GLOBALPLATFORM®
THE STANDARD FOR SECURE DIGITAL SERVICES AND DEVICES

GlobalPlatform Technology
Open Mobile API Specification
Version 3.3

Public Release

July 2018

Document Reference: GPD_SPE_075

THIS SPECIFICATION OR OTHER WORK PRODUCT IS BEING OFFERED WITHOUT ANY WARRANTY WHATSOEVER, AND IN PARTICULAR, ANY WARRANTY OF NON-INFRINGEMENT IS EXPRESSLY DISCLAIMED. ANY IMPLEMENTATION OF THIS SPECIFICATION OR OTHER WORK PRODUCT SHALL BE MADE ENTIRELY AT THE IMPLEMENTER'S OWN RISK, AND NEITHER THE COMPANY, NOR ANY OF ITS MEMBERS OR SUBMITTERS, SHALL HAVE ANY LIABILITY WHATSOEVER TO ANY IMPLEMENTER OR THIRD PARTY FOR ANY DAMAGES OF ANY NATURE WHATSOEVER DIRECTLY OR INDIRECTLY ARISING FROM THE IMPLEMENTATION OF THIS SPECIFICATION OR OTHER WORK PRODUCT.

Contents

Tables

Figures

1 Introduction

The API specified in this document enables mobile applications to access different SEs in mobile devices, such as SIMs or embedded SEs.

This specification provides interface definitions and UML diagrams to allow implementation on various mobile platforms and in different programming languages.

If the programming language supports namespace, it SHALL be `org.simalliance.openmobileapi`, except where explicitly changed in a Platform Binding document. For the procedural interface, the prefix '`OMAPI_`' is used instead.

Editor's Note: GlobalPlatform intends, in a future version of this specification, to split this document so that there will be a high-level functional API description with a set of platform/language bindings as a means to address ambiguities around Platform Bindings that have arisen in the past.

In particular, it is planned that the existing TEE SE API and WebAPI for Accessing SE should become Platform Bindings of OMAPI in future, with a new and separate document describing the widely deployed Java/Android binding.

Some text in this document – particularly in the area of Event Handlers – is intended to pave the way for such a split. This text is not intended to be used in the future to justify backward-incompatible changes to Platform Bindings of the Open Mobile API which have been deployed in the market.

1.1 Audience

This specification is intended for:

- Device manufacturers wishing to implement the interface in a device.
- Service providers developing an REE application in connection with a Secure Element application.

1.2 IPR Disclaimer

Attention is drawn to the possibility that some of the elements of this GlobalPlatform specification or other work product may be the subject of intellectual property rights (IPR) held by GlobalPlatform members or others. For additional information regarding any such IPR that have been brought to the attention of GlobalPlatform, please visit https://www.globalplatform.org/specificationsipdisclaimers.asp. GlobalPlatform shall not be held responsible for identifying any or all such IPR, and takes no position concerning the possible existence or the evidence, validity, or scope of any such IPR.

1.3 References

Table 1-1: Informative References

Standard / Specification	Description	Ref
GPC_SPE_034	GlobalPlatform Technology Card Specification v2.3	[GPCS]
GPD_SPE_013	GlobalPlatform Technology Secure Element Access Control v1.1 Specification for controlling access to SEs based on access policies that are stored in the SEs	[SEAC]

Standard / Specification	Description	Ref
GPD_SPE_024	GlobalPlatform Technology TEE Secure Element API v1.1.1	[TEE SE API]
GPP_SPE_002	GlobalPlatform Technology Open Mobile API – Android Binding	[Android Binding]
ISO/IEC 8825-1:2002 | ITU-T Recommendation X.690 (2002)	Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)	[ASN.1]
ISO/IEC 7816-3:2006	Identification cards – Integrated circuit cards – Part 3: Cards with contacts – Electrical interface and transmission protocols	[ISO 7816-3]
ISO/IEC 7816-4:2013	Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange	[ISO 7816-4]
ISO/IEC 7816-5:2004	Identification cards – Integrated circuit cards – Part 5: Registration of application providers	[ISO 7816-5]
ISO/IEC 7816-15:2004	Identification cards – Integrated circuit cards with contacts – Part 15: Cryptographic information application	[ISO 7816-15]
Java™ Cryptography Architecture API Specification & Reference	Go to the following website for JCA documentation: https://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html	[Java Crypto]
PKCS #11 v2.20	Cryptographic Token Interface Standard	[PKCS #11]
PKCS #15 v1.1	Cryptographic Token Information Syntax Standard	[PKCS #15]
RFC 2119	Key words for use in RFCs to Indicate Requirement Levels	[RFC 2119]

1.4 Terminology and Definitions

The following meanings apply to SHALL, SHALL NOT, MUST, MUST NOT, SHOULD, and MAY in this document (refer to [RFC 2119]):

- **SHALL** indicates an absolute requirement, as does **MUST**.
- **SHALL NOT** indicates an absolute prohibition, as does **MUST NOT**.
- **SHOULD** indicates a recommendation.
- **MAY** indicates an option.

Table 1-2: Terminology and Definitions

Term	Definition
Applet	A general term for an SE application. An application as described in [GPCS] which is installed in the SE and runs within the SE. For example a Java Card™ application or a native application.
Application	Device/terminal/mobile application. An application which is installed in and runs within the mobile device.
Channel	An open connection between an application on the device (e.g. mobile phone) and an applet on the SE.
Execution Environment	An environment hosting and executing software. This could be an REE such as Android, Linux, or Windows; it could be a GlobalPlatform TEE; or it could be an abstract platform such as a Web Browser.
Platform Binding	A concrete implementation of the Open Mobile API. This will typically fix some or all of the following characteristics: • The Execution Environment hosting the Open Mobile API • The programming language used to implement Applications. The Platform Binding will expose the APIs described in this specification as an API in this programming languages.
Secure Element	A tamper-resistant secure hardware component which is used in a device to provide the security, confidentiality, and multiple application environment required to support various business models. May exist in any form factor, such as embedded or integrated SE, SIM/UICC, smart card, smart microSD, etc.
Session	An open connection between an application on the device (e.g. mobile phone) and an SE.
Tamper-resistant secure hardware	Hardware designed to isolate and protect embedded software and data by implementing appropriate security measures. The hardware and embedded software meet the requirements of the latest Security IC Platform Protection Profile (BSI-PP-0084) including resistance to physical tampering scenarios described in that Protection Profile.

1.5 Abbreviations and Notations

Table 1-3: Abbreviations and Notations

Abbreviation / Notation	Meaning
AID	Application Identifier
APDU	Application Protocol Data Unit (as per ISO/IEC 7816)
API	Application Programming Interface
ASN.1	Abstract Syntax Notation One (as per [ASN.1])
ATR	Answer to Reset (as per ISO/IEC 7816)
CLF	Contactless Front-end
DER	Distinguished Encoding Rules of ASN.1
DF	Dedicated File
DM	Device Management
EF	Elementary File
FCP	File Control Parameters
FID	File Identifier
ISO	International Organization for Standardization
JCE	Java Crypto Extension
MF	Master File
REE	Rich Execution Environment
SFI	Short File Identifier
SPI	Serial Peripheral Interface bus
SS	Secure Storage
TEE	Trusted Execution Environment

1.6 Revision History

Note: This specification was developed by SIMalliance and transferred to GlobalPlatform in November 2016.

GlobalPlatform technical documents numbered *n*.0 are major releases. Those numbered *n*.1, *n*.2, etc., are minor releases where changes typically introduce supplementary items that do not impact backward compatibility or interoperability of the specifications. Those numbered *n.n*.1, *n.n*.2, etc., are maintenance releases that incorporate errata and precisions; all non-trivial changes are indicated, often with revision marks.

Table 1-4: Revision History

Date	Version	Description
28.02.2011	1.0	Initial Release 1.0
16.03.2011	1.01	Minor corrections
04.05.2011	1.1	Clarifications for several functions in the transport layer

Date	Version	Description
12.07.2011	1.2	Correction for open Basic Channel
30.09.2011	2.0	Adding descriptions of the service layer, corrections in the transport layer, adding getSelectResponse() to channel class
14.10.2011	2.01	Minor corrections
4.11.2011	2.02	Corrections in diagrams of the transport layer
19.06.2012	2.03	Clarification on Chapter 10 (since GlobalPlatform SEAC spec is released), on 6.4.4. and 6.7.6 / 6.7.7
15.07.2013	2.04	Clarification on Chapter 5, 6.2, 6.7.6, 6.7.7, 6.8.6, 7.1, 7.6.1, 7.6.3, 7.6.4, 7.6.5, 7.8.1 and Chapter 8. Added 6.4.5, 6.8.7
28.01.2014	2.05	Chapter 3, clarification on the namespace; Chapter 6.4.2: IllegalStateError added; Chapter 6.6.1 changed definition of reader name according request from GSMA; Chapter 6.8.6: clarification on handling of status words
26.08.2014	3.0	Procedural interface added to chapter 6; reference header for Ansi-C defined in Annex; P2 parameter added for openBasicChannel and openLogicalChannel (previous interface without P2 is kept)
16.04.2015	3.1	Updates in Chapter 1.1; Informative reference added in Chapter 2, Updates added in Chapter 6.1; The definition of getVersion method is changed; The definition of SEService constructor, openSession, openBasicChannel, openLogicalChannel and transmit methods is updated; Chapter 8 renamed; UML diagrams updated; Annex A updated.
03.02.2016	3.2	Updates in -6.2.1, 6.2.7f, 6.2.7h, 6.2.8a, 6.2.8g, 6.2.8h and 8 -new methods introduced: 6.2.6f, 6.2.6g, 6.2.8 f -Chapter 6.1.1 updated -Chapter 2 updated -new Chapters 6.1.2, 6.2.4, 6.2.5 are introduced for Handling of Events -matching updates in Procedural interface -UML diagrams are updated -Annex A updated
November 2016	3.2	Transfer of the technology from SIMalliance to GlobalPlatform Integration of SIMalliance Open Mobile API specification: Second Errata for v3.2 Service API now obsolete

Date	Version	Description
July 2018	3.3	Public Release; changes include: • Comment on the future split of the OMAPI specification into a main specification and appendices. • Text defining plug-ins. • Modified Architecture Overview. • Text giving implementation guidance for non-OO targets. • More detailed definitions of the Types used in the abstract specification. • Note that `Reader.openSession()` may not involve communication with SE. • Note that UICC may restrict the use of the Basic Channel. • Removed the C language binding. • Changed the architecture diagram to better reflect device connectivity options. • Clarified channel management behavior with openLogicalChannel. • Clarified illegal AID lengths which might lead to IllegalParameterError.

2 Architecture

Figure 2-1 provides an overview of the Open Mobile API architecture.

The architecture is divided into the following functional layers:

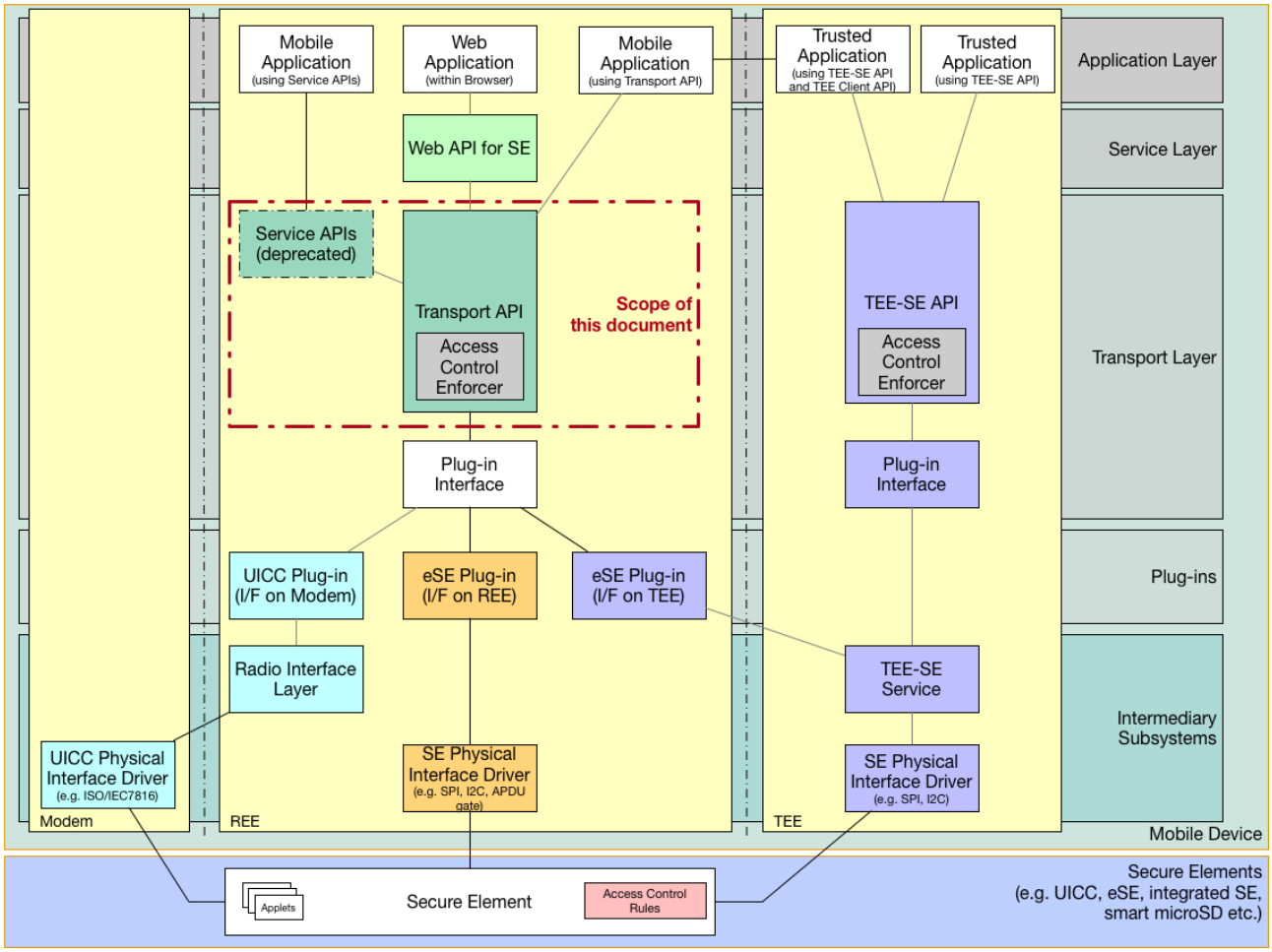
- The Transport Layer, when accessed via the Transport API, provides an application with general access to SEs. The Transport Layer uses APDUs to communicate with an SE (see section 4 for details).

  Beneath the Transport Layer are plug-ins which provide an interface abstraction between the core functionality of the Open Mobile API and low-level mechanisms for communication with Secure Elements.

  - Concrete bindings of the Open Mobile API to specific programming languages or software platforms may provide specific guidelines on plug-in implementations.
  - There is one instance of a plug-in per Reader.
  - Some plug-ins may offer direct and unmediated communication with the associated Secure Element. However, other plug-ins may communicate via intermediary subsystems, such as a modem or RIL.
  - There may be aspects of the behavior of plug-ins that are out of scope of this document; for example: error handling, power management, and recovery strategies. Plug-in developers are responsible for ensuring that such strategies do not impact applications using the Open Mobile API.
  - Plug-in functionality and behavior are discussed further in section 7.
- The Service Layer provides a more abstract interface to various functions on the SE. They may be easier to use by application developers than the generic Transport API. One example could be an email application that uses a `sign()` function of the Crypto API and lets the Crypto API do all the APDU exchange with the SE (rather than handle all the required APDUs directly in the email application).

  Note: The Service Layer in this document is deprecated.
- The Application Layer represents the various applications that make use of the Open Mobile API.

Figure 2-1: An Example of Possible Open Mobile API Architecture



A possible architecture for implementing the Open Mobile API on a Mobile Device is shown in Figure 2-1. Since aspects of an Open Mobile API implementation on a given device are REE and hardware dependent, some aspects of this architecture might look different depending on the REE.

The description of the APIs uses an abstract object-oriented execution environment supporting aggregation, exception handling, concurrency, callbacks, and inner classes. Some guidance is provided on mappings for environments that do not support some or all of these features.

In Figure 2-1, we show that plug-ins enabling connection to a Secure Element may communicate with Execution Environments other than the one hosting the Open Mobile API:

- A Secure Element may be connected to the Transport API via a plug-in which uses a physical interface which is hosted and controlled by the REE. Examples of such interfaces include I^2C, SPI, or an APDU gate hosted by the REE Service supporting a CLF.
- A Secure Element (generally supporting UICC functionality) may be connected to the Transport API via a plug-in which uses a physical interface, such as that defined in ISO/IEC 7816, which is hosted and controlled by a modem. In the diagram, a Radio Interface Layer provides a proxy for communication between the REE and the Modem.

- A Secure Element may be connected to the Transport API via a plug-in which uses a physical interface which is owned and controlled by a TEE, such as I^2C or SPI. The arrangement presented shows a way of allowing both Trusted Applications using the TEE Secure Element API specification [TEE SE API] and Mobile Applications using either the Service APIs or the Transport API to access the Secure Element. Note that in this scenario an REE Application may also communicate indirectly with the Secure Element via a Trusted Application which itself uses the TEE SE API (e.g. using the TEE Client API). Such communication is out of scope of this document.

Other arrangements of Secure Element connectivity from the REE or other Execution Environments are valid.

Secure Elements exist in many form factors including Embedded Secure Element, smartSD, smart microSD, UICC, eUICC, and integrated Secure Elements. When we refer to a Secure Element in Figure 2-1, we are referring to a GlobalPlatform compliant Secure Element in any form factor.

The Transport API includes an implementation of an Access Control Enforcer. In conjunction with ARA applets and/or and ARF in the connected Secure Elements, this enables the Open Mobile API to use the GlobalPlatform Secure Element Access Control specification ([SEAC]) to ensure, to the level of security provided by the Execution Environment hosting the Access Control Enforcer, that only applications authorized by the SE issuer or authorized SD owners can communicate with applets on the SE.

3 API Description

In general, the API described in this specification defines an abstract interface which enables a software platform that supports object-oriented concepts, such as exceptions or objects and instances, to access a SE.

The interface and data types are not bound to a specific software platform or programming language. Instead, they are defined through a logical type that can be mapped accordingly to the corresponding platform representation.

The methods are described as follows:

```
<return value type> <method name> ( <parameter1 type> <parameter1 name> …)
```

Concrete bindings of this specification for particular software platforms may be defined, for example:

- The Open Mobile API – Android Binding specification [Android Binding] defines the concrete binding of this specification on the Android platform.
- [TEE SE API] defines the concrete binding for this specification on a GlobalPlatform TEE.

The following types are used to describe return values, parameters, and errors. If supported by the platform, errors may be mapped to exceptions.

3.1 Programming Language vs. ISO 7816-4 Language Conventions

This document describes an API that can be used to implement use cases involving a device and an SE.

In this document, API definitions use Java-like language conventions.

Interactions with the SE use the conventions found in ISO/IEC 7816-4 [ISO 7816-4]. The following conventions, in particular, should be noted:

- The use of 'XX' or 'xx' denotes a byte, represented as a pair of hexadecimal digits, where each x can take one of the following values: ‘0’..‘9’ for the decimal values 0 to 9; ‘A’..‘F’ for the decimal values 10 to 15; ‘X’ for any decimal value between 0 and 15.
- To guarantee an unambiguous mapping to bytes, this syntax can only be used with an even number of hexadecimal digits (e.g. 'A34' is illegal).
- An unlimited number of bytes can be concatenated within a pair of straight single quotes.

Table 3-1 includes examples of the correspondence between C language and [ISO 7816-4].

Table 3-1: Language Conventions: Java-like and ISO 7816-4

Java-like convention	ISO 7816-4	Meaning
`0x2A`	'2A'	A single byte with hexadecimal value 2A
`{0x4E, 0x2A}`	'4E2A'	Two bytes, the first with the hexadecimal value 4E and the second with the hexadecimal value 2A
`{0x61, 0xXX}`	'61XX'	Two bytes, the first with the hexadecimal value 61 and the second with any legal value for a byte. Application developers should note that `0xXX` is not legal syntax in most programming languages but is used here for correspondence with ISO 7816-4.

3.2 Value Types

The value types in Table 3-2 are used during interactions with the API. These will be mapped to appropriate concrete types for any specific language or software Platform Binding.

Table 3-2: Value Types

Type	Definition
`Boolean`	A primitive type, can be `true` or `false`.
`Int`	A primitive type, mapped to the integer of the platform. No assumption is made about the endianness of the platform representation of the `Int` type.
`Byte[]`	An array (ordered sequence) of single byte (8 bit) values.
`String`	A string of characters
`Context`	An abstract representation of the execution context of an application
`Void`	Not a type, indicates that the method has no return value.

3.3 Error Types

The error types in Table 3-3 are propagated by the API to the caller using appropriate concrete mechanisms for any specific language binding.

Different mechanisms may be chosen for error propagation to the calling application:

- Errors can be mapped to exceptions in environments where this is the idiomatic mechanism for error propagation and handling (e.g. Java, C++, C#, Python)
  - E.g. (Java language) `public Session openSession() throws IOError`
- Errors can be mapped to function return values in environments where this is the idiomatic mechanism for error propagation and handling (e.g. C)
  - E.g. (C language) `TEE_Result TEE_SEReaderOpenSession(…)`
- Errors can be propagated via abstract data types in environments where this is the idiomatic mechanism for error propagation and handling
  - E.g. (F# language) `Either<'Session, 'Error> openSession()`

Note: The `Success` value is generally used only with the function return value idiom – other idioms usually do not require it to be used.

Table 3-3: Error/Return Types

Type	Definition
`Success`	No error was encountered.
`NullPointerError`	`Null` was given where data is required.
`IllegalParameterError`	A function or method was given an incorrect parameter (e.g. bad format for an APDU).
`IllegalStateError`	A function or method was used in the wrong context (e.g. being closed).
`SecurityError`	Security conditions were not satisfied.
`ChannelNotAvailableError`	The basic channel was blocked/busy or no logical channel was available.
`NoSuchElementError`	An AID could not be found.
`IllegalReferenceError`	A reference could not be found.
`OperationNotSupportedError`	An operation is not supported.
`IOError`	An error related to communication (I/O).
`GeneralError`	A general error occurred – no further diagnosis available.

3.4 Constants

- **Boolean constants**
  - The `Boolean` type can hold two values: `true` and `false`.
- **Int constants**
  - The `Int` type can hold signed values that fall within at least the range (-2^{31} to $2^{31} - 1$).
  - `Int` constants can be expressed in a hexadecimal format by preceding them with '0x', e.g. `0x3F0A`.
- **String constants**
  - Constants of `String` type consist of characters within quotations, e.g. "Hello World!".
  - No assumption is made regarding the character set that can be represented by a `String`.
- **Byte constants**
  - The `Byte` type can hold unsigned values between `0` and 255.
  - `Byte` constants can be expressed in a hexadecimal format by preceding them with '0x', e.g. `0x3A`.
- **Byte arrays**
  - Constant `Byte` arrays can be initialized by placing the individual values of each element between curly brackets (aka braces) '{' and '}', e.g. `{0x63, 0x02}`.
  - Unknown or don't care values can be represented by 'XX', e.g. `{0x63, 0xXX}`
- **Null**
  - The `Null` value indicates that no value is provided. `String`, `Byte[]`, and `Context` types can take `Null` values in this specification.

3.5 Asynchronous Notifications

This specification defines APIs to enable asynchronous notifications. The essential behavior that is captured by these APIs is that of the Observer pattern:

- A mechanism whereby observer objects can register with, or deregister with, some subject object.
- A mechanism whereby observer objects are asynchronously notified of changes in the state of some subject object.
- A mechanism whereby subject objects initiate notification to the observer objects.

In this document, asynchronous notifications are defined in terms of callbacks. While these are widely supported, they are not necessarily a preferred asynchronous notification mechanism on some systems and may not even be supportable in the form defined here. In such cases, the concrete binding for this specification to a particular software platform or programming language MAY provide an equivalent alternative implementation of the Observer pattern. Some examples include:

- Single threaded/co-operative multitasking software platforms could implement asynchronous notifications using an event queue.
- Platforms offering lightweight asynchronous broadcast mechanisms might use these in preference to callbacks.
- Message-passing systems might implement asynchronous notifications by sending messages with the required contents.

4 Transport API

The Transport API, as part of the Open Mobile API, provides a communication framework to SEs available in the Open Mobile device.

4.1 Overview

The role of the Transport API is to provide the means for applications to access the SE(s) available on the device. The access provided is based on the concepts defined by ISO/IEC 7816-4 [ISO 7816-4]:

- APDUs: The format of the messages that are exchanged with the SE, basically a sequence of bytes, is sent to the SE (or more precisely to an applet in the SE) and the SE responds with another sequence of bytes. For details of the exact formatting of such byte sequences, refer to [ISO 7816-4].
- Basic and Logical Channels: These channels provide a logical link between an entity on the device and an applet on the SE. Multiple Logical Channels may be opened at any time, and the OMAPI Transport Layer is responsible for any necessary access serialization.

This API relies on a ‘connection’ pattern. The client application (running on the device connected to the SE, e.g. the phone) opens a connection to the SE (a ‘session’) and then opens a logical or basic channel to an applet running in the SE.

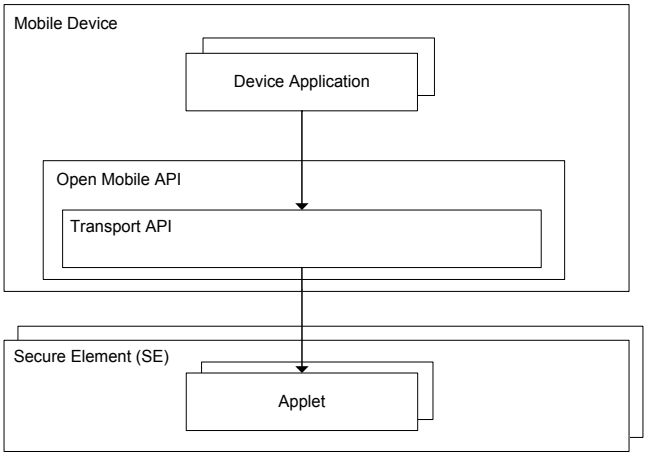
On top of this pattern, there are a number of constraints that are enforced by the system.

A major benefit of the Transport API is that it abstracts the management of communication channels across multiple relying applications using the channel management APDUs defined in [ISO 7816-4]. Applications are therefore forbidden from sending ‘channel management’ APDUs themselves, as this would break the isolation feature given by the channel. Once a channel is opened, it is allocated to communicate with one and only one applet in the SE. In the same manner, the SELECT by DF name APDU cannot be sent by the terminal application.

The restrictions for the system should be implemented in the modules that are directly handling the communication with the SE and not in the API itself, to ensure that attackers cannot overcome the APDU filters. Thus if possible, the baseband should take care of the filtering or at least the RIL that communicates with the baseband.

Once a logical channel is allocated to a mobile application, only APDU commands coming from this mobile application SHOULD be transmitted on this logical channel.

Figure 4-1: Transport API Overview



4.1.1 General Rules for Handling of Status Word

This specification is based on ISO specifications, in order to behave correctly with any applet across different sectors and applications (banking, transport, identity, etc.).

Besides ISO 7816, the OMAPI Specification does not integrate the specific behaviors of any other specifications. This is intentional and ensures compliant behavior with any type of applications.

Using Transport Layer T=1

The API or the underlying implementation SHALL handle the protocol T=1 as specified in ISO/IEC 7816-3 [ISO 7816-3] and returns data (if available) and received status word to the application. The API or underlying implementation SHALL NOT automatically issue GET-RESPONSE APDU on reception of any status word.

Using Transport Layer T=0

Unless otherwise specified, when sending a command APDU, this specification requires the following status word management for all methods defined in this document:

- For status word `{0x61, 0xXX}`, the API or underlying implementation SHALL issue a GET RESPONSE command as specified in [ISO 7816-3]. The same behavior has to be applied for the scenario where the SE returns overall response data of more than 256 bytes to the APDU. If an Error SW is received to GET RESPONSE command, all data received so far SHALL be discarded and the Error SW SHALL be returned.
- For the status word matching `{0x6C, 0xXX}`, the API or underlying implementation SHALL reissue the input command as specified in [ISO 7816-3]. If an Error SW is received to the reissued command, all data received so far SHALL be discarded and the Error SW SHALL be returned.
- For status words other than `{0x61, 0xXX}` and `{0x6C, 0xXX}`, the API (or underlying implementation) SHALL NOT handle the received status words internally (e.g. it SHALL NOT send GET RESPONSE automatically). The API (or underlying implementation) SHALL provide the status word together with data, if data is also received with the SW to the calling Mobile Application.
- Specific rules for handling of status word Warning (`{0x62, 0xXX}` and `{0x63, 0xXX}`)
  - In case of `transmit()` method for the status words `{0x62, 0xXX}` and `{0x63, 0xXX}`, the API or underlying implementation SHALL handle these status words according to the behavior set by the `setTransmitBehaviour(Boolean expectDataWithWarningSW)` method.
  - In case of any of the following:
    - `openBasicChannel(byte[] aid)`
    - `openBasicChannel(byte[] aid, byte P2)`
    - `openLogicalChannel(byte[] aid)`
    - `openLogicalChannel(byte[] aid, byte P2)`
    - `selectNext()`

    the GET RESPONSE with Le = 0x00 SHALL be sent also in case of SW warning (`{0x62, 0xXX}`, `{0x63, 0xXX}`) received as first response for the SELECT command and independently of the setting in the `setTransmitBehaviour(Boolean expectDataWithWarningSW)` method.

Note that the handling of GET RESPONSE specified above implies that the API (or underlying implementation) SHALL handle response data even if it is more than 256 bytes.

Note: Developers of Card Applets intended to be used from the Open Mobile API SHOULD avoid implementation of warning behavior which requires the use of `expectDataWithWarningSW` to avoid possible confusion on the part of device application developers on the operation of this special-case handling for case 4 APDU.

4.1.2 Handling of Events

The Open Mobile API provides mechanisms to notify Mobile Applications of changes to the availability of functionality hosted on SEs.

Most target platforms provide mechanisms which support notification of external events. The preferred embodiment of such mechanisms is often platform specific and this specification acknowledges that it is desirable that a platform mapping of the Open Mobile API is best implemented using the idioms of that platform.

Event handling is mandatory if the platform supports it. The mechanism and API used SHALL be documented in the Platform Binding.

Using the Event API, applications can be notified of asynchronous events that may occur on the Reader.

IOErrorEvent

When `IOError` occurs, the API SHALL implement the following behavior: Before returning the `IOError`, the API SHALL close ALL the opened sessions and channels of ALL applications on the related reader. When all the sessions and channels are closed, the API SHALL asynchronously notify all applications that registered to receive Open Mobile API Event notifications from the related reader.

Note: `IOError` is generally not recoverable as it is the consequence of an I/O failure. Some SE are removable and may become available on reinsertion into the device. Others are fixed in the device, in which case `IOError` may indicate a hardware problem.

SEInsertedEvent and SERemovalEvent

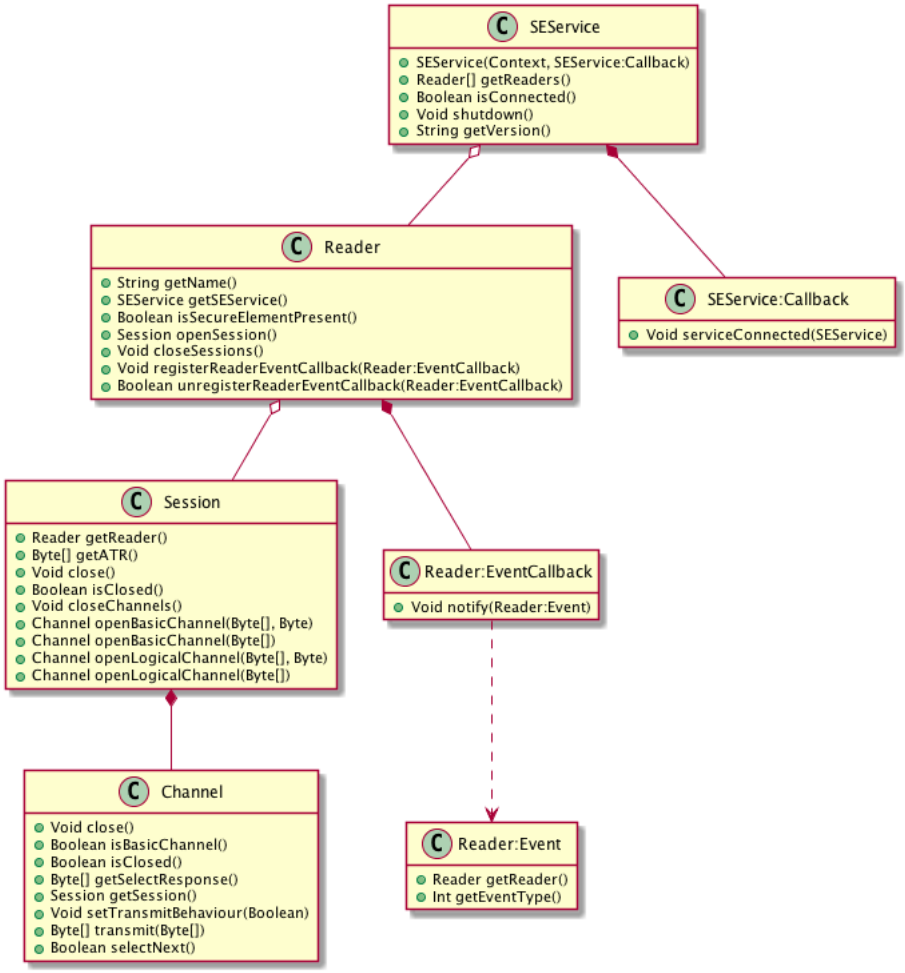
`SEInsertedEvent` and `SERemovalEvent` occurs upon insertion/removal of SE on a specific reader. With these events, applications can be notified about insertion/removal without polling the SE presence. Before issuing `SERemovalEvent`, the API SHALL close ALL the opened sessions and channels of ALL applications on the related reader. When all the sessions and channels are closed, the API SHALL asynchronously notify all applications that registered to receive Open Mobile API Event notifications from the related reader.

Reader instances for which the SE is not removable are not required to support `SEInsertedEvent` and `SERemovalEvent`.

4.2 Object Interface

This class diagram contains all classes of the Transport API. The `SEService` class realizes a connector to the SE framework system and can be used to retrieve all SE readers available in the system. The Reader class can be used to access the SE connected with the selected reader. The Session class represents a session to an SE established by the reader and allows different communication channels to be opened represented by the `Channel` class.

Figure 4-2: Transport API Class Diagram



4.2.1 Usage Pattern

The usage pattern of the Transport API is as follows:

1. The application gets access to the SE service(s):

   It creates an instance of `SEService` according to the Platform Binding API definition. Some Platform Bindings require the caller to provide an entity implementing the asynchronous notification API when creating the `SEService` instance.

   Application developers should note that the `SEService` instance does not represent a connection with the SE itself, but with the subsystem implementing the SE access functionality.

2. The Application enumerates the available readers.

   Readers are the slots where SEs are connected (in a removable or non-removable manner), and a device may support several Readers. There is usually one `Reader` instance for each SE. Many Platform Bindings enforce a naming convention for Readers to assist Application developers in selecting the required SE when there is more than one on the platform.

   Once the user or an application-specific algorithm has selected a `Reader`, then the application opens a session on this `Reader`.

3. Where the Platform Binding supports it, the Application may register for asynchronous notifications from the `Reader`, e.g. by registering `Reader.EventCallback` on `Reader` instances.

4. With the session opened in step 2, the application can retrieve the ATR of the SE, and if it matches with one of the known ATRs, it can start opening channels with applets in the SE.

5. To open a channel, the application will use the AID of the applet or use the default applet on the newly opened channel.

6. Then the terminal application can start transmitting APDUs to the applet.

7. Once done, the application can close any existing channels or even sessions, and its connection to the `SEService`.

4.2.2 Class: SEService

The `SEService` realizes the communication to available SEs on the device.

This is the entry point of this API. It is used to connect to the infrastructure and get access to a list of SE readers.

4.2.2.1 Constructor: SEService(Context context, SEService.CallBack listener)

Establishes a new connection that can be used to connect to all the SEs available in the system. The connection process can be quite long, so it happens in an asynchronous way. It is usable only if the specified listener is called or if `isConnected()` returns `true`.

The call-back object passed as a parameter will have its `serviceConnected()` method called when the connection actually happens.

Parameters

`context`	The context of the calling application. Cannot be `Null`.
`listener`	An `SEService:CallBack` object. Can be `Null`.

Errors

`NullPointerError`	If `context` is `Null`.

4.2.2.2 Method: Reader[] getReaders()

Returns the list of available SE readers. There must be no duplicated objects in the returned list. All available readers SHALL be listed even if no card is inserted.

Return value

The readers list, as an array of readers. If there are no readers, the returned array is of length `0`.

Errors

`NullPointerError`	If `context` is `Null`.
`IllegalStateError`	If the `SEService` object is not connected.

4.2.2.3 Method: Boolean isConnected()

Tells whether or not the service is connected.

Return value

`True` if the service is connected.

4.2.2.4 Method: Void shutdown()

Releases all SE resources allocated by this `SEService` (including any binding to an underlying service). As a result `isConnected()` will return `false` after `shutdown()` was called. After this method call, the `SEService` object is not connected.

It is recommended to call this method in the termination method of the calling application (or part of this application) which is bound to this `SEService`.

4.2.2.5 Method: String getVersion()

Returns the version of the Open Mobile API Specification this implementation is based on.

Return value

`String` containing the Open Mobile API version (e.g. "3.3" for Open Mobile API Specification version 3.3).

4.2.3 Class (or Interface): SEService:CallBack

Interface to receive asynchronous notifications when the service is connected.

If the Platform Binding supports it, then this SHOULD be an inner interface of the `SEService` class.

Note: Device/target bindings for the Open Mobile API MAY provide alternative asynchronous notification mappings where the use of callbacks is discouraged or not possible.

4.2.3.1 Method: Void serviceConnected(SEService service)

Called by the framework when the service is connected.

Parameters

`service`	The connected service.

4.2.4 Class (or Interface): Reader:EventCallBack

Interface to receive event when asynchronous event occurs on the specified reader.

If the Platform Binding supports it, then this SHOULD be an inner interface of the Reader class.

Note: Device/target bindings for the Open Mobile API MAY provide alternative asynchronous notification mappings where the use of callbacks is discouraged or not possible.

4.2.4.1 Method: Void notify(Reader:Event event)

Called by the framework when the event occurs.

Parameters

`event`	The event instance generated by the framework.

4.2.5 Class: Reader:Event

An instance of that class is generated by the framework

4.2.5.1 Method: Reader getReader()

Return value

Return the reader on which the event occurred.

4.2.5.2 Method: Int getEventType()

Return value

Return the value corresponding to the type of the event, as shown below.

EventType

`0x1001` for `Reader:IOErrorEventType`	An `IOError` occurred on the reader.
`0x2001` for `Reader:SEInsertedEventType`	An SE has been inserted in the reader.
`0x2002` for `Reader:SERemovalEventType`	An SE has been removed from the reader.

4.2.6 Class: Reader

Instances of this class represent SE readers supported by this device. These readers can be physical devices or virtual devices. They can be removable or not. They can contain one SE that can or cannot be removed.

4.2.6.1 Method: String getName()

Return the name of this reader.

- If this reader is a SIM reader, then its name must be “SIM[Slot]”.
- If the reader is a SD or micro SD reader, then its name must be “SD[slot]”.
- If the reader is an embedded SE reader, then its name must be “eSE[slot]”.

Slot is a decimal number without leading zeros. The numbering must start with 1 (e.g. SIM1, SIM2, … or SD1, SD2, … or eSE1, eSE2, …). The slot number 1 for a reader is optional (SIM and SIM1 are both valid for the first SIM reader, but if there are two readers then the second reader must be named SIM2). This applies also for other SD or SE readers.

Return value

The reader name, as a `String`.

4.2.6.2 Method: SEService getSEService()

Return the SE service this reader is bound to.

Return value

The `SEService` object.

4.2.6.3 Method: Boolean isSecureElementPresent()

Check if a SE is present in this reader.

Return value

`True` if the SE is present, `false` otherwise.

4.2.6.4 Method: Session openSession()

Connects to a SE in this reader.

This method prepares (initializes) the SE for communication before the session object is returned (i.e. powers the SE by ICC ON if it is not already on).

There might be multiple sessions opened at the same time on the same reader. The system ensures the interleaving of APDUs between the respective sessions.

A Reader instance might not communicate with the SE during calls to `openSession()`, depending on the implementation.

Return value

A Session object to be used to create channels. The creation of a Session object SHALL NOT depend upon the availability of the basic or logical channels.

Errors

`IOError`	If something went wrong when communicating with the SE or the reader. **Note:** Implementations which do not communicate with the SE during `openSession()` MAY NOT generate an `IOError`.

4.2.6.5 Method: Void closeSessions()

Close all the sessions opened on this reader. All the channels opened by all these sessions will be closed.

4.2.6.6 Method: Void registerReaderEventCallback(Reader:EventCallBack callback)

Register to receive asynchronous notifications from the `Reader`.

Where the Platform Binding uses callbacks, register the specified callback in the set of `Reader` callbacks.

In general, the order of dispatch of asynchronous notifications is undetermined.

If the asynchronous notification has been already registered the method does nothing.

Parameters

`callback`	The event callback to be notified when an event occurs.

Return value

`Void`

4.2.6.7 Method: Boolean unregisterReaderEventCallback(Reader:EventCallBack callback)

Unregister to receive asynchronous notifications from the Reader.

Parameters

`callback`	The event callback to be unregistered from the callback set.

Return value

`True`	The callback has been previously registered and has been unregistered.
`False`	The callback has not been previously registered.

4.2.7 Class: Session

Instances of this class represent a connection session to one of the SEs available on the device. These objects can be used to get a communication channel with an applet in the SE. This channel can be the basic channel or a logical channel.

4.2.7.1 Method: Reader getReader()

Get the reader that provides this session.

Return value

The `Reader` object.

4.2.7.2 Method: Byte[] getATR()

Get the ATR of this SE.

A `Null` value SHALL be returned if the ATR for this SE is not available.

Return value

The ATR as a byte array or `Null`.

4.2.7.3 Method: Void close()

Close the connection with the SE. This will close any channels opened by this application with this SE.

4.2.7.4 Method: Boolean isClosed()

Tells if this session is closed.

Return value

`True` if the session is closed, `false` otherwise.

4.2.7.5 Method: Void closeChannels()

Close any channels opened on this session.

4.2.7.6 Method: Channel openBasicChannel(Byte[] aid, Byte P2)

Get access to the basic channel, as defined in [ISO 7816-4] (the one that has number `0`). The obtained object is an instance of the channel class.

Once this channel has been opened by a device application, it is considered as 'locked' by this device application, and other calls to this method SHALL return `Null`, until the channel is closed.

Some SE plug-ins, such as those handling UICC, may prevent the use of the Basic Channel. In these cases, a `Null` value SHALL be returned.

When the SE plug-in supports use of the Basic Channel, the method SHALL be implemented as follows:

- If the AID is defined (the AID is not `Null` and the length of the AID is not `0`) and the channel is not locked then the corresponding applet SHALL be selected by sending a SELECT command as defined below with the requested AID.
- If the AID is a `0` length AID and the channel is not locked, the method will select the Issuer Security Domain of the SE by sending a SELECT command with a `0` length AID as defined in the GlobalPlatform Card Specification [GPCS].
- If the AID is `Null` and the channel is not locked, then there are four possibilities:
  - No applet has previously been selected (either explicitly by its AID or by selecting the Issuer Security Domain as described above) on this channel since the SE was powered on. The implicit default applet is therefore selected on the SE. In this case, the method SHALL NOT send a SELECT command to the SE and it SHALL lock the channel for the calling application.
  - An applet has previously been selected on this channel but a `channel.close()` (as defined in section 4.2.8.1) has been executed successfully using a MANAGE CHANNEL Reset (P1 = `0x40`) command (i.e. if this command is supported by the SE). The implicit default applet is therefore already selected on the SE. In this case, the method SHALL NOT send a SELECT command to the SE and it SHALL lock the channel for the calling application.
  - An applet has previously been selected on this channel but a `channel.close()` (as defined in section 4.2.8.1) has been executed successfully using a SELECT command with 0 length AID (i.e. because the MANAGE CHANNEL Reset command is not supported by the SE). The Issuer Security Domain is therefore implicitly selected and it is not possible to reselect the implicit default applet except by resetting the SE[1]. The method SHALL therefore return `Null`.
  - If an applet was selected on this channel and no `channel.close()` has been executed successfully, it is not possible to reselect the implicit default applet except by resetting the SE, so the method does nothing and returns `Null`.

Note: Some types of SE plug-in, such as UICC, may restrict the use of the Basic Channel at lower layers. In these cases a `Null` value SHALL be returned.

Note: Application developers using the Open Mobile API are advised not to rely on implicit selection due to potentially different behavior depending on SE configuration (as defined above). Explicit selection by AID avoids such issues.

P2 is normally `0x00`. The device SHOULD allow any value for P2 and SHALL allow the following values: `0x00`, `0x04`, `0x08`, `0x0C` (as defined in [ISO 7816-4]).

[1] [ISO 7816-4] allows the implicit default applet to be selected in the case where the SE is configured with the ISD as the implicit default applet. However, the application using OMAPI cannot determine which applet is selected by default; without this restriction, behavior seen by applications using OMAPI would depend on SE configuration.

The implementation of the underlying SELECT command within this method SHALL be based on [ISO 7816-4] with the following options:

CLA = 0x00

INS = 0xA4

P1 = 0x04 (Select by DF name/application identifier)

The select response data can be retrieved with `getSelectResponse()`.

For the SELECT command, the API handles the received status word as defined in section 4.1.1, except that for T=0 protocol, if an SW warning is received as the first response to the SELECT command, then the GET RESPONSE with Le = 0x00 SHALL also be sent. If the status word for the SELECT command indicates that the SE was able to open a channel (status word {0x90, 0x00} or status words referencing a warning {0x62, 0xXX} or {0x63, 0xXX}), the API SHALL keep the channel open and the next `getSelectResponse()` SHALL return the status word for the SELECT command together with data, if any data is received.

Other status words for the SELECT command which indicate that the SE was not able to open a channel indicate error conditions and the corresponding channel SHALL NOT be opened.

When a basic channel is opened the value of the attribute `expectDataWithWarningSW` SHALL be `false`.

The function without P2 as parameter is provided for backwards compatibility and will fall back to a select command with P2 = 0x00.

Parameters

`aid`	The AID of the applet to be selected on this channel, as a byte array, or `Null` if no applet is to be selected.
P2	The P2 parameter of the SELECT APDU executed on this channel.

Return value

An instance of channel if available or `Null`.

Errors

`IllegalParameterError`	If the AID length is not zero and is not within the range (5 to 16).
`IllegalStateError`	If an attempt is made to use an SE session that has been closed.
`SecurityError`	If the calling application cannot be granted access to this AID or the default applet on this session.
`NoSuchElementError`	If the AID on the SE is not available or cannot be selected.
`OperationNotSupportedError`	If the given P2 parameter is not supported by the device.
`IOError`	If there is a communication problem to the reader or the SE.

4.2.7.7 Method: Channel openBasicChannel(Byte[] aid)

This method is provided to ease the development of mobile applications and for backward compatibility with existing applications. This method is equivalent to `openBasicChannel(aid, P2=0x00)`.

4.2.7.8 Method: Channel openLogicalChannel(Byte[] aid, Byte P2)

Open a logical channel with the SE, selecting the applet represented by the given AID (when the AID is not `Null` and the length of the AID is not `0`).

This method SHALL first issue a MANAGE CHANNEL Open command followed, optionally, by a SELECT command.

If the length of the AID is `0`, the method will select the Issuer Security Domain of the SE by sending a SELECT command with `0` length AID as defined in [GPCS].

If the AID is `Null`, the method SHALL only send a MANAGE CHANNEL Open and SHALL NOT send a SELECT command. In this case, the default applet associated to the logical channel will be selected by default.

The MANAGE CHANNEL Open command SHALL be issued on basic channel (CLA encoding zero as channel number).

P2 is normally 0x00. The device SHOULD allow any value for P2 and SHALL allow values: `0x00`, `0x04`, `0x08`, `0x0C` (as defined in [ISO 7816-4]).

In the case of a UICC, calls to `openLogicalChannel()` without a specific AID SHALL be rejected by returning `Null`. This requirement does not apply to other types of Secure Element.

The implementation of the underlying SELECT command within this method SHALL be based on [ISO 7816-4] with the following options:

CLA = 0x01 to 0x03, 0x40 to 0x4F

INS = 0xA4

P1 = 0x04 (Select by DF name/application identifier)

The select response data can be retrieved with `Byte[] getSelectResponse()`.

For the SELECT command, the API SHALL handle received status word as defined in section 4.1.1, except that for T=0 protocol the GET RESPONSE with Le = 0x00 SHALL also be sent in case of SW warning received as first response of the SELECT command. If the status word for the SELECT command indicates that the SE was able to open a channel (status word {0x90, 0x00} or status words referencing a warning `{0x62, 0xXX}` or `{0x63, 0xXX}`), the API SHALL keep the channel open and the next `getSelectResponse()` SHALL return the status word for the SELECT command together with data, if any data is received.

Other status words for the SELECT command which indicate that the SE was not able to open a channel indicate error conditions and the corresponding channel SHALL NOT be opened.

When a logical channel is opened, the value of the attribute `expectDataWithWarningSW` SHALL be `false`.

The function without P2 as parameter is provided for backwards compatibility and will fall back to a select command with P2 = 0x00.

Parameters

`aid`	The AID of the applet to be selected on this channel, as a byte array.
P2	The P2 parameter of the SELECT APDU executed on this channel.

Return value

An instance of channel. `Null` if the SE is unable to provide a new logical channel or is unable to retrieve Access Control rules due to the lack of an available logical channel.

Errors

`IllegalParameterError`	If the length of `aid` is not zero and is not within 5 to 16 (inclusive).
`IllegalStateError`	If the SE is used after being closed.
`SecurityError`	If the calling application cannot be granted access to this AID or the default applet on this session. **Note:** Failure in retrieving rules due to the lack of an available logical channel (and only this failure) SHALL result in a `Null` return value and not a security exception. This is in line with [GPCS], as the access to the SE applet will be denied anyway.
`NoSuchElementError`	If the AID on the SE is not available or cannot be selected or a logical channel is already open to a non-multiselectable applet.
`OperationNotSupportedError`	If the given P2 parameter is not supported by the device.
`IOError`	If there is a communication problem to the reader or the SE.

4.2.7.9 Method: Channel openLogicalChannel(Byte[] aid)

This method is provided to ease the development of mobile applications and for backward compatibility with existing applications. This method is equivalent to `openLogicalChannel(aid, P2=0x00)`.

4.2.8 Class: Channel

Instances of this class represent an [ISO 7816-4] channel opened to a SE. It can be either a logical channel or the basic channel.

They can be used to send APDUs to the SE. Channels are opened by calling the `Session.openBasicChannel()` or `Session.openLogicalChannel()`methods.

4.2.8.1 Method: Void close()

Closes this channel to the SE. If the method is called when the channel is already closed, this method SHALL be ignored.

The `close()` method SHALL wait for completion of any pending `transmit(byte[] command)` before closing the channel.

If the channel is not the basic channel, then a MANAGE CHANNEL Close (P1 = `0x80`) command SHALL be sent to the Secure Element for closing the channel.

If the channel is the basic channel, then the following procedure must be followed for closing the channel:

1. Send a MANAGE CHANNEL Reset (P1 = 0x40) command, as defined by [ISO 7816-4].
2. If previous command is not supported by the card (i.e. card returns an error SW), then SELECT by DF Name (P1 = `0x04`; P2 = `0x00`) command SHALL be sent with an empty AID (Lc = `0x00`). All data returned for the SELECT by DF Name command SHALL be discarded.

In all cases, the last SW of the MANAGE CHANNEL Close; MANAGE CHANNEL Reset or SELECT by DF Name command SHALL be ignored and the channel object SHALL be closed by the API.

4.2.8.2 Method: Boolean isBasicChannel()

Returns a Boolean indicating whether this channel is the basic channel.

Return value

`True` if this channel is a basic channel.

`False` if this channel is a logical channel.

4.2.8.3 Method: Boolean isClosed()

Tells if this channel is closed.

Return value

`True` if the channel is closed, `false` otherwise.

4.2.8.4 Method: Byte[] getSelectResponse()

Returns the data as received from the application select command, including the status word received at applet selection.

The returned byte array contains the data bytes in the following order:

```
[<first data byte>, …, <last data byte>, <sw1>, <sw2>]
```

Return value

- The data as returned by the application select command inclusive of the status word.
- Only the status word if the application select command has no returned data.
- `Null` if an application select command has not been performed or the selection response cannot be retrieved by the reader implementation.

4.2.8.5 Method: Session getSession()

Get the session that has opened this channel.

Return value

The `Session` object this channel is bound to.

4.2.8.6 Method: Void setTransmitBehaviour(Boolean expectDataWithWarningSW)

Sets the expected behavior of the `transmit()` method for handling the first received warning SW ({0x62, 0xXX} or {0x63, 0xXX}). The method applies only for T=0 protocol and only when transmitting case 4 APDU and only for the first received SW warning ({0x62, 0xXX} or {0x63, 0xXX}). It applies only for this channel object. When opening a new channel object the default value of the `expectDataWithWarningSW` attribute SHALL be `false`.

Note: Developers of Card Applets intended to be used from the Open Mobile API SHOULD avoid implementation of warning behavior which requires the use of `expectDataWithWarningSW` to avoid possible confusion on the part of device application developers on the operation of this special-case handling for case 4 APDU.

Some Platform Bindings might not implement `setTransmitBehaviour()`. When a Platform Binding does not implement this API, the Platform Binding SHALL behave as though the `expectDataWithWarningSW` attribute is `false`.

Parameter

`expectDataWithWarningSW`	• When set to `false`: The underlying implementation of the `transmit()` method SHALL issue a GET RESPONSE command only after receiving a {0x61, 0xXX} SW (as defined in section 4.1.1). • When set to `true`: o If the SE returns data and warning SW ({0x62, 0xXX} or {0x63, 0xXX}), the returned data and the warning SW SHALL be provided to the calling Mobile Application and GET RESPONSE SHALL NOT be sent. o If the SE returns a warning SW ({0x62, 0xXX} or {0x63, 0xXX}) without data, the API SHALL issue a GET RESPONSE with Le = 0x00. After sending the GET RESPONSE with Le = 0x00, general rules specified in section 4.1.1 SHALL apply for retrieving the data. If all the available data was successfully retrieved – i.e. the last status word is {0x90, 0x00} or a warning status word ({0x63, 0xXX}, {0x63, 0xXX}) – then the API SHALL return the data together with the first received warning status word (i.e. the warning status word received for the command APDU sent in the `transmit()` method). In consequence the last success or warning status word SHALL be ignored. **Note:** If the SE has no data to provide back after sending the warning SW, the GET RESPONSE will trigger an error on the SE. So, when setting the value to `true`, it is strongly recommended to check that the SE applet will have data to provide back to the calling application in case of any warning status word returned by the applet.

4.2.8.7 Method: Byte[] transmit(Byte[] command)

Transmit an APDU command (as per ISO/IEC 7816) to the SE. The underlying layers generate as many TPDUs as necessary to transport this APDU. The transport part is invisible from the application. The generated response is the response of the APDU which means that all protocol related responses are handled inside the API or the underlying implementation.

The system ensures the synchronization between all the concurrent calls to this method, and that only one APDU will be sent at a time, irrespective of the number of TPDUs that might be required to transport it to the SE. The entire APDU communication to this SE is locked during the execution of the `transmit()` method.

The API SHALL ensure that all available data returned from the SE, including concatenated responses, are retrieved and made available to the calling application, as defined by ISO 7816. The OMAPI implementation, or underlying layer, SHALL handle the received status words as defined in section 4.1.1 and according to the attribute `expectDataWithWarningSW` – see section 4.2.8.6. The channel SHALL NOT be closed even if the SE answers with an error status word.

The API (or underlying implementation) will manage all the required information to ensure the APDU is transmitted on the appropriate channel:

- the channel information in the class byte of the APDU provided by the mobile application will be ignored.
- class byte will be updated with the correct channel information.

The API implementation SHALL be able to send all valid APDU commands as defined by ISO 7816. The only restrictions on the set of commands that can be sent by the `transmit()` method are:

- MANAGE_CHANNEL commands are not allowed.
- SELECT by DF Name (P1 = `0x04`) commands are not allowed.
- Invalid APDU commands according to ISO 7816 are not allowed.

The `transmit()` method SHALL support extended length APDU commands with a length of at least 2048 bytes.

Parameters

`command`	The APDU command to be transmitted, as a byte array.

Return value

The response received, as a byte array. The returned byte array contains the data bytes in the following order: `{<first data byte>, …, <last data byte>, <sw1>, <sw2>}`.

Errors

`NullPointerError`	If command is `Null`.
`IllegalParameterError`	• The command byte array is less than 4 bytes long. • Lc byte is inconsistent with the length of the byte array. • CLA byte is invalid according to [ISO 7816-4] (0xFF). • INS byte is invalid according to [ISO 7816-4] (0x6X or 0x9X).
`IllegalStateError`	If an attempt is made to use an SE session or channel that has been closed.
`SecurityError`	If the command is filtered by the security policy.
`IOError`	If there is a communication problem to the reader or the SE.

4.2.8.8 Method: Boolean selectNext()

Performs a selection of the next applet on this channel that matches to the partial AID specified in the `openBasicChannel()` or `openLogicalChannel()` method. This mechanism can be used by a device application to iterate through all applets matching to the same partial AID.

If `selectNext()` returns `true`, a new applet was successfully selected on this channel. If no further applet exists with matches to the partial AID, this method returns `false` and the applet already selected stays selected.

Since the API cannot distinguish between a partial and full AID, the API SHALL rely on the response of the SE for the return value of this method.

The implementation of the underlying SELECT command within this method SHALL use the same values as the corresponding `openBasicChannel()` or `openLogicalChannel()` command with the option:

P2 = 0x02 (Next occurrence)

For the SELECT command, the API SHALL handle received status word as defined in section 4.1.1, except that for T=0 protocol the GET RESPONSE SHALL also be sent in case of SW warning. If the status word for the SELECT command is `{0x90, 0x00}`, `{0x62, 0xXX}`, or `{0x63, 0xXX}`, the API SHALL keep the channel open and the next `getSelectResponse()` SHALL return the status word for the SELECT command together with data, if any data is received.

The select response stored in the channel object, SHALL be updated with the APDU response of the SELECT command.

Return value

`True` if a new applet was selected on this channel.

`False` if the applet already selected stays selected on this channel.

Errors

`IllegalStateError`	If an attempt is made to use an SE session or channel that has been closed.
`OperationNotSupportedError`	If this operation is not supported by the card.
`IOError`	If there is a communication problem to the reader or the SE.

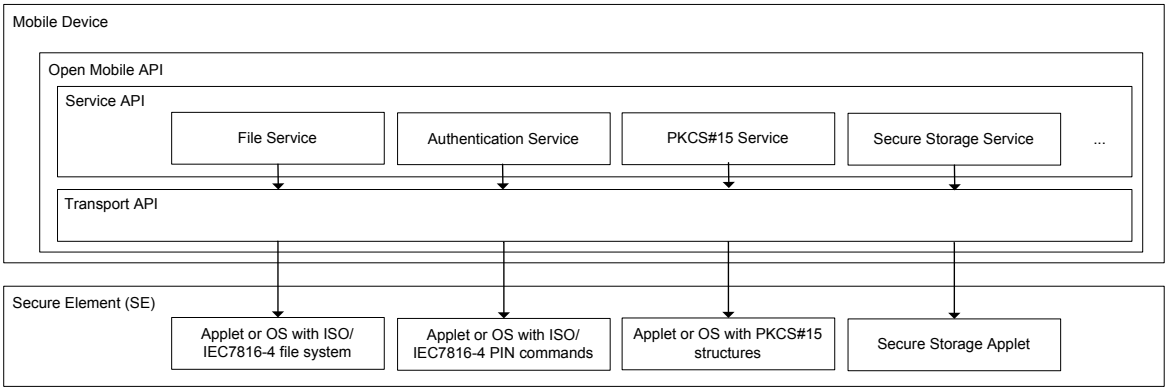
5 Service Layer APIs (Deprecated)

The Service APIs as part of the Open Mobile API provide a framework to access SEs available in the mobile device with high level interfaces. The Service APIs are deprecated, and will be removed from a future version of the Specification.

5.1 Overview

The Open Mobile API contains a Service API on top of the Transport API for providing high level API methods for different purposes. The Service API relies on the Transport API to establish communication with the applets within the SE. The Service API consists of different APIs specialized for different purposes. Normally each specialized API requires a counterpart on the SE side (e.g. an SE applet providing a defined set of APDUs).

Figure 5-1: Service API Overview



Note:

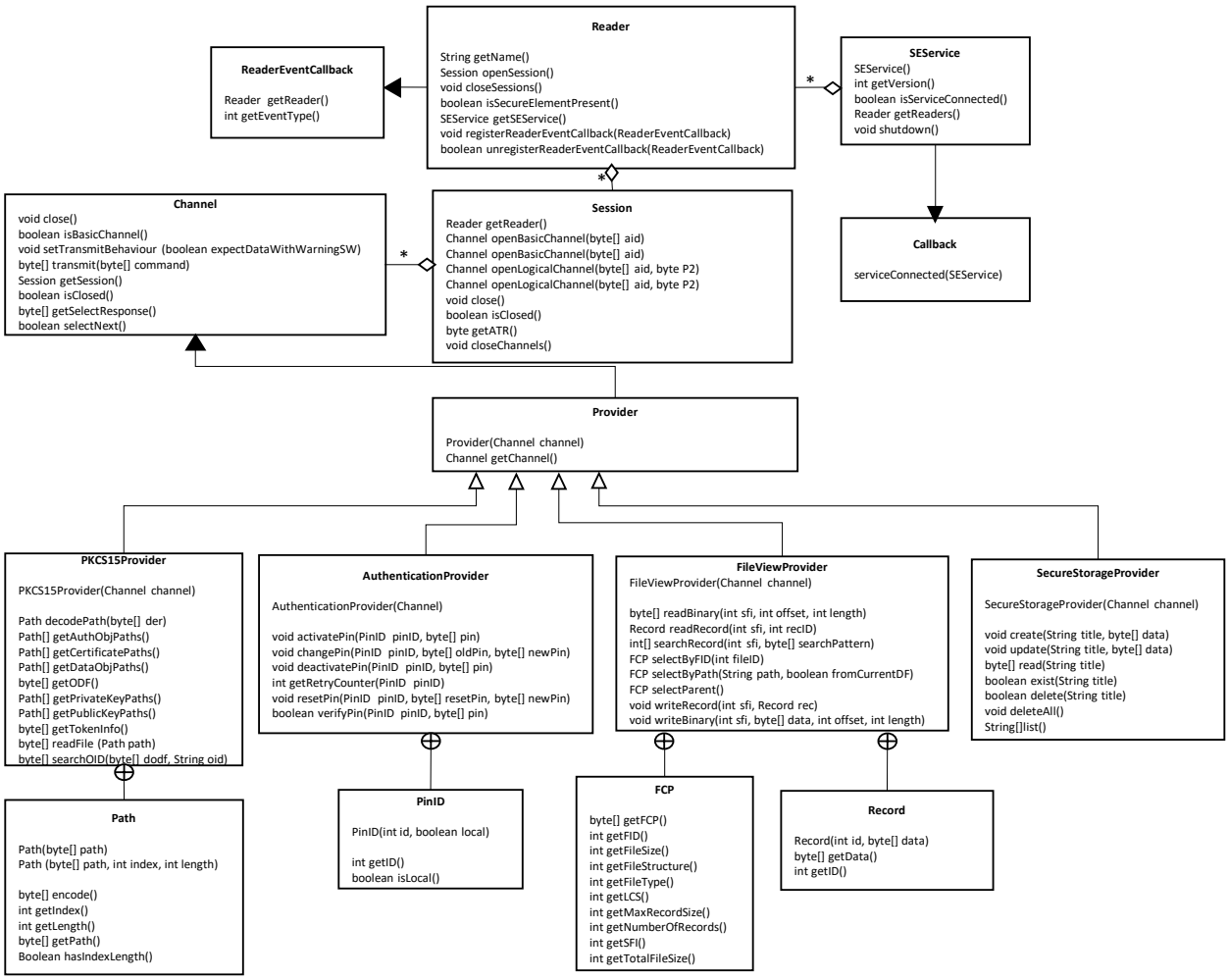
The use of the Service Layer API requires SE access rights (see section 8 for more information on SE access rights).

Since all operations within the Service API layer are based on the Transport API, all error conditions of the corresponding transport classes can be thrown in the service layer although they are not explicitly named. For example, a call to `AuthenticationProvider::verifyPin()` can cause an IOError because the implementation uses the `Channel.transmit()` API call internally.

5.2 Class Diagram

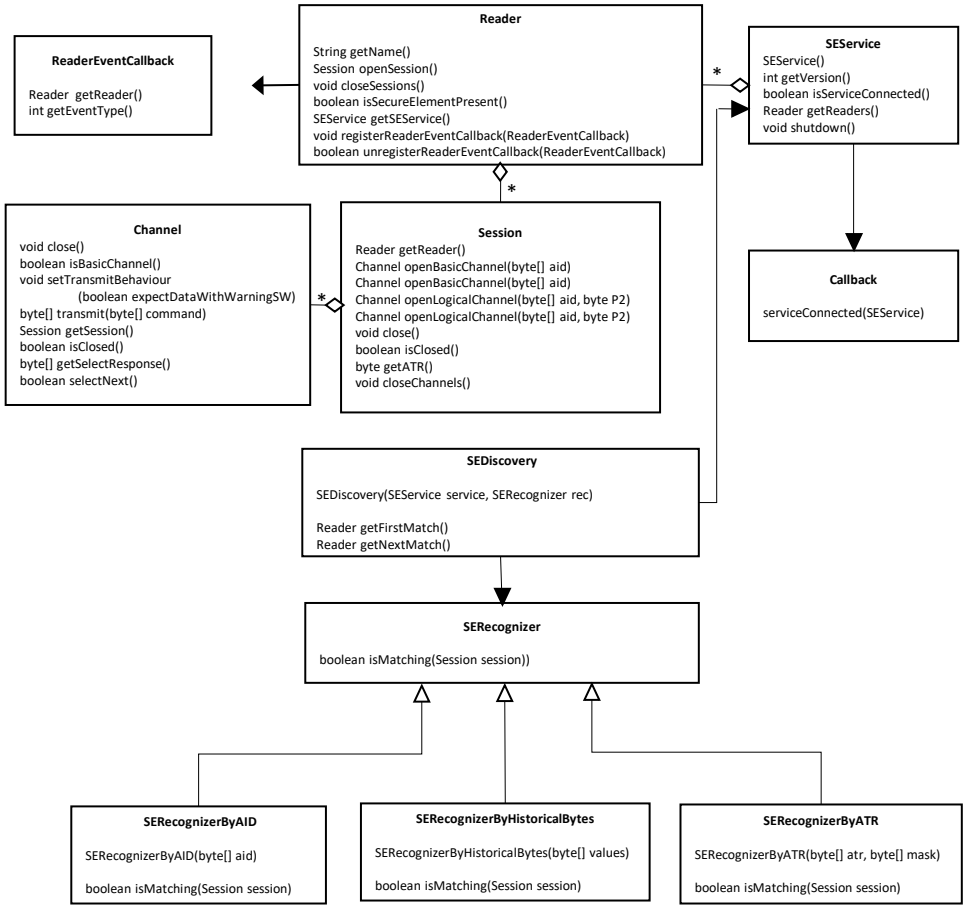
This class diagram contains the Service API besides the Transport API. The Service API consists of a set of classes derived from the base class Provider. The Crypto API is not part of this diagram, but it is also a component of the Service API. The Crypto API relies on existing APIs in the REE and realizes SE communication via the Transport API.

Figure 5-2: Service API Class Diagram with Provider Classes



Besides these Provider classes, the Service API contains a SEDiscovery class which provides a discovery mechanism that can be used for SE selection by defined criteria.

Figure 5-3: Service API Class Diagram with SEDiscovery Classes



5.3 Usage Pattern

The usage pattern of the Service API is as follows:

1. The application gets access to the SE service(s):

   It creates a SEService class, passing an object implementing the SEService.Callback interface, whose `serviceConnected()` method is called asynchronously when the connection is established. This does not represent a connection with the SE itself, but with the subsystem implementing the SE access functionality.

2. The application enumerates the available readers.

   Readers are the slots where SEs are connected (in a removable or non-removable manner).

   Once the user or an application-specific algorithm has selected a reader, then the application opens a session on this reader. The right reader can also be chosen by using the Discovery API. The Discovery API allows different criteria to be defined (ATR, AID, …) for finding an appropriate SE and applet in an SE. Finally, the Discovery API allows iterating through the readers containing an SE with the defined criteria.

3. With this session, the application can retrieve the ATR of the SE, and if it matches with one of the known ATRs, it can start opening channels with applets in the SE.

4. To open a channel, the application will use the AID of the applet or use the default applet on the newly opened channel. The application is in charge of selecting an applet which fits the specific Provider class that will be chosen in the next step for SE operations.

5. The application creates an instance of a certain Provider class depending on the application's intention. If the intention is to read files from the SE's file system, the FileViewProvider has to be chosen. The application has to consign the communication channel (established in the step before) to the Provider before the Provider instance can be used for SE operations.

6. Then the terminal application can start performing operations on the selected applet in the SE with the help of the provider's methods. If a FileViewProvider instance was created for reading files from the SE's file system, the application can use the FileViewProvider's methods `readBinary()` or `readRecord()`.

7. The terminal application can also use several Provider instances alternately on the same channel or it can use the `transmit()` method of the Transport Layer to send any APDUs directly to the SE on that channel. This option is especially useful if the application needs to perform different operations on the same channel. This could happen for example if the application needs to read a file from the SE's file system that requires a successful PIN verification on the same channel as before. In this case, the terminal application can instantiate an Authentication class which provides the `verifyPin()` method. After the successful PIN verification via the AuthenticationProvider, the FileViewProvider can be used to read the file from the SE's file system. Since the SE usually manages the PIN verification individually for each logical channel, it is important to apply the AuthenticationProvider and FileViewProvider on the same channel.

8. Once done, the application can close any existing channels, or even sessions, and its connection to the SEService.

5.4 Service API Framework

The Open Mobile API provides a set of service layer classes with high level methods for SE operations.

5.4.1 Class: Provider

This Provider class (realized as interface or abstract class) is the base class for all service layer classes. Each service layer class provides a set of methods for a certain aspect (file management, PIN authentication, [PKCS #15] structure handling, etc.) and acts as a provider for service routines. All Provider classes need an open channel for communication with the SE. As such, before a certain Provider class can be used for SE operations, the channel has to be consigned. For performing different operations (PIN authentication, file operation, etc.) the Provider classes can be easily combined by using the same channel for different Provider classes and alternately calling methods of these different providers. It has to be considered that each Provider class needs a counterpart on the SE side (e.g. an applet with a standardized APDU interface as required by the Provider class). The application using a Provider class for SE interactions is in charge of assigning a channel to the Provider where the Provider's SE counterpart applet is already preselected.

5.4.1.1 Constructor: Provider(Channel channel)

Encapsulates the defined channel by a Provider object that can be used for performing a service operation on it. This constructor has to be called by derived Provider classes during the instantiation.

Parameters

`channel`	The channel to be used by this Provider for service operations.

Errors

`IllegalStateError`	If an attempt is made to use an SE session that has been closed.

5.4.1.2 Method: Channel getChannel()

Returns the channel that is used by this provider.

This returned channel can also be used by other providers.

Return value

The channel instance that is used by this provider.

5.5 Crypto API

The Crypto API that is native to the mobile operating system SHALL be used.

For SE vendors, it guarantees that the crypto functionality offered by the SEs will be seen at the same level as others.

Application developers should be careful to choose the crypto functionality provided by the SE when enumerating the available crypto providers.

For example, in a Java environment, it is recommended to use the Java Crypto Extension as the Java binding of the Crypto API. More information on the JCE can be found in [Java Crypto].

[Java Crypto] serves two purposes:

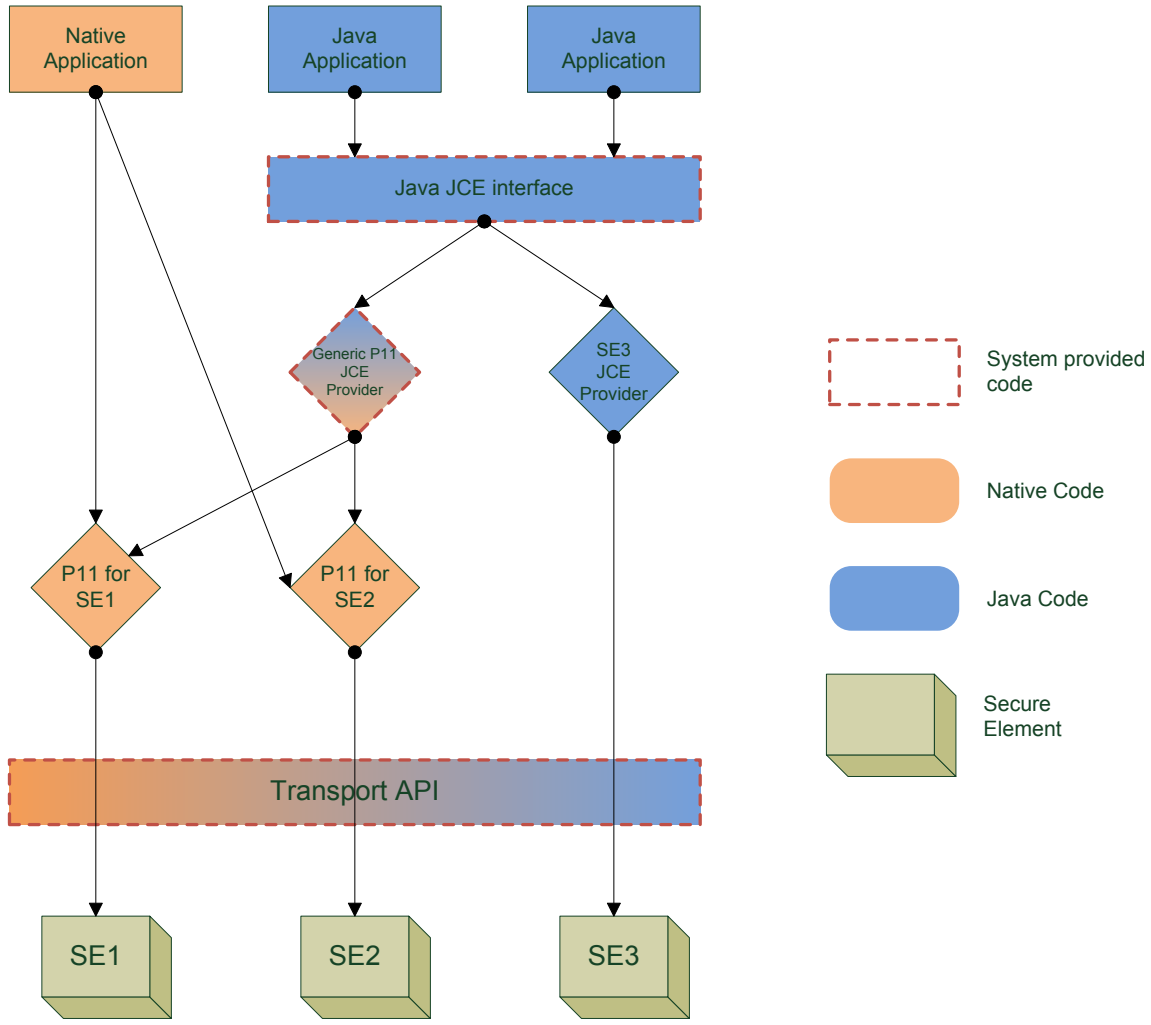
- It contains information about the API side of the Java Cryptography Architecture, that may be used by developers to learn how to use this API.
- It contains information about the SPI (Service Provider Interface) side of the Java Cryptography Architecture, i.e. how to add new cryptography capabilities to a Java platform, by implementing a Provider, and how to declare and use the new providers.

In a native environment, where C/C++ is used as the programming language, it is recommended to use PKCS #11 as the native binding of the Crypto API. More information on PKCS #11 can be found in the PKCS #11 standard [PKCS #11].

In a mixed environment where Java and C/C++ are cohabiting, the PKCS #11 implementations should be made available to the Java applications. This is typically done by defining a JCE Provider for PKCS #11, that behaves as a Java binding for PKCS #11.

The following is an example of such a mixed architecture.

Figure 5-4: Crypto API Architecture



5.5.1 Extensibility

A requirement of this API is to provide the functionality to add system wide crypto providers during runtime (without flashing the device) to support the different card implementations in the field.

For example, in a Java environment, the JCE provider architecture should be used to declare new providers (by SE vendors) and to lookup for JCE providers (by application developers).

In a native environment, a mechanism to declare new PKCS #11 implementation (typically as shared libraries) must be available (to be used by PKCS #11 implementers), and reciprocally a mechanism to choose between the available PKCS #11 libraries must be available (to be used by application developers).

5.5.2 Extending by Shared Libraries

On systems that support the use of shared libraries, this mechanism can be used to provide new implementations of crypto providers. For example, PKCS #11 implementations can be provided as shared libraries. The mechanism to register and discover these shared libraries is yet to be defined (it is not part of PKCS #11).

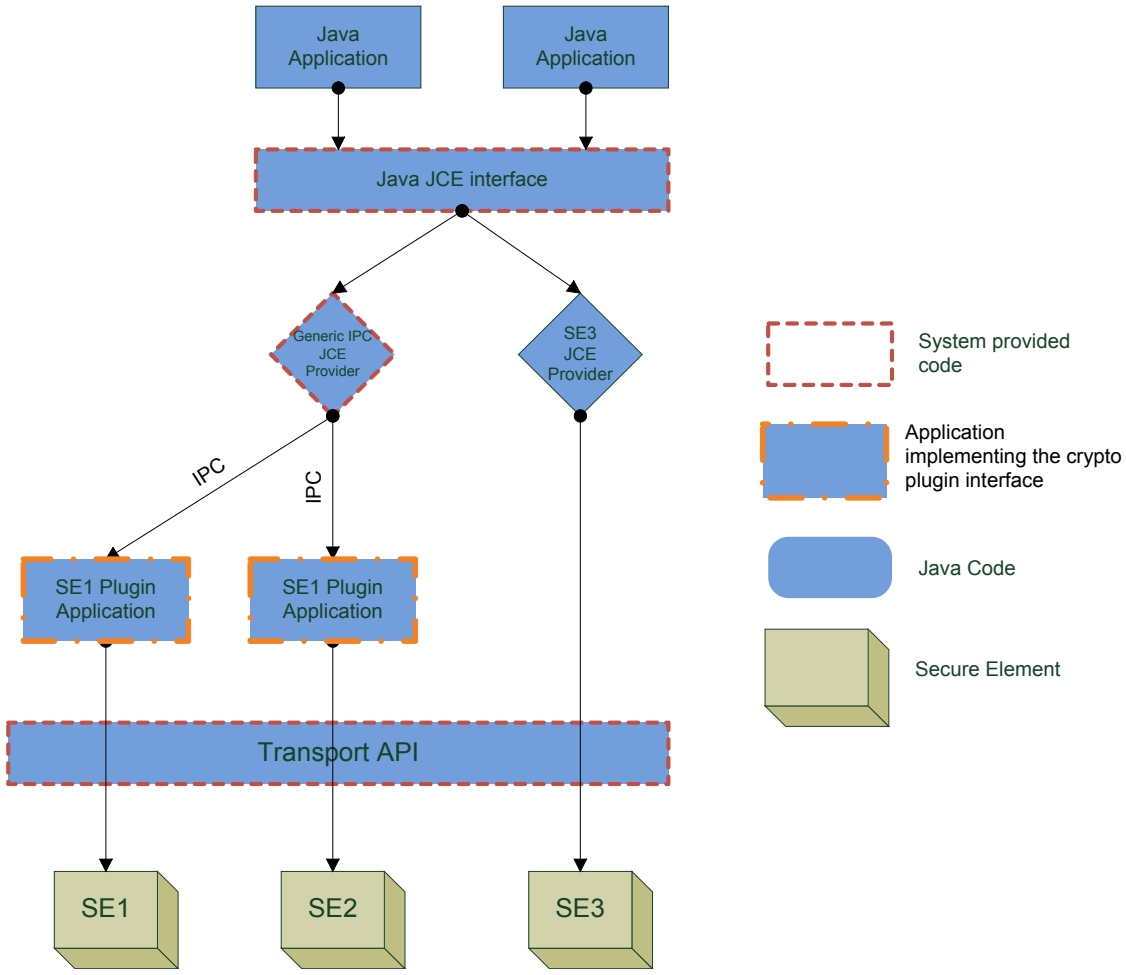
5.5.3 Extending Functionality via Applications

On systems where extensibility can only be achieved by installing new applications, and which provide a way to perform inter-application communication (e.g. IPC), a plug-in mechanism can be used.

A generic provider based on the IPC capabilities can be offered by the system, to be used by the applications. The role of this generic provider is to lookup for plug-in applications implementing a specific crypto service interface, list them to the crypto-aware applications (enumerated as regular crypto providers), and when a crypto-aware application selects a particular provider, establish the connection to the plug-in application actually implementing the crypto operations and forward all the requests thanks to the IPC mechanism.

The following is an illustration of such an implementation on a Java-based environment, using a JCE provider (the same could apply to a native-based environment, using a PKCS #11 library):

Figure 5-5: Crypto API Architecture with Plug-in Applications



With this architecture, when support for a new type of SE providing crypto services has to be added to the system, it is just a matter of installing a new plug-in application implementing the protocol to interact properly with this new type of crypto provider.

5.5.4 Integration with the Transport API

The Crypto API(s) should be implemented on top of the Transport API; if it is not, the system should behave as if it was. For example, if the Crypto API requires access to the basic channel of a SE, then it is subject to the same rules as the other applications: access to the basic channel is provided only if the application calling the Crypto API is authorized to access the basic channel, and if it is currently not locked. In the same way, if the basic channel is locked by the Crypto API, it is not available to other applications.

If the PKCS #11 libraries are implemented on top of a PC/SC layer or an equivalent layer that gives a reader-access level, then it is recommended that APDUs sent over this layer are filtered and processed to be translated into commands for the Transport API.

For example, if a native application sends a MANAGE_CHANNEL command and then a SELECT_BY_DF_NAME command, this should be translated into a call to `openLogicalChannel()`.

5.6 Discovery API

This API provides means for the applications to lookup a SE, based on a search criterion. The rationale behind such an API is to factorize this lookup code in a system-provided API, reducing the development cost of this part of each application.

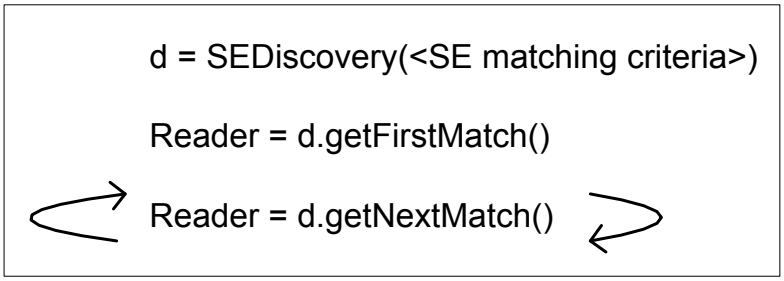
This API relies on an object-oriented approach: the lookup method is shared, but the criterion is implemented as a separate class that can be derived. A set of basic criterion class is provided, they include:

- Search by ATR
- Search by historical bytes
- Search by AID

If these basic research criteria are not sufficient to fulfil a specific application's needs, then the application can provide its own criterion object. This object will have its `isMatching()` method called for each SE present in the system, and will be able to use the Transport API (or any other Service API) to implement a specific matching algorithm. Examples of such specific algorithm are:

- Analysis of EFdir file (if available)
- Analysis of GlobalPlatform Status information (if available)

Figure 5-6: Discovery Mechanism



5.6.1 Class: SEDiscovery

Instances of this class must be created by the applications to start a discovery process.

When created, they are configured with a SEService and an object that will perform the recognition algorithm.

5.6.1.1 Constructor: SEDiscovery (SEService service, SERecognizer recognizer)

Creates a discovery object that will perform a discovery algorithm specified by the recognizer object, and will be applied to the given SEService.

Parameters

`service`	The `SEService` used to perform the discovery. Cannot be `Null`.
`recognizer`	An `SERecognizer` instance, whose `isMatching()` will be called. Cannot be `Null`.

Errors

`IllegalParameterError`	If one of the parameters is `Null`.

5.6.1.2 Method: Reader getFirstMatch()

Returns the first SE reader containing a SE that matches the search criterion.

Actually starts a full discovery process:

- SE readers are enumerated.
- For the first reader, if a SE is present, open a session.
- On this session, call the `isMatching()` method of the SERecognizer object given at construction time.
- Close the session.
- If the `isMatching()` method returns `false`, the process is continued with the next reader.
- If the `isMatching()` method returns `true`, the reader object is returned.

The sessions used by the discovery process are closed to avoid the risk of leaks: if they were opened and returned to the caller, there would be a risk that the caller would forget to close them.

Calling `getFirstMatch()` twice simply restarts the discovery process (e.g. probably returns the same result, unless a SE has been removed).

Return Value

The first matching SE reader, or `Null` if there is none.

Errors

NONE. All errors must be caught within the implementation.

5.6.1.3 Method: Reader getNextMatch()

Returns the next SE reader containing a SE that matches the search criterion.

Actually continues the discovery process:

- For the next reader in the enumeration, if a SE is present, open a session.
- On this session, call the `isMatching()` method of the SERecognizer object given at construction time.
- The session is closed.
- If the `isMatching()` method returns `false`, the process is continued with the next reader.
- If the `isMatching()` method returns `true`, the reader object is returned.

Return Value

The next matching SE reader, or `Null` if there is none.

Errors

`IllegalStateError`	If the `getNextMatch()` method is called without having previously called `getFirstMatch()` since the creation of the SEDiscovery object or since the last call to `getFirstMatch()` or `getNextMatch()` that returned `Null`.

5.6.2 Class: SERecognizer

Base class for recognizer classes.

Extended by system-provided recognizers, or by custom recognizers.

5.6.2.1 Method: Boolean isMatching(Session session)

This method will be called during the discovery process, once for each SE that is present. Application developers can use the given session object to perform any discovery algorithm they think is appropriate. They can use the Transport API or any other API, conforming to access control rules and policy, as they would with regular application code (i.e. this is not privileged code).

Parameters

`session`	A session object that is used to perform the discovery. Never `Null`.

Return value

A Boolean indicating whether the SE to which the given session has been open is matching with the recognition criterion implemented by this method.

Errors

NONE. All errors must be caught within the implementation of the method, and must be indicated by returning `false`.

5.6.3 Class: SERecognizerByATR

Instances of this class can be used to find a SE with a specific ATR (or ATR pattern).

5.6.3.1 Constructor: SERecognizerByATR (byte[] atr, byte[] mask)

Parameters

`atr`	Byte array containing the ATR bytes values that are searched for.
`mask`	Byte array containing an AND-mask to be applied to the SE ATR values before comparing them with the parameter `atr`.

Errors

`IllegalParameterError`	If ATR is invalid.

5.6.4 Class: SERecognizerByHistoricalBytes

Instances of this class can be used to find a SE with a specific value in their historical bytes.

5.6.4.1 Constructor: SERecognizerByHistoricalBytes (byte[] values)

Parameters

`values`	Byte array, to be checked for presence in the historical bytes.

Errors

`IllegalParameterError`	If historical bytes are invalid.

5.6.5 Class: SERecognizerByAID

Instances of this class can be used to find a SE implementing a specific applet, identified by its AID. The presence of such an applet is verified by trying to open a channel to this applet. The opened channel, if any, is closed before the end of the `isMatching()` method.

5.6.5.1 Constructor: SERecognizerByAID (byte[] aid)

Parameters

`aid`	Byte array holding the AID to be checked for presence.

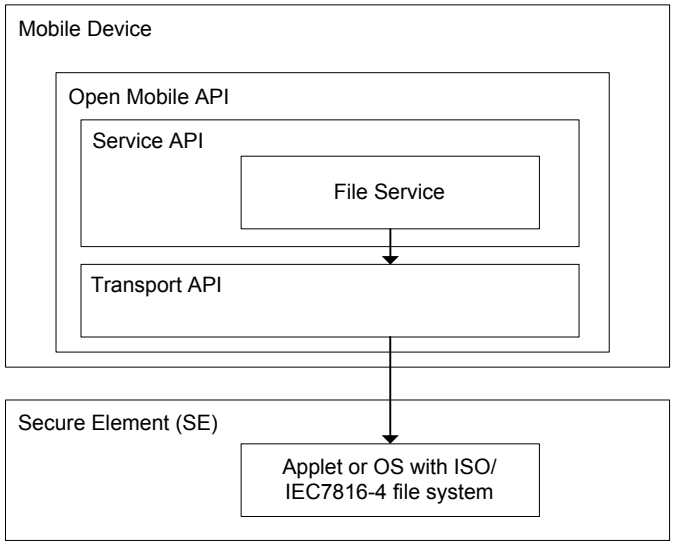
Errors

`IllegalParameterError`	If AID is invalid.

5.7 File Management

API for file management to read and write the content of files in an [ISO 7816-4] compliant file system provided by the SE's REE or applet (installed in the SE).

Figure 5-7: File Management Overview



5.7.1 Class: FileViewProvider

This Provider class simplifies file operations on SEs with a file structure specified in [ISO 7816-4].

Methods are provided that allow file content to be read or written. If the read or write operation is not allowed because security conditions are not satisfied, a SecurityError will be returned. It must be considered that a file operation can only be applied onto a file which has a corresponding structure.

Prerequisites

This Provider requires an [ISO 7816-4] compliant file system on the SE. If this file system is implemented by an applet within the SE then this applet must be preselected before this Provider can be used, in case that the applet is not already selected (e.g. the GSM Applet as a default selected applet on a UICC).

Notes:

- If used by multiple threads, synchronization is up to the application.
- Each operation needs access to the SE. If access cannot be granted because of a closed channel or a missing security condition, the called method will return an error.
- Using the basic channel for accessing the file system of the UICC (provided by the default selected GSM Applet) implies the risk of interferences from the baseband controller, as the baseband controller works internally on the basic channel and can modify the current file selected state on the basic channel anytime. This means a file selection performed by this FileViewProvider does not guarantee a permanent file selection state on the UICC's basic channel and care has to be taken to ensure the application using the FileViewProvider has the needed file selection state. The FileViewProvider itself cannot avoid interferences from the baseband controller on the basic channel, but the risk could be minimized if the application using the FileViewProvider performs implicit selections for the file operation or performs the file selection immediately before the file operation.

5.7.1.1 Constant: CURRENT_FILE

Indicates for file operation methods that the currently selected file SHALL be used for the file operation.

5.7.1.2 Constant: INFO_NOT_AVAILABLE

Indicates that the demanded information is not available.

5.7.1.3 Constructor: FileViewProvider(Channel channel)

Encapsulates the defined channel by a FileViewProvider object that can be used for performing file operations on it.

Parameters

`channel`	The channel to be used by this Provider for file operations.

Errors

`IllegalStateError`	If an attempt is made to use an SE session or channel that has been closed.

Note: A file must be selected before a file operation can be performed. The file can be implicitly selected via a short file identifier (SFI), by the file operation method itself, or explicitly by defining the file ID (FID) with `selectByFID()` or path with `selectByPath()`.

5.7.1.4 Method: FCP selectByPath(String path, Boolean fromCurrentDF)

Selects the file specified by a path.

The path references the SE file by a path (concatenation of file IDs and the order of the file IDs is always in the direction 'parent to child') in the following notation: "DF1:DF2:EF1"; e.g. "0023:0034:0043". The defined path is applied to the SE as specified in [ISO 7816-4].

Note: For performing read or write operations on a file, the last node in the path must reference an EF that can be read or written.

Parameters

`path`	The path that references a file (DF or EF) on the SE. This path SHALL NOT contain the current DF or MF at the beginning of the path.
`fromCurrentDF`	If `true` then the path is selected from the current DF, if `false` then the path is selected from the MF.

Return value

The FCP containing information about the selected file.

Errors

`IllegalParameterError`	If the defined path is invalid.
`IllegalStateError`	If an attempt is made to use an SE session or channel that has been closed.
`SecurityError`	If the operation is not allowed because the security conditions are not satisfied.
`IllegalReferenceError`	If the file could not be selected.
`OperationNotSupportedError`	If this operation is not supported.

Notes:

- A file must be selected before a file operation can be performed.
- This method is based on the [ISO 7816-4] command SELECT.

5.7.1.5 Method: FCP selectByFID(int fileID)

Selects the file specified by the FID.

The fileID references the SE file (DF or EF) by a FID. The FID consists of a two-byte value as defined in [ISO 7816-4].

Parameters

`fileID`	The FID that references the file (DF or EF) on the SE. The FID must be in the range of (0x0000-0xFFFF).

Return value

The FCP containing information about the selected file.

Errors

`IllegalParameterError`	If `fileID` is not valid.
`IllegalStateError`	If an attempt is made to use an SE session or channel that has been closed.
`SecurityError`	If the operation is not allowed because the security conditions are not satisfied.
`IllegalReferenceError`	If the file could not be selected.
`OperationNotSupportedError`	If this operation is not supported.

Notes:

- A file must be selected before a file operation can be performed.
- This method is based on the [ISO 7816-4] command SELECT.

5.7.1.6 Method: FCP selectParent()

Selects the parent DF of the current DF.

The parent DF of the currently selected file is selected according to [ISO 7816-4].

If the currently selected file has no parent, then nothing will be done.

Return value

The FCP containing information about the selected file.

Errors

`IllegalStateError`	If an attempt is made to use an SE session or channel that has been closed.
`SecurityError`	If the operation is not allowed because the security conditions are not satisfied.
`IllegalReferenceError`	If the file could not be selected.
`OperationNotSupportedError`	If this operation is not supported.

Notes:

- A file must be selected before a file operation can be performed.
- This method is based on the [ISO 7816-4] command SELECT.

5.7.1.7 Method: Record readRecord(int sfi, int recID)

Returns the record which corresponds to the specified record ID. If the record is not found, then `Null` will be returned.

Parameters

`sfi`	The SFI of the file to be selected for this read operation. CURRENT_FILE can be applied if the file is already selected. `sfi` must be in the range of (1-30).
`recID`	The record ID that references the record to be read.

Return value

The record which corresponds to the specified record ID.

Errors

`IllegalStateError`	• If the used channel is closed. • If no file is currently selected. • If the currently selected file is not a record based file. • If the record could not be read.
`SecurityError`	If the operation is not allowed because the security conditions are not satisfied.
`IllegalReferenceError`	If the file could not be selected via SFI.
`IllegalParameterError`	• If the defined SFI is not valid. • If the defined record ID is invalid.
`OperationNotSupportedError`	If this operation is not supported.

Note: This method is based on the [ISO 7816-4] command READ RECORD.

5.7.1.8 Method: void writeRecord(int sfi, Record rec)

Writes a record into the specified file.

Parameters

`sfi`	The SFI of the file to be selected for this write operation. CURRENT_FILE can be applied if the file is already selected. `sfi` must be in the range of (1-30).
`rec`	The Record to be written.

Errors

`IllegalStateError`	• If the used channel is closed. • If no file is currently selected. • If the currently selected file is not a record based file. • If the record could not be written.
`SecurityError`	If the operation is not allowed because the security conditions are not satisfied.
`IllegalReferenceError`	If the file could not be selected via SFI.
`IllegalParameterError`	• If the defined record is invalid. • If the defined SFI is not valid.
`OperationNotSupportedError`	If this operation is not supported.

Note: This method is based on the [ISO 7816-4] command APPEND RECORD and UPDATE RECORD (which replaces existing bytes).

5.7.1.9 Method: int[] searchRecord(int sfi, byte[] searchPattern)

Returns the record numbers that contain the defined search pattern.

Parameters

`sfi`	The SFI of the file to be selected for this search operation. CURRENT_FILE can be applied if the file is already selected. `sfi` must be in the range of (1-30).
`searchPattern`	The pattern to be matched with records.

Return value

A list of record numbers (position 1..n of the record in the file) of the records that match the search pattern. If no record matches, then `Null` will be returned.

Errors

`IllegalParameterError`	If the defined SFI is not valid.
`IllegalStateError`	• If the used channel is closed. • If no file is currently selected. • If the currently selected file is not a record based file. • If the search pattern is empty. • If the search pattern is too long. • If the data could not be searched.
`SecurityError`	If the operation is not allowed because the security conditions are not satisfied.
`IllegalReferenceError`	If the file could not be selected via SFI.
`OperationNotSupportedError`	If this operation is not supported.

Note: This method is based on the [ISO 7816-4] command SEARCH RECORD with simple search.

5.7.1.10 Method: byte[] readBinary(int sfi, int offset, int length)

Reads content of the selected transparent file at the position specified by offset and length.

Parameters

`sfi`	The SFI of the file to be selected for this read operation. CURRENT_FILE can be applied if the file is already selected. `sfi` must be in the range of (1-30).
`offset`	Defines the start point of the file where the data should be read.
`length`	Defines the length of the data which should be read.

Return value

The data read from the file or `Null` if no content is available.

Errors

`IllegalParameterError`	• If the defined SFI is not valid. • If the defined offset and length could not be applied.
`IllegalStateError`	• If the used channel is closed. • If no file is currently selected. • If the currently selected file is not a transparent file. • If the data could not be read.
`SecurityError`	If the operation is not allowed because the security conditions are not satisfied.
`IllegalReferenceError`	If the file could not be selected via SFI.
`OperationNotSupportedError`	If this operation is not supported.

Note: This method is based on the [ISO 7816-4] command READ BINARY.

5.7.1.11 Method: void writeBinary(int sfi, byte[] data, int offset, int length)

Writes the defined data into the selected file at the position specified by offset and length.

Parameters

`sfi`	The SFI of the file to be selected for this write operation. CURRENT_FILE can be applied if the file is already selected. `sfi` must be in the range of (1-30).
`data`	The data to be written.
`offset`	Defines the position in the file where the data should be stored.
`length`	Defines the length of the data to be written.

Errors

`IllegalParameterError`	• If the defined SFI is not valid. • If the defined data array is empty or too short. • If the defined offset and length could not be applied.
`IllegalStateError`	• If the used channel is closed. • If no file is currently selected. • If the currently selected file is not a transparent file. • If the data could not be written.
`SecurityError`	If the operation is not allowed because the security conditions are not satisfied.
`IllegalReferenceError`	If the file could not be selected via SFI.
`OperationNotSupportedError`	If this operation is not supported.

Note: This method is based on the [ISO 7816-4] command UPDATE BINARY.

5.7.2 Class: FileViewProvider:FCP

File control parameter contains information of a selected file. FCPs are returned after a file select operation. This class is based on the [ISO 7816-4] FCP returned by the SELECT command as specified in [ISO 7816-4] section 5.3.3 (File control information) table 12 (File control parameter data objects).

5.7.2.1 Method: byte[] getFCP()

Returns the complete FCP as byte array.

Return value

The complete FCP as byte array.

Note: This method is based on the FCP control parameter as specified in [ISO 7816-4] section 5.3.3 (File control information) table 12 (File control parameter data objects).

5.7.2.2 Method: int getFileSize()

Returns the file size of the selected file (number of data bytes in the file, excluding structural information).

Return value

The file size depending on the file type:

- Transparent EF: The length of the body part of the EF.
- Linear fixed or cyclic EF: Record length multiplied by the number of records of the EF.

INFO_NOT_AVAILABLE if the information is not available.

Note: This method is based on the FCP control parameter as specified in [ISO 7816-4] section 5.3.3 (File control information) table 12 (File control parameter data objects).

5.7.2.3 Method: int getTotalFileSize()

Returns the total file size of the selected file (number of data bytes in the file, including structural information if any).

Return value

The total file size depending on the file type:

- DF/MF: the total file size represents the sum of the file sizes of all the EFs and DFs contained in this DF, plus the amount of available memory in this DF. The size of the structural information of the selected DF itself is not included.
- EF: the total file size represents the allocated memory for the content and the structural information (if any) of this EF.

INFO_NOT_AVAILABLE if the information is not available.

Note: This method is based on the FCP control parameter as specified in [ISO 7816-4] section 5.3.3 (File control information) table 12 (File control parameter data objects).

5.7.2.4 Method: int getFID()

Returns the file identifier of the selected file.

Return value

The file identifier of the selected file.

INFO_NOT_AVAILABLE if the FID of the selected file is not available.

Note: This method is based on the FCP control parameter as specified in [ISO 7816-4] section 5.3.3 (File control information) table 12 (File control parameter data objects).

5.7.2.5 Method: int getSFI()

Returns the short file identifier of the selected EF file.

Return value

The short file identifier of the selected file.

INFO_NOT_AVAILABLE if selected file is not an EF or an SFI is not available.

Note: This method is based on the FCP control parameter as specified in [ISO 7816-4] section 5.3.3 (File control information) table 12 (File control parameter data objects).

5.7.2.6 Method: int getMaxRecordSize()

Returns the maximum record size in case of a record based EF.

Return value

The maximum record size in case of a record based EF.

INFO_NOT_AVAILABLE if the currently selected file is not record based or the information cannot be fetched.

Note: This method is based on the FCP control parameter as specified in [ISO 7816-4] section 5.3.3 (File control information) table 12 (File control parameter data objects).

5.7.2.7 Method: int getNumberOfRecords()

Returns the number of records stored in the EF in case of a record based EF.

Return value

The number of records stored in the EF in case of a record based EF.

INFO_NOT_AVAILABLE if the currently selected file is not record based or the information cannot be fetched.

Note: This method is based on the FCP control parameter as specified in [ISO 7816-4] section 5.3.3 (File control information) table 12 (File control parameter data objects).

5.7.2.8 Method: int getFileType()

Returns the file type of the currently selected file.

Return value

The file type:

- (0) DF
- (1) EF

INFO_NOT_AVAILABLE if the information cannot be fetched.

Note: This method is based on the FCP control parameter as specified in [ISO 7816-4] section 5.3.3 (File control information) table 12 (File control parameter data objects). The file type is based on the definition in table 14 (File descriptor byte).

5.7.2.9 Method: int getFileStructure()

Returns the structure type of the selected EF.

Return value

The structure type of the selected file:

- (0) NO_EF
- (1) TRANSPARENT
- (2) LINEAR_FIXED
- (3) LINEAR_VARIABLE
- (4) CYCLIC

INFO_NOT_AVAILABLE if the information cannot be fetched.

Note: This method is based on the FCP control parameter as specified in [ISO 7816-4] section 5.3.3 (File control information) table 12 (File control parameter data objects). The file structure is based on the definition in table 14 (File descriptor byte).

5.7.2.10 Method: int getLcS()

Returns the lifecycle state of the currently selected file.

Return value

The lifecycle state:

- (0) NO_INFORMATION_GIVEN
- (1) CREATION_STATE
- (2) INITIALISATION_STATE
- (3) OPERATIONAL_STATE_ACTIVATED
- (4) OPERATIONAL_STATE_DEACTIVATED
- (5) TERMINATION_STATE

INFO_NOT_AVAILABLE if the information is not available.

Note: This method is based on the FCP control parameter as specified in [ISO 7816-4] section 5.3.3 (File control information) table 12 (File control parameter data objects). The lifecycle status is based on the definition in table 13 (Life cycle status byte).

5.7.3 Class: FileViewProvider:Record

Record class serves as a container for record data. The created record (as immutable object) can be used to read record data from a file or to write record data to a file.

5.7.3.1 Constructor: Record(int id, byte[] data)

Creates a record instance which can be used to store record data.

Parameter

`id`	The record id to be stored.
`data`	The data to be stored.

5.7.3.2 Method: int getID()

Returns the record ID of this record.

Return value

The record ID of this record.

5.7.3.3 Method: byte[] getData()

Returns the data of this record.

Return value

The data of this record.

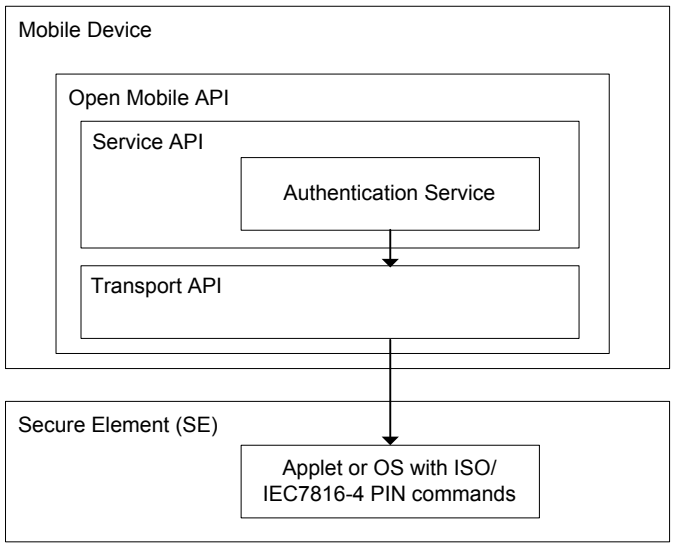
5.8 Authentication Service

Provides an API to perform a PIN authentication on the SE to enable an authentication state.

Use cases:

- Performing an operation on the SE which requires a user authentication.
  - For example, reading a file from the SE's file system which is PIN protected.
  - For example, using a key from the SE which requires a PIN authentication.
- Moreover, the API allows the management of SE PINs with management commands such as reset, change, activate, and deactivate.

Figure 5-8: Authentication Service Overview



5.8.1 Class: AuthenticationProvider

This Authentication class can be used to privilege a certain communication channel to the SE for operations that require a PIN authentication. Besides the PIN verification for authentication, this class also provides PIN management commands for changing, deactivating or activating PINs.

Prerequisites

The PIN operations performed by this AuthenticationProvider class are based on the [ISO 7816-4] specification and require a preselected applet on the specified communication channel to the SE that implements [ISO 7816-4] compliant PIN commands.

Notes

- If used by multiple threads, synchronization is up to the application.
- Each operation needs access to the SE. If access cannot be granted because of a closed channel or a missing security condition, the called method will return an error.

5.8.1.1 Constructor: AuthenticationProvider(Channel channel)

Encapsulates the defined channel by an AuthenticationProvider object that can be used for applying PIN commands on it.

Parameters

`channel`	The channel that should be privileged for operations which require a PIN authentication.

Errors

`IllegalStateError`	If an attempt is made to use an SE session or channel that has been closed.

5.8.1.2 Method: Boolean verifyPin(PinID pinID, byte[] pin)

Performs a PIN verification.

Parameters

`pinID`	A reference to the PIN in the SE to be used for the verification.
`pin`	The PIN to be verified.

Return value

`True` if the authentication was successful.

`False` if the authentication fails.

Errors

`IllegalParameterError`	If the PIN value has bad coding or wrong length (empty or too long).
`IllegalStateError`	If an attempt is made to use an SE session or channel that has been closed.
`IllegalReferenceError`	If the PIN referenced by `pinID` could not be found in the SE.
`OperationNotSupportedError`	If this operation is not supported.

Note:

This method is based on the [ISO 7816-4] command VERIFY.

When the PIN is blocked, the method returns `false`. Clients have to use `getRetryCounter()` to check for a blocked PIN.

5.8.1.3 Method: void changePin(PinID pinID, byte[] oldPin, byte[] newPin)

Changes the PIN.

Parameters

pinID	A reference to the PIN in the SE to be changed.
oldPin	The old PIN to be changed.
newPin	The PIN to be set as the new PIN.

Errors

IllegalParameterError	If oldPin or newPIN has bad coding or wrong length (empty or too long).
IllegalStateError	If an attempt is made to use an SE session or channel that has been closed.
SecurityError	If oldPin does not match the PIN stored in the SE. The PIN is not changed.
IllegalReferenceError	If the PIN referenced by pinID could not be found in the SE.
OperationNotSupportedError	If this operation is not supported.

Note: This method is based on the [ISO 7816-4] command CHANGE REFERENCE DATA.

5.8.1.4 Method: void resetPin(PinID pinID, byte[] resetPin, byte[] newPin)

Resets the PIN with the reset PIN or just resets the retry counter.

Parameters

pinID	A reference to the PIN in the SE to be reset.
resetPin	The reset PIN to be used for reset.
newPin	The PIN to be set as new PIN. Can be Null if only the reset counter is to be reset.

Errors

IllegalParameterError	If resetPin or newPIN has bad coding or wrong length (empty or too long).
IllegalStateError	If an attempt is made to use an SE session or channel that has been closed.
SecurityError	If resetPIN does not match the “reset PIN” stored in the SE. The PIN and reset counter are not changed.
IllegalReferenceError	If the PIN referenced by pinID could not be found in the SE.
OperationNotSupportedError	If this operation is not supported (e.g. PIN is not defined).

Note: This method is based on the [ISO 7816-4] command RESET RETRY COUNTER.

5.8.1.5 Method: int getRetryCounter(PinID pinID)

Returns the retry counter of the referenced PIN.

Parameters

`pinID`	A reference to the PIN in the SE and its retry counter.

Return value

The retry counter of the referenced PIN.

Errors

`IllegalStateError`	If an attempt is made to use an SE session or channel that has been closed.
`IllegalReferenceError`	If the PIN referenced by `pinID` could not be found in the SE.
`OperationNotSupportedError`	If this operation is not supported.

Note: This method is based on the [ISO 7816-4] command VERIFY.

5.8.1.6 Method: void activatePin(PinID pinID, byte[] pin)

Activates the PIN. Thus a deactivated PIN can be used again.

Parameters

`pinID`	A reference to the PIN in the SE to be activated.
`pin`	The verification PIN for activating the PIN if required. Can be `Null` if not required.

Errors

`IllegalParameterError`	If `pin` has bad coding or wrong length (empty or too long).
`IllegalStateError`	If an attempt is made to use an SE session or channel that has been closed.
`SecurityError`	If `pin` does not match the PIN needed for the activation. The PIN state will not be changed.
`IllegalReferenceError`	If the PIN referenced by `pinID` could not be found in the SE.
`OperationNotSupportedError`	If this operation is not supported.

Note: This method is based on the [ISO 7816-4] command ENABLE VERIFICATION REQUIREMENT.

5.8.1.7 Method: void deactivatePin(PinID pinID, byte[] pin)

Deactivates the PIN. Thus the objects which are protected by the PIN can now be used without this restriction until `activatePin()` is called.

Parameters

`pinID`	A reference to the PIN in the SE to be deactivated.
`pin`	The verification PIN for deactivating the pin if required. Can be `Null` if not required.

Errors

`IllegalParameterError`	If `pin` has bad coding or wrong length (empty or too long).
`IllegalStateError`	If an attempt is made to use an SE session or channel that has been closed.
`SecurityError`	If the defined PIN does not match the PIN needed for the deactivation. The PIN state will not be changed.
`IllegalReferenceError`	If the PIN referenced by `pinID` could not be found in the SE.
`OperationNotSupportedError`	If this operation is not supported.

Note: This method is based on the [ISO 7816-4] command DISABLE VERIFICATION REQUIREMENT.

5.8.2 Class: AuthenticationProvider:PinID

This PIN ID uniquely identifies a PIN in the SE system. The PIN ID is defined as specified in [ISO 7816-4] and can be used to reference a PIN in an [ISO 7816-4] compliant system.

5.8.2.1 Constructor: PinID(int id, Boolean local)

Creates a PIN ID (reference) to identify a PIN within a SE. The created PIN ID (as immutable object) can be specified on all PIN operation methods provided by the AuthenticationProvider class.

Parameters

`id`	The ID of the PIN (value from 0x00 to 0x1F).
`local`	Defines the scope (global or local). `True` if the PIN is local. Otherwise, `false`.

Errors

`IllegalParameterError`	If `id` is invalid.

Note: This constructor is based on the P2 reference data for PIN related commands as specified in [ISO 7816-4] section 7.5 (basic security handling). Local set to `true` indicates specific reference data and local set to `false` indicates global reference data according to [ISO 7816-4]. The ID indicates the number of the reference data (qualifier) according to [ISO 7816-4].

5.8.2.2 Method: int getID()

Returns the PIN ID.

Return value

The PIN ID.

Note: This method is based on the P2 reference data for PIN related commands as specified in [ISO 7816-4] section 7.5 (basic security handling). The ID indicates the number of the reference data (qualifier) according to [ISO 7816-4].

5.8.2.3 Method: Boolean isLocal()

Identifies if the PIN is local or global.

Return value

`True` if the PIN is local. Otherwise, `false`.

Note: This method is based on the P2 reference data for PIN related commands as specified in [ISO 7816-4] section 7.5 (basic security handling). Local set to `true` indicates specific reference data and local set to `false` indicates global reference data according to [ISO 7816-4].

5.9 PKCS #15 API

The PKCS #15 standard ([PKCS #15]) is a structured way to store and organize data. The PKCS #15 service API simplifies access to PKCS #15 file systems according to v1.1 of the PKCS #15 specification. Classes and methods are provided to retrieve the elementary PKCS #15 data structures, such as ODF (Object Directory File) and TokenInfo.

PKCS #15 file systems can be used to store cryptographic data, but also, any kind of data using application-specific files and OIDs. For example in the OMA-DM use case, the bootstrap data can be stored in the SE's file system using a dedicated PKCS #15 file structure. This PKCS #15 API can be used by an OMA-DM client application to retrieve the bootstrap data from a SE.

The preferred way to select a PKCS #15 file system is through the PKCS #15 AID (A0 00 00 00 63 50 4B 43 53 2D 31 35), however, a legacy file system can reference a PKCS #15 data structure though the EF(DIR).

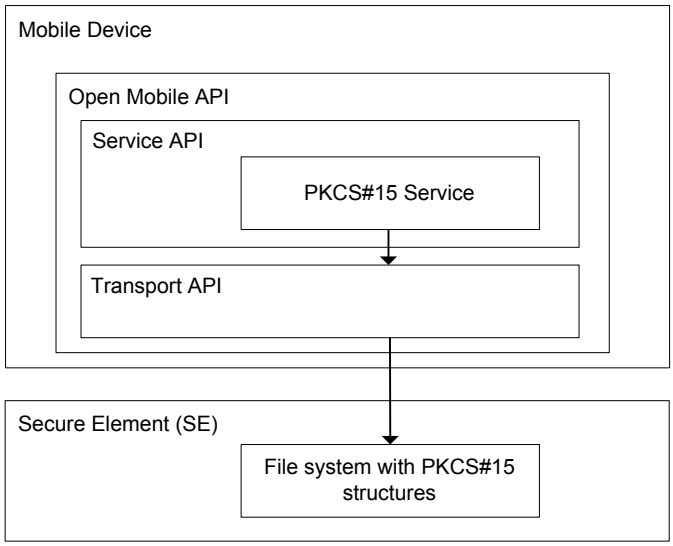
Example of a typical PKCS #15 file system:

```
ADF(PKCS#15)        : AID = A0 00 00 00 63 50 4B 43 53 2D 31 35
 |-EF(ODF)          : FID = 5031
 |-EF(TokenInfo)    : FID = 5032
 |-EF(PrKDF)        : optional, referenced by EF(ODF)
 |-EF(PuKDF)        : optional, referenced by EF(ODF)
 |-EF(CDF)          : optional, referenced by EF(ODF)
 |-EF(DODF)         : optional, referenced by EF(ODF)
 |-EF(AODF)         : optional, referenced by EF(ODF)
```

Example of a legacy file system with a PKCS #15 structure:

```
MF                  : FID = 3F00
 |-EF(DIR)          : FID = 2F00
 |-DF(PKCS#15)      : referenced by EF(DIR)
    |-EF(ODF)       : FID = 5031
    |-EF(TokenInfo) : FID = 5032
```

Figure 5-9: PKCS #15 Service Overview



5.9.1 Class: PKCS15Provider

This Provider class offers basic services to access a PKCS #15 file system. This Provider requires a PKCS #15 data structure on the SE and a channel instance allowing access to this PKCS #15 data structure.

5.9.1.1 Constant: byte[] AID_PKCS15

Default PKCS #15 AID (A0 00 00 00 63 50 4B 43 53 2D 31 35).

5.9.1.2 Constructor: PKCS15Provider(Channel channel)

Encapsulates the defined channel by a PKCS #15 file system object. This method checks the presence of the EF(ODF) (Object Directory File) with file identifier 5031 and of the EF(TokenInfo) with file identifier 5032. Both files are mandatory and must be present in a valid PKCS #15 file system.

This method must first try to select EF(ODF) and EF(TokenInfo) on the provided channel. If the select fails, this method must try to locate a DF(PKCS #15) in the legacy file system using the EF(DIR) according to the data structure described in section 5.4 of [PKCS #15].

Parameters

`channel`	The channel to be used by this Provider for file operations.

Errors

`IOError`	If no PKCS #15 file system is detected on the provided channel.

5.9.1.3 Method: byte[] getODF()

Returns the raw content of the EF(ODF) (Object Directory File).

Return value

The EF(ODF) as a byte array. Must not be null.

5.9.1.4 Method: byte[] getTokenInfo()

Returns the raw content of the EF(TokenInfo).

Return value

The EF(TokenInfo) as a byte array. Must not be null.

5.9.1.5 Method: Path[] getPrivateKeyPaths()

Returns an array of EF(PrKDF) paths (Private Key Directory Files). The PKCS #15 file system may contain zero, one or several EF(PrKDF).

Return value

The array of EF(PrKDF) paths. May be null if empty.

5.9.1.6 Method: Path[] getPublicKeyPaths()

Returns an array of EF(PuKDF) paths (Public Key Directory Files). The PKCS #15 file system may contain zero, one or several EF(PuKDF).

Return value

The array of EF(PuKDF) paths. May be null if empty.

5.9.1.7 Method: Path[] getCertificatePaths()

Returns an array of EF(CDF) paths (Certificate Directory Files). The PKCS #15 file system may contain zero, one or several EF(CDF).

Return value

The array of EF(CDF) paths. May be null if empty.

5.9.1.8 Method: Path[] getDataObjPaths()

Returns an array of EF(DODF) paths (Data Object Directory Files). The PKCS #15 file system may contain zero, one or several EF(DODF).

Return value

The array of EF(DODF) paths. May be null if empty.

5.9.1.9 Method: Path[] getAuthObjPaths()

Returns an array of EF(AODF) paths (Authentication Object Directory Files). The PKCS #15 file system may contain zero, one or several EF(AODF).

Return value

The array of EF(AODF) paths. May be null if empty.

5.9.1.10 Method: byte[] readFile(Path path)

Selects and reads a PKCS #15 file. The file may be a transparent or linear fixed EF. The ‘index’ and ‘length’ fields of the path instance will be used according to section 6.1.5 of [PKCS #15]. In the case of a transparent EF, ‘index’ is the start offset in the file and ‘length’ is the length to read. In the case of a linear fixed EF, ‘index’ is the record to read.

Parameters

`path`	Path of the file.

Return value

The file content as a byte array. Or null if the referenced path does not exist.

Errors

`SecurityError`	If the operation cannot be performed, if a security condition is not satisfied.
`OperationNotSupportedError`	If this operation is not supported.
`IOError`	If the PKCS #15 file cannot be selected or read.

5.9.1.11 Method: byte[] searchOID(byte[] dodf, String oid)

Parses the raw content of an EF(DODF) and searches for a specific OID Data Object. This method is a convenience method to simplify the access to OID Data Objects by applications, as described in section 6.7.4 of [PKCS #15]. In many cases, the EF(DODF) contains a simple OID Data Object with a Path object, in order to reference an application-specific EF. For example, the OMA-DM specification requires a EF(DODF) containing the OID 2.23.43.7.1, followed by a path object, referencing the EF(DM_Bootstrap).

Parameters

`dodf`	The raw content of an EF(DODF) to parse.
`oid`	The searched OID value (e.g. OMA-DM bootstrap OID is 2.23.43.7.1).

Return value

The raw object value if OID has been found, null if not found.

Errors

`IllegalParameterError`	If the OID is not correct.
`OperationNotSupportedError`	If this operation is not supported.

5.9.1.12 Method: Path decodePath(byte[] der)

Builds a path object using a DER-encoded (see ITU X.690 for DER-Coding) buffer.

Parameters

`der`	The DER-encoded path object as a byte array.

Return value

The path object.

Errors

`IllegalParameterError`	If the defined path is not a correctly DER-encoded buffer.
`OperationNotSupportedError`	If this operation is not supported.

5.9.2 Class: PKCS15Provider:Path

This class represents a path object as defined in [PKCS #15] section 6.1.5.

5.9.2.1 Constructor: Path(byte[] path)

Builds a path object without index and length (the path can be absolute as well as relative).

Parameters

`path`	The path as a byte array.

Errors

`IllegalParameterError`	If the path is not correct.

5.9.2.2 Constructor: Path(byte[] path, int index, int length)

Builds a path object with index and length (the path can be absolute as well as relative).

Parameters

`path`	The path as a byte array.
`index`	The index value.
`length`	The length value.

Errors

`IllegalParameterError`	If `path`, `index`, or `length` is not correct.

5.9.2.3 Method: byte[] getPath()

Returns the path field of this path object.

Return value

The path field.

5.9.2.4 Method: Boolean hasIndexLength()

Checks whether this path object has an index and length fields.

Return value

`True` if the index and length field is present, `false` otherwise.

5.9.2.5 Method: int getIndex()

Returns the index field of this path object. The value of this field is undefined if the method `hasIndexLength()` returns `false`.

Return value

The index field.

5.9.2.6 Method: int getLength()

Returns the length field of this path object. The value of this field is undefined if the method `hasIndexLength()` returns `false`.

Return value

The length field.

5.9.2.7 Method: byte[] encode()

Encodes this path object according to DER (see ITU X.690 for DER-Coding).

Return value

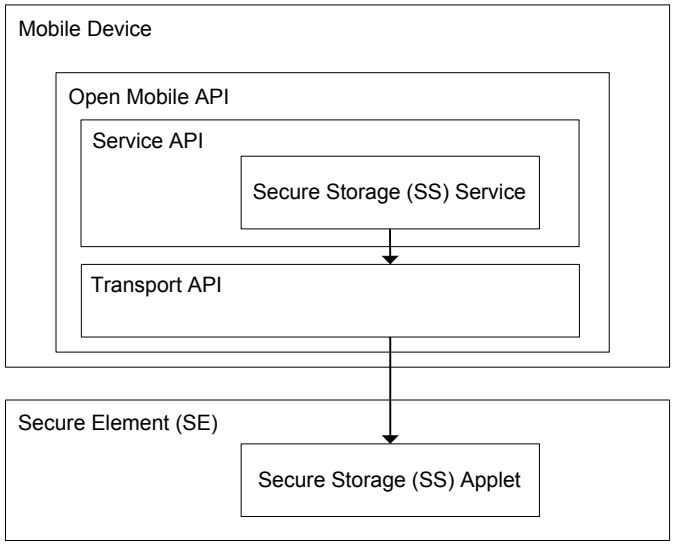
This path object as a DER-encoded byte array.

5.10 Secure Storage

The Secure Storage (SS) Service can be used to store and retrieve sensitive data on the SE. This API requires a SS applet on the SE with an APDU interface as defined below.

Data is stored in a dictionary format (string, value). It is simpler to store data in the SS than with a PKCS#15Provider or FileViewProvider, which can, in principle, also be used to store data securely but in a more elaborate way.

Figure 5-10: Secure Storage Service Overview



5.10.1 Class: SecureStorageProvider

This class provides an API to store and retrieve data on the SE which is protected in a secure environment. A default set of functionality that is always provided on every platform enables application developers to rely on this interface for secure data storage (e.g. credit card numbers, private phone numbers, passwords etc.). The interface should encapsulate any SE specifics as it is intended for device application developers who might not be familiar with SE or APDU internals.

Security Notes

PIN verification is required to grant access to the SS applet where the Authentication Provider can be reused for the PIN operations. The SS applet must separate the PIN verification on all logical channels to ensure that each device application needs to verify the PIN individually.

Prerequisites

The SS operations performed by this Provider class are based on a SS located in the SE. The SS is usually realized by an applet (providing the defined SS APDU interface) that must be preselected on the specified communication channel to the SE before this Provider can be used.

Notes:

- If used by multiple threads, synchronization is up to the application.
- Each operation needs access to the SE. If access cannot be granted because of a closed channel or a missing security condition, the called method will return an error.

5.10.1.1 Constructor: SecureStorageProvider(Channel channel)

Creates a SecureStorageProvider instance which will be connected to the preselected SE SS applet on a defined channel.

Parameters

`channel`	The channel to be used by this Provider for operations on the SS.

Errors

`IllegalStateError`	If an attempt is made to use an SE session or channel that has been closed.

5.10.1.2 Method: void create(String title, byte[] data)

This command creates a SS entry with the defined title and data. The data can contain an uninterpreted byte stream of an undefined max length (e.g. names, numbers, image, media data, etc.).

Parameters

`title`	The title of the entry to be written. The maximum title length is 60. All characters must be supported by UTF-8.
`data`	The data of the entry to be written. If data is empty or null then only the defined title will be assigned to the new entry.

Errors

`IllegalParameterError`	• If the title already exists. All entry titles must be unique within the SS. • If the title is incorrect: bad encoding or wrong length (empty or too long). • If the data chain is too long.
`IllegalStateError`	If an attempt is made to use an SE session or channel that has been closed.
`SecurityError`	If the PIN to access the SS applet was not verified.
`IOError`	If the entry could not be created because of an incomplete write procedure.

5.10.1.3 Method: void update(String title, byte[] data)

This command updates the data of the SS entry referenced by the defined title. The data can contain an uninterpreted byte stream of an undefined max length (e.g. names, numbers, image, media data, etc.).

Parameters

`title`	The title of the entry that must already exist. The maximum title length is 60. All characters must be supported by UTF-8.

`data`	The data of the entry to be written. If data is empty or null then the data of the existing entry (referenced by the title) will be deleted.

Errors

`IllegalParameterError`	• If `title` does not already exist. • If `title` is incorrect: bad encoding or wrong length (empty or too long). • If the data chain is too long.
`IllegalStateError`	If an attempt is made to use an SE session or channel that has been closed.
`SecurityError`	If the PIN to access the SS applet was not verified.
`IOError`	If the entry could not be updated because of an incomplete write procedure.

5.10.1.4 Method: byte[] read(String title)

This command reads and returns the byte stream of a data entry stored in the SE referenced by the title.

Parameters

`title`	The title of the entry to be read. The maximum title length is 60. All characters must be supported by UTF-8.

Return value

The data retrieved from the referenced entry. If the data does not exist in the SS entry referenced by the title then an empty byte array will be returned.

Errors

`IllegalParameterError`	If `title` is incorrect: bad encoding or wrong length (empty or too long).
`IllegalStateError`	If an attempt is made to use an SE session or channel that has been closed.
`SecurityError`	If the PIN to access the SS applet was not verified.
`IOError`	If the entry could not be read because of an incomplete read procedure.

5.10.1.5 Method: Boolean exist(String title)

This command checks if the SS entry with the defined title exists.

Parameters

`title`	The title of the entry to be checked. The maximum title length is 60. All characters must be supported by UTF-8.

Errors

`IllegalParameterError`	If `title` is incorrect: bad encoding or wrong length (empty or too long).
`IllegalStateError`	If an attempt is made to use an SE session or channel that has been closed.
`SecurityError`	If the PIN to access the SS applet was not verified.

Return value

`True` if the entry with the defined title exists. `False` if the entry does not exist.

5.10.1.6 Method: Boolean delete(String title)

This command deletes the SS entry referenced by the title. If the entry does not exist, nothing will be done.

Parameters

`title`	The title of the entry to be deleted. The maximum title length is 60. All characters must be supported by UTF-8.

Errors

`IllegalParameterError`	If `title` is incorrect: bad encoding or wrong length (empty or too long).
`IllegalStateError`	If an attempt is made to use an SE session or channel that has been closed.
`SecurityError`	If the PIN to access the SS applet was not verified.

Return value

`True` if the entry with the defined title is deleted. `False` if the entry does not exist.

5.10.1.7 Method: void deleteAll()

This command deletes all SS entry referenced. If no entries exist nothing will be done.

Errors

`IllegalStateError`	If an attempt is made to use an SE session or channel that has been closed.
`SecurityError`	If the PIN to access the SS applet was not verified.

5.10.1.8 Method: String[] list()

This command returns an entry list with all title-identifiers. The title is intended for the users to identify and to reference the SS entries.

Return value

A list of titles of all entries located in SS. An empty list will be returned if no entries exist in the SS.

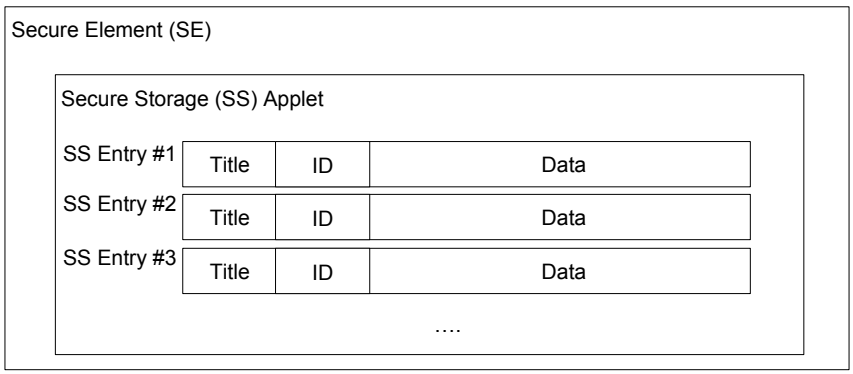
Errors

`IllegalStateError`	If an attempt is made to use an SE session or channel that has been closed.
`SecurityError`	If the PIN to access the SS applet was not verified.

5.10.2 Secure Storage APDU Interface

The SS applet has to provide this APDU command interface for adding entries to the SS and deleting entries from the SS. Each SS entry is a container for a SS data record consisting of a title and data attribute. The attribute title identifies the SS entry with a user readable text and must be unique. The SS entry title SHALL be defined by the user before the SS entry is created. Thus the user can identify the created SS entry within the SS afterwards. The data attribute contains the sensitive data which has to be stored into the SS. Besides the title, each SS entry can also be identified with a unique ID which is generated by the SS during the creation and has to be used to reference an SS entry within the SS. The title and ID must be unique within a SS, as each entry can be referenced by either the title or the ID.

Figure 5-11: Secure Storage Applet Overview



5.10.2.1 CREATE SS ENTRY Command Message

The CREATE SS ENTRY command can be used to create an entry in the SS. Each entry requires a unique title which must be specified in the command.

The CREATE SS ENTRY command message SHALL be coded according to the following table:

Table 5-1: CREATE SS ENTRY Command Message

Code	Value	Meaning
CLA	'80'	
INS	'E0'	CREATE SS ENTRY
P1 P2	'00 00'	
Lc	Length of title	
Data	Title	Title of the new entry in UTF-8. This title must be unique within the SS. If the defined title already exists in the SS, then the error code '6A 80' will be returned.
Le	2	

5.10.2.2 CREATE SS ENTRY Response Message

The CREATE SS ENTRY response SHALL contain a data field with response code '90 00' (successful operation) or an error response code.

Table 5-2: CREATE SS ENTRY Response Data

Value	Meaning	Presence
ID	The ID of the new entry created in the SS. This ID can be used to reference an SS entry. The length of this ID is always 2 bytes.	Mandatory

Table 5-3: CREATE SS ENTRY Response Code

SW1	SW2	Meaning
'6A'	'80'	Incorrect values in the command data (if the defined title does already exist)
'6A'	'82'	Security status not satisfied (if PIN verified state is not set)
'67'	'00'	Wrong length in Lc
'6A'	'86'	Incorrect P1 P2
'6D'	'00'	Invalid instruction
'6E'	'00'	Invalid class
'65'	'81'	Memory failure (if the creation of the entry fails due to memory issues)
'6A'	'84'	Not enough memory space (if not enough memory resources are available)

5.10.2.3 DELETE SS ENTRY Command Message

The DELETE SS ENTRY command can be used to delete an entry from the SS. The entry referenced in this command by an ID must exist in the SS otherwise an error code will be returned.

The DELETE SS ENTRY command message shall be coded according to the following table:

Table 5-4: DELETE SS ENTRY Command Message

Code	Value	Meaning
CLA	'80'	
INS	'E4'	DELETE SS ENTRY
P1 P2	P1: ID high byte P2: ID low byte	The ID of the SS entry which has to be deleted. If the SS entry couldn't be found, then '6A 88' is returned.
Lc	-	
Data	-	
Le	-	

5.10.2.4 DELETE SS ENTRY Response Message

The DELETE SS ENTRY response shall contain a response code '90 00' (successful operation) or an error response code.

Table 5-5: DELETE SS ENTRY Response Code

SW1	SW2	Meaning
'6A'	'82'	Security status not satisfied (if PIN verified state is not set)
'6A'	'88'	Referenced data not found (if the referenced SS entry does not exist)
'6D'	'00'	Invalid instruction
'6E'	'00'	Invalid class
'65'	'81'	Memory failure (if the operation fails due to memory issues)

5.10.2.5 SELECT SS ENTRY Command Message

The SELECT SS ENTRY has to be used to select an SS ENTRY in the SS for a write or read operation.

The SELECT SS ENTRY command message shall be coded according to the following table:

Table 5-6: SELECT SS ENTRY Command Message

Code	Value	Meaning
CLA	'80'	
INS	'A5'	SELECT SS ENTRY
P1 P2	P1: Reference parameter P2: '00'	Reference parameter: Select ID('00'): Select the entry referenced by the ID Select First('01'): Select the first SS entry Select Next('02'): Select the next available SS entry
Lc	2 or -	The length of the ID of the SS entry (only needed with P1 = Select ID)
Data	ID or -	The ID of the SS entry to be selected (only needed with P1 = Select ID)
Le	'00'	

5.10.2.6 SELECT SS ENTRY Response Message

The SELECT SS ENTRY response shall contain a data field with response code '90 00' (successful operation) or an error response code.

Table 5-7: SELECT SS ENTRY Response Data

Value	Meaning	Presence
Title	The title of the referenced SS entry in UTF-8.	Mandatory

Table 5-8: SELECT SS ENTRY Response Code

SW1	SW2	Meaning
'6A'	'80'	Incorrect values in the command data (if the data field has a length other than 2 bytes)
'6A'	'82'	Security status not satisfied (if PIN verified state is not set)
'6A'	'88'	Referenced data not found (if the referenced SS entry does not exist)
'67'	'00'	Wrong length in Lc
'6A'	'86'	Incorrect P1 P2
'6D'	'00'	Invalid instruction
'6E'	'00'	Invalid class

5.10.2.7 PUT SS ENTRY DATA Command Message

The PUT SS ENTRY DATA command message can be used to store sensitive data into the currently selected SS entry. An SS entry can be selected with the command SELECT SS ENTRY DATA. For very long data the command PUT SS ENTRY DATA can be used iteratively by applying the command several times with an appropriate P1 parameter (first) and (next).

Note:

Before data can be stored into the SS entry with PUT SS ENTRY DATA, the data size has to be specified. Otherwise an error code will be returned.

The transmitted data will only be stored into the SS entry if all data parts are transferred (this means the sum of all transferred parts fits exactly to the defined data length). As long as the transferred data is not complete, the data has to be temporarily buffered within the SS application. If the succeeding APDU is not a PUT SS ENTRY DATA command or the succeeding PUT SS ENTRY DATA command does not contain the following data as expected, then the buffered data has to be discarded.

The PUT SS ENTRY DATA command message shall be coded according to the following table:

Table 5-9: PUT SS ENTRY Data Command Message

Code	Value	Meaning
CLA	'80'	
INS	'DA'	PUT SS ENTRY DATA
P1 P2	P1: size (0), first (1), next (2) P2: '00'	size (0): The whole size of the data to be stored. first (1): DATA contains the first data part. next (2): DATA contains the next data part (append mode).
Lc	Data length	
Data	Data	P1 = size (0): Defines the data size. P1 = first (1) or next (2): Sensitive data (or a part of the data) to be stored to the currently selected SS entry.
Le	-	

5.10.2.8 PUT SS ENTRY DATA Response Message

The PUT SS ENTRY ID response shall contain a response code '90 00' (successful operation) or an error response code.

Table 5-10: PUT SS ENTRY DATA Response Code

SW1	SW2	Meaning
'6A'	'80'	Incorrect values in the command data
'6A'	'82'	Security status not satisfied (if PIN verified state is not set)
'6A'	'88'	Referenced data not found (if no SS entry is currently selected)
'67'	'00'	Wrong length in Lc
'6A'	'86'	Incorrect P1 P2 (if the defined P1/P2 are invalid or cannot be applied)
'6D'	'00'	Invalid instruction
'6E'	'00'	Invalid class
'65'	'81'	Memory failure (if the defined data exceeds the defined size or a size was not defined)
'6A'	'84'	Not enough memory space (if not enough memory resources are available)

5.10.2.9 GET SS ENTRY DATA Command Message

The GET SS ENTRY DATA command message can be used to retrieve data from the currently selected SS entry. An SS entry can be selected with the command SELECT SS ENTRY DATA. For very long data, the command GET SS ENTRY DATA can be used iteratively by applying the command several times with an appropriate P1 parameter (first) and (next). If the succeeding APDU is not a GET SS ENTRY DATA command, then an outstanding retrieve procedure must be reset by the SS application.

The GET SS ENTRY DATA command message shall be coded according to the following table:

Table 5-11: GET SS ENTRY DATA Command Message

Code	Value	Meaning
CLA	'80'	
INS	'CA'	GET SS ENTRY DATA
P1 P2	P1: size (0), first (1), next (2) P2: '00'	size (0): Response contains the whole size of the data to be read. first (1): Response contains the first data part. next (2): Response contains the next data part.
Lc	-	
Data	-	
Le	'00'	

5.10.2.10 GET SS ENTRY DATA Response Message

The GET SS ENTRY DATA response shall contain a data field with response code '90 00' (successful operation) or an error response code.

Table 5-12: GET SS ENTRY DATA Response Data

Value	Meaning	Presence
Data	The data (or a part) of the currently selected SS entry. or Whole size of the data stored in the currently selected SS entry.	Mandatory

Table 5-13: GET SS ENTRY DATA Response Code

SW1	SW2	Meaning
'6A'	'82'	Security status not satisfied (if PIN verified state is not set)
'6A'	'86'	Incorrect P1 P2 (if the defined P1/P2 are invalid or cannot be applied)
'6A'	'88'	Referenced data not found (if no SS entry is currently selected)
'6D'	'00'	Invalid instruction
'6E'	'00'	Invalid class
'65'	'81'	Memory failure (if further data is demanded but no further data exists).

5.10.2.11 GET SS ENTRY ID Command Message

The GET SS ENTRY ID command message can be used to retrieve the ID of an SS entry referenced by the title.

The GET SS ENTRY ID command message shall be coded according to the following table:

Table 5-14: GET SS ENTRY ID Command Message

Code	Value	Meaning
CLA	'80'	
INS	'B2'	GET ENTRY ID
P1 P2	'00 00'	
Lc	Length of title	
Data	Title	Title of the entry
Le	'02'	

5.10.2.12 GET SS ENTRY ID Response Message

The READ SS ENTRY ID response shall contain a data field with response code '90 00' (successful operation) or an error response code.

Table 5-15: GET SS ENTRY ID Response Data

Value	Meaning	Presence
ID	The ID of the entry in the SS referenced by the defined title. The length of this ID is always 2 bytes.	Mandatory

Table 5-16: GET SS ENTRY ID Response Code

SW1	SW2	Meaning
'6A'	'82'	Security status not satisfied (if PIN verified state is not set)
'6A'	'88'	Referenced data not found (if the referenced SS entry does not exist)
'6A'	'86'	Incorrect P1 P2
'6D'	'00'	Invalid instruction
'6E'	'00'	Invalid class

5.10.2.13 DELETE ALL SS ENTRIES Command Message

The DELETE ALL SS ENTRIES command can be used to delete all entries from the SS.

The DELETE ALL SS ENTRIES command message shall be coded according to the following table:

Table 5-17: DELETE ALL SS ENTRIES Command Message

Code	Value	Meaning
CLA	'80'	
INS	'E5'	DELETE ALL SS ENTRIES
P1 P2	'00 00'	
Lc	-	
Data	-	
Le	-	

5.10.2.14 DELETE ALL SS ENTRIES Response Message

The DELETE SS ENTRIES response shall contain a response code '90 00' (successful operation) or an error response code.

Table 5-18: DELETE ALL SS ENTRIES Response Code

SW1	SW2	Meaning
'6A'	'82'	Security status not satisfied (if PIN verified state is not set)
'6D'	'00'	Invalid instruction
'6E'	'00'	Invalid class
'65'	'81'	Memory failure (if the operation fails due to memory issues)

5.10.3 Secure Storage APDU Transfer

This section describes how the SS APDU interface has to be applied for performing the SS service operations provided by the SecureStorageProvider.

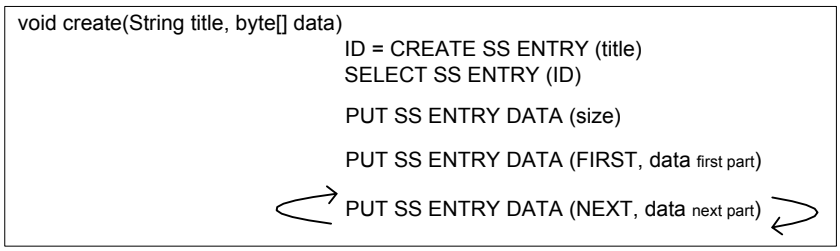
Note: All SS operations have to be realized in an atomic way. This means if an error occurs during a SS operation (e.g. if an error occurs on a certain APDU) all modifications made on the SS (in the previous steps within a SS operation) have to be reversed.

5.10.3.1 Create Operation

The create method includes the creation and selection of an SS entry with a succeeding SS entry data update. The following steps are needed:

- CREATE SS ENTRY (title) creates the SS entry in the SS with the defined title.
- SELECT SS ENTRY (ID) selects the SS entry in the SS for the data update.
- PUT SS ENTRY DATA (size) to define the data size.
- PUT SS ENTRY DATA (data) writes the data to the selected SS entry. A long data chain (which cannot be transferred via one APDU command) can be written by applying the command iteratively by using PUT SS ENTRY DATA(P1P2=NEXT) several times after performing PUT SS ENTRY DATA(P1P2=FIRST).

Figure 5-12: Create SS Entry Operation



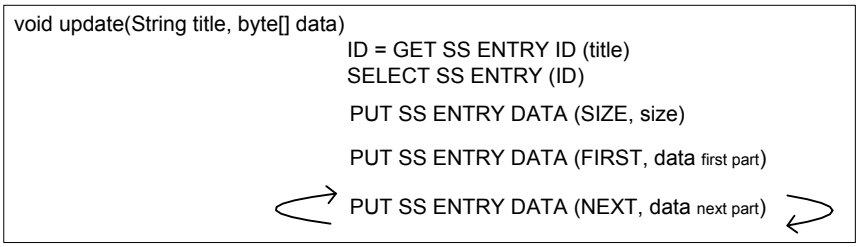
Note: If an error occurs during the create operation, all previous steps must be reversed to obtain the same SS state as before. For example, if the creation of the SS entry was successful but the storage of the SS entry data fails, then this newly created SS entry has to be deleted again.

5.10.3.2 Update Operation

The `update()` method includes the selection of an SS entry with a succeeding SS entry data update. The following steps are needed:

- GET ENTRY ID (title) returns the internal SS entry ID to the defined title.
- SELECT SS ENTRY (ID) selects the SS entry in the SS for the data update.
- PUT SS ENTRY DATA (size) to define the data size.
- PUT SS ENTRY DATA (data) writes the data to the selected SS entry. A long data chain (which cannot be transferred via one APDU command) can be written by applying the command iteratively by using PUT SS ENTRY DATA (P1P2=NEXT) several times after performing PUT SS ENTRY DATA (P1P2=FIRST).

Figure 5-13: Update SS Entry Operation



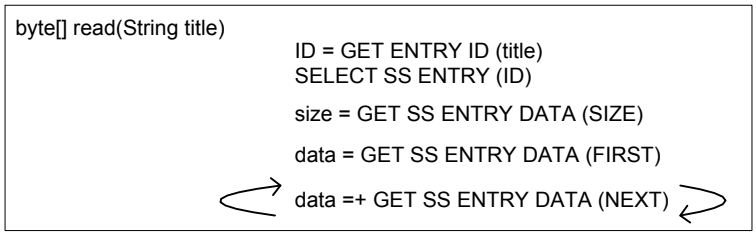
Note: If an error occurs during the update operation, all previous steps must be reversed to obtain the same SS state as before. For example, if the update of some data parts was successful but the update of a following data part fails then the SS entry has to be set to the previous state (e.g. by reassigning the previously stored data to the SS entry).

5.10.3.3 Read Operation

The read method includes the selection of an SS entry with a succeeding read SS entry data operation. The following steps are needed:

- GET ENTRY ID (title) returns the internal SS entry ID to the defined title.
- SELECT SS ENTRY (ID) selects the SS entry in the SS for the read operation.
- GET SS ENTRY DATA (size) to determine the whole size of the data to be read.
- GET SS ENTRY DATA (data) reads the data to the selected SS entry. A long data chain (which cannot be transferred via one APDU command) can be read by applying the command iteratively by using GET SS ENTRY DATA (P1P2=NEXT) several times after performing GET SS ENTRY DATA (P1P2=FIRST).

Figure 5-14: Read SS Entry Operation

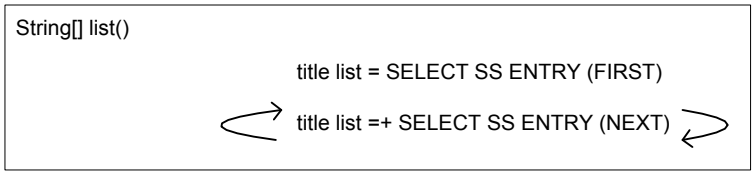


5.10.3.4 List Operation

The list method includes an iterative selection of all SS entries. The following steps are needed:

- SELECT SS ENTRY (FIRST) selects the first SS entry and returns its title.
- SELECT SS ENTRY (NEXT) selects the next SS entry and returns its title. This command has to be applied iteratively until all SS entry titles are retrieved.

Figure 5-15: List SS Entries Operation



5.10.3.5 Delete Operation

The delete method includes a delete operation. The following steps are needed:

- GET ENTRY ID (title) returns the internal SS entry ID to the defined title.
- DELETE SS ENTRY (ID) deletes the SS entry referenced by the defined ID.

Figure 5-16: Delete SS Entry Operation

5.10.3.6 Delete All Operation

The delete all method includes a delete operation which deletes all entries from the SS. The following steps are needed:

- DELETE ALL SS ENTRIES

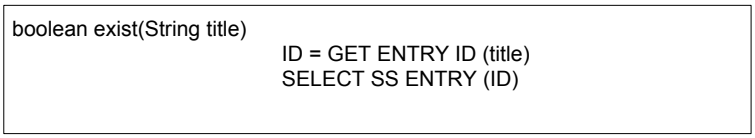
Figure 5-17: Delete All SS Entries Operation

5.10.3.7 Exist Operation

The exist method checks if a certain SS entry exists. The following steps are needed:

- GET ENTRY ID (title) returns the internal SS entry ID to the defined title.
- SELECT SS ENTRY (ID) indicates if the SS entry to the defined ID exists or not.

Figure 5-18: Exist SS Entry Operation



5.10.4 Secure Storage PIN Protection

Before a SS command can be performed, PIN verification has to be performed towards the SS applet. The SS applet allows the execution of an APDU command only after a successful PIN verification. Therefore the SS applet stores an internal PIN verified state for each logical channel. Thus an established logical channel has to be authorized with PIN verification before this channel can be used for the SecureStorageService. The internal PIN verified state for a channel is kept until this channel is closed. If a SS APDU command is used without being authorized with a PIN, the SS applet has to return an error code indicating the missing privilege.

To provide the SS access restriction based on PIN authentication, the SS applet must provide the [ISO 7816-4] commands VERIFY, CHANGE REFERENCE DATA, and RESET RETRY COUNTER to support the following Authentication Service methods: `verifyPin(), changePin(), resetPin().`

The SS can also provide other PIN related [ISO 7816-4] commands such as DISABLE VERIFICATION REQUIREMENT and ENABLE VERIFICATION REQUIREMENT to allow extended PIN management with the Authentication Service methods `deactivatePin()` and `activatePin()`.

6 Minimum Set of Functionality

Applications running on other environments often use the Secure Element as a means of enhancing system security. As a result, any mobile device compliant with the Open Mobile API SHOULD provide access to all SEs on the device. The Transport API is therefore mandatory for Open Mobile API compliant devices.

The most common SEs today are SIM cards, smart SD cards, and embedded SEs, but new SEs may emerge in the future. The SE provider interface is therefore mandatory, to ensure that the device can support future SEs.

The Transport API SHALL support the maximum number of extended logical channels according to [ISO 7816-4] (19 logical channels in addition to the basic channel).

In case the ATR is not available, or the ATR is available and indicates support of 8 or more channels (including the basic channel), the API SHALL try to open logical channels, provided no error is indicated.

In case the ATR is available and indicates support of 7 or less channels (including the basic channel), the API SHALL manage as many logical channels as indicated by the ATR, either by using the ATR information or by opening logical channels, provided no error is indicated.

The Transport API SHALL support extended length APDU commands with a length of at least 2048 bytes.[2]

[2] Developers using OMAPI should be aware that not all Secure Elements support extended length APDU commands.

7 Plug-ins

Section 2 defines plug-ins as providing an interface abstraction between the core functionality of the Open Mobile API and low-level mechanisms for communication with Secure Elements.

There is a plug-in instance associated with each `Reader`.

The role of the plug-in is to abstract platform-specific behavior from the core functionality of the Open Mobile API specification, as described in this document, so that the core functionality is independent of the hardware on which it executes.

In practice, this may include some or all of the following:

- Power management for the Secure Element.
- Handling lower-layer protocol implementation details (e.g. T=0 or T=1 protocol for communication with the Secure Element – this is completely abstracted by the Open Mobile API)
- Interfacing with device drivers for physical interfaces (e.g. SPI, I^2C, APDU gate in an HCI network) to a Secure Element.
- Communicating with a modem abstraction layer such as a RIL, when the Secure Element is a UICC. Since different modem implementations behave differently, this might include handling for the `expectDataWithWarningSW` attribute.
- Supporting “Plug-and-Play” mechanisms that provide a way to add and remove drivers for SEs at runtime, by installing or removing downloadable application and/or driver packages.

The API of such a plug-in interface might be implementation-specific or might be defined in a Platform Binding document for a specific programming language or software platform.

The following aspects SHOULD be considered in the concrete definition of a plug-in interface:

- The implementation SHOULD enforce that it is only accessible from the Transport Layer and not directly from applications, to ensure that the channel management and security mechanism in the Transport Layer cannot be bypassed.
- The plug-in API SHOULD allow different types of Secure Element to interface with the same Open Mobile API Transport Layer. Typical plug-in APIs MAY include
  - The API called when a terminal connects to the Secure Element.
  - The API called when a terminal disconnects from the Secure Element.
  - The API called when a logical channel is opened and/or closed.
  - The API called when the basic channel is opened.
  - The API called when APDUs are transceived.
  - The API called when a plug-in is loaded and/or unloaded.
- The implementation SHOULD ensure that built-in plug-ins cannot be replaced by dynamically loaded plug-ins.

If the plug-in API is intended to support plug-in authoring by third parties, it is recommended that a reference plug-in implementation be provided.

8 Access Control

The GlobalPlatform Secure Element Access Control specification [SEAC]) defined a mechanism to prevent non-authorized device applications from accessing SE applications. Secure Element access rules are stored in the SE and read and applied by the Access Control Enforcer that is implemented in the Transport layer of the Open Mobile API.

When a channel is opened, the OMAPI implementation SHALL update the Access Control rules on the device in compliance with [SEAC].

The APIs in this specification will indicate errors (e.g. declaring security errors) when access control rules are not met.