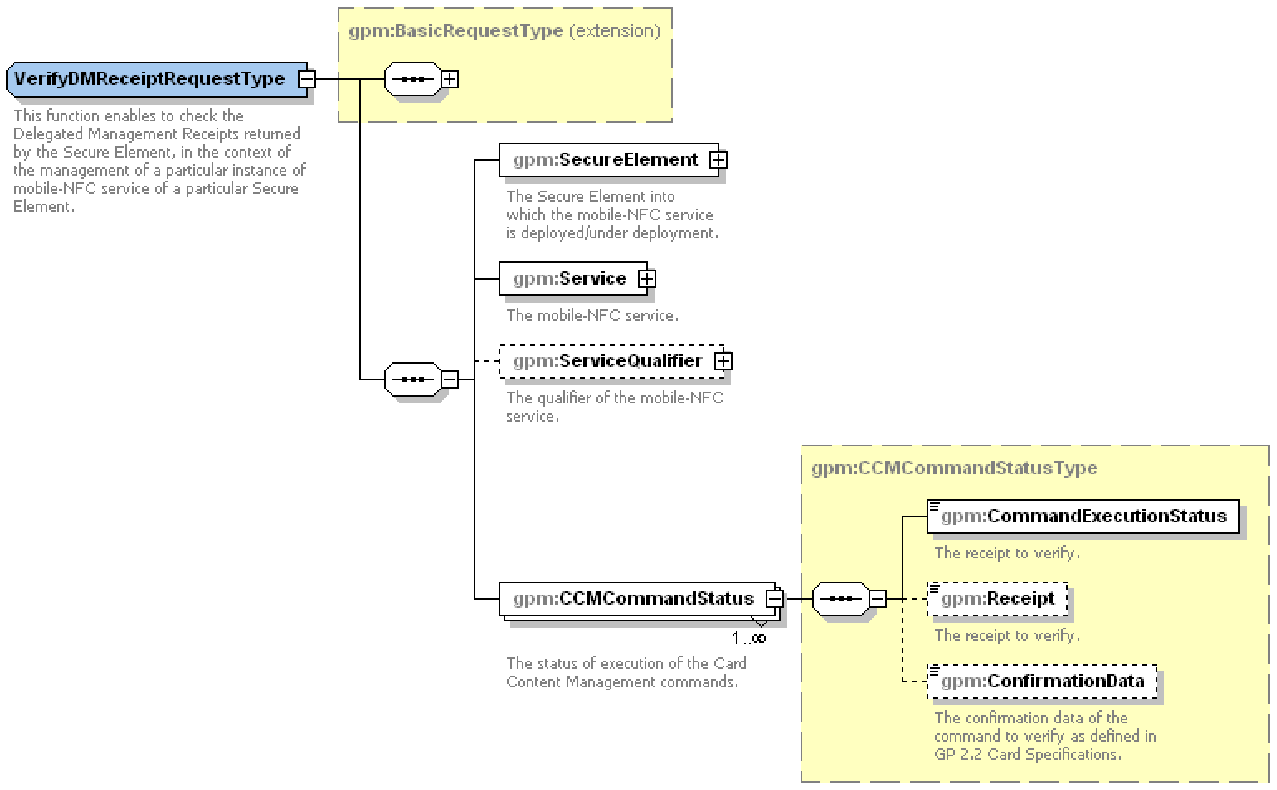
Where:

- The `APDU` element is of type `APDUType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "GenerateDMTokenResponse".

4.5.5.2.2 The "VerifyDMReceipt" Function

The input data for the `VerifyDMReceipt` function defined in section 3.5.2.2 SHALL be mapped to the `VerifyDMReceiptRequest` element that is of type `VerifyDMReceiptRequestType` described in the following figure:



gpm:BasicRequestType (extension)
VerifyDMReceiptRequestType
This function enables to check the Delegated Management Receipts returned by the Secure Element, in the context of the management of a particular instance of mobile-NFC service of a particular Secure Element.
gpm:SecureElement
The Secure Element into which the mobile-NFC service is deployed/under deployment.
gpm:Service
The mobile-NFC service.
gpm:ServiceQualifier
The qualifier of the mobile-NFC service.
gpm:CCMCommandStatusType
gpm:CommandExecutionStatus
The receipt to verify.
gpm:CCMCommandStatus
1..∞
The status of execution of the Card Content Management commands.
gpm:Receipt
The receipt to verify.
gpm:ConfirmationData
The confirmation data of the command to verify as defined in GP 2.2 Card Specifications.

Figure 4-115: `VerifyDMReceiptRequestType` type

Where:

- The `SecureElement` element is of type `SEIdentifierType` (see section 4.5.1.2).
- The `Service` element is of type `ServiceIdentifierType` (see section 4.5.1.2).
- The `ServiceQualifier` element is of type `ServiceQualifierType` (see section 4.5.1.2).
- The `CommandExecutionStatus` element is of type `CommandExecutionStatusType` (see section 4.5.1.4).

The value of the `GPHeader.Type` associated to this element SHALL be "VerifyDMReceiptRequest".

The output data of the `VerifyDMReceipt` function defined in section 3.5.2.2 SHALL be mapped to the `VerifyDMReceiptResponse` element that is of type `VerifyDMReceiptResponseType` described in the following figure:

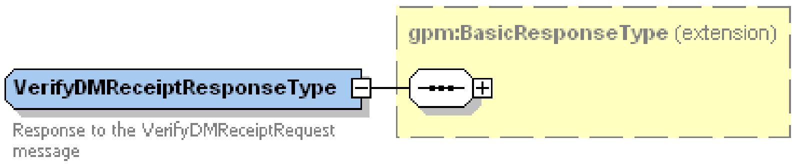

gpm:BasicResponseType (extension)
VerifyDMReceiptResponseType
Response to the VerifyDMReceiptRequest message

Figure 4-116: `VerifyDMReceiptResponseType` type

The value of the `GPHeader.Type` associated to this element SHALL be "VerifyDMReceiptResponse".

4.5.5.3 Scripts Sending Functions

4.5.5.3.1 The "BeginConversation" Function

The input data for the `BeginConversation` function defined in section 3.5.3.1 SHALL be mapped to the `BeginConversationRequest` element that is of type `BeginConversationRequestType` described in the following figure:

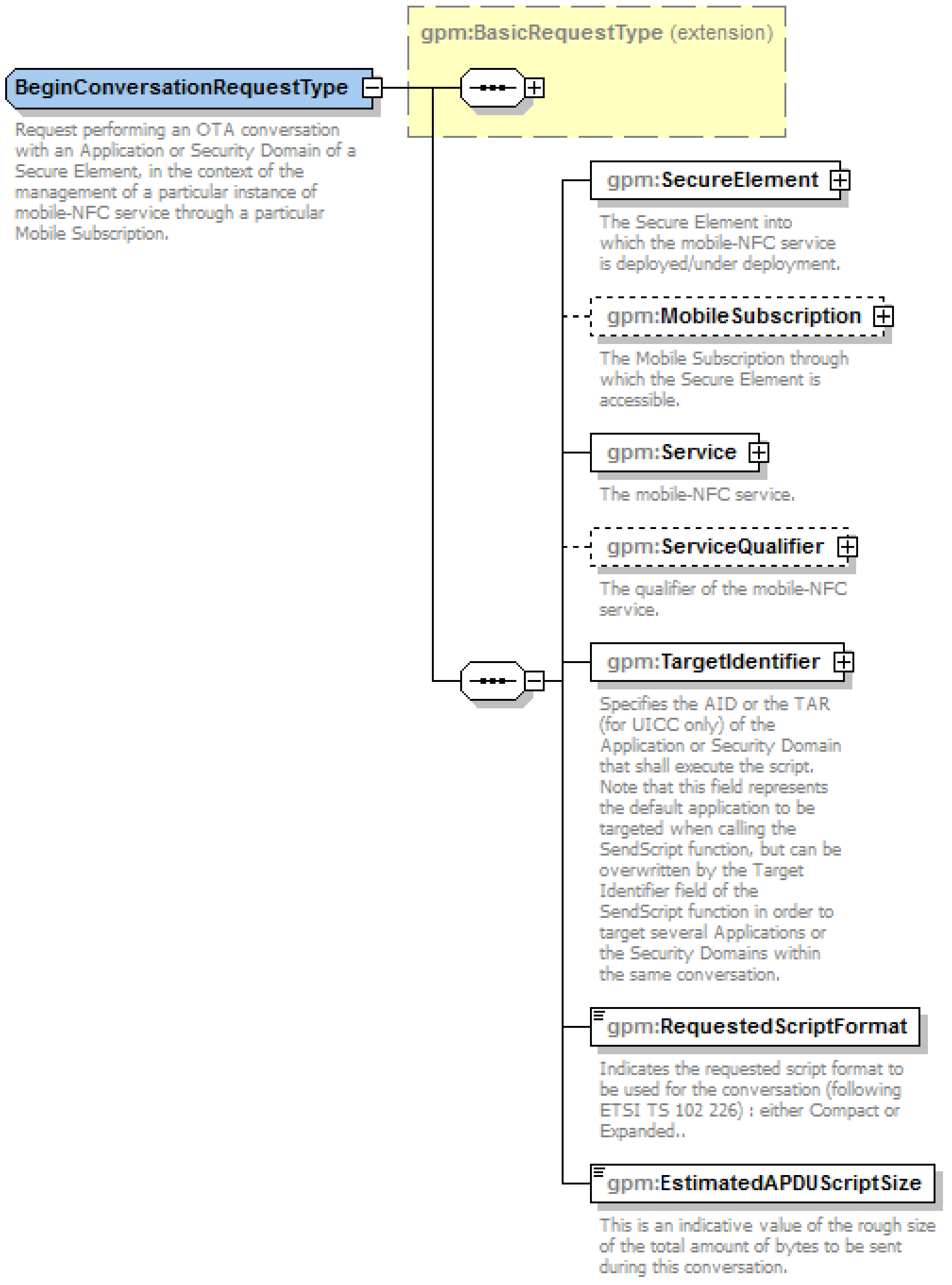

gpm:BasicRequestType (extension)
BeginConversationRequestType
Request performing an OTA conversation with an Application or Security Domain of a Secure Element, in the context of the management of a particular instance of mobile-NFC service through a particular Mobile Subscription.
gpm:SecureElement
The Secure Element into which the mobile-NFC service is deployed/under deployment.
gpm:MobileSubscription
The Mobile Subscription through which the Secure Element is accessible.
gpm:Service
The mobile-NFC service.
gpm:ServiceQualifier
The qualifier of the mobile-NFC service.
gpm:TargetIdentifier
Specifies the AID or the TAR (for UICC only) of the Application or Security Domain that shall execute the script. Note that this field represents the default application to be targeted when calling the SendScript function, but can be overwritten by the Target Identifier field of the SendScript function in order to target several Applications or the Security Domains within the same conversation.
gpm:RequestedScriptFormat
Indicates the requested script format to be used for the conversation (following ETSI TS 102 226) : either Compact or Expanded..
gpm:EstimatedAPDUScriptSize
This is an indicative value of the rough size of the total amount of bytes to be sent during this conversation.

Figure 4-117: `BeginConversationRequestType` type

Where:

- The `MobileSubscription` element is of type `MobileSubscriptionIdentifierType` (see section 4.5.1.2).
- The `SecureElement` element is of type `SEIdentifierType` (see section 4.5.1.2).
- The `Service` element is of type `ServiceIdentifierType` (see section 4.5.1.2).
- The `ServiceQualifier` element is of type `ServiceQualifierType` (see section 4.5.1.2).
- The `TargetIdentifier` element is of type `SEApplicationIdentifierType` (see section 4.5.1.2)

The value of the `GPHeader.Type` associated to this element SHALL be "BeginConversationRequest".

The output data of the `BeginConversation` function defined in section 3.5.3.1 SHALL be mapped to the `BeginConversationResponse` element that is of type `BeginConversationResponseType` described in the following figure:

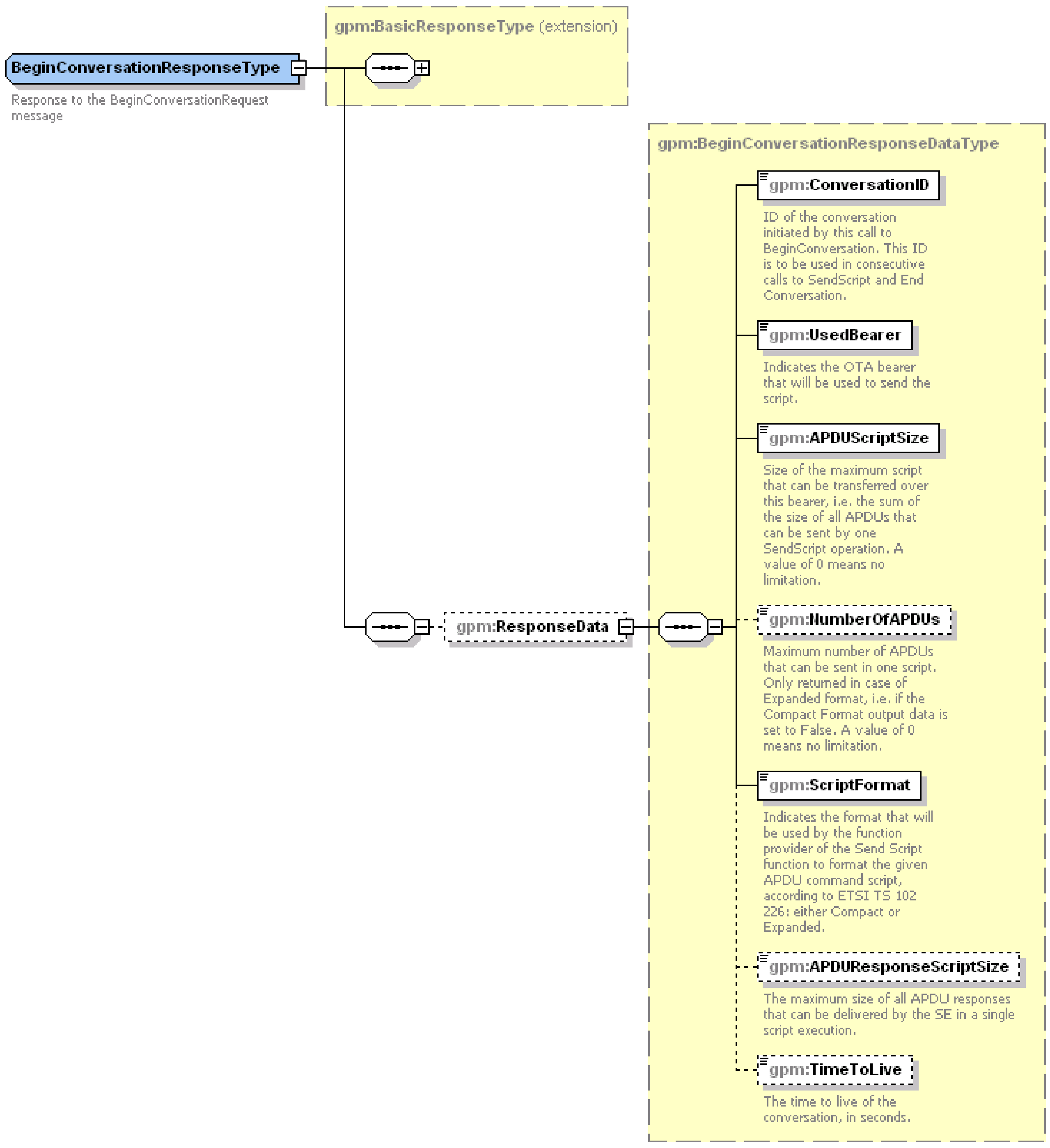

gpm:BasicResponseType (extension)
BeginConversationResponseType
Response to the BeginConversationRequest message
gpm:ResponseData
gpm:BeginConversationResponseDataType
gpm:ConversationID
ID of the conversation initiated by this call to BeginConversation. This ID is to be used in consecutive calls to SendScript and End Conversation.
gpm:UsedBearer
Indicates the OTA bearer that will be used to send the script.
gpm:APDUScriptSize
Size of the maximum script that can be transferred over this bearer, i.e. the sum of the size of all APDUs that can be sent by one SendScript operation. A value of 0 means no limitation.
gpm:NumberOfAPDUs
Maximum number of APDUs that can be sent in one script. Only returned in case of Expanded format, i.e. if the Compact Format output data is set to False. A value of 0 means no limitation.
gpm:ScriptFormat
Indicates the format that will be used by the function provider of the Send Script function to format the given APDU command script, according to ETSI TS 102 226: either Compact or Expanded.
gpm:APDUResponseScriptSize
The maximum size of all APDU responses that can be delivered by the SE in a single script execution.
gpm:TimeToLive
The time to live of the conversation, in seconds.

Figure 4-118: `BeginConversationResponseType` type

The value of the `GPHeader.Type` associated to this element SHALL be "BeginConversationResponse".

4.5.5.3.2 The "SendScript" Function

The input data for the `SendScript` function defined in section 3.5.3.2 SHALL be mapped to the `SendScriptRequest` element that is of type `SendScriptRequestType` described in the following figure:

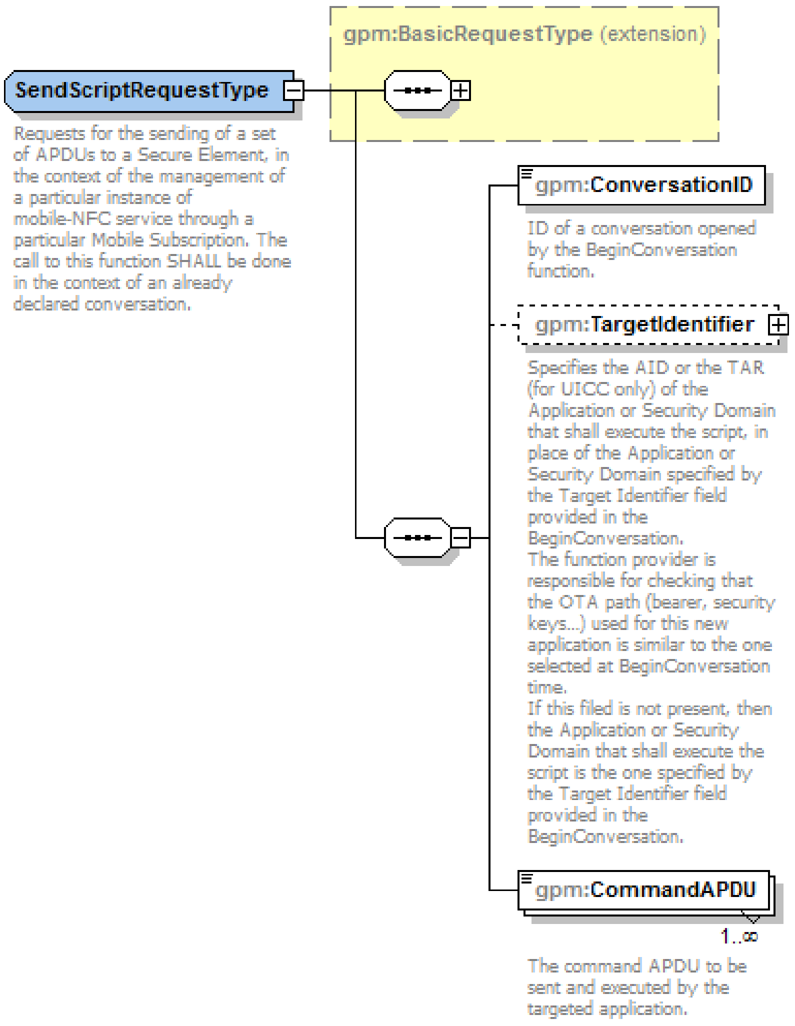

gpm:BasicRequestType (extension)
SendScriptRequestType
Requests for the sending of a set of APDUs to a Secure Element, in the context of the management of a particular instance of mobile-NFC service through a particular Mobile Subscription. The call to this function SHALL be done in the context of an already declared conversation.
gpm:ConversationID
ID of a conversation opened by the BeginConversation function.
gpm:TargetIdentifier
Specifies the AID or the TAR (for UICC only) of the Application or Security Domain that shall execute the script, in place of the Application or Security Domain specified by the Target Identifier field provided in the BeginConversation.
The function provider is responsible for checking that the OTA path (bearer, security keys...) used for this new application is similar to the one selected at BeginConversation time.
If this filed is not present, then the Application or Security Domain that shall execute the script is the one specified by the Target Identifier field provided in the BeginConversation.
gpm:CommandAPDU
1..∞
The command APDU to be sent and executed by the targeted application.

Figure 4-119: `SendScriptRequestType` type

Where;

- The `TargetIdentifier` element is of type `SEApplicationIdentifierType` (see section 4.5.1.2)
- The `CommandAPDU` element is of type `APDUType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "SendScriptRequest".

The output data of the `SendScript` function defined in section 3.5.3.2 SHALL be mapped to the `SendScriptResponse` element that is of type `SendScriptResponseType` described in the following figure:

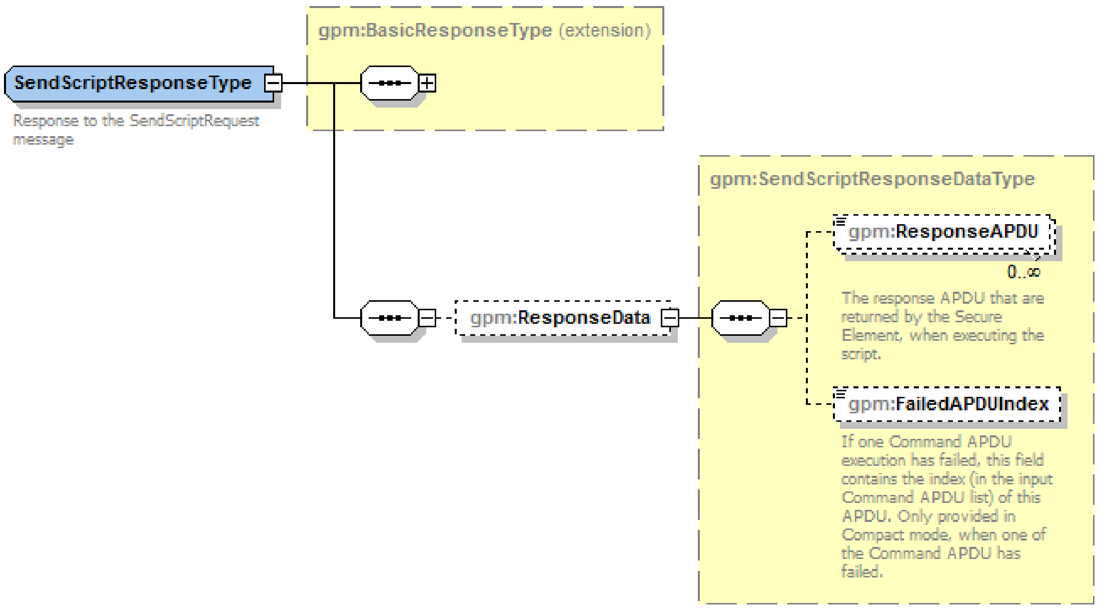

gpm:BasicResponseType (extension)
SendScriptResponseType
Response to the SendScriptRequest message
gpm:SendScriptResponseDataType
gpm:ResponseAPDU
0..∞
The response APDU that are returned by the Secure Element, when executing the script.
gpm:ResponseData
gpm:FailedAPDUIndex
If one Command APDU execution has failed, this field contains the index (in the input Command APDU list) of this APDU. Only provided in Compact mode, when one of the Command APDU has failed.

Figure 4-120: `SendScriptResponseType` type

Where:

- The `ResponseAPDU` element is of type `APDUResponseType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "SendScriptResponse".

4.5.5.3.3 The "EndConversation" Function

The input data for the `EndConversation` function defined in section 3.5.3.3 SHALL be mapped to the `EndConversation` Request element that is of type `EndConversationRequestType` described in the following figure:

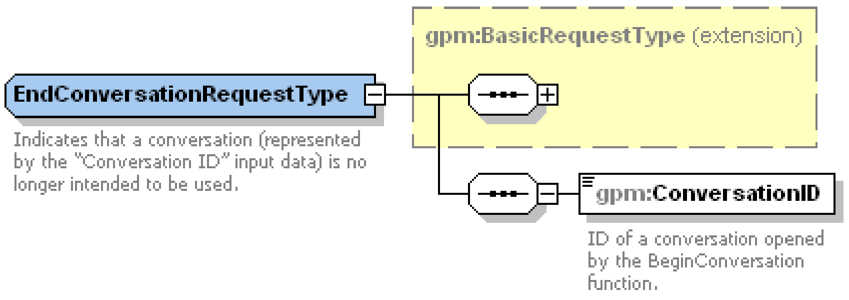

gpm:BasicRequestType (extension)
EndConversationRequestType
Indicates that a conversation (represented by the "Conversation ID" input data) is no longer intended to be used.
gpm:ConversationID
ID of a conversation opened by the BeginConversation function.

Figure 4-121: `EndConversationRequestType` type

The value of the `GPHeader.Type` associated to this element SHALL be "EndConversationRequest".

The output data of the `EndConversation` function defined in section 3.5.3.3 SHALL be mapped to the `EndConversationResponse` element that is of type `EndConversationResponseType` described in the following figure:

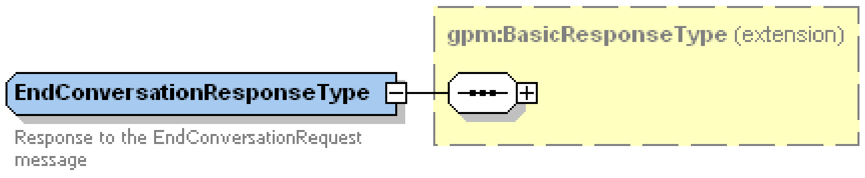

gpm:BasicResponseType (extension)
EndConversationResponseType
Response to the EndConversationRequest message

Figure 4-122: `EndConversationResponseType` type

The value of the `GPHeader.Type` associated to this element SHALL be "EndConversationResponse".

4.5.5.4 SE Audit Functions

4.5.5.4.1 The "GetApplicationOrELFStatus" Function

The input data for the `GetApplicationOrELFStatus` function defined in section 3.5.4.1 SHALL be mapped to the `GetApplicationOrELFStatusRequest` element that is of type `GetApplicationOrELFStatusRequestType` described in the following figure:

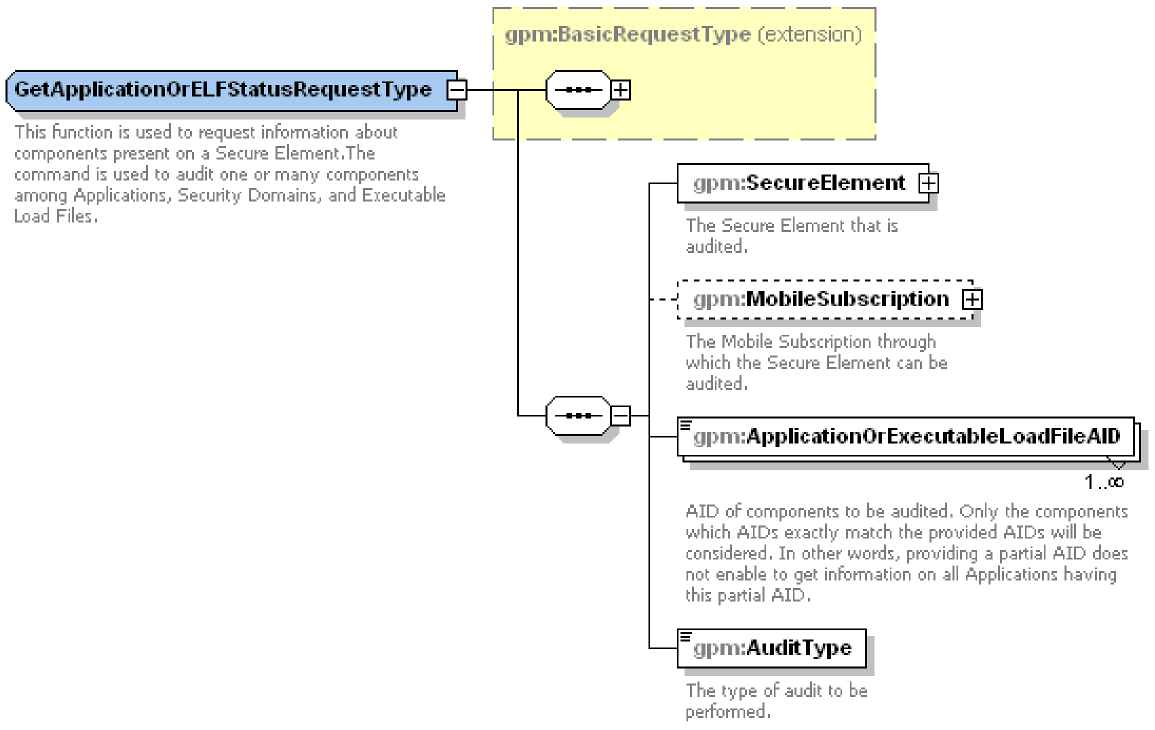

gpm:BasicRequestType (extension)
GetApplicationOrELFStatusRequestType
This function is used to request information about components present on a Secure Element.The command is used to audit one or many components among Applications, Security Domains, and Executable Load Files.
gpm:SecureElement
The Secure Element that is audited.
gpm:MobileSubscription
The Mobile Subscription through which the Secure Element can be audited.
gpm:ApplicationOrExecutableLoadFileAID
1..∞
AID of components to be audited. Only the components which AIDs exactly match the provided AIDs will be considered. In other words, providing a partial AID does not enable to get information on all Applications having this partial AID.
gpm:AuditType
The type of audit to be performed.

Figure 4-123: `GetApplicationOrELFStatusRequestType` type

Where:

- The `SecureElement` element is of type `SEIdentifierType` (see section 4.5.1.2).
- The `MobileSubscription` element is of type `MobileSubscriptionIdentifierType` (see section 4.5.1.2).
- The `ApplicationOrExecutableLoadFileAID` element is of type `AIDType` (see section 4.5.1.2).
- The `AuditType` is an enumeration of the possible audit types (`SEAuditType`).

The value of the `GPHeader.Type` associated to this element SHALL be "GetApplicationOrELFStatusRequest".

The output data of the `GetApplicationOrELFStatus` function defined in section 3.5.4.1 SHALL be mapped to the `GetApplicationOrELFStatusResponse` element that is of type `GetApplicationOrELFStatusResponseType` described in the following figure:

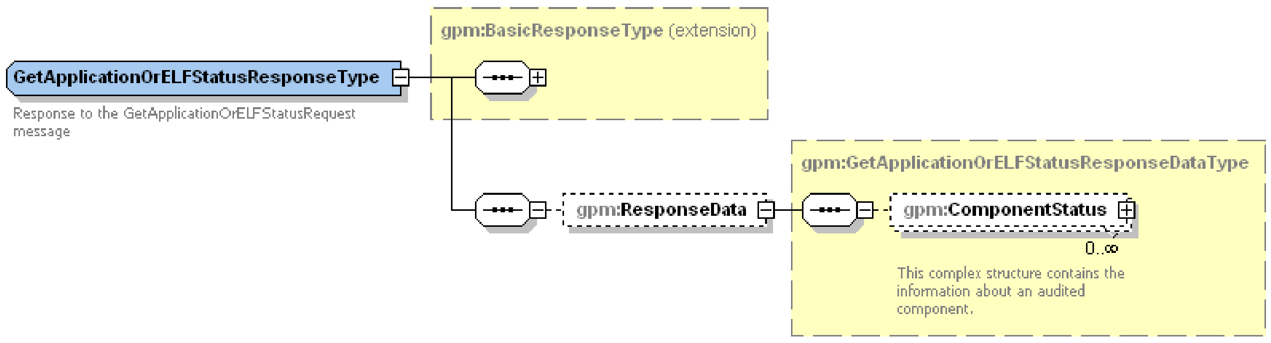

GetApplicationOrELFStatusResponseType
Response to the GetApplicationOrELFStatusRequest message
gpm:BasicResponseType (extension)
gpm:ResponseData
gpm:GetApplicationOrELFStatusResponseDataType
gpm:ComponentStatus
0..∞
This complex structure contains the information about an audited component.

Figure 4-124: `GetApplicationOrELFStatusResponseType` type

Where:

- `ComponentStatus` is an element of type `ComponentStatusType` a base abstract type for the possible component audit statuses. Each concrete component audit status extends this `ComponentStatusType` type

The value of the `GPHeader.Type` associated to this element SHALL be "GetApplicationOrELFStatusResponse".

The component status for an Application defined in section 3.5.4.1 SHALL be mapped to the `Application_ComponentStatusType` type described in the following figure. This `Application_ComponentStatusType` type extends the `ComponentStatusType` type.

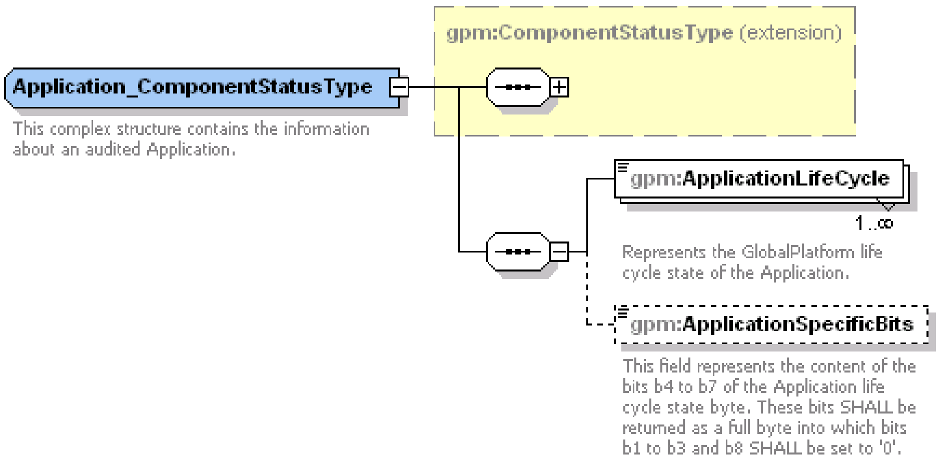

gpm:ComponentStatusType (extension)
Application_ComponentStatusType
This complex structure contains the information about an audited Application.
gpm:ApplicationLifeCycle
1..∞
Represents the GlobalPlatform life cycle state of the Application.
gpm:ApplicationSpecificBits
This field represents the content of the bits b4 to b7 of the Application life cycle state byte. These bits SHALL be returned as a full byte into which bits b1 to b3 and b8 SHALL be set to '0'.

Figure 4-125: `Application_ComponentStatusType` type

Where:

- `ApplicationLifeCycle` is an enumeration of the possible Application life cycle states (`ApplicationLifeCycleType`)

The component status for a Security Domain defined in section 3.5.4.1 SHALL be mapped to the `SD_ComponentStatusType` type described in the following figure. This `SD_ComponentStatusType` type extends the `ComponentStatusType` type.

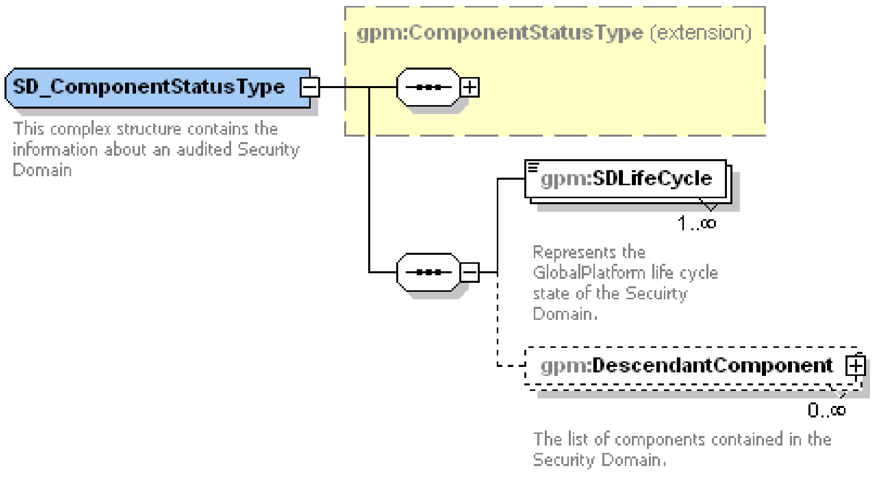

gpm:ComponentStatusType (extension)
SD_ComponentStatusType
This complex structure contains the information about an audited Security Domain
gpm:SDLifeCycle
1..∞
Represents the GlobalPlatform life cycle state of the Security Domain.
gpm:DescendantComponent
0..∞
The list of components contained in the Security Domain.

Figure 4-126: `SD_ComponentStatusType` type

Where:

- `SDLifeCycle` is an enumeration of the possible Security Domain life cycle states (`SDLifeCycleType`)
- `DescendantComponent` is a list of `ComponentStatusType`

The component status for an Executable Load File defined in section 3.5.4.1 SHALL be mapped to the `ELF_ComponentStatusType` type described in the following figure. This `ELF_ComponentStatusType` type extends the `ComponentStatusType` type.

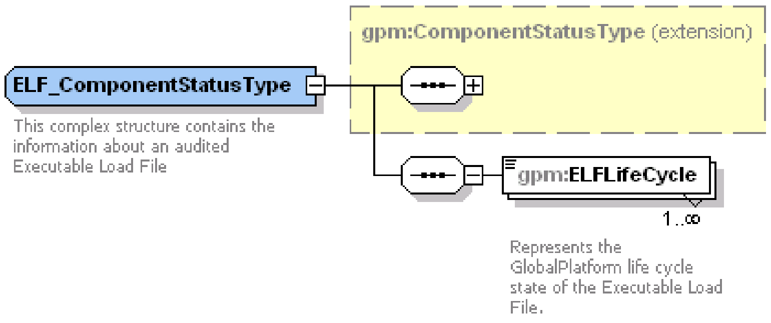

gpm:ComponentStatusType (extension)
ELF_ComponentStatusType
This complex structure contains the information about an audited Executable Load File
gpm:ELFLifeCycle
1..∞
Represents the GlobalPlatform life cycle state of the Executable Load File.

Figure 4-127: `ELF_ComponentStatusType` type

Where:

- `ELFLifeCycle` is an enumeration of the possible Executable Load File life cycle states (`ELFLifeCycleType`)

4.5.5.4.2 The "GetSDFreeMemory" Function

The input data for the `GetSDFreeMemory` function defined in section 3.5.4.2 SHALL be mapped to the `GetSDFreeMemoryRequest` element that is of type `GetSDFreeMemoryRequestType` described in the following figure:

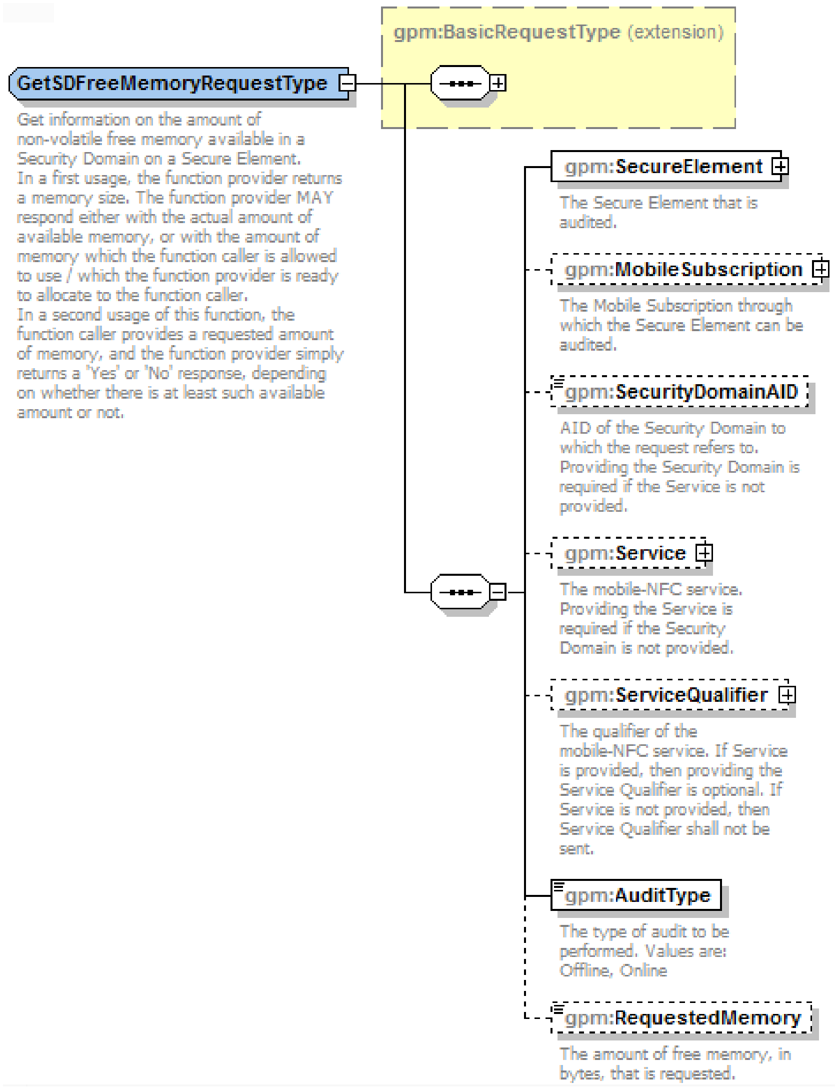

gpm:BasicRequestType (extension)
GetSDFreeMemoryRequestType
Get information on the amount of non-volatile free memory available in a Security Domain on a Secure Element. In a first usage, the function provider returns a memory size. The function provider MAY respond either with the actual amount of available memory, or with the amount of memory which the function caller is allowed to use / which the function provider is ready to allocate to the function caller. In a second usage of this function, the function caller provides a requested amount of memory, and the function provider simply returns a 'Yes' or 'No' response, depending on whether there is at least such available amount or not.
gpm:SecureElement
The Secure Element that is audited.
gpm:MobileSubscription
The Mobile Subscription through which the Secure Element can be audited.
gpm:SecurityDomainAID
AID of the Security Domain to which the request refers to. Providing the Security Domain is required if the Service is not provided.
gpm:Service
The mobile-NFC service. Providing the Service is required if the Security Domain is not provided.
gpm:ServiceQualifier
The qualifier of the mobile-NFC service. If Service is provided, then providing the Service Qualifier is optional. If Service is not provided, then Service Qualifier shall not be sent.
gpm:AuditType
The type of audit to be performed. Values are: Offline, Online
gpm:RequestedMemory
The amount of free memory, in bytes, that is requested.

Figure 4-128: `GetSDFreeMemoryRequestType` type

Where:

- The `SecureElement` element is of type `SEIdentifierType` (see section 4.5.1.2).
- The `SecurityDomain` element is of type `AIDType` (see section 4.5.1.2).
- The `MobileSubscription` element is of type `MobileSubscriptionIdentifierType` (see section 4.5.1.2).
- The `Service` element is of type `ServiceIdentifierType` (see section 4.5.1.2).
- The `ServiceQualifier` element is of type `ServiceQualifierType` (see section 4.5.1.2).
- The `AuditType` is an enumeration of the possible audit types (`SEAuditType`).

The value of the `GPHeader.Type` associated to this element SHALL be "GetSDFreeMemoryRequest".

The output data of the `GetSDFreeMemory` function defined in section 3.5.4.2 SHALL be mapped to the `GetSDFreeMemoryResponse` element that is of type `GetSDFreeMemoryResponseType` described in the following figure:

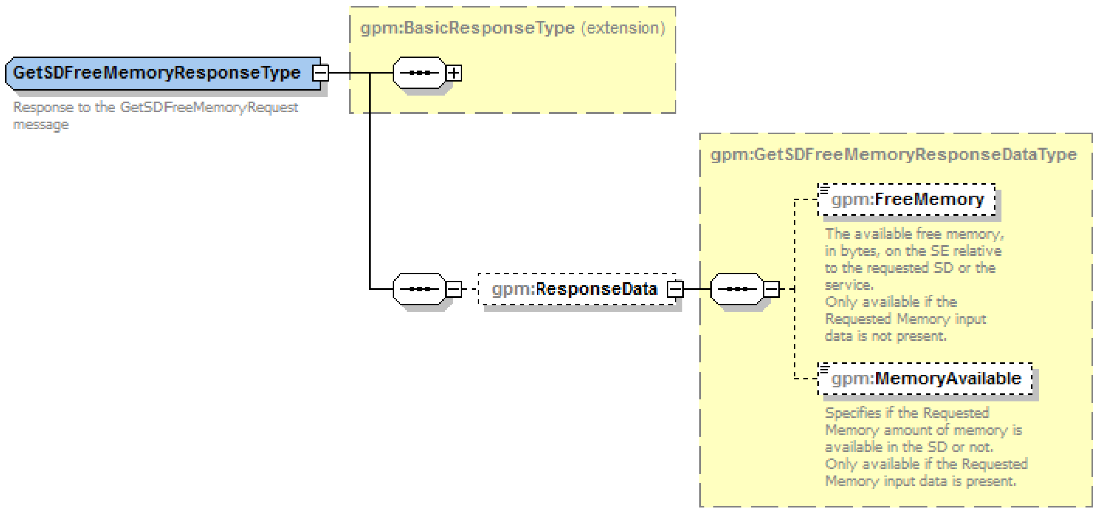

gpm:BasicResponseType (extension)
GetSDFreeMemoryResponseType
Response to the GetSDFreeMemoryRequest message
gpm:GetSDFreeMemoryResponseDataType
gpm:FreeMemory
The available free memory, in bytes, on the SE relative to the requested SD or the service.
Only available if the Requested Memory input data is not present.
gpm:ResponseData
gpm:MemoryAvailable
Specifies if the Requested Memory amount of memory is available in the SD or not.
Only available if the Requested Memory input data is present.

Figure 4-129: `GetSDFreeMemoryResponseType` type

The value of the `GPHeader.Type` associated to this element SHALL be "GetSDFreeMemoryResponse".

4.5.5.5 SCWS Service Portal Functions

4.5.5.5.1 The "LoadSCWSServicePortal" Function

The input data for the `LoadSCWSServicePortal` function defined in section 3.5.5.1 SHALL be mapped to the `LoadSCWSServicePortalRequest` element that is of type `LoadSCWSServicePortalRequestType` described in the following figure:

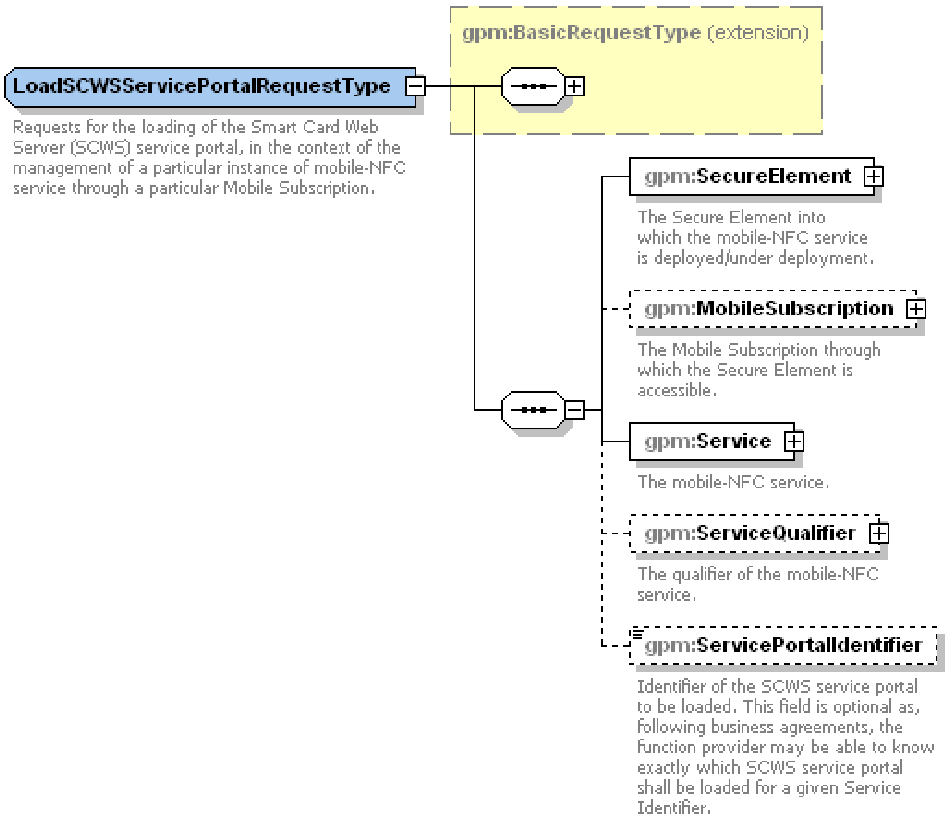

gpm:BasicRequestType (extension)
LoadSCWSServicePortalRequestType
Requests for the loading of the Smart Card Web Server (SCWS) service portal, in the context of the management of a particular instance of mobile-NFC service through a particular Mobile Subscription.
gpm:SecureElement
The Secure Element into which the mobile-NFC service is deployed/under deployment.
gpm:MobileSubscription
The Mobile Subscription through which the Secure Element is accessible.
gpm:Service
The mobile-NFC service.
gpm:ServiceQualifier
The qualifier of the mobile-NFC service.
gpm:ServicePortalIdentifier
Identifier of the SCWS service portal to be loaded. This field is optional as, following business agreements, the function provider may be able to know exactly which SCWS service portal shall be loaded for a given Service Identifier.

Figure 4-130: `LoadSCWSServicePortalRequestType` type

Where:

- The `MobileSubscription` element is of type `MobileSubscriptionIdentifierType` (see section 4.5.1.2).
- The `SecureElement` element is of type `SEIdentifierType` (see section 4.5.1.2).
- The `Service` element is of type `ServiceIdentifierType` (see section 4.5.1.2).
- The `ServiceQualifier` element is of type `ServiceQualifierType` (see section 4.5.1.2).
- The `ServicePortalIdentifier` element is of type `SCWSPortalIdentifierType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "LoadSCWSServicePortalRequest".

The output data of the `LoadSCWSServicePortal` function defined in section 3.5.5.1 SHALL be mapped to the `LoadSCWSServicePortalResponse` element that is of type `LoadSCWSServicePortalResponseType` described in the following figure:

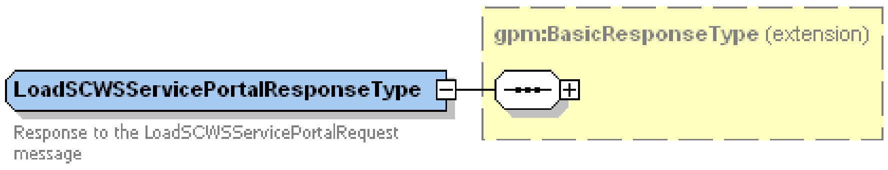

gpm:BasicResponseType (extension)
LoadSCWSServicePortalResponseType
Response to the LoadSCWSServicePortalRequest message

Figure 4-131: `LoadSCWSServicePortalResponseType` type

The value of the `GPHeader.Type` associated to this element SHALL be "LoadSCWSServicePortalResponse".

4.5.5.5.2 The "DeleteSCWSServicePortal" Function

The input data for the `DeleteSCWSServicePortal` function defined in section 3.5.5.2 SHALL be mapped to the `DeleteSCWSServicePortalRequest` element that is of type `DeleteSCWSServicePortalRequestType` described in the following figure:

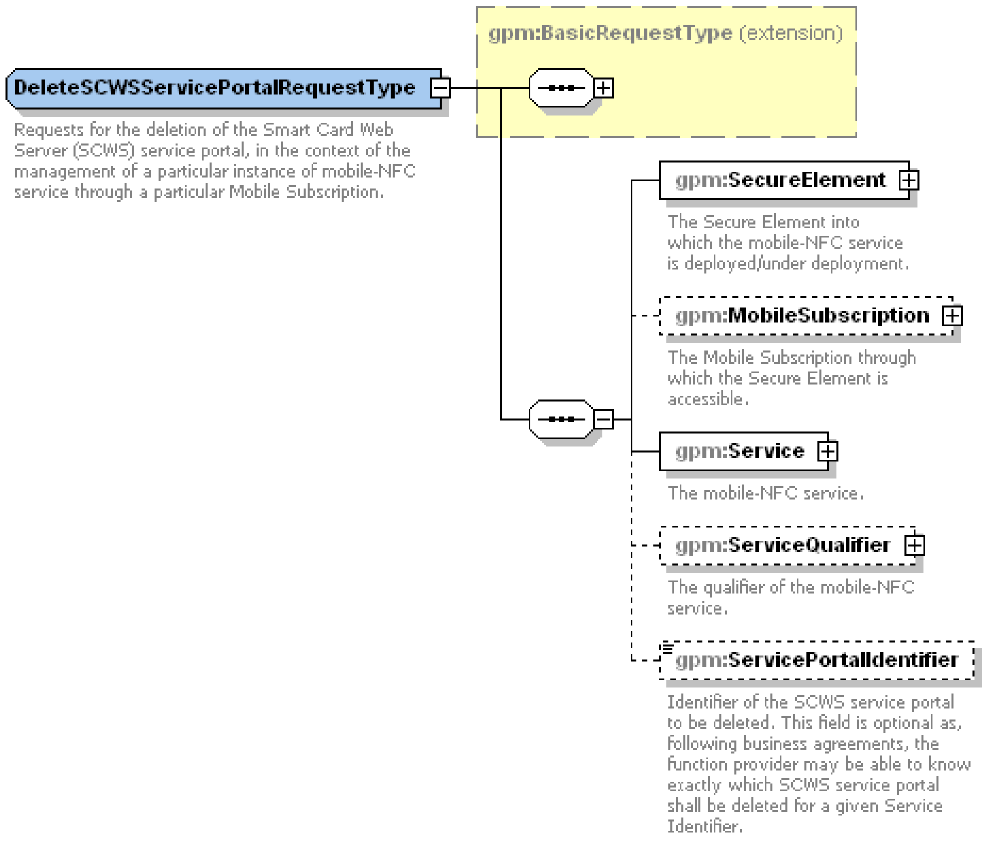

gpm:BasicRequestType (extension)
DeleteSCWSServicePortalRequestType
Requests for the deletion of the Smart Card Web Server (SCWS) service portal, in the context of the management of a particular instance of mobile-NFC service through a particular Mobile Subscription.
gpm:SecureElement
The Secure Element into which the mobile-NFC service is deployed/under deployment.
gpm:MobileSubscription
The Mobile Subscription through which the Secure Element is accessible.
gpm:Service
The mobile-NFC service.
gpm:ServiceQualifier
The qualifier of the mobile-NFC service.
gpm:ServicePortalIdentifier
Identifier of the SCWS service portal to be deleted. This field is optional as, following business agreements, the function provider may be able to know exactly which SCWS service portal shall be deleted for a given Service Identifier.

Figure 4-132: `DeleteSCWSServicePortalRequestType` type

Where:

- The `MobileSubscription` element is of type `MobileSubscriptionIdentifierType` (see section 4.5.1.2).
- The `SecureElement` element is of type `SEIdentifierType` (see section 4.5.1.2).
- The `Service` element is of type `ServiceIdentifierType` (see section 4.5.1.2).
- The `ServiceQualifier` element is of type `ServiceQualifierType` (see section 4.5.1.2).
- The `ServicePortalIdentifier` element is of type `SCWSPortalIdentifierType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "DeleteSCWSServicePortalRequest".

The output data of the `DeleteSCWSServicePortal` function defined in section 3.5.5.2 SHALL be mapped to the `DeleteSCWSServicePortalResponse` element that is of type `DeleteSCWSServicePortalResponseType` described in the following figure:

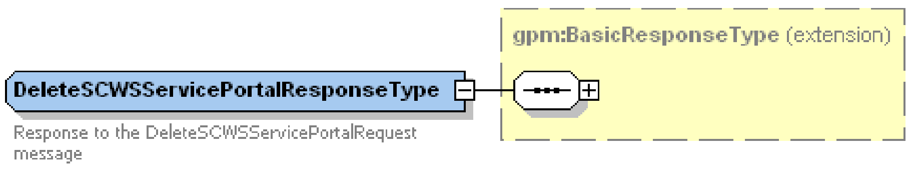

gpm:BasicResponseType (extension)
DeleteSCWSServicePortalResponseType
Response to the DeleteSCWSServicePortalRequest message

Figure 4-133: `DeleteSCWSServicePortalResponseType` type

The value of the `GPHeader.Type` associated to this element SHALL be "DeleteSCWSServicePortalResponse".

4.5.6 Functions for the Device Applications Management

4.5.6.1 The "LoadDeviceApplication" Function

The input data for the `LoadDeviceApplication` function defined in section 3.6.1 SHALL be mapped to the `LoadDeviceApplicationRequest` element that is of type `LoadDeviceApplicationRequestType` described in the following figure:

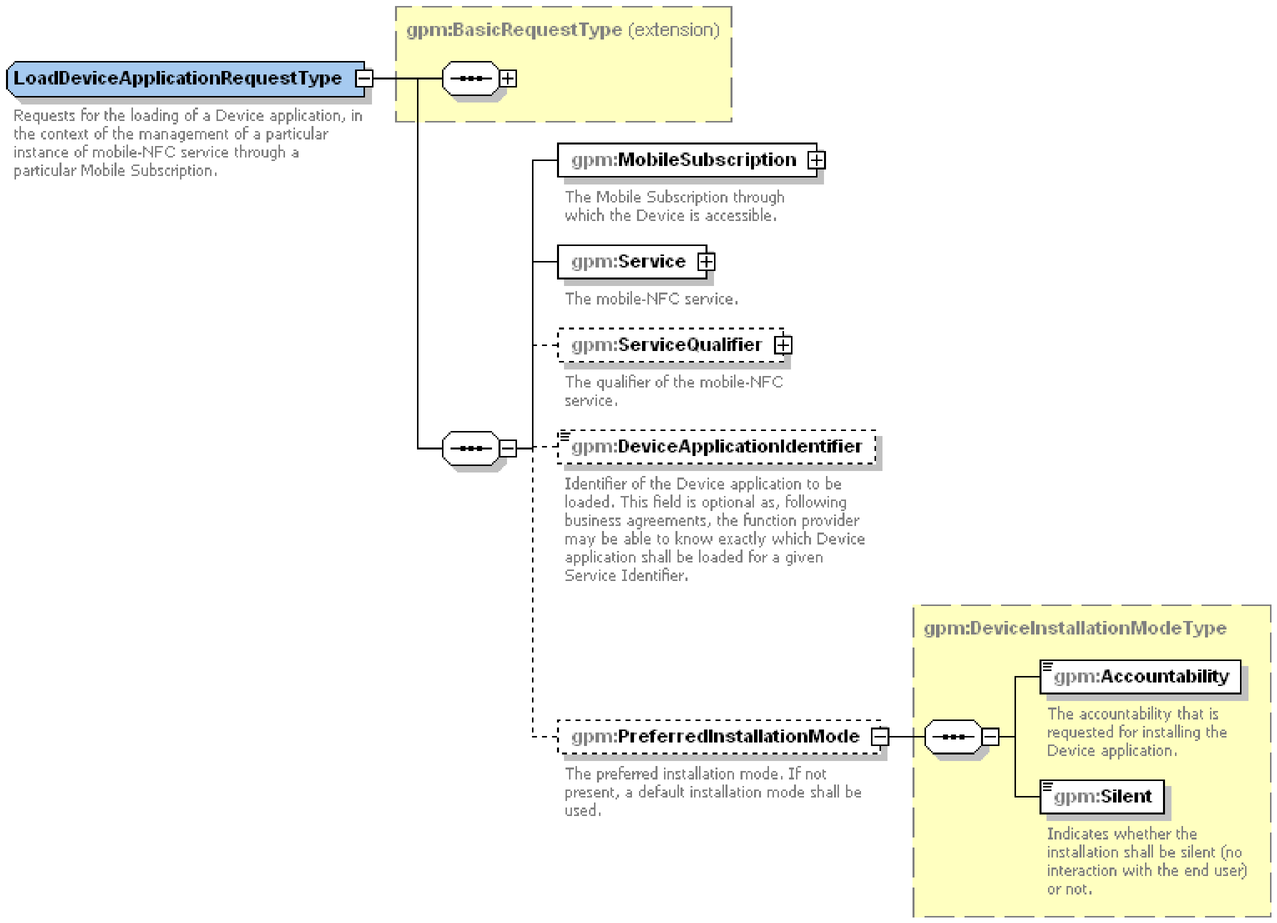

gpm:BasicRequestType (extension)
LoadDeviceApplicationRequestType
Requests for the loading of a Device application, in the context of the management of a particular instance of mobile-NFC service through a particular Mobile Subscription.
gpm:MobileSubscription
The Mobile Subscription through which the Device is accessible.
gpm:Service
The mobile-NFC service.
gpm:ServiceQualifier
The qualifier of the mobile-NFC service.
gpm:DeviceApplicationIdentifier
Identifier of the Device application to be loaded. This field is optional as, following business agreements, the function provider may be able to know exactly which Device application shall be loaded for a given Service Identifier.
gpm:DeviceInstallationModeType
gpm:Accountability
The accountability that is requested for installing the Device application.
gpm:PreferredInstallationMode
The preferred installation mode. If not present, a default installation mode shall be used.
gpm:Silent
Indicates whether the installation shall be silent (no interaction with the end user) or not.

Figure 4-134: `LoadDeviceApplicationRequestType` type

Where:

- The `MobileSubscription` element is of type `MobileSubscriptionIdentifierType` (see section 4.5.1.2).
- The `Service` element is of type `ServiceIdentifierType` (see section 4.5.1.2).
- The `ServiceQualifier` element is of type `ServiceQualifierType` (see section 4.5.1.2).
- The `DeviceApplicationIdentifier` element is of type `DeviceApplicationIdentifierType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "LoadDeviceApplicationRequest".

The output data of the `LoadDeviceApplication` function defined in section 3.6.1 SHALL be mapped to the `LoadDeviceApplicationResponse` element that is of type `LoadDeviceApplicationResponseType` described in the following figure:

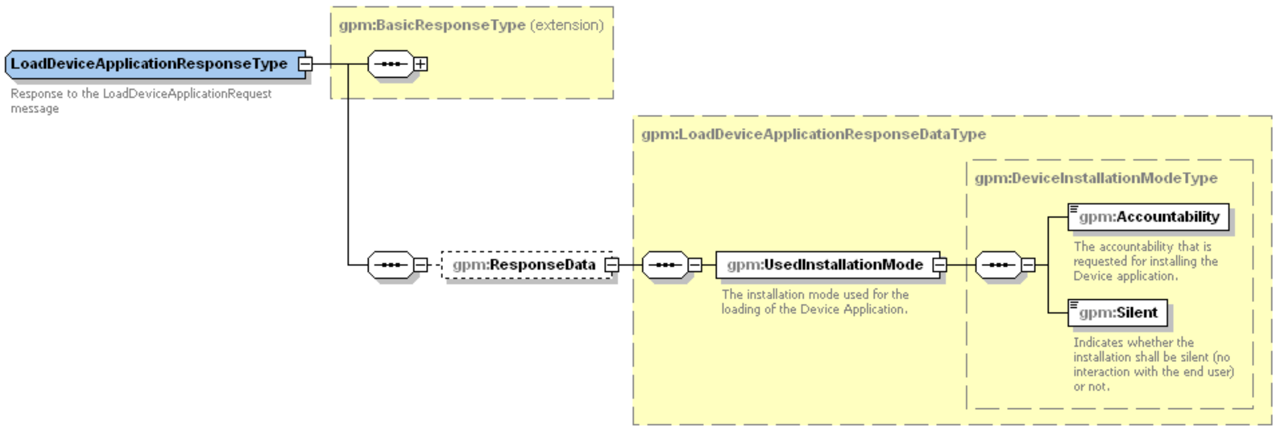

gpm:BasicResponseType (extension)
LoadDeviceApplicationResponseType
Response to the LoadDeviceApplicationRequest message
gpm:LoadDeviceApplicationResponseDataType
gpm:DeviceInstallationModeType
gpm:Accountability
The accountability that is requested for installing the Device application.
gpm:ResponseData
gpm:UsedInstallationMode
The installation mode used for the loading of the Device Application.
gpm:Silent
Indicates whether the installation shall be silent (no interaction with the end user) or not.

Figure 4-135: `LoadDeviceApplicationResponseType` type

The value of the `GPHeader.Type` associated to this element SHALL be "LoadDeviceApplicationResponse".

4.5.6.2 The "BindDeviceApplicationToSEApplication" Function

The input data for the `BindDeviceApplicationToSEApplication` function defined in section 3.6.2 SHALL be mapped to the `BindDeviceApplicationToSEApplicationRequest` element that is of type `BindDeviceApplicationToSEApplicationRequestType` described in the following figure:

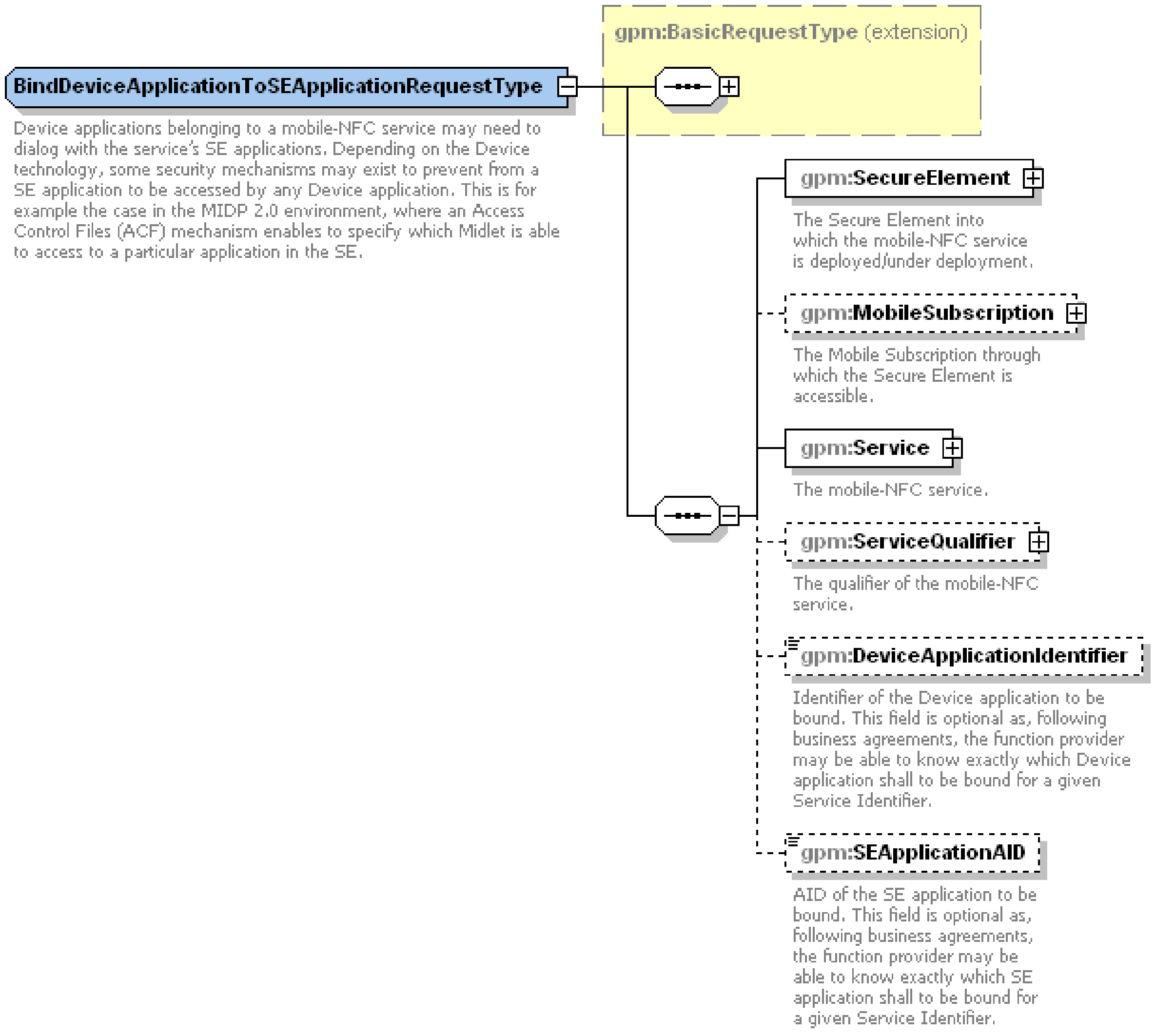

gpm:BasicRequestType (extension)
BindDeviceApplicationToSEApplicationRequestType
Device applications belonging to a mobile-NFC service may need to dialog with the service's SE applications. Depending on the Device technology, some security mechanisms may exist to prevent from a SE application to be accessed by any Device application. This is for example the case in the MIDP 2.0 environment, where an Access Control Files (ACF) mechanism enables to specify which Midlet is able to access to a particular application in the SE.
gpm:SecureElement
The Secure Element into which the mobile-NFC service is deployed/under deployment.
gpm:MobileSubscription
The Mobile Subscription through which the Secure Element is accessible.
gpm:Service
The mobile-NFC service.
gpm:ServiceQualifier
The qualifier of the mobile-NFC service.
gpm:DeviceApplicationIdentifier
Identifier of the Device application to be bound. This field is optional as, following business agreements, the function provider may be able to know exactly which Device application shall to be bound for a given Service Identifier.
gpm:SEApplicationAID
AID of the SE application to be bound. This field is optional as, following business agreements, the function provider may be able to know exactly which SE application shall to be bound for a given Service Identifier.

Figure 4-136: `BindDeviceApplicationToSEApplicationRequestType` type

Where:

- The `MobileSubscription` element is of type `MobileSubscriptionIdentifierType` (see section 4.5.1.2).
- The `SecureElement` element is of type `SEIdentifierType` (see section 4.5.1.2).
- The `Service` element is of type `ServiceIdentifierType` (see section 4.5.1.2).
- The `ServiceQualifier` element is of type `ServiceQualifierType` (see section 4.5.1.2).
- The `DeviceApplicationIdentifier` element is of type `DeviceApplicationIdentifierType` (see section 4.5.1.2).
- The `SEApplicationAID` element is of type `AIDType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "BindDeviceApplicationToSEApplicationRequest".

The output data of the `BindDeviceApplicationToSEApplication` function defined in section 3.6.2 SHALL be mapped to the `BindDeviceApplicationToSEApplicationResponse` element that is of type `BindDeviceApplicationToSEApplicationResponseType` described in the following figure:

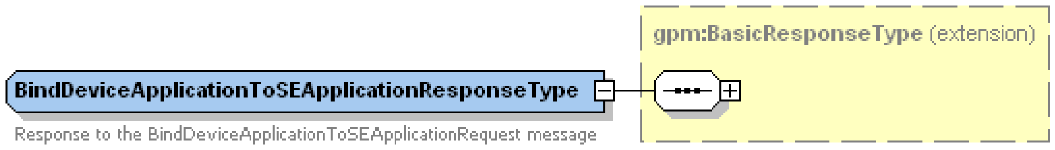
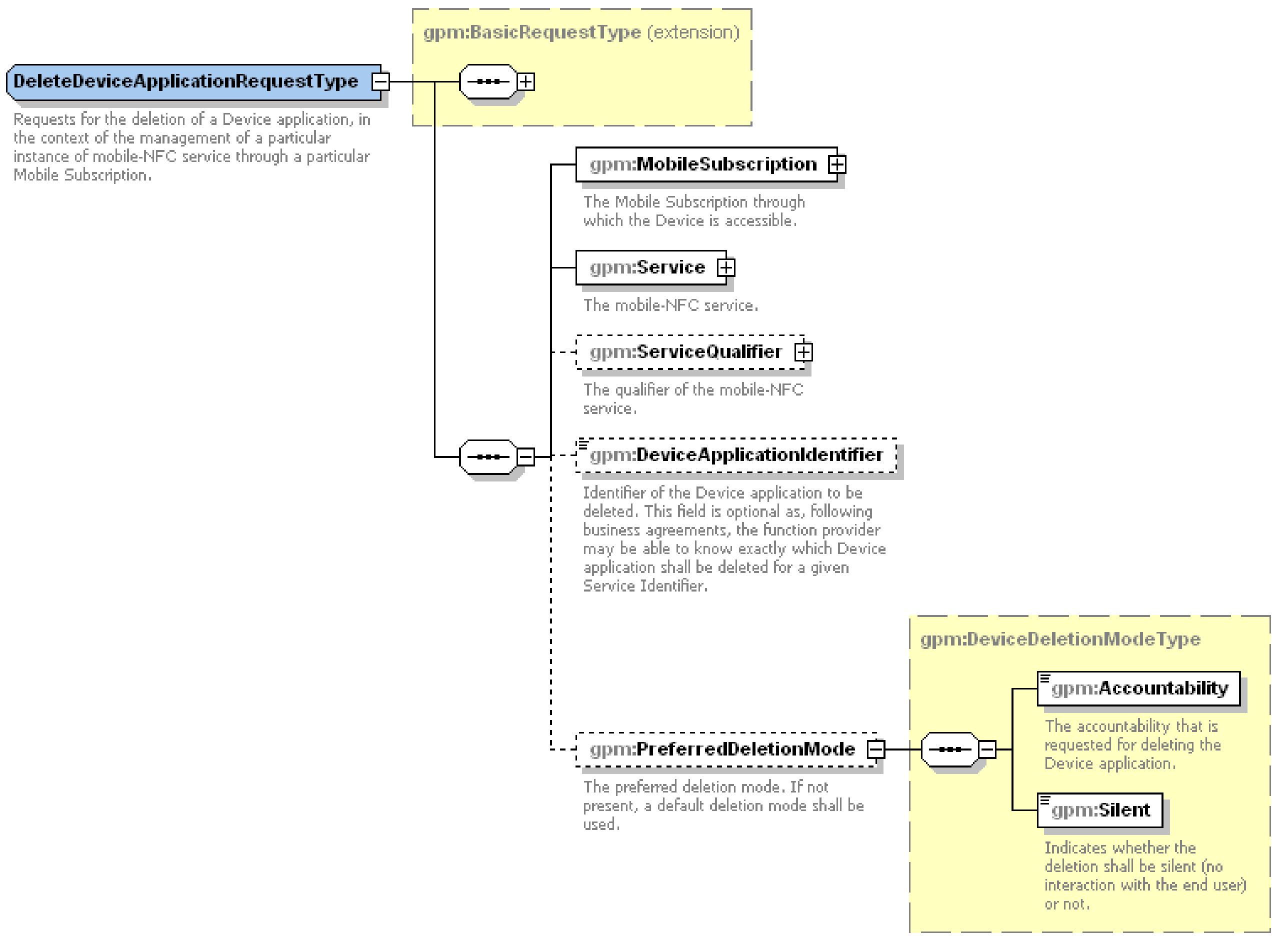

gpm:BasicResponseType (extension)
BindDeviceApplicationToSEApplicationResponseType
Response to the BindDeviceApplicationToSEApplicationRequest message

Figure 4-137: `BindDeviceApplicationToSEApplicationResponseType` type

The value of the `GPHeader.Type` associated to this element SHALL be "BindDeviceApplicationToSEApplicationResponse".

4.5.6.3 The "DeleteDeviceApplication" Function

The input data for the `DeleteDeviceApplication` function defined in section 3.6.3 SHALL be mapped to the `DeleteDeviceApplicationRequest` element that is of type `DeleteDeviceApplicationRequestType` described in the following figure:



gpm:BasicRequestType (extension)
DeleteDeviceApplicationRequestType
Requests for the deletion of a Device application, in the context of the management of a particular instance of mobile-NFC service through a particular Mobile Subscription.
gpm:MobileSubscription
The Mobile Subscription through which the Device is accessible.
gpm:Service
The mobile-NFC service.
gpm:ServiceQualifier
The qualifier of the mobile-NFC service.
gpm:DeviceApplicationIdentifier
Identifier of the Device application to be deleted. This field is optional as, following business agreements, the function provider may be able to know exactly which Device application shall be deleted for a given Service Identifier.
gpm:DeviceDeletionModeType
gpm:Accountability
The accountability that is requested for deleting the Device application.
gpm:Silent
Indicates whether the deletion shall be silent (no interaction with the end user) or not.
gpm:PreferredDeletionMode
The preferred deletion mode. If not present, a default deletion mode shall be used.

Figure 4-138: `DeleteDeviceApplicationRequestType` type

Where:

- The `MobileSubscription` element is of type `MobileSubscriptionIdentifierType` (see section 4.5.1.2).
- The `Service` element is of type `ServiceIdentifierType` (see section 4.5.1.2).
- The `ServiceQualifier` element is of type `ServiceQualifierType` (see section 4.5.1.2).
- The `DeviceApplicationIdentifier` element is of type `DeviceApplicationIdentifierType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "DeleteDeviceApplicationRequest".

The output data of the `DeleteDeviceApplication` function defined in section 3.6.3 SHALL be mapped to the `DeleteDeviceApplicationResponse` element that is of type `DeleteDeviceApplicationResponseType` described in the following figure:

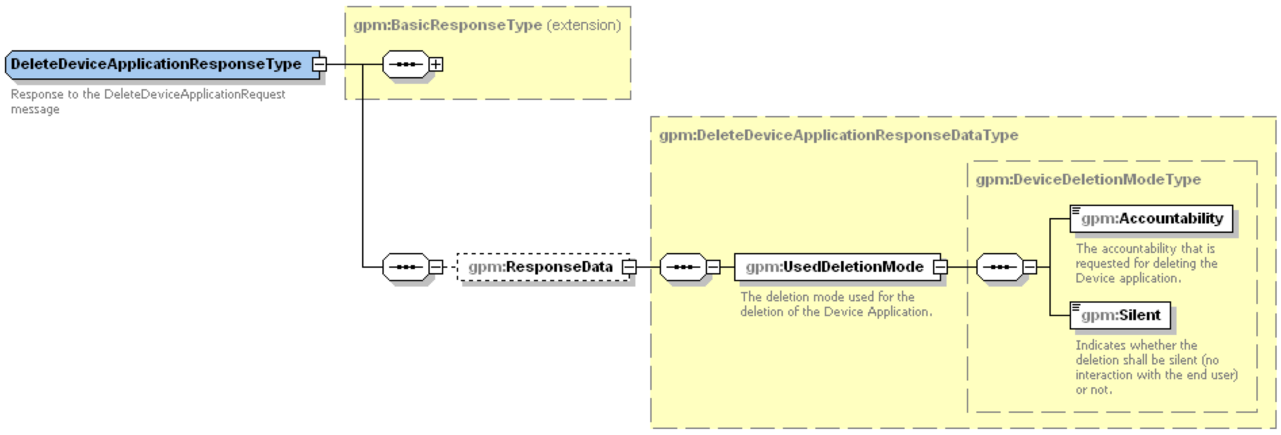

gpm:BasicResponseType (extension)
DeleteDeviceApplicationResponseType
Response to the DeleteDeviceApplicationRequest message
gpm:DeleteDeviceApplicationResponseDataType
gpm:DeviceDeletionModeType
gpm:Accountability
The accountability that is requested for deleting the Device application.
gpm:ResponseData
gpm:UsedDeletionMode
The deletion mode used for the deletion of the Device Application.
gpm:Silent
Indicates whether the deletion shall be silent (no interaction with the end user) or not.

Figure 4-139: `DeleteDeviceApplicationResponseType` type

The value of the `GPHeader.Type` associated to this element SHALL be "DeleteDeviceApplicationResponse".

4.5.6.4 The "UnbindDeviceApplicationToSEApplication" Function

The input data for the `UnbindDeviceApplicationToSEApplication` function defined in section 3.6.4 SHALL be mapped to the `UnbindDeviceApplicationToSEApplicationRequest` element that is of type `UnbindDeviceApplicationToSEApplicationRequestType` described in the following figure:

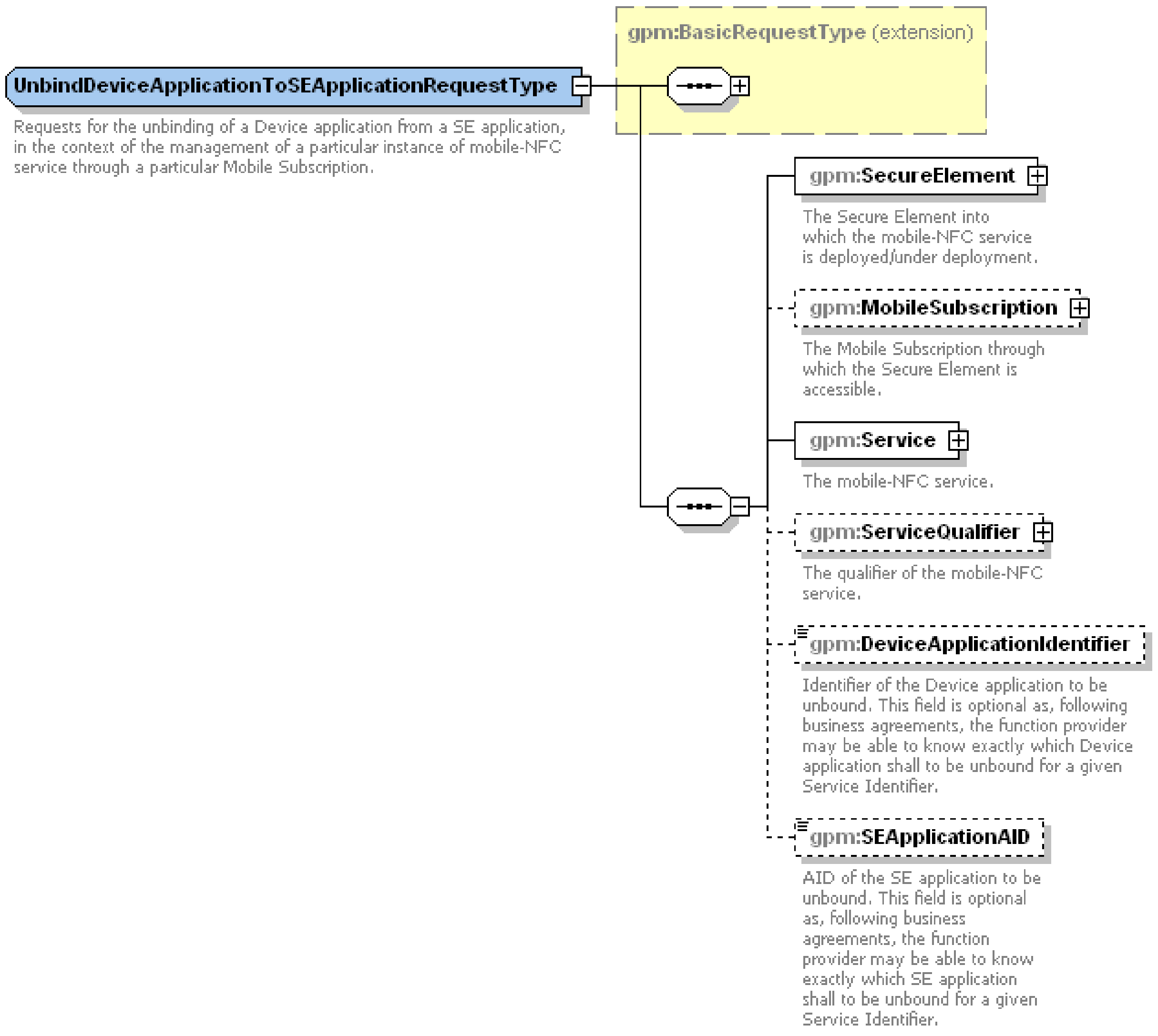

gpm:BasicRequestType (extension)
UnbindDeviceApplicationToSEApplicationRequestType
Requests for the unbinding of a Device application from a SE application, in the context of the management of a particular instance of mobile-NFC service through a particular Mobile Subscription.
gpm:SecureElement
The Secure Element into which the mobile-NFC service is deployed/under deployment.
gpm:MobileSubscription
The Mobile Subscription through which the Secure Element is accessible.
gpm:Service
The mobile-NFC service.
gpm:ServiceQualifier
The qualifier of the mobile-NFC service.
gpm:DeviceApplicationIdentifier
Identifier of the Device application to be unbound. This field is optional as, following business agreements, the function provider may be able to know exactly which Device application shall to be unbound for a given Service Identifier.
gpm:SEApplicationAID
AID of the SE application to be unbound. This field is optional as, following business agreements, the function provider may be able to know exactly which SE application shall to be unbound for a given Service Identifier.

Figure 4-140: `UnbindDeviceApplicationToSEApplicationRequestType` type

Where:

- The `MobileSubscription` element is of type `MobileSubscriptionIdentifierType` (see section 4.5.1.2).
- The `SecureElement` element is of type `SEIdentifierType` (see section 4.5.1.2).
- The `Service` element is of type `ServiceIdentifierType` (see section 4.5.1.2).
- The `ServiceQualifier` element is of type `ServiceQualifierType` (see section 4.5.1.2).
- The `DeviceApplicationIdentifier` element is of type `DeviceApplicationIdentifierType` (see section 4.5.1.2).
- The `SEApplicationAID` element is of type `AIDType` (see section 4.5.1.2).

The value of the `GPHeader.Type` associated to this element SHALL be "UnbindDeviceApplicationToSEApplicationRequest".

The output data of the `UnbindDeviceApplicationToSEApplication` function defined in section 3.6.4 SHALL be mapped to the `UnbindDeviceApplicationToSEApplicationResponse` element that is of type `UnbindDeviceApplicationToSEApplicationResponseType` described in the following figure:

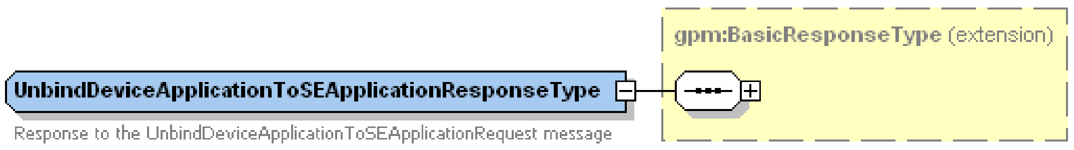

gpm:BasicResponseType (extension)
UnbindDeviceApplicationToSEApplicationResponseType
Response to the UnbindDeviceApplicationToSEApplicationRequest message

Figure 4-141: `UnbindDeviceApplicationToSEApplicationResponseType` type

The value of the `GPHeader.Type` associated to this element SHALL be "UnbindDeviceApplicationToSEApplicationResponse".

4.5.6.5 The "HandleActionDoneOnDeviceApplicationNotification" Notification Function

The input data for the `HandleActionDoneOnDeviceApplicationNotification` notification function defined in section 3.6.5 SHALL be mapped to the `HandleActionDoneOnDeviceApplicationNotificationRequest` element that is of type `HandleActionDoneOnDeviceApplicationNotificationRequestType` described in the following figure:

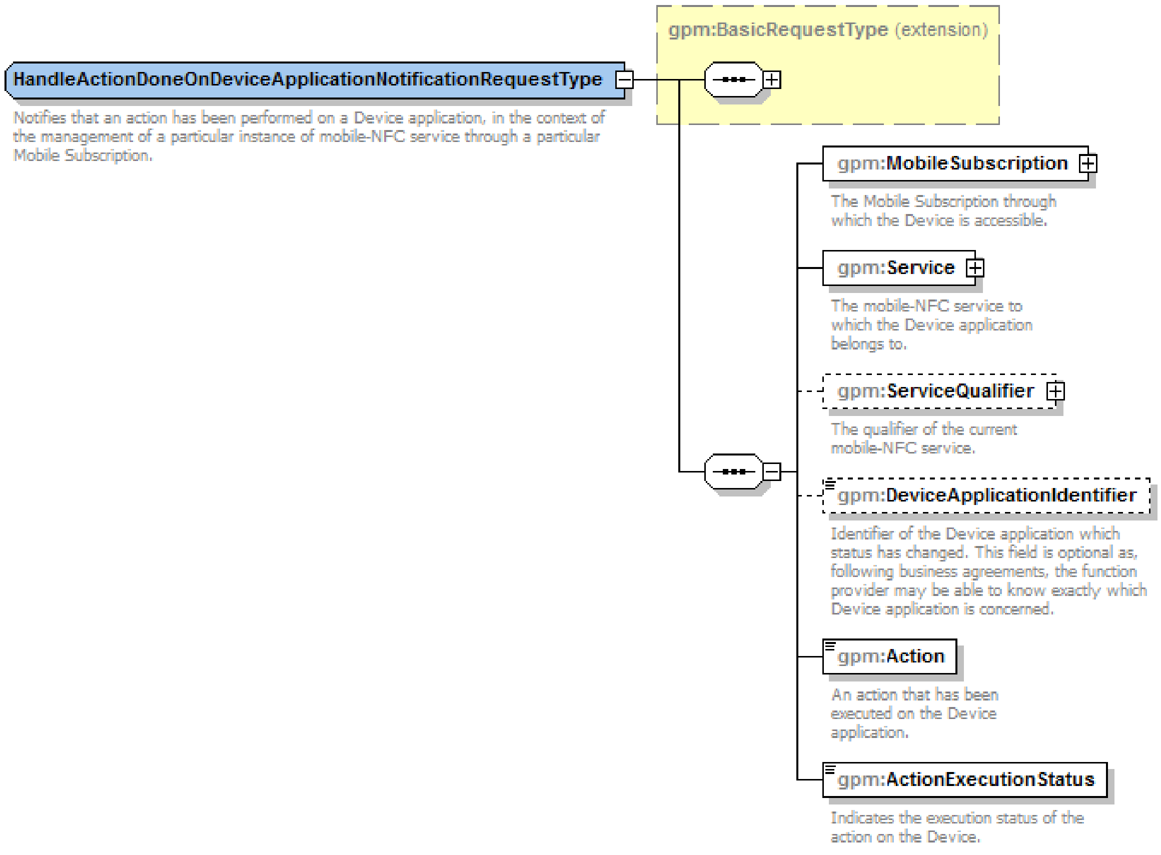

gpm:BasicRequestType (extension)
HandleActionDoneOnDeviceApplicationNotificationRequestType
Notifies that an action has been performed on a Device application, in the context of the management of a particular instance of mobile-NFC service through a particular Mobile Subscription.
gpm:MobileSubscription
The Mobile Subscription through which the Device is accessible.
gpm:Service
The mobile-NFC service to which the Device application belongs to.
gpm:ServiceQualifier
The qualifier of the current mobile-NFC service.
gpm:DeviceApplicationIdentifier
Identifier of the Device application which status has changed. This field is optional as, following business agreements, the function provider may be able to know exactly which Device application is concerned.
gpm:Action
An action that has been executed on the Device application.
gpm:ActionExecutionStatus
Indicates the execution status of the action on the Device.

Figure 4-142: `HandleActionDoneOnDeviceApplicationNotificationRequestType` type

Where:

- The `MobileSubscription` element is of type `MobileSubscriptionIdentifierType` (see section 4.5.1.2).
- The `Service` element is of type `ServiceIdentifierType` (see section 4.5.1.2).
- The `ServiceQualifier` element is of type `ServiceQualifierType` (see section 4.5.1.2).
- The `DeviceApplicationIdentifier` element is of type `DeviceApplicationIdentifierType` (see section 4.5.1.2).
- The `Action` is an enumeration of the possible actions (`ActionOnDeviceApplicationType`).
- The `ActionExecutionStatus` is an enumeration of the possible actions (`ActionOnDeviceExecutionStatusType`).

The value of the `GPHeader.Type` associated to this element SHALL be "HandleActionDoneOnDeviceApplicationNotificationRequest".

5 Binding to SOA Environment (normative)

This section provides the binding of the GlobalPlatform messages defined in section 4 into a SOA infrastructure.

GlobalPlatform recommended infrastructure for deploying mobile-NFC service management systems in a SOA environment is **the Web Services technology, following the OASIS and W3C WS-* standard**, because of interoperability and loose coupling between both parties named as "message requester" and "message receiver".

However GlobalPlatform does not prevent from using another type of technology if it is suitable for a specific deployment. For sure, it implies that both message requester and message receiver uses the same technology.

Note that in case Web Services is used, this section is normative and implementation SHALL comply with the requirements provided in this section. In particular, the WSDL and the supplied "*.wsdl" files are normative.

It is however important to note that, as a complete WSDL file contains implementation specific data such as the specific address location of a service (e.g. <soap12:address location="http://globalplatform.org/systems-messaging/seinfo"/>) it is not possible to claim that the entire WSDL is binary (verbatim) normative. Normative in the context of this specification includes the following:

- Namespace definitions
- Message name and definitions
- Port types and operations
- Existing binding definition including all sub elements.

Implementers are allowed to add additional elements, but SHALL NOT remove elements from the existing WSDL file, for example WS-Policy elements could be added as required by their implementation. Additionally, implementers SHALL obviously define the correct endpoint address of the deployed service.

This binding relies on the specification provided in the "Web Services Profile for GlobalPlatform Messaging document" [22] for usage of OASIS and W3C WS-* standard usage. In case no specific indication is given in this section, the specification given in [22] is applicable.

In particular, document [22] specifies the possible Message Exchange Patterns (MEP):

- "Synchronous Request-Response": maps to synchronous request-response functions defined in section 3.
- "Asynchronous Request":
    - With Asynchronous callback: maps to asynchronous request-response functions defined in section 3.
    - With Asynchronous Polling: not used in this version of the specification.
- "One Way (In-only)": maps to notifications defined in section 3.

Note that for the "Asynchronous Request with callback" MEP, document [22] recommends modeling the WSDL in two parts:

- One service definition ('XxxxService') with input only operations: provided by the function provider in order the function caller to request for the execution of the function,
- And one service definition ('XxxxServiceCallBack') with input only operations: provided by the function caller in order the function provider to send back the function execution result,

Systems are now highly multi-threaded. It is consequently possible for a function caller to perform massive parallel processing, and thus to call several Web Services in parallel. However, to avoid implementation and integration issues, this specification mandates that implementations SHALL NOT perform parallel Web Services calls when they are targeting the same Secure Element or Mobile Subscription or Service instance.

Web Services related to the same SE, Mobile Subscription or Service SHALL be serialized by the function caller. For example, to avoid application instantiation before completion of the executable load file loading, to avoid receiving several notifications in parallel, to avoid requesting several set of Delegated Management Tokens before presenting the associated Receipts, etc.

If several Web Service call are received by the function provider for the same Secure Element, Mobile Subscription, or Service instance, then the function provider SHOULD return one of the following exception: 'Function for the same Secure Element is already in process', or 'Function for the same Mobile Subscription is already in process', or 'Function for the same Device is already in process', or 'Function for the same mobile-NFC service is already in process'.

5.1 SOAP Binding

5.1.1 Common Binding

This paragraph describes the content of common elements of the SOAP message (introduced in document [22]), in the specific context of this specification.

5.1.1.1 Message Binding

The binding of the messages defined in section 4 to SOAP SHALL follow the rules defined in document [22]:

- **SOAP Header:**
  - The information contained in the `GPHeader` of the message SHALL be transferred into the SOAP header, according to what is defined in document [22] and in section 5.1.1.2 of this specification.
- **GP Body:**
  - Only the element contained in the `GPBody` structure SHALL be sent into the SOAP Body (see section 4.5). It means that:
    - The full `GPHeader` structure SHALL NOT be sent.

        - The `GPMessage` envelope SHALL NOT be sent.
        - The `GPBody` envelope SHALL NOT be sent.
    - As a consequence, only one message SHALL be sent at a time. Mixed content is not supported.

Note that, based on the information provided in the SOAP Header and Body, it SHALL still be possible to rebuild a full `GPMessage` structure (except the signature) that complies with definition given in section 4. No information is consequently lost when binding GlobalPlatform messages to SOAP.

5.1.1.2 WS-Addressing Attributes

The following bullet point provides precisions on the usage of WS-Addressing attributes (introduced in document [22]), in the specific context of this specification:

- `/wsa:Action` – Mandatory

  In the context of this specification, the format of the `/wsa:Action` SHALL be:

  [target namespace][delimiter][interface name][delimiter][operation name][direction token]

  Where:

  - [target namespace] = target namespace of the relevant WSDL file of the function group
  - [interface name]:
    - For Synchronous Request-Response MEP, for notification MEP, and for the request of the Asynchronous with callback MEP, the [interface name] value SHALL be filled with the name of the functions group (see Table 3-1 and Table 3-2). Possible values are:
      - "GlobalEligibilityInfo"
      - "DeviceInfo"
      - "SEInfo"
      - "MobileSubscriptionInfo"
      - "ServiceEnvironmentChangeNotification"
      - "ServiceLifeCycleNotification"
      - "MobileSubscriptionLifeCycleNotification"
      - "SELifeCycleNotification"
      - "GlobalServiceManagement"
      - "CAInfo"
      - "CCCMCertificatesManagement"
      - "CardContentManagement"
      - "DelegatedManagement"

        - "ScriptSending"
        - "SEAudit"
        - "SCWSManagement"
        - "DeviceApplicationManagement"
        - "DeviceApplicationBinding"
        - "DeviceApplicationLifeCycleNotification"
    - For Asynchronous with callback MEP, the [interface name] value SHALL be filled with the name of the functions group (see Table 3-1 and Table 3-2) appended with the "CallBack" string. Possible values are:
        - "GlobalServiceManagementCallBack"
        - "CAInfoCallBack"
        - "CardContentManagementCallBack"
        - "ScriptSendingCallBack"
        - "SEAuditCallBack"
        - "SCWSManagementCallBack"
        - "DeviceApplicationManagementCallBack"
        - "DeviceApplicationBindingCallBack"
- [operation name] = the name of the function (as defined in section 3 and referenced in Table 3-1 and Table 3-2).

  Note that in case of Asynchronous with Callback MEP, the operation name value of the callback response SHALL be identical to the operation name value of the request.
- [direction token] = Follows OASIS WS-* specifications, i.e.:
    - For Synchronous Request-Response MEP: the [direction token] SHALL be filled with the "Request" string for the request, and with the "Response" string for the response
    - For One-Way MEP: no direction Token (empty string)
    - For Asynchronous with callback MEP: as this MEP is indeed mapped to two one-way service calls (see introduction of section 5), then there is no direction token, neither for the request, nor for the asynchronous response (empty strings).
- [delimiter] = "/"

Examples:

- For the `GetDeviceCapabilityProfileId` function that belongs to the `DeviceInfo` functions group, the relevant `/wsa:Action` shall be (Synchronous Request-Response MEP):
  - For the request:
```
<wsa:Action>
     http://globalplatform.org/deviceinfo/DeviceInfo/GetDeviceCapabilityPro
     fileIdRequest
</wsa:Action>
```
  - For the response:
```
<wsa:Action>
     http://globalplatform.org/deviceinfo/DeviceInfo/GetDeviceCapabilityPro
     fileIdResponse
</wsa:Action>
```
- For the `HandleSERenewalNotification` function that belongs to the `SELifeCycleNotification` functions group, the relevant `/wsa:Action` shall be (One-Way MEP):
  - For the request:
```
<wsa:Action>
     http://globalplatform.org/selifecyclenotification/SELifeCycleNotificat
     ion/HandleSERenewalNotification
</wsa:Action>
```
- For the `SendScript` function that belongs to the `ScriptSending` functions group, the relevant `/wsa:Action` shall be (Asynchronous with callback MEP, mapped to two one-way calls):
  - For the request:
```
<wsa:Action>
     http://globalplatform.org/scriptsending/ScriptSending/SendScript
</wsa:Action>
```
  - For the response:
```
<wsa:Action>
     http://globalplatform.org/scriptsending/ScriptSendingCallBack/SendScri
     pt
</wsa:Action>
```

Note that document [22] mandates the client (i.e. the function caller) to set the WS-Addressing headers, but does not mandate the server (i.e. the function provider) to use/interpret the WS-Addressing headers in case of "Synchronous Request-Response" Message Exchange Pattern (MEP). MEP for each message defined in this specification is specified in section 5.2.

5.1.1.3 WS-MakeConnection

If usage of polling is required by actors, the binding to WS-MakeConnection defined in document [22] is fully applicable.

5.1.2 Security

The binding of security to transport level security (TLS) and WS-Security defined in document [22] is fully applicable.

Note that it is not required that identity used at transport (TLS) or WS-Security layer matches the identity provided in the `/wsa:From` attribute of the WS-Addressing layer and defined in document [22], as:

- The `/wsa:From` attribute defines a company/organization identity
- Transport or WS-Security identity represents a technical end point that might either represent a company/organization, but also any individual or technical member of the company/organization.

So pure identity check is not possible.

5.1.3 Exception Management

As mentioned in section 3.1.3, exceptions might either be errors or warnings.

Exceptions are neither mapped to any programming languge "exception", nor mapped to "SOAP faults"; they are rather mapped to a `StatusCodeDataType` structure.

In case of exception, the function provider SHALL then return a `StatusCodeDataType` structure, which is embedded inside the `BasicResponseType` (that is a basic type that is extended by each `Response` element) that is defined in section 4.5.1.4. This `StatusCodeData` provides details on the reason of the exception.

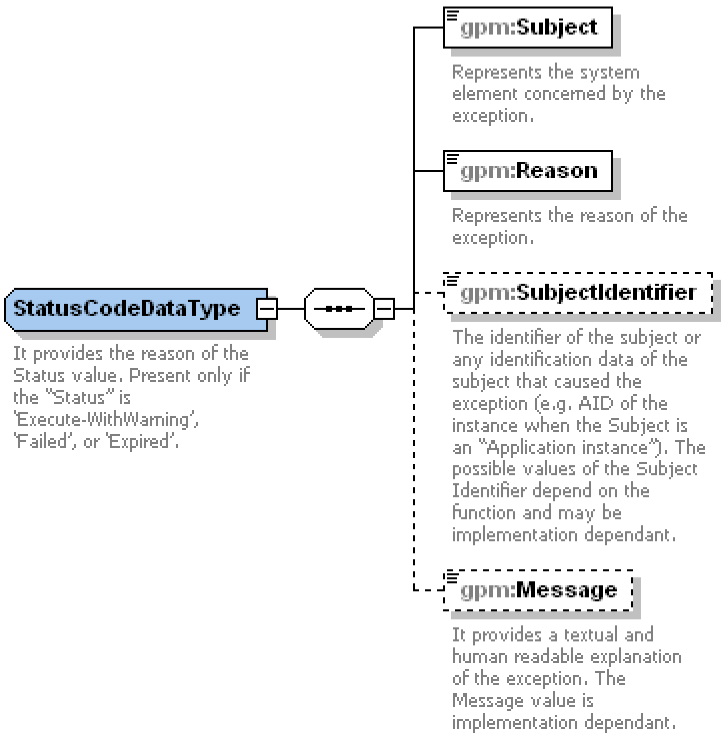

gpm:Subject
Represents the system element concerned by the exception.
gpm:Reason
Represents the reason of the exception.
StatusCodeDataType
It provides the reason of the Status value. Present only if the "Status" is 'Execute-WithWarning', 'Failed', or 'Expired'.
gpm:SubjectIdentifier
The identifier of the subject or any identification data of the subject that caused the exception (e.g. AID of the instance when the Subject is an "Application instance"). The possible values of the Subject Identifier depend on the function and may be implementation dependant.
gpm:Message
It provides a textual and human readable explanation of the exception. The Message value is implementation dependant.

Figure 5-1: `StatusCodeDataType` type

5.2 WSDL Binding

This section specifies the list of Services and of Operations that corresponds to the functions defined in section 3, using the messages defined in section 4.

The following sub-sections provide, per Service, a table listing the Operations of the Service, their recommended Message Exchange Pattern (as specified in [22]), and their request and response types (that refer to XML elements defined in section 4.5).

5.2.1 "GlobalEligibilityInfo" Service

Operation name		**Message Exchange Pattern**
	Request type	
	Response type	
`CheckGlobalEligibility`		Synchronous Request-Response
	`CheckGlobalEligibilityRequest`	
	`CheckGlobalEligibilityResponse`	

Table 5-1: The "GlobalEligibilityInfo" Service operations

The `GlobalEligibilityInfo.wsdl` file defines the WSDL interface of the "GlobalEligibilityInfo" Service. This WSDL file is normative.

5.2.2 "DeviceInfo" Service

Operation name		**Message Exchange Pattern**
	Request type	
	Response type	
`GetDeviceCapabilityProfileId`		Synchronous Request-Response
	`GetDeviceCapabilityProfileIdRequest`	
	`GetDeviceCapabilityProfileIdResponse`	

Table 5-2: The "DeviceInfo" Service operations

The `DeviceInfo.wsdl` file defines the WSDL interface of the "DeviceInfo" Service. This WSDL file is normative.

5.2.3 "SEInfo" Service

Operation name		**Message Exchange Pattern**
	Request type	
	Response type	
`GetSECapabilityProfileId`		Synchronous Request-Response
	`GetSECapabilityProfileIdRequest`	
	`GetSECapabilityProfileIdResponse`	
`GetSEMobileSubscriptionIdentifier`		Synchronous Request-Response
	`GetSEMobileSubscriptionIdentifierRequest`	
	`GetSEMobileSubscriptionIdentifierResponse`	

Table 5-3: The "SEInfo" Service operations

The `SEInfo.wsdl` file defines the WSDL interface of the "SEInfo" Service. This WSDL file is normative.

5.2.4 "MobileSubscriptionInfo" Service

Operation name		**Message Exchange Pattern**
	Request type	
	Response type	
`CheckMobileSubscriptionEligibility`		Synchronous Request-Response
	`CheckMobileSubscriptionEligibilityRequest`	
	`CheckMobileSubscriptionEligibilityResponse`	
`GetMobileSubscriptionAlternateIdentifier`		Synchronous Request-Response
	`GetMobileSubscriptionAlternateIdentifierRequest`	
	`GetMobileSubscriptionAlternateIdentifierResponse`	
`GetMobileSubscriptionSEIdentifiers`		Synchronous Request-Response
	`GetMobileSubscriptionSEIdentifiersRequest`	
	`GetMobileSubscriptionSEIdentifiersResponse`	

Table 5-4: The "MobileSubscriptionInfo" Service operations

The `MobileSubscriptionInfo.wsdl` file defines the WSDL interface of the "MobileSubscriptionInfo" Service. This WSDL file is normative.

5.2.5 "ServiceEnvironmentChangeNotification" Service

Operation name		Message Exchange Pattern
	Request type	
	Response type	
`HandleServiceEnvironmentChangeNotification`		One Way
	`HandleServiceEnvironmentChangeNotificationRequest`	
	No response	
`HandleActionDoneOnServiceNotification`		One Way
	`HandleActionDoneOnServiceNotificationRequest`	
	No response	

Table 5-5: The "ServiceEnvironmentChangeNotification" Service operations

The `ServiceEnvironmentChangeNotification.wsdl` file defines the WSDL interface of the "ServiceEnvironmentChangeNotification" Service. This WSDL file is normative.

5.2.6 "ServiceLifeCycleNotification" Service

Operation name		Message Exchange Pattern
	Request type	
	Response type	
`HandleStartServiceStateChangeNotification`		One Way
	`HandleStartServiceStateChangeNotificationRequest`	
	No response	
`HandleEndServiceStateChangeNotification`		One Way
	`HandleEndServiceStateChangeNotificationRequest`	
	No response	

Table 5-6: The "ServiceLifeCycleNotification" Service operations

The `ServiceLifeCycleNotification.wsdl` file defines the WSDL interface of the "ServiceLifeCycleNotification" Service. This WSDL file is normative.

5.2.7 "MobileSubscriptionLifeCycleNotification" Service

Operation name		Message Exchange Pattern
	Request type	
	Response type	
`HandleMobileSubscriptionIdentifierChangedNotification`		One Way
	`HandleMobileSubscriptionIdentifierChangedNotificationRequest`	
	No response	
`HandleMobileSubscriptionStatusChangeNotification`		One Way
	`HandleMobileSubscriptionStatusChangeNotificationRequest`	
	No response	

Table 5-7: The "MobileSubscriptionLifeCycleNotification" Service operations

The `MobileSubscriptionLifeCycleNotification.wsdl` file defines the WSDL interface of the "MobileSubscriptionLifeCycleNotification" Service. This WSDL file is normative.

5.2.8 "SELifeCycleNotification" Service

Operation name		Message Exchange Pattern
	Request type	
	Response type	
`HandleSERenewalNotification`		One Way
	`HandleSERenewalNotificationRequest`	
	No response	
`HandleSEDeviceChangedNotification`		One Way
	`HandleSEDeviceChangedNotificationRequest`	
	No response	
`HandleSEDeviceStatusChangeNotification`		One Way
	`HandleSEDeviceStatusChangeNotificationRequest`	
	No response	
`HandleSEMobileSubscriptionChangedNotification`		One Way
	`HandleSEMobileSubscriptionChangedNotificationRequest`	
	No response	
`HandleSEStatusChangeNotification`		One Way
	`HandleSEStatusChangeNotificationRequest`	
	No response	

Table 5-8: The "SELifeCycleNotification" Service operations

The `SELifeCycleNotification.wsdl` file defines the WSDL interface of the "SELifeCycleNotification" Service. This WSDL file is normative.

5.2.9 "GlobalServiceManagement" Service

Operation name		**Message Exchange Pattern**
	Request type	
	Response type	
`LookupServiceInstanceReference`		Synchronous Request-Response
	`LookupServiceInstanceReferenceRequest`	
	`LookupServiceInstanceReferenceResponse`	
`DeclareServiceInstanceReference`		Synchronous Request-Response
	`DeclareServiceInstanceReferenceRequest`	
	`DeclareServiceInstanceReferenceResponse`	
`GetServiceInstanceReferenceDescriptor`		Synchronous Request-Response
	`GetServiceInstanceReferenceDescriptorRequest`	
	`GetServiceInstanceReferenceDescriptorResponse`	
`GetServiceState`		Synchronous Request-Response
	`GetServiceStateRequest`	
	`GetServiceStateResponse`	
`DeployService`		Asynchronous with callback
	`DeployServiceRequest`	
	`DeployServiceResponse`	
`UpgradeService`		Asynchronous with callback
	`UpgradeServiceRequest`	
	`UpgradeServiceResponse`	
`ExchangeServiceData`		Asynchronous with callback
	`ExchangeServiceDataRequest`	
	`ExchangeServiceDataResponse`	
`SuspendOrResumeService`		Asynchronous with callback
	`SuspendOrResumeServiceRequest`	
	`SuspendOrResumeServiceResponse`	
`TerminateService`		Asynchronous with callback
	`TerminateServiceRequest`	
	`TerminateServiceResponse`	

Table 5-9: The "GlobalServiceManagement" Service operations

The `GlobalServiceManagement.wsdl` file defines the WSDL interface of the "GlobalServiceManagement" Service. This WSDL file is normative.

5.2.10 "CAInfo" Service

Operation name		Message Exchange Pattern
	Request type	
	Response type	
`GetCAInformation` **– Deprecated –**		Synchronous Request-Response
	`GetCAInformationRequest`	
	`GetCAInformationResponse`	
`AuditCAInformation`		Asynchronous with callback
	`AuditCAInformationRequest`	
	`AuditCAInformationResponse`	

Table 5-10: The "CAInfo" Service operations

Note that the `GetCAInformation` function was initially used for retrieving the CA information. It is now replaced by the `AuditCAInformation` function that provides the same feature but using an asynchronous Message Exchange Pattern, allowing to dialog up to the Secure Element, for retrieval of the CA SD information.

For backward compatibility, the `GetCAInformation` operation is kept, but is marked as deprecated. The `AuditCAInformation` operation should be used instead.

As recommended by [22], the asynchronous with callback MEP is mapped to two WSDLs: one for function invocation (with `XxxxRequest` as input only parameter), and one for result notification (with `XxxxResponse` as input only parameter).

Those `CAInfo.wsdl` and `CAInfoCallBack.wsdl` files define the WSDL interface of the "CAInfo" Service. These WSDL files are normative.

5.2.11 "CCCMCertificatesManagement" Service

Operation name		Message Exchange Pattern
	Request type	
	Response type	
`EnrollSSDOwnerCertificate`		Synchronous Request-Response
	`EnrollSSDOwnerCertificateRequest`	
	`EnrollSSDOwnerCertificateResponse`	

Table 5-11: The "CCCMCertificatesManagement" Service operations

The `CCCMCertificatesManagement.wsdl` file defines the WSDL interface of the "CCCMCertificatesManagement" Service. This WSDL file is normative.

5.2.12 "CardContentManagement" Service

Operation name		Message Exchange Pattern
	Request type	
	Response type	
`SECommandsGenerationAndRemoteExecution`		Asynchronous with callback
	`SECommandsGenerationAndRemoteExecutionRequest`	
	`SECommandsGenerationAndRemoteExecutionResponse`	

Table 5-12: The "CardContentManagement" Service operations

As recommended by [22], the asynchronous with callback MEP is mapped to two WSDLs: one for function invocation (with `XxxxRequest` as input only parameter), and one for result notification (with `XxxxResponse` as input only parameter).

Those `CardContentManagement.wsdl` and `CardContentManagementCallBack.wsdl` files define the WSDL interface of the "CardContentManagement" Service. These WSDL files are normative.

5.2.13 "DelegatedManagement" Service

Operation name		Message Exchange Pattern
	Request type	
	Response type	
`GenerateDMToken`		Synchronous Request-Response
	`GenerateDMTokenRequest`	
	`GenerateDMTokenResponse`	
`VerifyDMReceipt`		Synchronous Request-Response
	`VerifyDMReceiptRequest`	
	`VerifyDMReceiptResponse`	

Table 5-13: The "DelegatedManagement" Service operations

The `DelegatedManagement.wsdl` file defines the WSDL interface of the "DelegatedManagement" Service. This WSDL file is normative.

5.2.14 "ScriptSending" Service

Operation name		Message Exchange Pattern
	Request type	
	Response type	
`BeginConversation`		Asynchronous with callback
	`BeginConversationRequest`	
	`BeginConversationResponse`	
`SendScript`		Asynchronous with callback
	`SendScriptRequest`	
	`SendScriptResponse`	
`EndConversation`		Synchronous Request-Response
	`EndConversationRequest`	
	`EndConversationResponse`	

Table 5-14: The "ScriptSending" Service operations

As recommended by [22], the asynchronous with callback MEP is mapped to two WSDLs: one for function invocation (with `XxxxRequest` as input only parameter), and one for result notification (with `XxxxResponse` as input only parameter).

Those `ScriptSending.wsdl` and `ScriptSendingCallBack.wsdl` files define the WSDL interface of the "ScriptSending" Service. These WSDL files are normative.

5.2.15 "SEAudit" Service

Operation name		Message Exchange Pattern
	Request type	
	Response type	
`GetApplicationOrELFStatus`		Asynchronous with callback
	`GetApplicationOrELFStatusRequest`	
	`GetApplicationOrELFStatusResponse`	
`GetSDFreeMemory`		Asynchronous with callback
	`GetSDFreeMemoryRequest`	
	`GetSDFreeMemoryResponse`	

Table 5-15: The "SEAudit" Service operations

As recommended by [22], the asynchronous with callback MEP is mapped to two WSDLs: one for function invocation (with `XxxxRequest` as input only parameter), and one for result notification (with `XxxxResponse` as input only parameter).

Those `SEAudit.wsdl` and `SEAuditCallBack.wsdl` files define the WSDL interface of the "SEAudit" Service. These WSDL files are normative.

5.2.16 "SCWSManagement" Service

Operation name		Message Exchange Pattern
	Request type	
	Response type	
`LoadSCWSServicePortal`		Asynchronous with callback
	`LoadSCWSServicePortalRequest`	
	`LoadSCWSServicePortalResponse`	
`DeleteSCWSServicePortal`		Asynchronous with callback
	`DeleteSCWSServicePortalRequest`	
	`DeleteSCWSServicePortalResponse`	

Table 5-16: The "SCWSManagement" Service operations

As recommended by [22], the asynchronous with callback MEP is mapped to two WSDLs: one for function invocation (with `XxxxRequest` as input only parameter), and one for result notification (with `XxxxResponse` as input only parameter).

The `SCWSManagement.wsdl` and `SCWSManagementCallBack.wsdl` files define the WSDL interface of the "SCWSManagement" Service. These WSDL files are normative.

5.2.17 "DeviceApplicationManagement" Service

Operation name		Message Exchange Pattern
	Request type	
	Response type	
`LoadDeviceApplication`		Asynchronous with callback
	`LoadDeviceApplicationRequest`	
	`LoadDeviceApplicationResponse`	
`DeleteDeviceApplication`		Asynchronous with callback
	`DeleteDeviceApplicationRequest`	
	`DeleteDeviceApplicationResponse`	

Table 5-17: The "DeviceApplicationManagement" Service operations

As recommended by [22], the asynchronous with callback MEP is mapped to two WSDLs: one for function invocation (with `XxxxRequest` as input only parameter), and one for result notification (with `XxxxResponse` as input only parameter).

The `DeviceApplicationManagement.wsdl` and `DeviceApplicationManagementCallBack.wsdl` files define the WSDL interface of the "DeviceApplicationManagement" Service. These WSDL files are normative.

5.2.18 "DeviceApplicationBinding" Service

Operation name		**Message Exchange Pattern**
	Request type	
	Response type	
`BindDeviceApplicationToSEApplication`		Asynchronous with callback
	`BindDeviceApplicationToSEApplicationRequest`	
	`BindDeviceApplicationToSEApplicationResponse`	
`UnbindDeviceApplicationToSEApplication`		Asynchronous with callback
	`UnbindDeviceApplicationToSEApplicationRequest`	
	`UnbindDeviceApplicationToSEApplicationResponse`	

Table 5-18: The "DeviceApplicationBinding" Service operations

As recommended by [22], the asynchronous with callback MEP is mapped to two WSDLs: one for function invocation (with `XxxxRequest` as input only parameter), and one for result notification (with `XxxxResponse` as input only parameter).

The `DeviceApplicationBinding.wsdl` and `DeviceApplicationBindingCallBack.wsdl` files define the WSDL interface of the "DeviceApplicationBinding" Service. These WSDL files are normative.

5.2.19 "DeviceApplicationLifeCycleNotification" Service

Operation name		**Message Exchange Pattern**
	Request type	
	Response type	
`HandleActionDoneOnDeviceApplicationNotification`		One Way
	`HandleActionDoneOnDeviceApplicationNotificationRequest`	
	No response	

Table 5-19: The "DeviceApplicationLifeCycleNotification" Service operations

The `DeviceApplicationLifeCycleNotification.wsdl` file defines the WSDL interface of the "DeviceApplicationLifeCycleNotification" Service. This WSDL file is normative.

6 Exception Codes (Normative)

Status codes are used in a function call to indicate that an ***exception*** occurred during the processing of the function.

A "status code" is then sent by the function provider to the function requester and contains information about the exception (the `Status Code Data` output data, as described in section 3.1.3).

In this document, the status codes are representing any exception from a simple warning to an error.

- When an error is raised, it means that the expected functional behavior has not been completed.
- When a warning is raised, it means that the expected functional behavior has been completed, but under specific conditions that should be pointed out by the function provider.

Note that the behavior of the function caller receiving a status code may depend on the context. For example the status code representing "Application not present" can be considered a simple warning when the function caller is requesting the deletion of the application. On the other hand, the same status code can be considered as an error if the function caller requests an activation of the application.

6.1 Status Code Representation

6.1.1 Identification

The `Subject` and `Reason` fields contained in the `Status Code Data` of the function output header (see section 3.1.3) are represented by an OID (Object IDentifier).

These identifiers refer to a list of pre-defined elements and reasons (see below for detail) and are represented as integers separated with dots (e.g.: '1.2', '3.4.5').

The two lists are defined as a tree representation in order to achieve three objectives:

- Manage a reasonable list of status codes while the possible exceptions are numerous,
- Allow creating new status codes without altering the status codes numeric identifiers (for example, keep an ordered list after the addition of a new element),
- Allow the creation of "sub codes" on project basis; in this configuration the normative part concerns only the prefix of the status code and the proprietary part represents the suffix.

Exception mappings and bindings to Web Services over HTTP are provided in section 5.

6.1.2 Subject

The `Subject` represents, from the function provider perspective, the subject on which the exception occurred. The subject can either be its own system (e.g.: an internal error), a part of the mobile equipment (e.g.: Device, Secure Element, Application) or even the function caller itself (e.g.: Identification issue).

The subjects are regrouped in categories mapped in order to follow the functions groups. The categories (root elements in the tree representation) are:

1. Generic
2. Card Content Management
3. Secure Element
4. Smart Card Web Server
5. Mobile Subscription
6. NFC Service
7. Device

The possible values for the `Subject` are defined as follow:

1. Generic

- 1.1. Function Requester
- 1.2. Function Provider
  - 1.2.1. Validity Period
- 1.3. Protocol
  - 1.3.1. Protocol Format
  - 1.3.2. Protocol Version
- 1.4. External Resource
- 1.5. Extension Resource
- 1.6. Function

2. Card Content Management

- 2.1. Command
- 2.2. Security Domain
  - 2.2.1. Owner Certificate
  - 2.2.2. Security Domain AID
  - 2.2.3. Security Domain Lifecycle State
- 2.3. Executable Load File
  - 2.3.1. Executable Load File AID
  - 2.3.2. Data Authentication Pattern (DAP)
  - 2.3.3. Load Parameters Profile
- 2.4. Module

2.5. Application Instance

2.5.1. Application Instance AID

2.5.2. Application Instance TAR

2.5.3. Application Instance Lifecycle State

2.5.4. Installation Parameters

2.6. Script

2.6.1. Script Size

2.6.2. APDU Size

2.6.3. Conversation

2.6.4. Target Identifier

2.7. Audit

2.8. Delegated Management

2.8.1. Token

2.8.1.1. CCM Command

2.8.2. Receipt

2.8.2.1. Confirmation Data

2.8.3. Delegated Management Privileges

2.8.4. Receipt Verification Privileges

2.9. Confidential Card Content Management

2.9.1. Certification Request

2.9.2. Controlling Authority Certificate

3. Secure Element

3.1. Secure Element Profile

3.2. Secure Element Identifier type

3.3. Key

3.3.1. Key Provisioning Mode

3.3.2. KeySet Description

3.3.3. Key Generation Mode

3.3.4. KCV

4. Smart Card Web Server

4.1. Service Portal

5. Mobile Subscription

5.1. Mobile Subscription Identifier Type

6. NFC Service

6.1. NFC Service Version

6.2. NFC Service Qualifier

6.3. NFC Service Instance

6.4. Service Instance Reference

6.4.1. New Service Instance Reference

6.5. Service Command

6.6. Exchange Service Data Command

7. Device

7.1. Device Application

7.2. Device Installation Mode

7.3. Device Deletion Mode

7.4. Device Memory

7.5. Device Profile

7.6. Device Security Binding Mechanism

6.1.3 Reason

The `Reason` represents, from the function provider perspective, the reason why the exception occurred. The reason is selected among a list of pre-defined values regrouped in the following categories (root elements in the tree representation):

1. Access Error
2. Format Error
3. Conditions Of Use Not Satisfied
4. Processing Error
5. Transport Error
6. Security Error

The possible values for the `Reason` are defined as follow:

1. Access Error

1.1. Unknown (Identification)

1.2. Not Allowed (Authorization)

2. Format Error

2.1. Invalid Format

2.2. Mandatory Element Missing

2.3. Conditional Element Missing

3. Conditions Of Use Not Satisfied

3.1. Unsupported

  - 3.2. Maximum Size Exceeded
  - 3.3. Already In Use (Uniqueness)
  - 3.4. Invalid Destination
  - 3.5. Invalid Transition
  - 3.6. Related Objects Exists
  - 3.7. Unavailable
  - 3.8. Refused
  - 3.9. Not Consistent

4. **Processing Error**
  - 4.1. Function Already In Progress
  - 4.2. Execution Error
  - 4.3. Stopped On Warning
  - 4.4. Busy
  - 4.5. Operation Already Processed
  - 4.6. Not Present (Missing)
  - 4.7. Generation Not Possible
  - 4.8. Insufficient Memory
  - 4.9. Unassigned
5. **Transport Error**
  - 5.1. Inaccessible
  - 5.2. Timeout
  - 5.3. Time To Live Expired
6. **Security Error**
  - 6.1. Verification Failed
  - 6.2. Decipher Failed
  - 6.3. Certificate Expired
  - 6.4. Certificate Revoked

6.1.4 Status Codes Examples

Depending on the function provider implementation and on the context, the status code returned can be different. In the following examples, the function provider returns the `Subject` and `Reason` that it founds the most appropriate regarding the situation.

Identification issue example:

State: The function requester tries to access a function, but its credentials are not known by the function provider

Function processing: The function provider raises an internal exception, as the function requester couldn't be identified

Returned Status Code:

- `Subject`: **1.1** – Function requester
- `Reason`: **1.1** – Unknown

Authorization issue example:

State: The function requester tries to access a function and is known in the function provider system, but its credentials don't allow him to manage the subject. For example, the function requester tries to manage an Executable Load File he's not authorized to manage.

Function processing: The function provider raises an internal exception when the preliminary verifications are made.

Returned Status Code:

- `Subject`: **1.1** – Function requester
- `Reason`: **1.2** – Not Allowed

Application management issue example:

State: The function requester tries to use the `ExtraditeCommand` to extradite an application instance. The application instance is not present on the secure element.

Function processing: The function provider raises an internal exception when the preliminary verifications are made.

Returned Status Code:

- `Subject`: **2.5** – Application Instance
- `Reason`: **4.6** – Not Present

Application management issue example:

State: The function requester tries to use the `LoadELFCommand` to load an Executable Load File. The ELF is however already present on the secure element.

Function processing: The function provider raises an internal exception when the preliminary verifications are made.

Returned Status Code:

- `Subject`: **2.3** – Executable Load File
- `Reason`: **4.5** – Operation already processed

Note that this Status Code may either be considered as a warning or as an error, depending on the Stop On Warning field of the `SECommandsGenerationAndRemoteExecution` function.

Validity Period issue example:

State: The function requester is using the `SendScript` function but provides a Validity Period that is not manageable by the function provider.

Function processing: The function provider raises an internal exception when he verifies the validity period duration.

Returned Status Code:

- `Subject`: **1.2.1** – Validity Period
- `Reason`: **3.8** – Refused

6.2 Status Code List

The normative list of status codes that can be raised per function is provided in the attached "GPS_Messaging_Specification_for_Mobile_NFC_Services-Status_Code_Matrix-v1.1.xls" file.

7 Annex A: Sequence Diagram (Informative)

The following diagrams present the global mobile-NFC service deployment. For better readability, this diagram is split into 5 phases that are similar to the steps defined in section 2.2.2:

- Phase 1: Initialization
- Phase 2: Card Content Management
- Phase 3: Personalization
- Phase 4: UI Management
- Phase 5: Deployment Process Finalization

The order of functions and notification calls is only provided as an example. It may differ from one deployment to another. Moreover, Phases 2, 3 and 4 might be executed several times, and in a different order.

The following roles are presented: AP, SDM, SEI, and DMSR. Note that up to three SDMs might be present in the diagrams: a SDM for the SE Provider (most probably holding the ISD), a SDM for the TSM (most probably holding a SD_{TSM}), and potentially a SDM for the SP (holding the SD_{SP}). The SE Provider SDM is not presented in Dual mode. No SP SDM exists in Service Deployment Mode #1.

Only the main input data (mainly the identifiers) of the functions and notifications are displayed. The function outputs are mostly opaque.

7.1 Phase 1: Initialization

This first diagram presents a usual initialization phase.

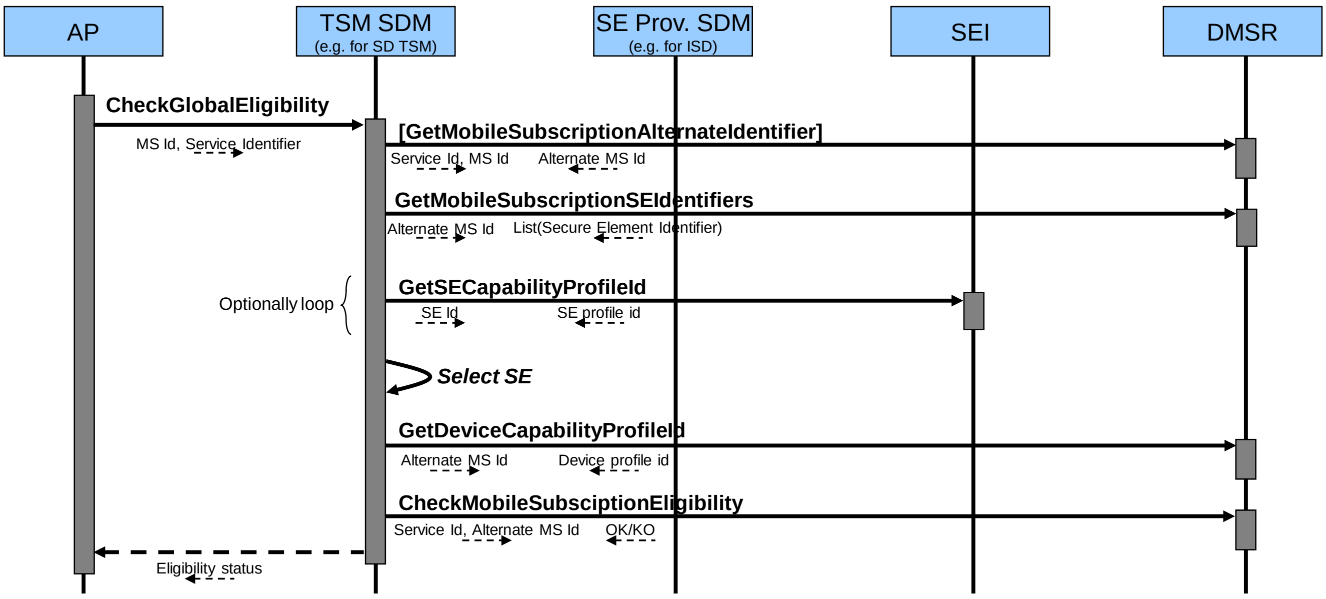


Figure 7-1: Global sequence diagram – Phase 1 Initialization

At the entry point of this first example, the AP knows the Mobile Subscription Id but not the Secure Element Id. During this initialization phase, the SDM TSM then needs to retrieve the list of SE Ids that are accessible through the MS Id, with the help of the DMSR.

Of course, in this example, it is assumed that the DMSR has the capability to know this "MS Id → List of SE Id" association. However, the way how the DMSR knows this information is out of scope of the specification, but it can, for example, be performed through an automatic detection mechanism, or by a direct call from the end-user to the DMSR.

The TSM SDM then has the responsibility to select one of the SE of the list received from the DMSR. This can be done by several ways:

- Checking the technical eligibility of each of the Secure Element, and choosing the one that is the most suitable
- And/or selecting a SE through some business criteria previously agreed with the SP or the SE Provider (e.g. preferably use the UICC; preferably use the Secure Element of the SP; etc.)

This process is out of scope of the specification, but the function for retrieving the technical information on the SE is in scope.

As another assumption of this example, the provided MS Id is not the final one to be used for inter-role communication. TSM SDM also has the responsibility to first request an alternate MS Id from the DMSR, and then to use it in place of the former one, for any further call (including the SE Id retrieval).

In a second example depicted below, the SE Id is known to the AP but not the MS Id.

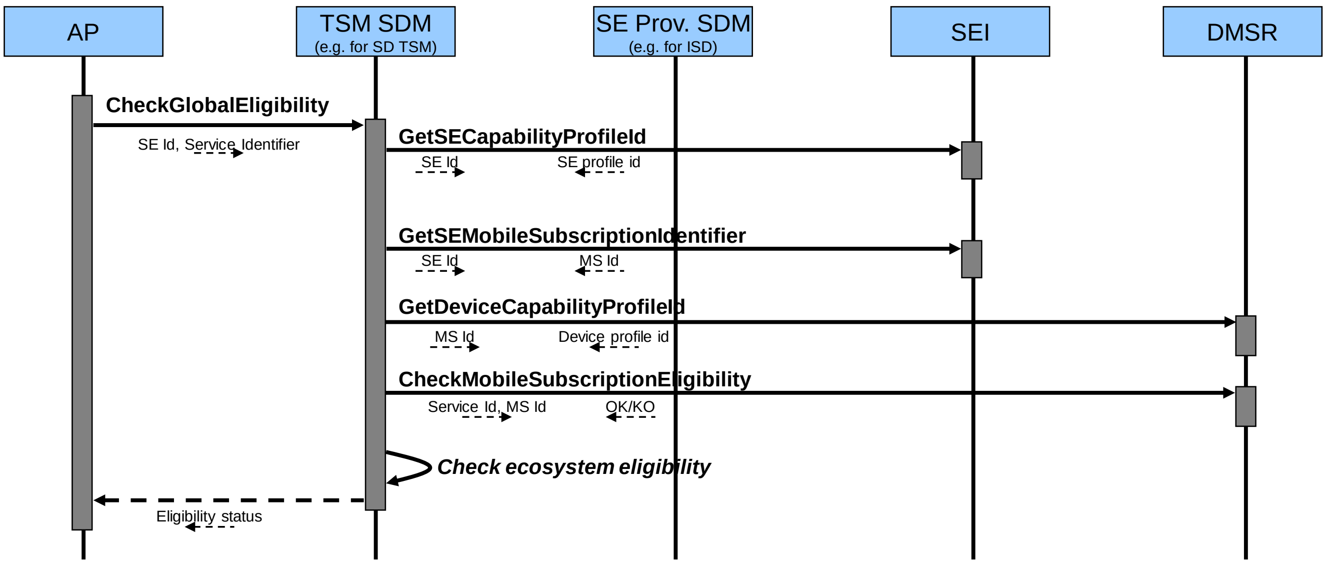


Figure 7-2: Global sequence diagram – Phase 1 Initialization (alternate example)

The SDM TSM needs to retrieve the MS Id that should be used to access to the Secure Element, with the help of the Secure Element Issuer

Of course, in this example, it is assumed that the SEI has the capability to know this information. However, the way that the SEI knows this "SE Id → MS Id " association is out of scope of the specification, but it can, for example, be performed through an automatic detection mechanism, or by a direct call from the end-user to the SEI.

After the initialization phase is done, the service deployment may be requested by the AP.

7.2 Phase 2: Card Content Management

As mentioned in the introduction to roles to actors allocation (see section 2.2.1.2), one of three different Service Deployment Modes may be used when looking at the SP to TSM relation. The following three sub sections show examples of phase 2 scenarios for each of the different Service Deployment Modes. These examples also refer to personalization details which will be explained later in this annex.

Most examples given in this section assume "Simple mode" is used as Card Content Management Mode. For the purpose of highlighting how a different CCM mode may impact the service deployment scenario, the first sub section, for Service Deployment Mode #1, has two diagrams: one in Simple mode and another in Delegated mode.

7.2.1 Service Deployment Mode #1

The Service Provider delegates the full operational deployment of the mobile-NFC service to the TSM (Eligibility and Service Management, and service personalization).

In this case, the SP plays the AP role only, and this AP calls the `DeployService` function on the TSM SDM, with the `InstallServiceCommand`, the `PersonalizeServiceCommand`, and finally the `ActivateServiceCommand`, as depicted in the following figure. During its processing, the TSM SDM delegates the single Card Content Management operations to the SE Provider SDM, in Simple mode.

The personalization is performed through the call to the script sending functions.

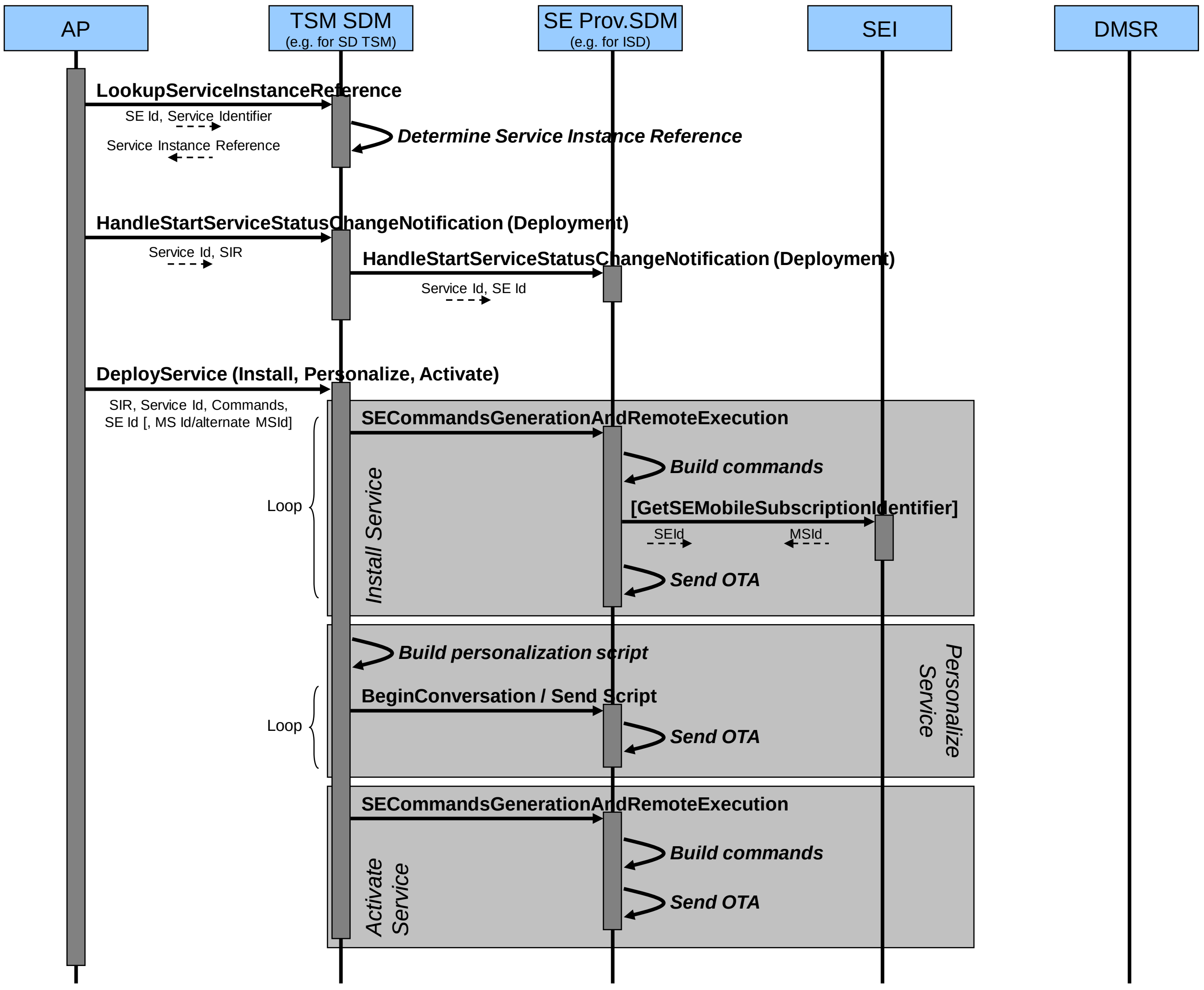

AP
TSM SDM (e.g. for SD TSM)
SE Prov.SDM (e.g. for ISD)
SEI
DMSR
LookupServiceInstanceReference
SE Id, Service Identifier
Service Instance Reference
Determine Service Instance Reference
HandleStartServiceStatusChangeNotification (Deployment)
Service Id, SIR
HandleStartServiceStatusChangeNotification (Deployment)
Service Id, SE Id
DeployService (Install, Personalize, Activate)
SIR, Service Id, Commands, SE Id [, MS Id/alternate MSId]
SECommandsGenerationAndRemoteExecution
Build commands
[GetSEMobileSubscriptionIdentifier]
SEId
MSId
Send OTA
Loop
Install Service
Build personalization script
BeginConversation / Send Script
Send OTA
Loop
Personalize Service
SECommandsGenerationAndRemoteExecution
Build commands
Send OTA
Activate Service

Figure 7-3: Global sequence diagram – Phase 2 – Service Deployment Mode #1

Note that if the MS Id (or the Alternate MS Id) is not provided by the AP nor provided by the TSM SDM, then the SE Provider SDM can retrieve it by calling the `GetSEMobileSubscriptionIdentifier` on the SEI.

Service deployment can be split into several steps depending on the deployment scenario, but the above diagram only shows the initial full deployment request (Install + Personalize + Activate) as an example. The following diagrams shows different ways to split up the deployment request and do other activities between these calls.

The following example also uses Service Deployment Mode #1, but CCM mode is now Delegated mode, and for the installation stage only. In this case the TSM SDM builds the commands and requests a DM token from the SE Provider SDM, before sending the APDU commands over-the-air, either through an own OTA capability, or by calling the script sending functions.

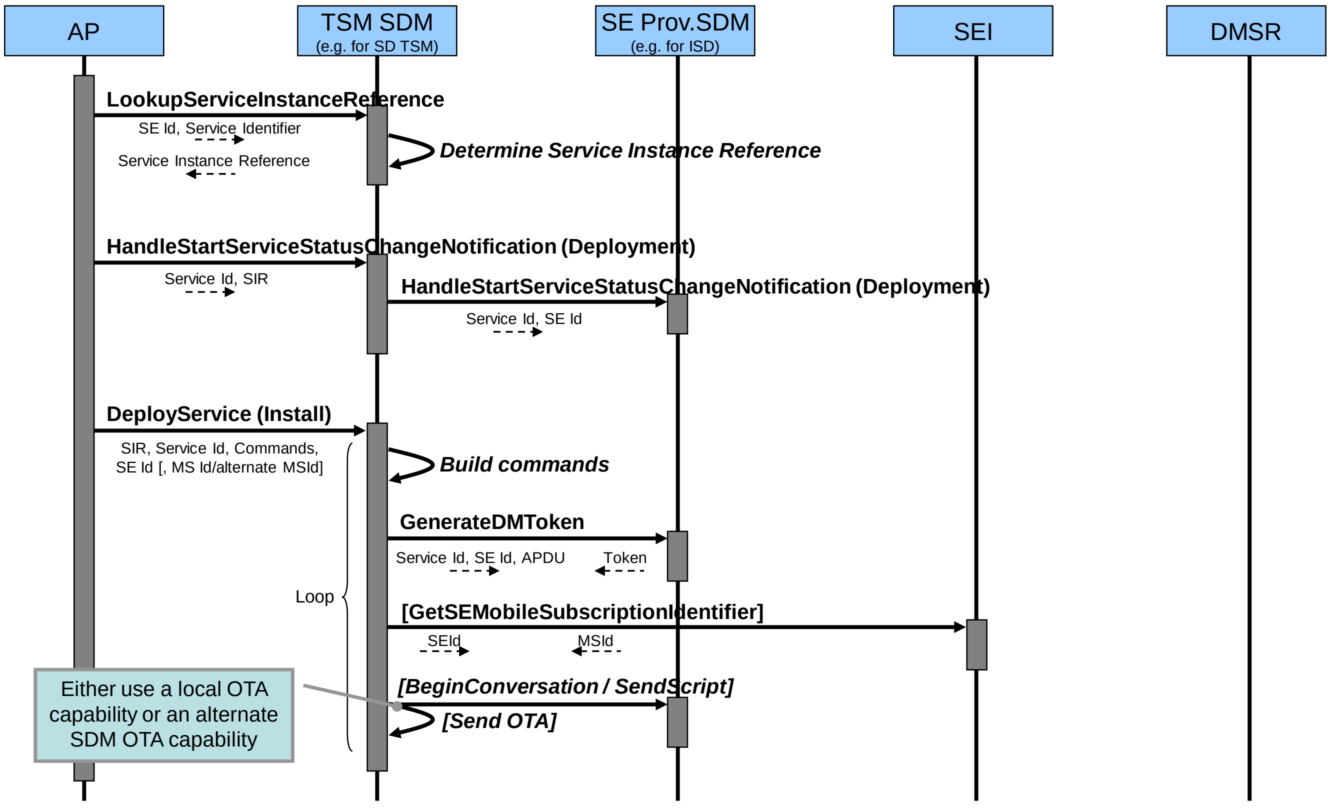

AP
TSM SDM
(e.g. for SD TSM)
SE Prov.SDM
(e.g. for ISD)
SEI
DMSR
LookupServiceInstanceReference
SE Id, Service Identifier
Service Instance Reference
Determine Service Instance Reference
HandleStartServiceStatusChangeNotification (Deployment)
Service Id, SIR
HandleStartServiceStatusChangeNotification (Deployment)
Service Id, SE Id
DeployService (Install)
SIR, Service Id, Commands, SE Id [, MS Id/alternate MSId]
Build commands
GenerateDMToken
Service Id, SE Id, APDU
Token
Loop
[GetSEMobileSubscriptionIdentifier]
SEId
MSId
[BeginConversation / SendScript]
[Send OTA]
Either use a local OTA capability or an alternate SDM OTA capability

Figure 7-4: Global sequence diagram – Phase 2 – Service Deployment Mode #1 – Delegated Mode

7.2.2 Service Deployment Mode #2

In this mode, the Service Provider performs the service personalization by itself. The SP thus remains the owner of the mobile-NFC service personalization keys, and then also performs its Security Domain creation and personalization. The rest of the service installation is delegated to the TSM.

In this case, the SP plays both the AP and SDM (Personalization Script Management facet) roles. As a sequence:

- The SP SDM first calls the `SECommandsGenerationAndRemoteExecution` function on the TSM SDM for the SSD creation (`InstantiateApplicationCommand` SE Command) and personalization (`CreateFirstSSDKeysetCommand` SE Command).
- Then, the AP calls the `DeployService` function on the TSM SDM, with the `InstallServiceCommand`.
- Then, the SP SDM builds the personalization script and calls the script sending functions on the TSM SDM for the service personalization.

- Finally, the AP calls the `DeployService` function on the TSM SDM, with the `ActivateServiceCommand`.

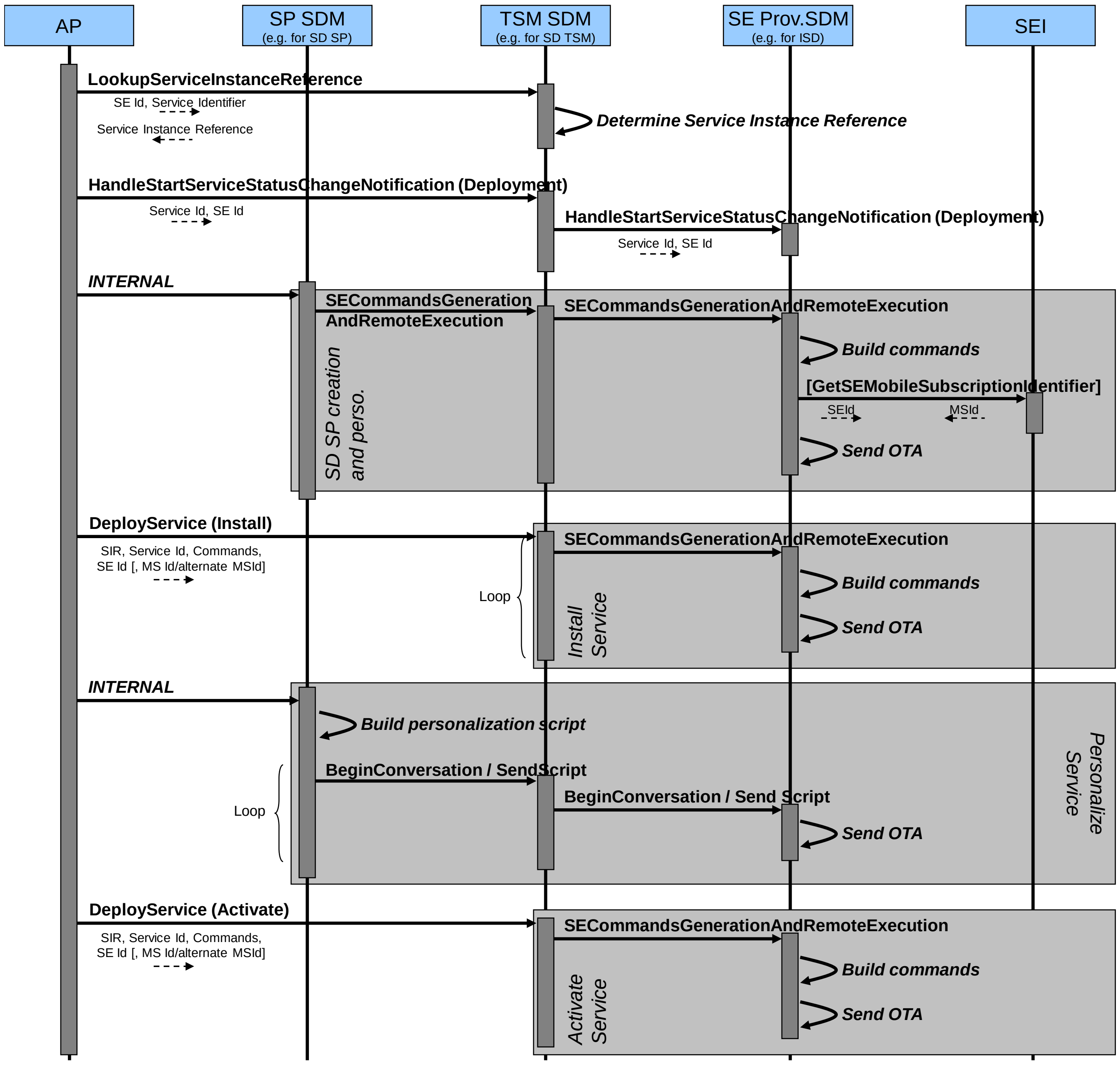

AP
SP SDM (e.g. for SD SP)
TSM SDM (e.g. for SD TSM)
SE Prov.SDM (e.g. for ISD)
SEI
LookupServiceInstanceReference
SE Id, Service Identifier
Service Instance Reference
Determine Service Instance Reference
HandleStartServiceStatusChangeNotification (Deployment)
Service Id, SE Id
HandleStartServiceStatusChangeNotification (Deployment)
Service Id, SE Id
INTERNAL
SECommandsGeneration AndRemoteExecution
SECommandsGenerationAndRemoteExecution
Build commands
[GetSEMobileSubscriptionIdentifier]
SEId
MSId
Send OTA
SD SP creation and perso.
DeployService (Install)
SIR, Service Id, Commands, SE Id [, MS Id/alternate MSId]
Loop
SECommandsGenerationAndRemoteExecution
Build commands
Send OTA
Install Service
INTERNAL
Build personalization script
BeginConversation / SendScript
BeginConversation / Send Script
Loop
Send OTA
Personalize Service
DeployService (Activate)
SIR, Service Id, Commands, SE Id [, MS Id/alternate MSId]
SECommandsGenerationAndRemoteExecution
Build commands
Send OTA
Activate Service

Figure 7-5: Global sequence diagram – Phase 2 – Service Deployment Mode #2

7.2.3 Service Deployment Mode #3

The Service Provider manages the global deployment of the mobile-NFC service, and just delegates the single Card Content Management operations to the TSM.

In this case, the SP plays both the AP and SDM (Eligibility and Service Management, and Personalization Script Management facets) roles, but only the SDM role is directly involved in the service deployment process. As a sequence:

- The SP SDM first calls the `SECommandsGenerationAndRemoteExecution` function on the TSM SDM for the SSD creation and the various steps of the service deployment.
- Then, the SP SDM builds the personalization script and calls the script sending functions on the TSM SDM for the service personalization.
- Finally, the SP SDM calls the `SECommandsGenerationAndRemoteExecution` function on the TSM SDM to finalize the mobile-NFC service activation, if required.

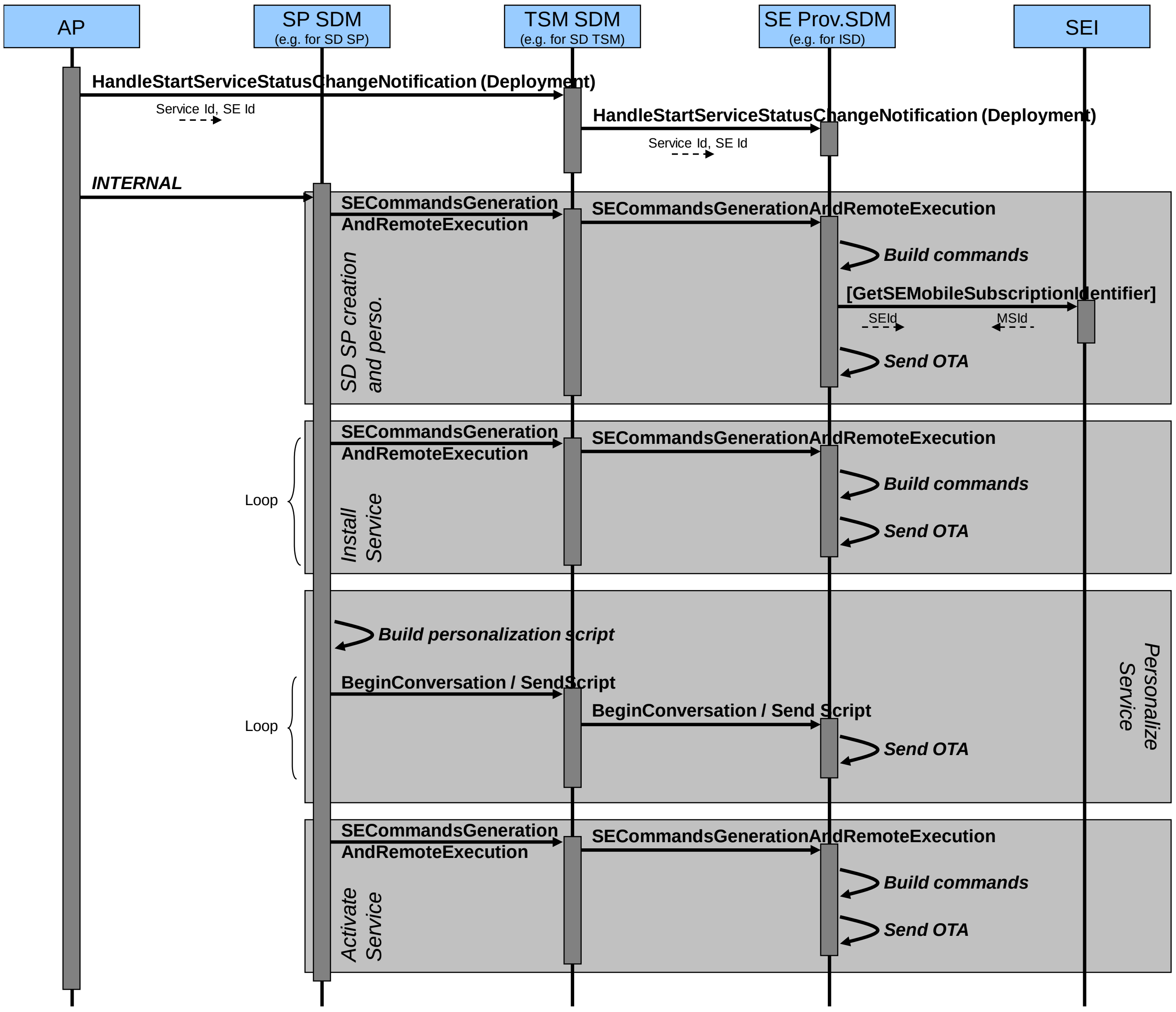

AP
SP SDM (e.g. for SD SP)
TSM SDM (e.g. for SD TSM)
SE Prov.SDM (e.g. for ISD)
SEI
HandleStartServiceStatusChangeNotification (Deployment)
Service Id, SE Id
HandleStartServiceStatusChangeNotification (Deployment)
Service Id, SE Id
INTERNAL
SECommandsGeneration AndRemoteExecution
SECommandsGenerationAndRemoteExecution
Build commands
[GetSEMobileSubscriptionIdentifier]
SEId
MSId
Send OTA
SD SP creation and perso.
SECommandsGeneration AndRemoteExecution
SECommandsGenerationAndRemoteExecution
Build commands
Send OTA
Loop
Install Service
Build personalization script
BeginConversation / SendScript
BeginConversation / Send Script
Send OTA
Loop
Personalize Service
SECommandsGeneration AndRemoteExecution
SECommandsGenerationAndRemoteExecution
Build commands
Send OTA
Activate Service

Figure 7-6: Global sequence diagram – Phase 2 – Service Deployment Mode #3

7.3 Phase 3: Personalization

This section presents more details related to SD and Secure Element application personalization. The sequence diagrams below are to be understood as details of the one presented in section 7.2, for phase 2 – Card Content Management.

Both examples are presented with the TSM SDM managing the personalization. Use case where the Service Provider SDM is managing the personalization is fully equivalent, the SP SDM replacing the TSM SDM in the diagrams.

<u>The confidential SD creation:</u>

A dialog with the CA is required to handle the AP certificates.

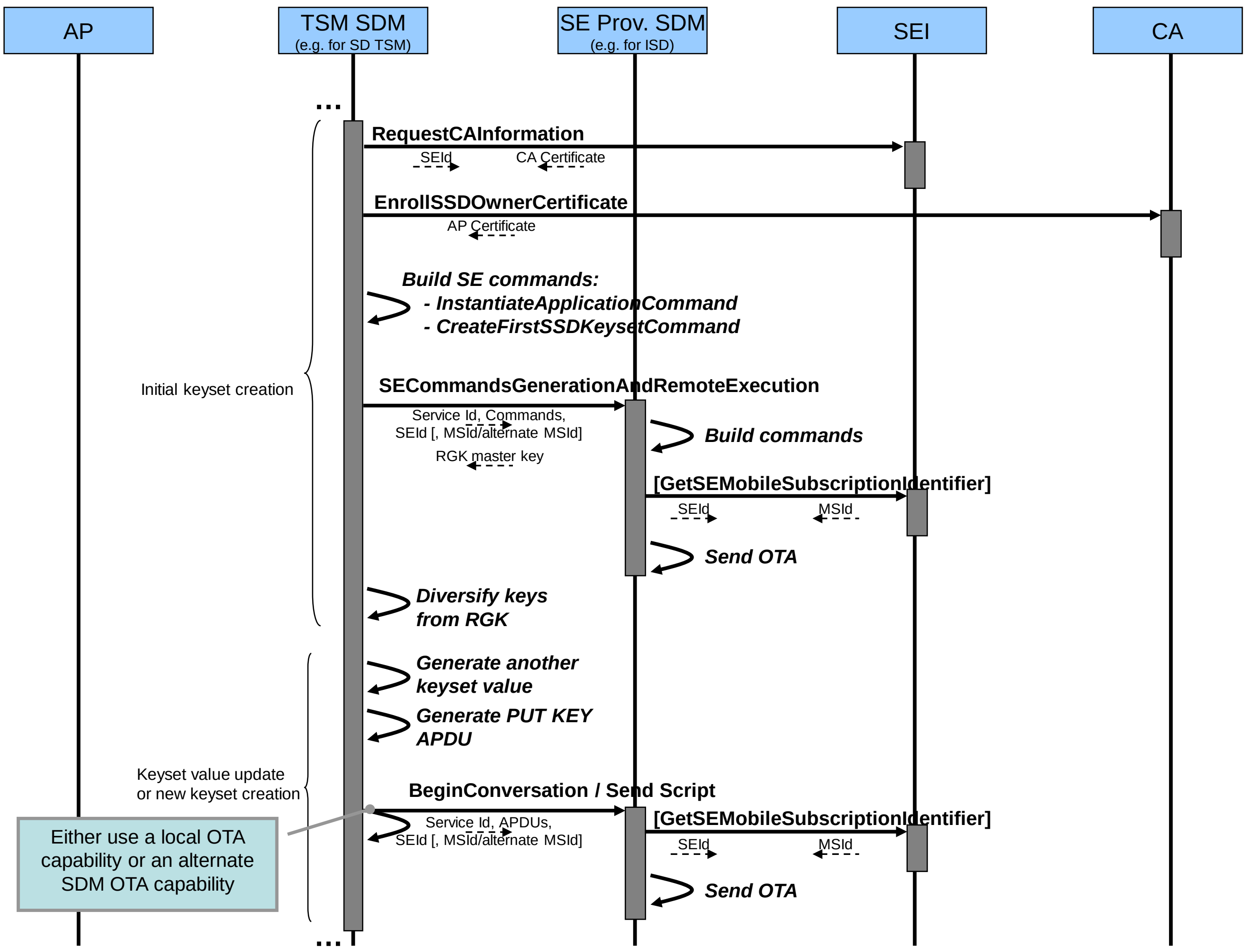

AP
TSM SDM (e.g. for SD TSM)
SE Prov. SDM (e.g. for ISD)
SEI
CA
RequestCAInformation
SEId
CA Certificate
EnrollSSDOwnerCertificate
AP Certificate
Build SE commands:
- InstantiateApplicationCommand
- CreateFirstSSDKeysetCommand
Initial keyset creation
SECommandsGenerationAndRemoteExecution
Service Id, Commands, SEId [, MSId/alternate MSId]
RGK master key
Build commands
[GetSEMobileSubscriptionIdentifier]
SEId
MSId
Send OTA
Diversify keys from RGK
Generate another keyset value
Generate PUT KEY APDU
Keyset value update or new keyset creation
BeginConversation / Send Script
Service Id, APDUs, SEId [, MSId/alternate MSId]
[GetSEMobileSubscriptionIdentifier]
SEId
MSId
Send OTA
Either use a local OTA capability or an alternate SDM OTA capability

Figure 7-7: Global sequence diagram – Phase 3 – Zoom on SD creation

In the scenario presented above, the on board key generation mechanism is used so that a RGK is returned by the Secure Element to the server. Other modes exist where the keys are created off-card and are pushed to the Secure Element.

Again, if the MS Id (or the Alternate MS Id) is not provided by the TSM SDM, then the SE Provider SDM can retrieve it by calling the `GetSEMobileSubscriptionIdentifier` on the SEI.

When the first keyset is created, the TSM SDM may want to change the key value, or to create another keyset in the SD. To do so, he shall open a secure channel using the keyset he just created in the previous phase and perform a normal PUT KEY that is protected by its own keys. If the TSM SDM has its own OTA capability (and if the first keyset that have been created is an OTA keyset), then he can do it by itself; else he shall rely on SE Provider SDM OTA capability, using the script sending functions (the provided script being secured by the first keyset values).

The SE Application personalization:

SE application personalization may be done autonomously by the TSM SDM, or request the help of another SDM to send the personalization scripts OTA.

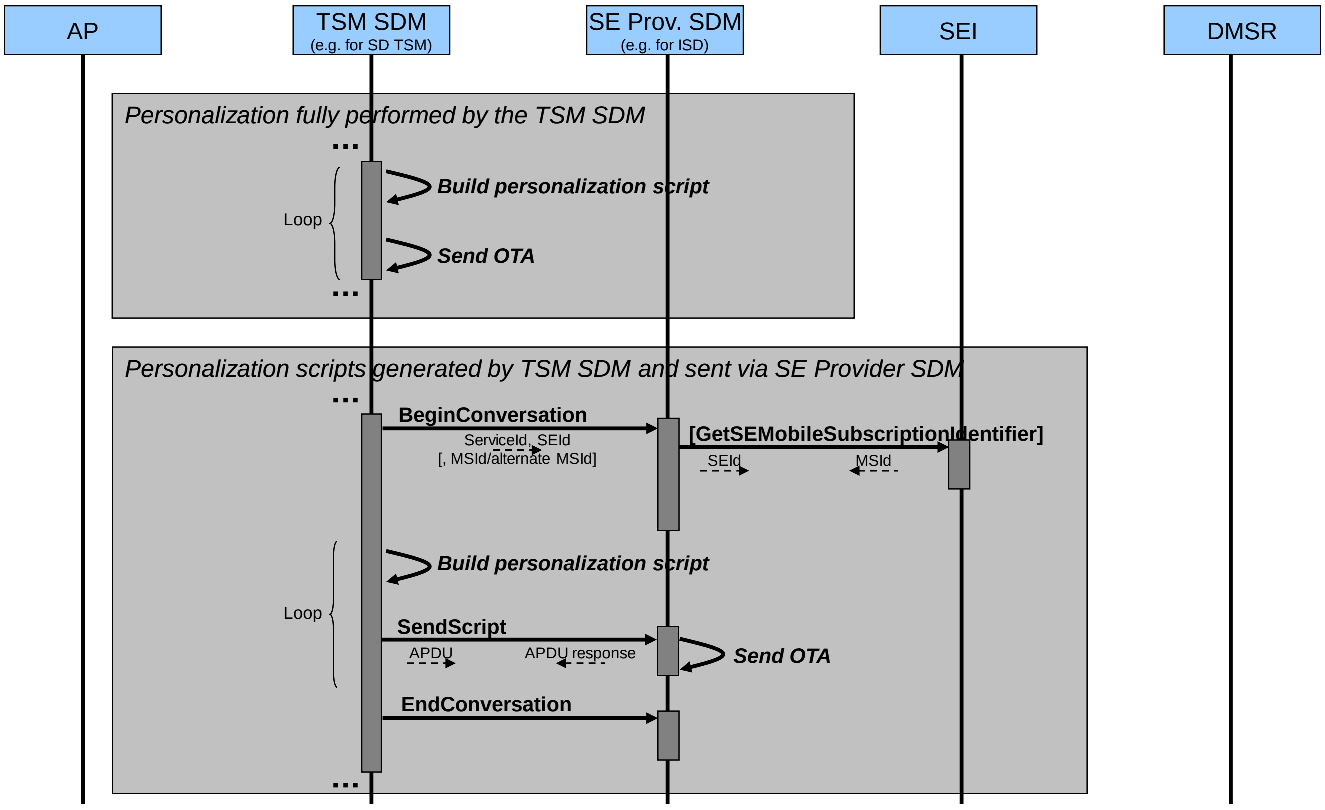

AP
TSM SDM (e.g. for SD TSM)
SE Prov. SDM (e.g. for ISD)
SEI
DMSR
Personalization fully performed by the TSM SDM
Loop
Build personalization script
Send OTA
Personalization scripts generated by TSM SDM and sent via SE Provider SDM
BeginConversation
ServiceId, SEId [, MSId/alternate MSId]
[GetSEMobileSubscriptionIdentifier]
SEId
MSId
Build personalization script
Loop
SendScript
APDU
APDU response
Send OTA
EndConversation

Figure 7-8: Global sequence diagram – Phase 3 – Zoom on SE Application personalization

Two scenarios are presented above:

- Either the TSM SDM is autonomous in the personalization script generation and sending OTA,
- Or the TSM SDM only generates the personalization script and uses the OTA capability of the SE Provider SDM (script sending functions).

The call to the `BeginConversation` function does not necessary open the true OTA communication. However, it shall ensure that the OTA communication will be possible. In particular, if the SE Provider SDM does not receive the MS Id from TSM SDM, the SE Provider SDM shall retrieve it by calling the `GetSEMobileSubscriptionIdentifier` on the SEI.

7.4 Phase 4: UI Management

Deployment of the User Interface of the mobile-NFC service may be required.

This section presents details related to the Device application management. The sequence diagrams below are to be inserted within the "install service" stage of the sequences presented in section 7.2, for phase 2 – Card Content Management.

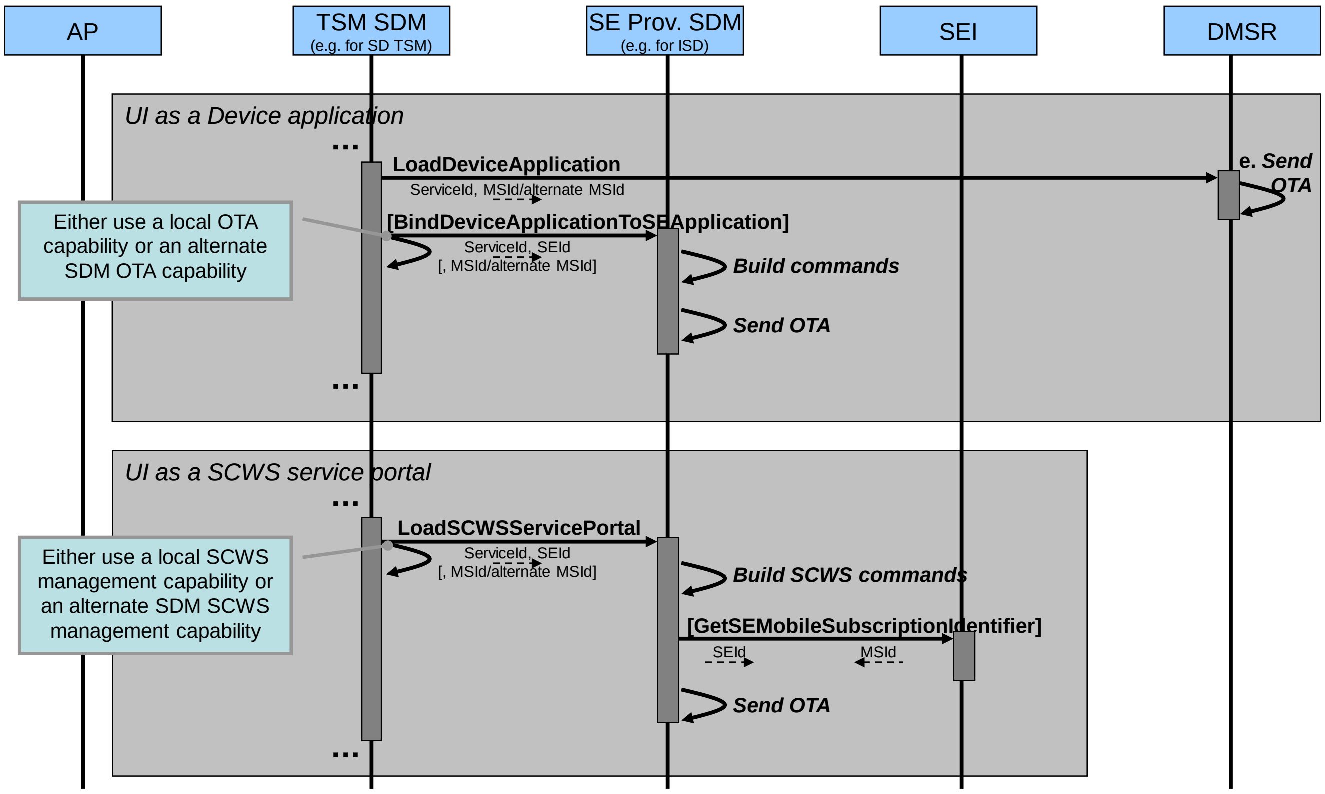

AP
TSM SDM
(e.g. for SD TSM)
SE Prov. SDM
(e.g. for ISD)
SEI
DMSR
UI as a Device application
LoadDeviceApplication
ServiceId, MSId/alternate MSId
e. Send OTA
Either use a local OTA capability or an alternate SDM OTA capability
[BindDeviceApplicationToSEApplication]
ServiceId, SEId
[, MSId/alternate MSId]
Build commands
Send OTA
UI as a SCWS service portal
LoadSCWSServicePortal
ServiceId, SEId
[, MSId/alternate MSId]
Either use a local SCWS management capability or an alternate SDM SCWS management capability
Build SCWS commands
[GetSEMobileSubscriptionIdentifier]
SEId
MSId
Send OTA

Figure 7-9: Global sequence diagram – Phase 4

If the UI is a Device application then the TSM SDM requests the DMSR to perform the Device application loading. Then, in case it is necessary to make a security binding of this Device application to a particular Secure Element application of the mobile-NFC service, then this binding is either done by the TSM SDM itself (if it has sufficient privileges and OTA capabilities), or the TSM SDM requests it to SE Provider SDM.

If the UI is a Smart Card Web Server service portal (UICC use case only), then either the TSM SDM performs the SCWS service portal download by itself (if it has SCWS administration capabilities), or requests it to the SE Provider SDM.

Again, for the call of the `LoadSCWSServicePortal`, if the MS Id (or the Alternate MS Id) is not known by the TSM SDM, then the SE Provider SDM can retrieve it by calling the `GetSEMobileSubscriptionIdentifier` on the SEI.

7.5 Phase 5

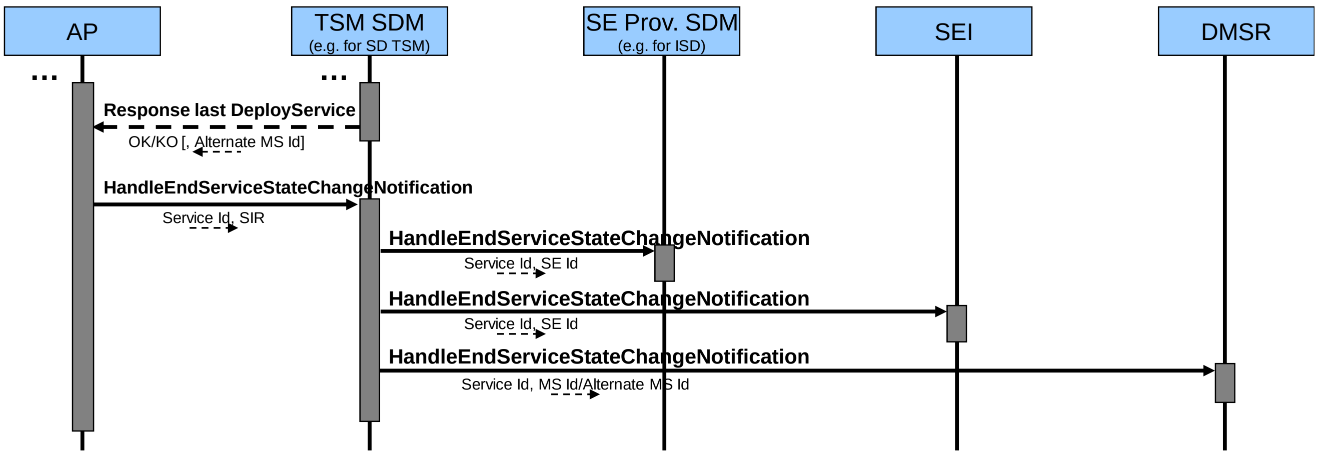

AP
TSM SDM
(e.g. for SD TSM)
SE Prov. SDM
(e.g. for ISD)
SEI
DMSR
Response last DeployService
OK/KO [, Alternate MS Id]
HandleEndServiceStateChangeNotification
Service Id, SIR
HandleEndServiceStateChangeNotification
Service Id, SE Id
HandleEndServiceStateChangeNotification
Service Id, SE Id
HandleEndServiceStateChangeNotification
Service Id, MS Id/Alternate MS Id

Figure 7-10: Global sequence diagram – Phase 5

The global mobile-NFC service management operation is finalized by notifying the end of the service deployment. Depending on the targeted role, the SE Id, the MS Id (or an alternate MS Id) are sent.

Note also that the notification to the SEI can be performed either by the TSM SDM or the SE Provider SDM, as a deployment choice.

8 Annex B: Common Types and Functions Track Changes (Informative)

In order to help implementers and users of this specification, the following tables identify the version of the specification for which a common type or a function has been Added (A), Updated (U), Deprecated (D) or Removed (R).

By common type *Updated*, we mean a change in the definition of the type (structure changed, allowed values range changed, etc.)

By function *Updated*, we mean any direct or indirect change of the function:

- Direct change: addition of a new parameter, for example
- Indirect change: update of a parameter type, addition of a new sub type/type extension of a parameter; update of the functional meaning/allowed values of a parameter; update of the functional behavior of the function.

Clarification of the functional description that does not change the global behavior of the function is not considered as a function update.

Deprecated indicates that the common type or the function will be removed in a future release.

Common types updates:

Impacted functions are mentioned only in case of update, deprecation or removal of the common type.

Common Type	A/U/D/R version	Impacted Functions
Mobile Subscription Identifier	A: 1.0	
SE Identifier	A: 1.0	
Service Identifier	A: 1.0	
Service Qualifier	A: 1.0 U: 1.0.2	`SECommandsGenerationAndRemoteExecution`, `GenerateDMToken`, `VerifyDMReceipt`, `BeginConversation`, `GetSEMobileSubscriptionIdentifier`, `CheckMobileSubscriptionEligibility`, `GetMobileSubscriptionAlternateIdentifier`, `LoadDeviceApplication`, `DeleteDeviceApplication`, `BindDeviceApplicationToSEApplication`, `UnbindDeviceApplicationToSEApplication`, `LoadSCWSServicePortal`, `DeleteSCWSServicePortal`, `HandleStartServiceStateChangeNotification`, `HandleEndServiceStateChangeNotification`
Service Instance Reference	A: 1.1	

Common Type	A/U/D/R version	Impacted Functions
Name Value Pair	A: 1.1	
Functions Input Header	A: 1.0	
Functions Output Header	A: 1.0	

Table 8-1: Change tracking on common types

Request-response functions:

Does not include the impacts due to common types updates (see Table 8-1 for such impacts).

Functions groups	Functions	A/U/D/R version
Global Eligibility Info	`CheckGlobalEligibility`	A: 1.1
Global Service Management	`LookupServiceInstanceReference`	A: 1.1
	`DeclareServiceInstanceReference`	A: 1.1 U: 1.1.2
	`GetServiceInstanceReferenceDescriptor`	A: 1.1
	`GetServiceState`	A: 1.1
	`DeployService`	A: 1.1 U: 1.1.2
	Service Commands:	
	• `InstallServiceCommand`	A: 1.1
	• `PersonalizeServiceCommand`	A: 1.1
	• `ActivateServiceCommand`	A: 1.1
	`UpgradeService`	A: 1.1
	`ExchangeServiceData`	A: 1.1
	`SuspendOrResumeService`	A: 1.1
	`TerminateService`	A: 1.1
Card Content Management	`SECommandsGenerationAndRemoteExecution`	A: 1.0 U: 1.0.1, 1.0.2
	SE Commands:	
	• `CreateFirstSSDKeysetCommand`	A: 1.0 U: 1.0.2
	• `LoadELFCommand`	A: 1.0
	• `ExtraditeCommand`	A: 1.0
	• `InstantiateApplicationCommand`	A: 1.0
	• `MakeSelectableApplicationCommand`	A: 1.0
	• `ApplicationRegistryUpdateCommand`	A: 1.0
	• `SetStatusCommand`	A: 1.0
	• `DeleteCommand`	A: 1.0

Functions groups	Functions	A/U/D/R version
Delegated Management	`GenerateDMToken`	A: 1.0
	`VerifyDMReceipt`	A: 1.0
Script Sending	`BeginConversation`	A: 1.0 U: 1.0.1
	`SendScript`	A: 1.0 U: 1.0.1, 1.0.2
	`EndConversation`	A: 1.0
SE Info	`GetSECapabilityProfileId`	A: 1.0
	`GetSEMobileSubscriptionIdentifier`	A: 1.0 U:1.0.2
SE Audit	`GetApplicationOrELFStatus`	A: 1.0 U: 1.0.1
	`GetSDFreeMemory`	A: 1.1 U: 1.1.2
CA Info	`AuditCAInformation`	`GetCAInformation`: A: 1.0 D: 1.0.2 `AuditCAInformation`: A: 1.0.2
Device Info	`GetDeviceCapabilityProfileId`	A: 1.0
Mobile Subscription Info	`CheckMobileSubscriptionEligibility`	A: 1.0
	`GetMobileSubscriptionAlternateIdentifier`	A: 1.0
	`GetMobileSubscriptionSEIdentifiers`	A: 1.1
Device Application Management	`LoadDeviceApplication`	A: 1.0
	`DeleteDeviceApplication`	A: 1.0
Device Application Binding	`BindDeviceApplicationToSEApplication`	A: 1.0
	`UnbindDeviceApplicationToSEApplication`	A: 1.0
SCWS Management	`LoadSCWSServicePortal`	A: 1.0
	`DeleteSCWSServicePortal`	A: 1.0
CCCM Certificates Management	`EnrollSSDOwnerCertificate`	A: 1.0

Table 8-2: Change tracking on request-response functions

Notification handler functions:

Functions groups	Notification handler functions	A/U/D/R version
Service Environment Change Notification	`HandleServiceEnvironmentChangeNotification`	A: 1.1
	`HandleActionDoneOnServiceNotification`	A: 1.1
Service Life Cycle Notification	`HandleStartServiceStateChangeNotification`	A: 1.0 U:1.0.2, 1.1.2

Functions groups	Notification handler functions	A/U/D/R version
	`HandleEndServiceStateChangeNotification`	A: 1.0 U:1.0.2, 1.1.2
Mobile Subscription Life Cycle Notification	`HandleMobileSubscriptionIdentifierChangedNotification`	A: 1.0 U:1.0.2
	`HandleMobileSubscriptionStatusChangeNotification`	A: 1.0 U:1.0.2
SE Life Cycle Notification	`HandleSERenewalNotification`	A: 1.0
	`HandleSEDeviceChangedNotification`	A: 1.0 U:1.0.2
	`HandleSEDeviceStatusChangeNotification`	A: 1.1
	`HandleSEMobileSubscriptionChangedNotification`	A: 1.0 U:1.0.2
	`HandleSEStatusChangeNotification`	A: 1.0 U:1.0.2
Device Application Life Cycle Notification	`HandleActionDoneOnDeviceApplicationNotification`	A: 1.1

Table 8-3: Change tracking on notification functions

9 Annex C: Device Capabilities (Informative)

This is a possible list of capabilities. This list is provided only as an example to help understanding the type of information that may be available in a profile.

Data name	Description	Type
NFC Chip	If the Device hardware includes a NFC chip component	`Boolean`
NFC Antenna	If the Device hardware includes a NFC antenna component	`Boolean`
SE Types	The list of supported SE types	`Enumeration {UICC, SMC, ESE}`
microSD placement	If the microSD placement makes it suitable for use when the microSD also includes NFC antenna	`Boolean`
Bluetooth	If the Device hardware includes a Bluetooth component	`Boolean`
BIP Server support	If BIP TCP Server Mode Class 'e' is supported by the Device	`Boolean`
BIP UDP Client support	If BIP UDP Client Mode Class 'e' is supported by the Device	`Boolean`
BIP TCP Client support	If BIP TCP Client Mode Class 'e' is supported by the Device	`Boolean`
HCI support	If HCI - ETSI TS 102 622 [16] – is supported by the Device	`Boolean`
HCI version	The version/release of ETSI TS 102 622 [16] implemented by the device. Conditional to the support of the HCI feature. Examples of possible versions are: Version 7.5	`Version`
SWP support	If SWP - ETSI TS 102 613 [15] - is supported by the Device	`Boolean`
SWP	The version/release of ETSI TS 102 613 [15] implemented by the device. Conditional to the support of the SWP feature. Examples of possible versions are: Version 7.7 Version 8.0 Version 9.0	`Version`
Screen Color	Number of colors supported by screen	`Integer`
Screen Size	Number of pixels in two dimensions	`Screen Size`
Screen re-Orientation Ability	Indicate the ability to reorient the screen	`Boolean`

Data name	Description	Type
Screen security	The ability of the device to protect NFC associated data when displayed to the user, expressed in terms of the entity providing the data isolation	`Enumeration {None, HLOS isolated, TEE isolated, ESE isolated, [others]}`
Input security	The ability of the device to protect NFC associated data when input from the user, expressed in terms of the entity providing the data isolation	`Enumeration {None, HLOS isolated, TEE isolated, ESE isolated, [others]}`
JSR177– SATSA-APDU support	If the SATSA-APDU mode of the JSR 177 is supported by the Device	`Boolean`
JSR177 – SATSA-JCRMI support	If the SATSA-JCRMI mode of the JSR 177 is supported by the Device	`Boolean`
JSR177 – SATSA-PKI support	If the SATSA-PKI mode of the JSR 177 is supported by the Device	`Boolean`
JSR177 – SATSA-CRYPTO support	If the SATSA-CRYPTO mode of the JSR 177 is supported by the Device	`Boolean`
JSR257 support	If the JSR 257 is supported by the Device	`Boolean`

Table 9-1: Device capabilities data list

Where Screen Size is:

Data name	Description	Type
X axis	Screen size in pixels on x axis	`Integer`
Y axis	Screen size in pixels on y axis	`Integer`

Table 9-2: Screen size

10 Annex D: Secure Element Capabilities (Informative)

This is a possible list of capabilities. This list is provided only as an example to help understanding the type of information that may be available in a profile.

Data name	Description	Type
NFC Chip	If the SE hardware includes a NFC chip component	`Boolean`
NFC Antenna	If the SE hardware includes a NFC antenna component	`Boolean`
SE type	Whether the SE is an UICC, Secure Memory Card or an Embedded Secure Element.	`Enumeration {UICC, SMC, ESE}`
UICC specific capabilities	The various capabilities regarding UICC as a SE	`UICC specific capabilities`
HCI support	If HCI according to ETSI TS 102 622 [16] is supported by the Device	`Boolean`
HCI version	The version/release of ETSI TS 102 622 [16] implemented by the SE. Conditional to the support of the HCI feature. Examples of possible versions are: Version 7.5	`Version`
SWP support	If SWP according to ETSI TS 102 613 [15] is supported by the Device	`Boolean`
SWP version	The version/release of ETSI TS 102 613 [15] implemented by the SE. Conditional to the support of the SWP feature. Examples of possible versions are: Version 7.7 Version 8.0 Version 9.0	`Version`
APDU script size limitations	Maximum size (in bytes) of the APDU scripts that can be processed by the SE.	`APDU script size limitations`
GP version	The version of GlobalPlatform Card specification implemented by the SE. Examples of possible versions are: Version 2.0.1 Version 2.1.1 Version 2.2 Version 2.2.1	`Version`

Data name	Description	Type
GP 2.2 Amendment A support	If amendment A of GP version 2.2 is supported. Conditional to the support of the version 2.2 of the GP Card specification.	`Boolean`
GP 2.2 Amendment A version	The version of amendment A that is implemented by the SE. Conditional to the support of the GP Amendment A feature. Examples of possible versions are: Version 1.0 Version 1.0.1	`Version`
GP 2.2 Amendment B support	If amendment B of GP version 2.2 is supported. Conditional to the support of the version 2.2 of the GP Card specification	`Boolean`
GP 2.2 Amendment B version	The version of amendment B that is implemented by the SE. Conditional to the support of the GP Amendment B feature. Examples of possible versions are: Version 1.0 Version 1.1 Version 1.1.1	`Version`
GP 2.2 Amendment C support	If amendment C of GP version 2.2 is supported. Conditional to the support of the version 2.2 of the GP Card specification 2.2	`Boolean`
GP 2.2 Amendment C version	The version of amendment C that is implemented by the SE. Conditional to the support of the GP Amendment C feature. Examples of possible versions are: Version 1.0 Version 1.0.1 Version 1.1	`Version`
GP 2.2 Amendment D support	If amendment D of GP version 2.2 is supported. Conditional to the support of the version 2.2 of the GP Card specification 2.2	`Boolean`

Data name	Description	Type
GP 2.2 Amendment D version	The version of amendment D that is implemented by the SE. Conditional to the support of the GP Amendment D feature. Examples of possible versions are: Version 1.0	`Version`
GP 2.2 Amendment E support	If amendment E of GP version 2.2 is supported. Conditional to the support of the version 2.2 of the GP Card specification 2.2	`Boolean`
GP 2.2 Amendment E version	The version of amendment E that is implemented by the SE. Conditional to the support of the GP Amendment E feature. Examples of possible versions are: Version 1.0	`Version`

Table 10-1: Secure Element capabilities data list

Where APDU script size limitations is:

Data name	Description	Type
Bearer	Defines the type of buffer	`Enumeration {SMS, CAT-TP to UICC, Secure packet over TCP to UICC, HTTP to UICC, ESE/SMC}`
Maximum script size	Number of bytes available for payload	`Integer`

Table 10-2: Maximum APDU script size

Where UICC specific capabilities is defined as:

Data name	Description	Type
UICC RAM version	The version of ETSI TS 102 226 [14] specification implemented by the UICC Examples of possible versions are: Version Rel5 Version Rel6 Version Rel7 Version Rel8	`Version`
UICC RAM Expanded mode support	If the Expanded mode of the UICC RAM is supported by the UICC.	`Boolean`
BIP support	If BIP Class 'e' is supported by the UICC.	`Boolean`
CAT-TP support	If CAT-TP according to ETSI TS 102 127 Rel [12] is supported by the UICC.	`Boolean`
CAT-TP version	The version of CAT-TP according to ETSI TS 102 127 Rel 6 [12] that is implemented by the UICC. Conditional to the support of the CAT-TP feature. Examples of possible versions are: Version 6.0.0 Version 6.1.0 Version 6.2.0 Version 6.3.0 Version 6.4.0 Version 6.5.0 Version 6.6.0 Version 6.7.0 Version 6.7.1 Version 6.8.0 Version 6.8.1 Version 6.9.0 Version 6.10.0 Version 6.11.0 Version 6.12.0 Version 6.13.0	`Version`
Secure packet over TCP support	If Secure packet over TCP according to ETSI TS 102 225 [13] and ETSI TS 102 226 [14] is supported.	`Boolean`

Data name	Description	Type
Secure packet over TCP version	The version of Secure packet over TCP according to ETSI TS 102 225 [13] and ETSI TS 102 226 [14] that is implemented by the UICC. Conditional to the support of the Secure packet over TCP feature. Examples of possible versions are: Version Rel9	`Version`
OTA HTTP support	If the RAM over HTTP according to ETSI TS 102 225 [13] and ETSI TS 102 226 [14] is supported by the UICC.	`Boolean`
OTA HTTP version	The version of the ETSI TS 102 225 Rel 9 [13] and ETSI TS 102 226 Rel 9 [14] that is implemented by the UICC. Conditional to the support of the OTA HTTP feature. Examples of possible versions are: Version Rel9	`Version`
OMA SCWS support	If the OMA SCWS is supported.	`Boolean`
OMA SCWS version	The version of SCWS implemented by the UICC. Conditional to the support of the OMA SCWS feature. Examples of possible versions are: Version 1.0 Version 1.1 Version 1.2	`Version`

Table 10-3: UICC-specific capabilities

11 Annex E: "Service Environment Change" Notifications Examples (Informative)

The current annex presents examples of notifications sent in case of Service environment change (see section 3.3.3)

11.1 Use cases around Secure Element

SE lost – device not lost:

`HandleServiceEnvironmentChangeNotification` raised:

- Event: 22 ⇨ SE Lost
- New Service Instance State: `Missing Mobile Environment Component`

AP is responsible for locking its backend system, if required.

Optionally, AP is responsible for requesting the deployment of the service when another SE will be available.

Later, SE recovered (if service not invalidated by AP in the meantime):

`HandleServiceEnvironmentChangeNotification` raised:

- Event: 121 ⇨ SE recovered
- New Service Instance State: `Active` (or in the status before SE was lost)

AP is responsible for unlocking its backend system, if required.

SE lost – SE Provider has terminated the service, for safety:

`HandleServiceEnvironmentChangeNotification` raised:

- Event: 22 ⇨ SE Lost
- New Service Instance State: `Terminated`

`HandleActionDoneOnServiceNotification` raised:

- Executed Action: 12 ⇨ SE application deleted

AP is responsible for requesting the deployment of the service when another SE will be available.

SE renewed by SE Provider, SDM not aware about SE eligibility info:

`HandleServiceEnvironmentChangeNotification` raised:

- Event: 122 ⇨ SE Renewed – No eligibility info
- New Service Instance State: `Not Deployed`

AP is responsible for requesting the deployment of the service.

11.2 Use cases around Device

Device lost – SE not lost:

`HandleServiceEnvironmentChangeNotification` raised:

- Event: 32 ⇨ Device Lost
- New Service Instance State: `Missing Mobile Environment Component`

AP is responsible for locking its backend system, if required.

Later, Device recovered (if service not invalidated by AP in the meantime):

`HandleServiceEnvironmentChangeNotification` raised:

- Event: 131 ⇨ Device recovered
- New Service Instance State: `Active` (or in the status before Device was lost)

AP is responsible for unlocking its backend system, if required.

Device changed – automatic re-delivery of Device application done by SDM:

`HandleServiceEnvironmentChangeNotification` raised:

- Event: 133 ⇨ Device changed - Device is eligible
- New Service Instance State: `Missing Mobile Environment Component`

`HandleActionDoneOnServiceNotification` raised:

- Executed Action: 101 ⇨ Device application re-deployed
- New Service Instance State: `Active` (or in the status before Device has changed)

Device changed – new device not eligible:

`HandleServiceEnvironmentChangeNotification` raised:

- Event: 134 ⇨ Device changed - Device is not eligible
- New Service Instance State: `Incomplete Deployment`

Device changed – no info on new device:

`HandleServiceEnvironmentChangeNotification` raised:

- Event: 132 ⇨ Device changed - No eligibility info
- New Service Instance State: `Incomplete Deployment`

AP is responsible for performing a new Check Global Eligibility and then requesting for the re-delivery of the service.

11.3 Use cases around Mobile Subscription

Mobile subscription suspended, but service still operational:

`HandleServiceEnvironmentChangeNotification` raised:

- Event: 1 ⇨ Mobile subscription suspended
- New Service Instance State: `Active` (or in the status before Mobile Subscription has been suspended)

Mobile subscription restored:

`HandleServiceEnvironmentChangeNotification` raised:

- Event: 101 ⇨ Mobile subscription restored
- New Service Instance State: `Active` (or in the status before Mobile Subscription has been suspended)

Mobile subscription suspended, service suspended by SDM:

`HandleServiceEnvironmentChangeNotification` raised:

- Event: 1 ⇨ Mobile subscription suspended
- New Service Instance State: `Active` (or in the status before Mobile Subscription has been suspended)

`HandleActionDoneOnServiceNotification` raised:

- Executed Action: 11 ⇨ SE application locked
- New Service Instance State: `Suspended`

Mobile subscription restored, service re-enabled by SDM:

`HandleServiceEnvironmentChangeNotification` raised:

- Event: 101 ⇨ Mobile subscription restored
- New Service Instance State: `Suspended`

`HandleActionDoneOnServiceNotification` raised:

- Executed Action: 111 ⇨ SE application un-locked
- New Service Instance State: `Active` (or in the status before Mobile Subscription has been suspended)

Or Mobile subscription restored, service to be re-enabled by AP:

`HandleServiceEnvironmentChangeNotification` raised:

- Event: 101 ⇨ Mobile subscription restored
- New Service Instance State: `Suspended`

AP is responsible to resume the service.

11.4 Use cases around both Secure Element and Device

SE and Device lost:

`HandleServiceEnvironmentChangeNotification` raised:

- Event: 42 ⇨ SE + Device Lost
- New Service Instance State: `Missing Mobile Environment Component` (no longer in end-user hands)

AP is responsible for locking its backend system, if required.

AP is responsible for performing a new Check Global Eligibility when new Device and SE will be available, and then request for the re-delivery of the service.

Later, Device recovered but not SE:

`HandleServiceEnvironmentChangeNotification` raised:

- Event: 131 ⇨ Device recovered
- New Service Instance State: `Missing Mobile Environment Component` (because SE still not available so service is not operational yet)

Or, SE recovered but not Device:

`HandleServiceEnvironmentChangeNotification` raised:

- Event: 121 ⇨ SE recovered
- New Service Instance State: `Missing Mobile Environment Component` (because Device still not available, so service is not operational yet)

Or, SE and Device recovered:

`HandleServiceEnvironmentChangeNotification` raised:

- Event: 141 ⇨ SE + Device recovered
- New Service Instance State: `Active` (or in the status before SE + Device has been lost)

AP is responsible for unlocking its backend system, if required.

SE and Device renewed:

2 `HandleServiceEnvironmentChangeNotification` events to be raised: SE renewed + Device renewed (see relevant use cases)

11.5 Use cases with events in the future

NFC option will be terminated in 5 days, but service still operational:

`HandleServiceEnvironmentChangeNotification` raised:

- Event: 12 ⇨ NFC mobile option terminated
- Date: current date + 5 days
- New Service Instance State: `Active`

AP is responsible to retrieve service information or update its backend for the future end of the service.

5 days later: NFC option is now terminated; service is terminated by SDM:

`HandleServiceEnvironmentChangeNotification` raised:

- Event: 12 ⇨ NFC mobile option terminated
- New Service Instance State: `Active`

`HandleActionDoneOnServiceNotification` raised:

- Executed Action: 12 ⇨ SE application deleted
- New Service Instance State: `Terminated`

Or 5 days later: NFC option is now terminated; service termination/suspension is under AP responsibility:

`HandleServiceEnvironmentChangeNotification` raised:

- Event: 12 ⇨ NFC mobile option terminated
- New Service Instance State: `Active`

AP is responsible to suspend or terminate the service.

12 Annex F: Guidelines to extend the XML schema (Informative)

To help implementers and users of this specification, this annex provides guidelines to extend the GlobalPlatform Messaging Specification schema.

This specification provides two ways of extending the XML schema:

- Where `<any data type>` is defined in various places among this specification as field type. In this case, function providers are allowed to use their own type definitions as well as existing type definitions of this specification (for example the Name Value pair type).

  See sections 3.3.2.5.3.2 and 3.3.2.7 for examples of usage of `<any data type>`.

- Where `ExtensionsType` is defined in various places among this specification to make the types extensible. In this case, function providers are allowed to encapsulate their own type definitions in order to extend existing types of this specification. However, function providers are not allowed to use existing type definitions of this specification within `ExtensionsType`.

  See section 4.5.1.1 for the `ExtensionsType` definition, and sections 4.5.1.2, 4.5.1.3, and 4.5.1.4 for examples of usage of `ExtensionsType`.

In both cases, it is recommended for function providers to create an XSD document describing the schema for own type definitions. That XSD document shall be given to function callers as part of the project documentation. A sample provider specific data type could be defined as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:gpx="http://myCompany.com/gp/system-messaging/extensions"
      targetNamespace="http://myCompany.com/gp/system-messaging/extensions"
      elementFormDefault="qualified"
      attributeFormDefault="unqualified">

      <xsd:element name="MyData" type="gpx:MyDataType"/>

      <xsd:complexType name="MyDataType">
            <xsd:sequence>
                  <xsd:element name="MyTransactionId" type="xsd:string"/>
            </xsd:sequence>
      </xsd:complexType>
</xsd:schema>
```

And these types can be used inside any `<any data type>` as follows (taking the `ServiceData` input data of the `ExchangeServiceData` function as an example):

```
<gpm:ServiceData>
      <gpx:MyData>
            <gpx:MyTransactionId>s1000998888</gpx:MyTransactionId>
      </gpx:MyData>
</gpm:ServiceData>
```

Please note that sample XSD and XML are informative only and function providers should define their own extensions to match their own needs.

13 Annex G: AFSCM Legacy

13.1 Introduction

The specification handles the AFSCM (Association Française du Sans Contact Mobile) specification [23] as valuable inputs for use cases and interface specification. However, contrary to AFSCM, GlobalPlatform does not specify business processes between parties: the use cases are only given as informative.

Nevertheless, it can be stated that:

- All AFSCM processes starting from process #3 are covered by the GlobalPlatform use cases,
- GlobalPlatform functions may be used to manage those AFSCM processes, as described in section 13.2.

13.2 AFSCM Processes by Using GlobalPlatform Functions

GSMA has delivered a specific guideline of this topic named "NFC MNO-SP interface Business Process implementation guidelines using GP protocols v1.0" [26].

This document sets out requirements for the implementation of AFSCM processes. It proposes an alternative implementation of AFSCM Processes defined in [23] using GP functions.

13.3 Convergences and Divergences

Convergence	Divergence
AFSCM use cases are covered (GlobalPlatform functions can be used to implement AFSCM processes)	GlobalPlatform allows also different use cases
Each AFSCM required functionality has an equivalent in GlobalPlatform (it might be required to have multiple GlobalPlatform calls to implement an AFSCM call)	AFSCM and GlobalPlatform functions are not directly compatible.
	AFSCM specifies business processes and API, whereas GlobalPlatform only specifies an API, based on informative use cases.

Table 13-1: Convergences and divergences

END OF DOCUMENT