GlobalPlatform Technology
UICC API for Internet of Things
Version 0.0.0.17

Public Review

April 2020

Document Reference: GPP_SPE_008

Contents

Figures

Tables

1 Introduction

This document specifies a low-level API for the minimum UICC functionality needed in low-cost constrained IoT devices. This includes functions for network attachment and functions for cellular subscription profile download and management. To reduce the cost of a constrained IoT device, the UICC functionality is likely to be an integrated part of the application processor or the modem of the IoT device. The realization of the UICC functionality may vary with the device vendor or modem vendor. The API hides the realization and location of the UICC functionality from its users, even when a dedicated component is used to realize the UICC functionality.

1.1 Audience

This specification is suitable for software developers implementing applications or software components that interact with the component offering the UICC functionality; examples include the cellular network, the EAP module, and the assistant for profile management operations. This specification is also suitable for manufacturers of the UICC functionality that may provide a hardware abstraction layer to connect the interface to their realization of the UICC functionality.

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://globalplatform.org/specifications/ip-disclaimers/. 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

The table below lists references applicable to this specification. The latest version of each reference applies unless a publication date or version is explicitly stated.

Table 1-1: Normative References

Standard / Specification	Description	Ref
GPD_SPE_075	GlobalPlatform Technology Open Mobile API Specification	[OMAPI]
3GPP TS 31.102	3rd Generation Partnership Project; Technical Specification Group Core Network and Terminals; Characteristics of the Universal Subscriber Identity Module (USIM) application (Release 15)	[31.102]
3GPP TS 33.102	3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Security architecture (Release 15)	[33.102]
3GPP TS 33.401	3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3GPP System Architecture Evolution (SAE); Security architecture (Release 15)	[33.401]

Standard / Specification	Description	Ref
3GPP TS 33.501	3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Security architecture and procedures for 5G system (Release 15)	[33.501]
ISO/IEC 9899:1999	Programming languages – C	[C99]
IETF RFC 5448	Arkko, J., Eronen, P., “Improved Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA')”, IETF specification, May 2009, https://tools.ietf.org/html/rfc5448	[RFC 5448]
GSMA SGP.22	GSMA Remote SIM Provisioning (RSP) Technical Specification v2.2.1, December 2018	[SGP.22]
IETF RFC 2119	Key words for use in RFCs to Indicate Requirement Levels	[RFC 2119]
BSI-CC-PP-0084-2014	Security IC Platform BSI Protection Profile 2014 with Augmentation Packages.	[PP-0084]

1.4 Terminology and Definitions

The following meanings apply to SHALL, SHALL NOT, MUST, MUST NOT, SHOULD, SHOULD NOT, 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** and **SHOULD NOT** indicate recommendations.
- **MAY** indicates an option.

Selected terms used in this document are included in Table 1-2.

Table 1-2: Terminology and Definitions

Term	Definition
Application Programming Interface (API)	A set of rules that software programs can follow to communicate with each other.
Regular Execution Environment (REE)	An Execution Environment comprising at least one Regular OS and all other components of the device (SoCs, other discrete components, firmware, and software) which execute, host, and support the Regular OS (excluding any Secure Components included in the device). From the viewpoint of a Secure Component, everything in the REE is considered untrusted, though from the Regular OS point of view there may be internal trust structures. (Formerly referred to as a *Rich Execution Environment (REE)*.) Contrast *Trusted Execution Environment (TEE)*.
Regular OS	An OS executing in a Regular Execution Environment. May be anything from a large OS such as Linux down to a minimal set of statically linked libraries providing services such as a TCP/IP stack. (Formerly referred to as a *Rich OS* or *Device OS*.) Contrast *Trusted OS*.

Term	Definition
Secure Component	GlobalPlatform terminology to represent either a Secure Element or a Trusted Execution Environment.
Secure Element (SE)	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	See *UICC session*.
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 ([PP-0084]) including resistance to physical tampering scenarios described in that Protection Profile.
Trusted Execution Environment (TEE)	An Execution Environment that runs alongside but isolated from an REE. A TEE has security capabilities and meets certain security-related requirements: It protects TEE assets against a set of defined threats which include general software attacks as well as some hardware attacks, and defines rigid safeguards as to data and functions that a program can access. There are multiple technologies that can be used to implement a TEE, and the level of security achieved varies accordingly. Contrast *Regular Execution Environment (REE)*.
Trusted OS	An OS executing in a Secure Component. Contrast *Regular OS*.
Trusted Platform Service (TPS)	A service provided by a Secure Component to applications running on a REE.
UICC API	A low-level API for the minimum UICC functionality needed in low-cost constrained IoT devices. Includes functions for network attachment and functions for cellular subscription profile download and management. Hides the realization and location of the UICC functionality from its users.
UICC client	An entity that uses the UICC API to access services provided by a UICC device.
UICC device	A trusted environment that provides UICC functionality for use by one or more UICC clients. The term UICC device here includes integrated UICC (iUICC), embedded UICC (eUICC), removable UICC, and Smart Secure Platform (SSP).
UICC device context	An abstraction of the logical connection that exists between a UICC client and a UICC device. Used by the API to route incoming function calls toward the correct UICC device.
UICC session	Represents a logical connection between a UICC client and a specific UICC device. Sessions are established within initialized UICC device contexts. Multiple sessions are possible within an initialized UICC device context.

1.5 Abbreviations and Notations

Table 1-3: Abbreviations and Notations

Abbreviation / Notation	Meaning
3GPP	3rd Generation Partnership Project
AKA	Authentication and Key Agreement
API	Application Programming Interface
BPP	Bound Profile Package
CIPKID	Certificate Issuer Public Key Identifier
CK	Ciphering Key
CRT	Control Reference Template
EAP	Extensible Authentication Protocol
eSIM	Embedded SIM
ETSI	European Telecommunications Standards Institute
eUICC	Embedded UICC
GSMA	Global System for Mobile Communications Association
HAL	Hardware Abstraction Layer
ICCID	Integrated Circuit Card ID
IK	Integrity Key
IMSI	International Mobile Subscriber Identity
IoT	Internet of Things
ISD-P	Issuer Security Domain Profile
iSIM	Integrated SIM
iSSP	Integrated SSP
iUICC	Integrated UICC
MNC	Mobile Network Code
NAA	Network Access Application
NB-IoT	Narrowband IoT
PCB	Printed Circuit Board
PE	Profile Element
PLMN	Public Land Mobile Network
PPR	Profile Policy Rule
RAT	Rules Authorization Table
REE	Regular Execution Environment
RSP	Remote SIM Provisioning

Abbreviation / Notation	Meaning
SIM	Subscriber Identity Module
SM-DP+	Subscription Manager Data Preparation
SM-DS	Subscription Manager Discovery Server
SoC	System-on-Chip
SSP	Smart Secure Platform
TEE	Trusted Execution Environment
TPS	Trusted Platform Service(s)
UICC	Universal Integrated Circuit Card
UPP	Unprotected Profile Package

1.6 Revision History

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
June 2019	0.0.0.4	Committee Review
December 2019	0.0.0.8	Member Review
April 2020	0.0.0.17	Public Review
TBD	1.0	Public Release

2 Overview

This document specifies a low-level API for the UICC functionality needed for network connectivity in low-cost constrained IoT devices. This is primarily cellular network connectivity (e.g. NB-IoT). The 3GPP Authentication and Key Agreement (AKA) is a generic lightweight technology for the authentication of subscribers, which may be used for cellular network connectivity as well as over other radio technologies such as WiFi and Bluetooth. For details about AKA, see 3rd Generation Partnership Project specifications [33.102], [33.401], and [33.501].

Historically, network authentication has relied on plastic SIM cards, also known as UICC, containing subscriber credentials and algorithm implementation, but this solution has some drawbacks for IoT including costs associated with manually changing SIM cards when swapping operators and the increased cost and size of the physical device due to inclusion of a SIM card reader. The embedded SIM technology in use today allows remote provisioning of subscriber credentials to an embedded UICC (eUICC), a dedicated chip soldered into the device during manufacture. Using an eUICC reduces the size compared to a SIM card, but the cost of using eUICC may still be too high for low-cost constrained IoT devices; a dedicated chip still adds to the required PCB size and may increase power consumption. Another possible solution for low-cost constrained IoT devices is an integrated SIM (iSIM) solution where the SIM functionality, including the remote provisioning support, is implemented in a trusted environment within the System-on-Chip. This is referred to as an integrated UICC (iUICC). Currently, there is no clear definition of iUICC and one might expect several different hardware vendor specific realizations. This API hides the realization and location of the UICC functionality; provides interoperability between devices, modems, and iUICC vendors; and allows easy integration of components from different vendors.

This API implements the bare minimum of what is needed to handle network access authentication and profile management. Limiting the UICC functionality in this way minimizes the profile packages that are downloaded to the device, which saves storage space on the IoT device and reduces the device's energy consumption.

The API in this specification is referred to as the "UICC API" and is applicable to:

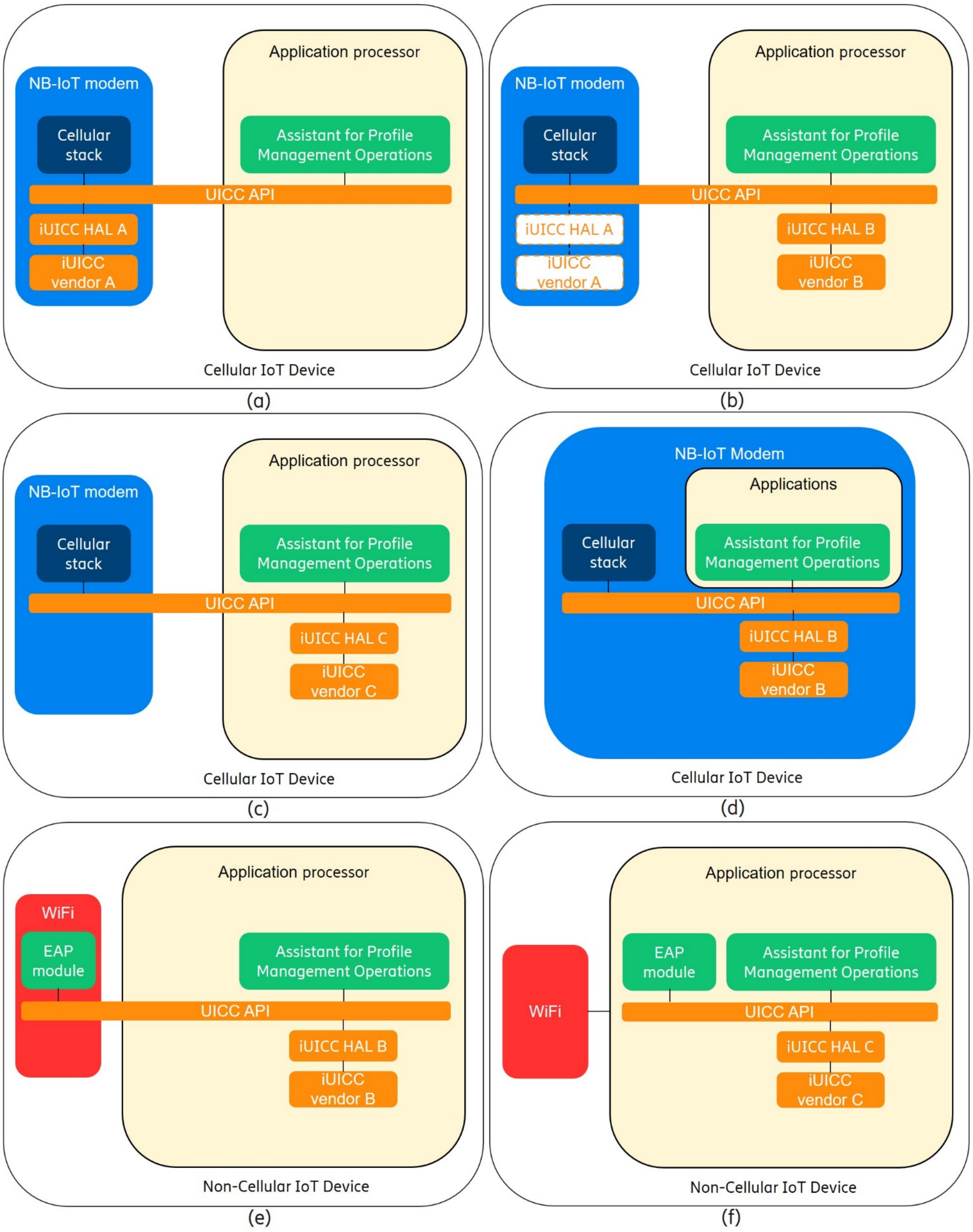
- Integrated UICC realizations
- Non-integrated UICC solutions such as eUICC and removable UICC
- ETSI-defined Smart Secure Platform (SSP) with a telecom secondary platform bundle supporting UICC functionality

Note that for SSP there are also different realization options; for example, there is an integrated SSP (iSSP) option where the SSP is integrated into the System-on-Chip.

2.1 Users of the UICC API

Users of this API include network stacks and assistants for profile management operations. Some examples of users located at different subsystems are shown in Figure 2-1.

Figure 2-1: Example iUICC Deployments Illustrating Location of iUICC and UICC API Users



2.1.1 Network Stack

In the case of a cellular IoT device, the cellular network stack, typically located as part of the modem processor or subsystem (refer to Figure 2-1 (a) – (d)), uses the UICC API to retrieve IMSI and network related configuration data, and to perform the 3GPP network Authentication and Key Agreement (AKA) computations using the cellular subscription credentials stored in the UICC device. (For more information on AKA, refer to [33.102], [33.401], and [33.501].)

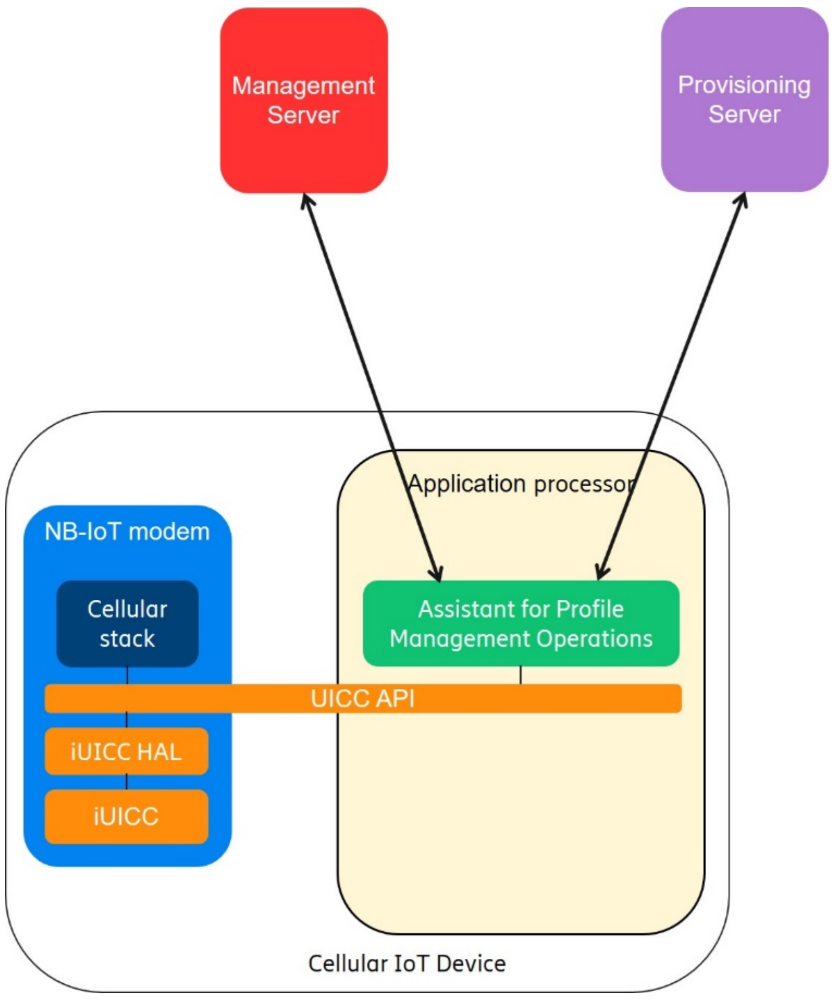
Alternatively, the Extensible Authentication Protocol is used during network access authentication. In this case an implementation of the EAP-AKA algorithm, typically part of an EAP module, is the user of the API to retrieve IMSI and perform the AKA algorithm. (For more information on EAP-AKA, refer to [RFC 5448].) The use of EAP-AKA is part of 3GPP 5G standardization (refer to [33.501]) and is also the likely way that network authentication using UICC functionality will be performed for non-3GPP IoT devices connecting via, for example, WiFi. The EAP module may be part of the cellular modem or WiFi modem subsystem (see Figure 2-1 (e)), or it may be part of the application processor OS (see Figure 2-1 (f)).

2.1.2 Assistant for Profile Management

In order to support the download of new subscriber credentials and management of already downloaded subscriber credentials, there is typically an assistant running in the IoT device handling interactions with provisioning and management servers. An example setup is shown in Figure 2-2. The provisioning server is responsible for the profile download and the management server is controlling the assistant running in the IoT device on behalf of the IoT device owner (e.g. an enterprise). The management server may, for example, provide details needed by the assistant to contact the provisioning server to download a profile. This may include the address of the provisioning server and a token granting permission for profile download. Profile management operations may also originate from the management server. The management server may be a dedicated server for connectivity management or it may be a combined device and connectivity management server. Typically, the management is configured to manage hundreds or thousands of identical IoT devices belonging to the same enterprise. Secure communication is established between the assistant and both the provisioning server and the management server. The specification of the assistant and its secure interaction with the two servers is out of scope of this document.

The assistant for profile management is typically part of the application processor, as shown in Figure 2-1. It is likely to be a rather standard component that may even be an integrated part of the application processor OS.

Figure 2-2: Example of Assistant for Profile Management and Its Interactions with Servers



2.2 Profile Download and Profile Management

For profile management, the API follows the GSMA Remote SIM Provisioning (RSP) protocol, particularly the variant for consumer devices. This variant has a more mature architecture than the Machine-to-Machine variant and, through the assistant for profile management (Local Profile Assistant in GSMA RSP terminology), it allows better integration with the existing native protocol stack of the IoT device. The current version of the API reflects the GSMA RSP consumer variant version 2.2.1.

Editor’s note: To be updated to GSMA RSP consumer variant v3.0 when released.

The consumer variant involves interaction with the user of the consumer devices. For an IoT device, the user of the IoT device typically maps to the owner of the IoT device, and interactions are handled via the management server as described in section 2.1.2.

2.3 Network Attachment

The network attachment part of the API includes support for the 3GPP Authentication and Key Agreement (AKA) algorithm (refer to [33.102], [33.401], and [33.501]) and functions to retrieve IMSI and network related configuration data. The minimum required network configuration information supported by a UICC realization for use in cellular and non-cellular IoT devices is described in section 4.3.6.

3 Principles and Concepts

This section details the underlying principles and concepts of the UICC API.

3.1 Design Principles

The key design principles of the API are:

- **C language**

  C is the common denominator for practically all the application frameworks and operating systems hosting applications that would use the UICC API.

- **Blocking functions**

  Most application developers are familiar with synchronous functions in which the calling code is blocked until the underlying task is completed. A synchronous interface is generally easier to use and it is assumed that multi-threading support is available on target platforms so that cancellation of blocking functions is possible.

- **Source-level portability**

  To enable compile-time and design-time optimization, this standard places no requirement on binary compatibility. Application developers will need to recompile their code against an appropriate implementation-defined version of the API headers in order to function correctly on that implementation.

3.2 Concepts

This section introduces key concepts and terminology.

3.2.1 UICC Device

A UICC device is a trusted environment that provides UICC functionality for use by one or more UICC clients. The term UICC device here includes integrated UICC (iUICC), embedded UICC (eUICC), removable UICC, and Smart Secure Platform (SSP).

3.2.2 UICC Client

A UICC client is an entity that uses the UICC API to access services provided by a UICC device. The UICC API allows a UICC client to use services from one or more UICC devices.

Note: For constrained IoT devices it is unlikely that there is more than one UICC device active at a time.

3.2.3 UICC API Contexts and Sessions

A UICC device context is an abstraction of the logical connection that exists between a UICC client and a UICC device. The UICC device context is used by the API to route incoming function calls toward the correct UICC device.

A UICC client needs to first establish a UICC device context with a UICC device and may then open one or more sessions with the UICC device. A session context is established for each session to allow a UICC client to keep track of the different sessions.

The allowed number of concurrent sessions a UICC device may have with different UICC clients is implementation-defined, depends on the design of the UICC device, and may depend on runtime resource constraints.

3.3 API Architecture

The API functions are divided into four groups:

- **Common functions for all UICC devices**

  Functions for use by UICC clients to find information about available UICC devices and to select a particular UICC device for use.

- **Generic functions, specific for each UICC device**

  Functions related to session handling and UICC data management.

- **Functions for profile management, specific for each UICC device**

  Functions for profile download and management of downloaded profiles. Support for profile download and profile management is further described in section 2.2.

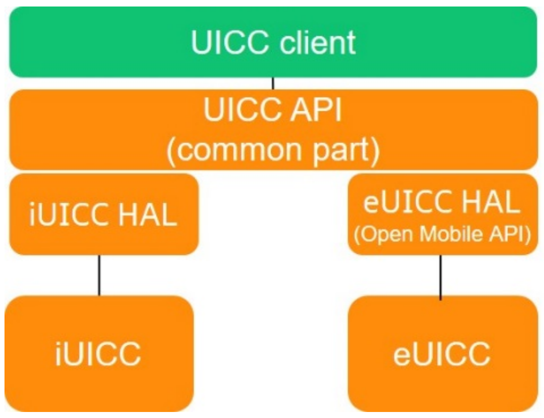
- **Functions related to network attachment, specific for each UICC device**

  Functions for network authentication and for obtaining network related configuration information. Support for network attachment is further described in section 2.3.

An example of API implementation supporting two UICC devices is shown in Figure 3-1, where one UICC device is an iUICC and the other is an eUICC. An API implementation consists of a common part plus a specific part for each UICC device, referred to as a Hardware Abstraction Layer (HAL) in Figure 3-1.

- The common part handles the first of the four function groups described above. The common part of the implementation must also, for the functions in the three other groups, be able to forward the function call to the correct device. This is done by analyzing the session context parameter provided in each function call of the functions belonging to groups 2-4. The way this is done is implementation defined.
- The device-specific parts handle the communication with the UICC device where the implementation of each function in groups 2-4 resides. The communication with the eUICC uses APDUs and may use the Open Mobile API [OMAPI]. The communication with the iUICC does not use APDUs. The device-specific layer may handle, on behalf of several UICC clients, several sessions with the UICC, if multiple sessions are supported. The session context is used by the device-specific part to map an incoming UICC API call to a particular session.

Figure 3-1: Example API Implementation Supporting Two UICC Devices



3.4 Security

The implementation of the UICC API must treat any input from a UICC client as potentially malicious. The UICC devices must assume that UICC clients may be compromised by attack or may be purposefully malicious.

It is assumed that the UICC device is running in a separate operating system, i.e. within a Secure Component, that exists in parallel to the Regular Operating System that runs the UICC clients. It is important that the integration of the UICC device alongside the Regular OS cannot be used to weaken the security of the Regular OS itself. The implementation of the UICC device must ensure that UICC clients cannot use the features they expose to bypass the security sandbox used by the Regular OS to isolate processes.

Securing the communication between a UICC client that is using the API on one subsystem (such as an application processor) and a UICC device located on another subsystem (such as a cellular modem) is implementation specific and out of scope of this API specification.

4 UICC API Definition

This section describes the UICC API.

4.1 Header File

The header file for the UICC API must have the name "`uicc_iot_api.h`".

```
#include "uicc_iot_api.h"
```

4.2 Data Types

4.2.1 Basic Types

This specification makes use of standard C data types, including the fixed width integer types from ISO/IEC 9899:1999 ([C99]). The following standard C types are used:

• `uint32_t`:	a 32-bit unsigned integer
• `uint16_t`:	a 16-bit unsigned integer
• `uint8_t`:	an 8-bit unsigned integer
• `size_t`:	an unsigned integer large enough to hold the size of an object in memory

4.2.2 UICC_Result_t

This type contains the result of invoking a UICC API function. Section 4.3.1 lists result codes defined by this specification.

```
typedef uint32_t UICC_Result_t;
```

4.2.3 UICC_Version_t

This type denotes the version of the UICC API as supported by the implementation.

```
typedef struct
{
  uint8_t major;
  uint8_t minor;
} UICC_Version_t;
```

The structure fields have the following meanings:

- `major` is the major version value.
- `minor` is the minor version value.

The current version is defined in section 4.3.2.

4.2.4 UICC_Buffer_t

This type is used for data buffers.

```
typedef struct
{
  uint8_t *p;
  size_t len;
  size_t allocated;
} UICC_Buffer_t;
```

The structure fields have the following meanings:

- `p` is a pointer to the first byte of the data buffer.
- `len` denotes the number of bytes with valid data in the data buffer.
- `allocated` denotes the number of bytes that have been allocated for the data buffer.

4.2.5 UICC_DataType_t

This type indicates a type of UICC data. Section 4.3.3 lists data types defined by this specification.

```
typedef uint8_t UICC_DataType_t;
```

4.2.6 UICC_ProfileMetadataType_t

This type indicates a type of profile metadata. Section 4.3.4 lists metadata types defined by this specification.

```
typedef uint8_t UICC_ProfileMetadataType_t;
```

4.2.7 UICC_CancelSessionReason_t

This type indicates a reason to cancel a GSMA RSP profile download session (see [SGP.22] for details). Section 4.3.5 lists cancel reasons defined by this specification.

```
typedef uint8_t UICC_CancelSessionReason_t;
```

4.2.8 UICC_FileId_t

This type denotes a file identifier. Section 4.3.6 lists file identifier used by UICC API.

```
typedef uint16_t UICC_FileId_t;
```

4.2.9 UICC_DevCtx_t

This type denotes a UICC device context as described in section 3.2.3.

```
typedef void* UICC_DevCtx_t;
```

4.2.10 UICC_SessCtx_t

This type denotes a session context as described in section 3.2.3.

```
typedef void* UICC_SessCtx_t;
```

4.2.11 UICC_Operation_t

This type indicates an operation to be performed within a particular function. Section 4.3.7 lists notification management operations and section 4.3.10 lists profile management operations defined by this specification.

```
typedef uint8_t UICC_Operation_t;
```

4.2.12 UICC_NotificationsSelection_t

This type denotes a set of notifications to be operated on. The type is a bit mask. Section 4.3.8 defines the individual bits of the bit mask.

```
typedef uint16_t UICC_NotificationsSelection_t;
```

4.2.13 UICC_SequenceNumber_t

This type denotes a sequence number.

```
typedef uint32_t UICC_SequenceNumber_t;
```

4.2.14 UICC_NotificationsParams_t

This type specifies data to be used in a notification management operation.

```
typedef struct
{
  UICC_Operation_t op;
  UICC_NotificationsSelection_t selection;
  UICC_SequenceNumber_t seq_nbr;
} UICC_NotificationsParams_t;
```

The structure fields have the following meanings:

- `op` denotes the notification management operation to be performed. Section 4.3.7 lists operations for managing notifications defined by this specification.
- `selection` denotes the notifications to be returned or operated on. This is a bit mask. Section 4.3.8 defines the individual bits of the bit mask.
- `seq_nbr` denotes the sequence number of a particular notification to be operated on.

4.2.15 UICC_ProfileSearchType_t

This type indicates the type of identifier to be used in searching for one or a set of profiles. Section 4.3.9 lists profile search types defined by this specification.

```
typedef uint8_t UICC_ProfileSearchType_t;
```

4.2.16 UICC_ProfileMgmtOperation_t

This type defines data to be used in a profile management operation.

```
typedef struct
{
  UICC_Operation_t op;
  UICC_ProfileSearchType_t type;
  UICC_Buffer_t buf;
} UICC_ProfileMgmtOperation_t;
```

The structure fields have the following meanings:

- `op` denotes the profile management operation to be performed. Section 4.3.10 lists operations for managing profiles defined by this specification.
- `type` indicates if ICCID or ISD-P AID is used to locate the profile on which the operation is to be performed. Section 4.3.9 lists profile search types defined by this specification.
- `buf` is a data buffer containing the value of the ICCID or ISD-P AID for the profile on which the operation is to be performed.

4.2.17 UICC_InfoList_t

This type indicates a set of profile information. The type is a bit mask. Section 4.3.11 defines the individual bits in the bit mask.

```
typedef uint16_t UICC_InfoList_t;
```

4.2.18 UICC_SearchFilter_t

This type defines data to be used when retrieving profile information.

```
typedef struct
{
  UICC_ProfileSearchType_t type;
  UICC_InfoList_t info_list;
  UICC_Buffer_t buf;
} UICC_SearchFilter_t;
```

The structure fields have the following meanings:

- `type` indicates for what profile or profiles information is to be returned: all available profiles, a set of profiles belonging to a particular profile class, or an individual profile as defined by an ICCID or ISD-P AID. Section 4.3.9 lists profile search types defined by this specification.
- `info_list` indicates what profile information is to be returned for each available profile matching the search filter given by `type` and `buf`.
- `buf` is a data buffer containing the value of the ICCID, ISD-P AID, or profile class for the profile(s) for which profile information is to be returned.

4.2.19 UICC_MemoryReset_t

This type indicates a set of memory reset options. The type is a bit mask. Section 4.3.12 defines the individual bits in the bit mask.

```
typedef uint16_t UICC_MemoryReset_t;
```

4.2.20 UICC_AkaChallengeParams_t

This type defines data to be used in a 3GPP Authentication and Key Agreement (AKA) operation (refer to [33.102] and [33.401]).

```
typedef struct
{
  uint8_t rand[AKA_RAND_LEN];
  uint8_t autn[AKA_AUTN_LEN];
  uint8_t res[AKA_RES_MAX];
  size_t  res_len;
  uint8_t ck[AKA_CK_LEN];
  uint8_t ik[AKA_IK_LEN];
  uint8_t auts[AKA_AUTS_LEN];
} UICC_AkaChallengeParams_t;
```

The structure fields have the following meanings:

- `rand` contains the random challenge RAND of `AKA_RAND_LEN` bytes.
- `autn` contains the authentication token AUTN of `AKA_AUTN_LEN` bytes.
- `res` contains the response RES to the authentication challenge, upon successful execution of the AKA algorithm. The maximum length of RES is `AKA_RES_MAX` bytes and the actual length is given by `res_len`.
- `res_len` denotes the length of `res`, i.e. the number of valid bytes.
- `ck` contains the ciphering key CK of `AKA_CK_LEN` bytes, upon successful execution of the AKA algorithm.
- `ik` contains the integrity key IK of `AKA_IK_LEN` bytes, upon successful execution of the AKA algorithm.
- `auts` contains the synchronization failure parameter AUTS of `AKA_AUTS_LEN` bytes, in case of synchronization error while executing the AKA algorithm.

The lengths of `AKA_RAND_LEN`, etc., are defined in section 4.3.13.

4.3 Constants

4.3.1 Result Codes

The result code of each function defined in this specification is of type `UICC_Result_t` (see section 4.2.2) and is divided in two parts: a status and a value. The following bit masks are defined to indicate the value part, the status part, and individual status bits as defined by this specification.

Table 4-1: Bit Masks for Extracting Parts of the Result Code

Name	Value	Description / Cause
`UICC_VALUE_MASK`	`0x0000FFFF`	Bit mask used to extract value part of the result code.
`UICC_STATUS_MASK`	`0xFF000000`	Bit mask used to extract status part of the result code.
`UICC_STATUS_MASK_ REPLY_DATA_REMAINING`	`0x01000000`	Bit mask indicating that there is remaining reply data that could not be fit into the given input buffer and that SHOULD be obtained by a second separate call to the same function immediately following this call.
`UICC_STATUS_MASK_ NONPROCESSED_DATA_REMAINING`	`0x02000000`	Bit mask indicating that there is non-processed BPP data remaining that MUST be provided in the next call to the same function concatenated with the next BPP part.
Reserved	`0xFC000000`	Bits reserved for future GlobalPlatform use
Reserved	`0x00FF0000`	Bits reserved for implementation-specific use

The following values of the result code are defined by this specification.

Table 4-2: API Result Code Constants

Name	Value	Description / Cause
`UICC_NO_ERROR`	`0`	Successful execution of the function.
`UICC_ARGUMENT_ERROR`	`1`	Error in function argument.
`UICC_INVALID_DEVICE_NBR`	`2`	Invalid device number given as input.
`UICC_TOO_SMALL_BUFFER`	`3`	Input buffer provided is too small for the data to be returned by the function.
`UICC_UNSUPPORTED_TYPE`	`4`	Unsupported type given as input.
`UICC_INVALID_TYPE`	`5`	Invalid type given as input.
`UICC_INVALID_CONTEXT`	`6`	Invalid context given as input.
`UICC_UNSUPPORTED_OPERATION`	`7`	Unsupported operation given as input.
`UICC_INVALID_OPERATION`	`8`	Invalid operation given as input

Name	Value	Description / Cause
`UICC_MAX_SESSIONS_REACHED`	9	Maximum number of allowed sessions already in use for the UICC device linked to the UICC context provided as input.
`UICC_UNSUPPORTED_FUNCTION`	10	Function is not supported by the UICC device.
`UICC_NOT_ENOUGH_MEMORY`	11	Not enough memory is available for the function to complete its operation.
`UICC_AKA_MAC_ERROR`	12	During 3GPP AKA authentication, error in verifying the MAC contained in AUTN.
`UICC_AKA_SYNC_ERROR`	13	Synchronization error in 3GPP AKA authentication.
`UICC_FILE_NOT_FOUND`	14	File is not found.
`UICC_INVALID_FILE`	15	Invalid file.
`UICC_INVALID_TRANS_ID`	16	Invalid transaction identifier. The transaction identifier identifies a profile download and installation transaction (see [SGP.22]).
`UICC_INVALID_OID`	17	Invalid object identifier. The object identifier does not match the already stored object identifier for the ongoing transaction (see [SGP.22]).
`UICC_CHALLENGE_MISMATCH`	18	The challenge in the provided data structure does not match the challenge returned in `UICC_InitiateAuthentication` (see [SGP.22]).
`UICC_INVALID_CERTIFICATE`	19	A certificate in the provided data structure is invalid (see [SGP.22]).
`UICC_UNSUPPORTED_CURVE`	20	The elliptic curve in the provided data structure is not supported (see [SGP.22]).
`UICC_UNSUPPORTED_CIPKID`	21	The requested Certificate Issuer Public Key Identifier is not known/supported (see [SGP.22]).
`UICC_NO_RSP_SESSION_CONTEXT`	22	No GSMA RSP session context, i.e. no ongoing authentication/profile download session (see [SGP.22]).
`UICC_INVALID_SIGNATURE`	23	The provided data structure contains an invalid signature (see [SGP.22]).
`UICC_INCORRECT_INPUT_VALUES`	24	The provided data structure contains incorrect input values (see [SGP.22]).
`UICC_UNSUPPORTED_ REMOTE_OPERATION_TYPE`	25	The remote operation type in the provided data structure is not supported (see [SGP.22]).
`UICC_UNSUPPORTED_CRT_VALUES`	26	The Control Reference Template (CRT) values in the provided data structure are not supported (see [SGP.22]).

Name	Value	Description / Cause
`UICC_ICCID_ALREADY_EXISTS`	27	The ICCID of the provided profile package already exists (see [SGP.22]).
`UICC_INSTALL_FAILED_ PPR_NOT_ALLOWED`	28	Profile installation failed because Profile Policy Rule (PPR) is not allowed according to UICC Rules Authorization Table (RAT) (see [SGP.22]).
`UICC_INSTALL_FAILED_ DATA_MISMATCH`	29	Profile installation failed due to data mismatch in provided profile elements (see [SGP.22]).
`UICC_INSTALL_FAILED_ INSUFFICIENT_MEMORY`	30	Profile installation failed due to insufficient memory (see [SGP.22]).
`UICC_INSTALL_INTERRUPTED`	31	Profile installation was interrupted (see [SGP.22]).
`UICC_INSTALL_FAILED_ PE_PROCESSING_ERROR`	32	Profile installation failed due to Profile Element (PE) processing error (see [SGP.22]).
`UICC_INSTALL_FAILED_ INVALID_NAA_KEY`	33	Profile installation failed due to invalid Network Access Application (NAA) key (see [SGP.22]).
`UICC_UNSUPPORTED_PROFILE_CLASS`	34	The profile class of the provided profile is not supported (see [SGP.22]).
`UICC_SCP03T_STRUCT_ERROR`	35	SCP03T structural error (see [SGP.22]).
`UICC_SCP03T_SEC_ERROR`	36	SCP03T security error (see [SGP.22]).
`UICC_NOTHING_TO_DELETE`	37	Nothing to delete matching the provided input parameters (see [SGP.22]).
`UICC_ERROR_BUSY`	38	Error due to UICC is busy (see [SGP.22]).
`UICC_ICCID_OR_AID_NOT_FOUND`	39	No profile could be found that matches the ICCID or ISD-P AID given as input (see [SGP.22]).
`UICC_PROFILE_NOT_DISABLED`	40	The selected profile is not disabled, which prevents the requested operation (see [SGP.22]).
`UICC_PROFILE_NOT_ENABLED`	41	The selected profile is not enabled, which prevents the requested operation (see [SGP.22]).
`UICC_NOT_ALLOWED_BY_POLICY`	42	The requested operation is not allowed according to Profile Policy Rules (see [SGP.22]).
`UICC_WRONG_PROFILE_REENABLE`	43	The requested enable operation is blocked because the currently enabled profile A (e.g. a test profile) forces a previously enabled profile B to be re-enabled, and the selected profile is not the previously enabled profile B (see [SGP.22]).
Reserved	44 - 99	Reserved for future GlobalPlatform use

Name	Value	Description / Cause
`UICC_LOAD_PROFILE_COMPLETED`	`100`	Profile download and installation completed successfully.
Reserved	`101 – 126`	Reserved for future GlobalPlatform use.
`UICC_UNDEFINED_ERROR`	`127`	Undefined error.
Reserved	`128 – 255`	Reserved for implementation-specific use.
Reserved	`256 – 65535`	Reserved for future GlobalPlatform use.

4.3.2 Version Information

The version of the API described in this document is 1.0; i.e. major version value is 1 and minor version value is 0. (See `UICC_Version_t`, section 4.2.3.)

4.3.3 Data Types

This specification defines the following types of UICC data, `UICC_DataType_t` (see section 4.2.5).

Table 4-3: UICC Data Types

Name	Value	Description / Cause
`UICC_DATATYPE_DEFAULT_SMDP_ADDRESS`	`0`	Default SM-DP+ address (see [SGP.22]) formatted as a UTF8 string
`UICC_DATATYPE_INFO1`	`1`	The eUICCInfo1 data structure (see [SGP.22])
`UICC_DATATYPE_INFO2`	`2`	The eUICCInfo2 data structure (see [SGP.22])
`UICC_DATATYPE_EID`	`3`	EID (identifier of UICC) (see [SGP.22])
`UICC_DATATYPE_ROOT_SMDS_ADDRESS`	`4`	Root SM-DS address (see [SGP.22]) formatted as a UTF8 string
`UICC_DATATYPE_RAT`	`5`	Rules Authorization Table (RAT) (see [SGP.22])
`UICC_DATATYPE_CERTIFICATE_CHAIN`	`6`	Certificate chain starting with UICC certificate followed by its CA certificate, followed by its CA, and so on up until but not including the root CA certificate
`UICC_DATATYPE_UICC_CERTIFICATE`	`7`	UICC certificate
Reserved	`8 – 127`	Reserved for future GlobalPlatform use
Reserved	`128 – 255`	Reserved for implementation-specific use

4.3.4 Profile Metadata Types

This specification defines the following types of profile metadata, `UICC_ProfileMetadataType_t` (see section 4.2.6).

Table 4-4: Profile Metadata Types

Name	Value	Description / Cause
`UICC_PROFILEDATATYPE_NICKNAME`	`0`	Profile nickname
Reserved	`1 – 127`	Reserved for future GlobalPlatform use
Reserved	`128 – 255`	Reserved for implementation-specific use

4.3.5 Cancel Session Reasons

Table 4-5 defines the reasons for cancelling a GSMA RSP profile download session (see [SGP.22]). The `UICC_CancelSessionReason_t` type, defined in section 4.2.7, SHALL take one of these values.

Table 4-5: Cancel Session Reasons

Name	Value	Description / Cause
`UICC_CSREASON_ENDUSER_REJECTION`	`0`	End-user rejection
`UICC_CSREASON_POSTPONED`	`1`	Postponed
`UICC_CSREASON_TIMEOUT`	`2`	Timeout
`UICC_CSREASON_PPR_NOT_ALLOWED`	`3`	PPR not allowed
`UICC_CSREASON_METADATA_MISMATCH`	`4`	Metadata mismatch
`UICC_CSREASON_LOAD_BPP_EXECUTION_ERROR`	`5`	Execution error while loading BPP
Reserved	`6 – 126`	Reserved for future GlobalPlatform use
`UICC_CSREASON_UNDEFINED`	`127`	Undefined reason
Reserved	`128 – 255`	Reserved for implementation-specific use

4.3.6 File Identifiers

The following tables list the file identifiers for network configuration information stored as elementary files. (See section 4.2.8 and section 4.4.22.) The file identifiers are specified as part of the 3GPP USIM application in [31.102].

- For use with a cellular modem, a UICC device MUST support at least the files listed in Table 4-6.
- For use with a 5G cellular modem, a UICC device MUST support the files listed in Table 4-6 and Table 4-7.
- For non-3GPP devices, at least EF_{IMSI} SHALL be supported.

Table 4-6: File Identifiers

Name	File Identifier	Description
EF_{ACC}	`0x6F78`	Contains the assigned access control class(es)
EF_{AD}	`0x6FAD`	Contains administrative data, including whether MNC length is 2 or 3
EF_{HPPLMN}	`0x6F31`	Contains the interval of time between searches for a higher priority PLMN
EF_{IMSI}	`0x6F07`	Contains the IMSI
$EF_{THRESHOLD}$	`0x6F5C`	Contains information about maximum life time of CK and IK
EF_{UST}	`0x6F38`	Contains USIM services table

Table 4-7: 5G File Identifiers

Name	File Identifier	Description
$EF_{SUCI\_Calc\_Info}$	`0x4F07`	Contains information needed by the UICC client for the support of subscription identifier privacy as defined in [33.501]
$EF_{Routing\_Indicator}$	`0x4F0A`	Contains Routing Indicator needed by the UICC client for support of subscription identifier privacy as defined in [33.501]

4.3.7 Notification Management Operations

This specification defines the following operations, `UICC_Operation_t`, for notification management (see section 4.2.11 and section 4.2.14).

Table 4-8: Notification Management Operations

Name	Value	Description / Cause
`UICC_NOTIFICATION_MANAGEMENT_OP_LIST`	`0`	List notification metadata
`UICC_NOTIFICATION_MANAGEMENT_OP_RETRIEVE`	`1`	Retrieve notifications
`UICC_NOTIFICATION_MANAGEMENT_OP_DELETE`	`2`	Delete notification
Reserved	`3 – 127`	Reserved for future GlobalPlatform use
Reserved	`128 – 255`	Reserved for implementation-specific use

4.3.8 Bit Mask Definition for Notifications Selection

This specification defines the following values for the individual bits of the bit mask `UICC_NotificationsSelection_t` (section 4.2.12), which specifies the notifications that management operations will be performed on (see section 4.2.14).

Table 4-9: Bit Mask Definition for Notifications Selection

Name	Value	Description / Cause
`UICC_NOTIFICATION_MASK_PROFILE_INSTALL`	`0x0001`	Notifications for profile installation
`UICC_NOTIFICATION_MASK_PROFILE_ENABLE`	`0x0002`	Notifications for local profile enable operation
`UICC_NOTIFICATION_MASK_PROFILE_DISABLE`	`0x0004`	Notifications for local profile disable operation
`UICC_NOTIFICATION_MASK_PROFILE_DELETE`	`0x0008`	Notifications for local profile delete operation
Reserved	`0x0FF0`	Bits reserved for future GlobalPlatform use
Reserved	`0x7000`	Bits reserved for implementation-specific use
`UICC_NOTIFICATION_MASK_SEQNUMB`	`0x8000`	Notification is selected based on sequence number

4.3.9 Profile Search Types

This specification defines the following profile search types, `UICC_ProfileSearchType_t` (see section 4.2.15), which indicate the type of identifier to be used in searching for a profile or a set of profiles.

Table 4-10: Profile Search Types

Name	Value	Description / Cause
`UICC_PROFILE_SEARCHTYPE_ALL`	`0`	All available profiles
`UICC_PROFILE_SEARCHTYPE_ICCID`	`1`	A profile defined by ICCID
`UICC_PROFILE_SEARCHTYPE_ISDP_AID`	`2`	A profile defined by ISD-P AID
`UICC_PROFILE_SEARCHTYPE_CLASS`	`3`	A set of profiles belonging to a particular profile class
Reserved	`4 - 127`	Reserved for future GlobalPlatform use
Reserved	`128 - 255`	Reserved for implementation-specific use

4.3.10 Profile Management Operations

This specification defines the following operations, `UICC_Operation_t`, for profile management (see section 4.2.11 and section 4.2.16).

Table 4-11: Profile Management Operations

Name	Value	Description / Cause
`UICC_PROFILE_MANAGEMENT_OP_ENABLE`	`0`	Enable profile
`UICC_PROFILE_MANAGEMENT_OP_DISABLE`	`1`	Disable profile
`UICC_PROFILE_MANAGEMENT_OP_DELETE`	`2`	Delete profile
Reserved	`3 - 127`	Reserved for future GlobalPlatform use
Reserved	`128 - 255`	Reserved for implementation-specific use

4.3.11 Bit Mask Definition for Profile Information List

This specification defines the following values for the individual bits of the bit mask `UICC_InfoList_t` (see section 4.2.17), which specifies the profile information to be returned for each profile.

Table 4-12: Bit Mask Definition for Profile Information List

Name	Value	Description / Cause
`UICC_PROFILE_INFOLIST_MASK_ICCID`	`0x0001`	ICCID
`UICC_PROFILE_INFOLIST_MASK_ISDP_AID`	`0x0002`	ISD-P AID
`UICC_PROFILE_INFOLIST_MASK_PROFILE_STATE`	`0x0004`	Profile state (enabled/disabled)
`UICC_PROFILE_INFOLIST_MASK_PROFILE_NICKNAME`	`0x0008`	Profile nickname
`UICC_PROFILE_INFOLIST_MASK_SP_NAME`	`0x0010`	Service provider name
`UICC_PROFILE_INFOLIST_MASK_PROFILE_NAME`	`0x0020`	Profile name
`UICC_PROFILE_INFOLIST_MASK_ICON_TYPE`	`0x0040`	Icon type
`UICC_PROFILE_INFOLIST_MASK_ICON`	`0x0080`	Icon
`UICC_PROFILE_INFOLIST_MASK_PROFILE_CLASS`	`0x0100`	Profile class
`UICC_PROFILE_INFOLIST_MASK_NOTIFICATION_CI`	`0x0200`	Notification configuration information
`UICC_PROFILE_INFOLIST_MASK_PROFILE_OWNER`	`0x0400`	Profile owner
`UICC_PROFILE_INFOLIST_MASK_SMDP_PROP_DATA`	`0x0800`	SM-DP+ proprietary data
`UICC_PROFILE_INFOLIST_MASK_PPR`	`0x1000`	Profile Policy Rules
Reserved	`0xE000`	Bits reserved for future GlobalPlatform use

4.3.12 Bit Mask Definition for Memory Reset

This specification defines the following values for the individual bits of the bit mask `UICC_MemoryReset_t` (see section 4.2.19), which specifies memory reset options.

Table 4-13: Bit Mask Definition for Memory Reset

Name	Value	Description / Cause
`UICC_MEMRST_MASK_DELETE_OPERATIONAL_PROFILES`	`0x0001`	Operational profiles
`UICC_MEMRST_MASK_DELETE_FIELDLOADED_TEST_PROFILES`	`0x0002`	Field loaded test profiles
`UICC_MEMRST_MASK_RESET_DEFAULT_SMDP_ADDRESS`	`0x0004`	Default SM-DP+ address
`UICC_MEMRST_MASK_DELETE_ALL_PROFILES`	`0x0003`	All profiles
Reserved	`0x0FF8`	Bits reserved for future GlobalPlatform use
Reserved	`0xF000`	Bits reserved for implementation-specific use

4.3.13 AKA Parameters Length Definitions

This specification defines the following values for the length of AKA parameters (see section 4.2.20).

Table 4-14: AKA Parameters Length Definitions

Name	Value	Description / Cause
`AKA_AUTN_LEN`	16	Length of AUTN in bytes
`AKA_AUTS_LEN`	14	Length of AUTS in bytes
`AKA_CK_LEN`	16	Length of CK in bytes
`AKA_IK_LEN`	16	Length of IK in bytes
`AKA_RAND_LEN`	16	Length of RAND in bytes
`AKA_RES_MAX`	16	Maximum length of RES in bytes

4.4 Functions

The following sub-sections specify the behavior of the functions within the UICC API.

4.4.1 Documentation Format

```
Function Prototype
```

Description

This topic describes the behavior of the function.

Parameters

This topic describes each function parameter.

Return

This topic lists the possible return values. Note that this list is not comprehensive, and often leaves some choice over error returns to the Implementation. However, if restrictions do exist, then this topic will document them.

Programmer Error

This topic documents cases of programmer error – error cases that MAY be detected by the implementation, but that MAY also perform in an unpredictable manner. This topic is not exhaustive, and does not document cases such as passing an invalid pointer or a NULL pointer where the body text states that the pointer must point to a valid structure.

Implementer Notes

This topic highlights key points about the intended use of the function.

4.4.2 UICC_GetVersion

```
UICC_Result_t UICC_GetVersion(
   UICC_Version_t *version_p)
```

Description

This function returns the version value of the common part of the API, as discussed in section 3.3, API Architecture.

Parameters

- version_p: A pointer to a UICC_Version_t structure (see section 4.2.3) where the version of the API is returned.

Return

- UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_UNDEFINED_ERROR

Programmer Error

None

Implementer Notes

This returns the version value of the common part of the UICC API. A UICC device may support another version of the UICC API; that value can be obtained using the UICC_GetDeviceInfo function. The implementer notes of the UICC_GetDeviceInfo function in section 4.4.4 describe how differences in UICC API versions are handled.

4.4.3 UICC_GetNumberOfDevices

```
UICC_Result_t UICC_GetNumberOfDevices(
   uint8_t *nbr_p)
```

Description

This function returns the number of available UICC devices.

Parameters

- nbr_p: A pointer to a uint8_t where the number of available UICC devices is returned.

Return

- UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_UNDEFINED_ERROR

Programmer Error

None

Implementer Notes

None

4.4.4 UICC_GetDeviceInfo

```
UICC_Result_t UICC_GetDeviceInfo(
   uint8_t nbr,
   UICC_Buffer_t *vendor_p,
   UICC_Version_t *version_p)
```

Description

This function returns information about UICC device `i`, `1 ≤ i ≤ n`, where `n` is the number of devices as returned by `UICC_GetNumberOfDevices`.

Parameters

- `nbr`: The number of the UICC device for which information is requested, where `1 ≤ nbr ≤ n` and where `n` is the number of devices as returned by `UICC_GetNumberOfDevices`. The order of the UICC devices is not guaranteed to be the same following a reset/reboot.
- `vendor_p`: A pointer to a `UICC_Buffer_t` structure where the UICC vendor (and possibly also model) information is given as a UTF8 string. The maximum length of the string is 64 bytes. If the allocated buffer is not large enough to fit the complete string, an error is returned. Upon function return, the buffer length is updated to indicate the length of the string. A call to this function with the buffer pointer set to NULL returns the required minimum length of the buffer to query the information string.
- `version_p`: A pointer to a `UICC_Version_t` structure where the version of the API as supported by the UICC device is returned.

Return

- `UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_INVALID_DEVICE_NBR, UICC_TOO_SMALL_BUFFER, UICC_UNDEFINED_ERROR`

Programmer Error

None

Implementer Notes

UICC device information may be hard coded as part of the UICC API common implementation or the common part may have queried the UICC device itself or its HAL to obtain the information.

The implementation MUST ensure that a UICC client can distinguish two UICC devices of the same model from the same vendor. This can be done by encoding information about the physical link to the UICC device into the vendor string.

Unless explicitly specified, a newer version of the UICC API is intended to be backward compatible and to handle data structures of older versions. To support that compatibility, the version returned by this function MUST NOT exceed the version returned by `UICC_GetVersion`, even if the UICC device handles a later version. Therefore:

- From the UICC client point of view, the UICC device supports the lower version.
- When interacting with the UICC device, the HAL MUST adapt to any differences in the API functions between the two versions. In particular, if the version of the API supported by the UICC device is lower than the common UICC API version, then for any new function that is not present in the older version, the HAL returns `UICC_UNSUPPORTED_FUNCTION`.

4.4.5 UICC_InitializeDeviceContext

```
UICC_Result_t UICC_InitializeDeviceContext(
   uint8_t nbr,
   UICC_DevCtx_t *dctx_p)
```

Description

This function initializes a UICC device context with the selected UICC device. After this function, the context will contain information that supports mapping to the correct device.

Parameters

- `nbr`: The number of the UICC device for which initialization is requested, where `1 ≤ nbr ≤ n` and where `n` is the number of devices as returned by `UICC_GetNumberOfDevices`. The order of the UICC devices is not guaranteed to be the same following a reset/reboot.
- `dctx_p`: A pointer to a `UICC_DevCtx_t` type representing a UICC device context that is initialized by this function.

Return

- `UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_INVALID_DEVICE_NBR, UICC_UNDEFINED_ERROR`

Programmer Error

None

Implementer Notes

The internal structure of the UICC device context is implementation defined and may, for example, serve only as a reference to an internal context kept by the implementation.

4.4.6 UICC_FinalizeDeviceContext

```
UICC_Result_t UICC_FinalizeDeviceContext(
   UICC_DevCtx_t dctx)
```

Description

This function ends communication between a UICC client and the UICC device identified by the provided UICC device context.

Parameters

- `dctx`: A UICC device context of type `UICC_DevCtx_t`.

Return

- `UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_INVALID_CONTEXT, UICC_UNDEFINED_ERROR`

Programmer Error

None

Implementer Notes

The implementation SHALL ensure that a previously valid UICC device context is no longer valid following the call of this function. The implementation SHALL also ensure that any open session connected to this UICC device context is closed and that the corresponding session context is made invalid.

4.4.7 UICC_OpenSession

```
UICC_Result_t UICC_OpenSession(
   UICC_DevCtx_t dctx,
   UICC_SessCtx_t *sctx_p)
```

Description

This function opens a session between a UICC client and the UICC device identified by the provided UICC device context.

Parameters

- `dctx`: A UICC device context of type `UICC_DevCtx_t`.
- `sctx_p`: A pointer to a `UICC_SessCtx_t` type representing a session context.

Return

- `UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_INVALID_CONTEXT, UICC_MAX_SESSIONS_REACHED, UICC_UNDEFINED_ERROR`

Programmer Error

None

Implementer Notes

It is implementation defined whether multiple sessions or only a single session is supported.

It is implementation defined how the link between the session context and the UICC device context is established.

4.4.8 UICC_CloseSession

```
UICC_Result_t UICC_CloseSession(
   UICC_SessCtx_t sctx)
```

Description

This function closes an open session between a UICC client and the UICC device identified by the provided session context.

Parameters

- `sctx`: A session context of type `UICC_SessCtx_t`.

Return

- `UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_INVALID_CONTEXT, UICC_UNDEFINED_ERROR`

Programmer Error

None

Implementer Notes

None

4.4.9 UICC_GetUICCData

```
UICC_Result_t UICC_GetUICCData(
   UICC_SessCtx_t sctx,
   UICC_DataType_t type,
   UICC_Buffer_t *buffer_p)
```

Description

This function returns UICC data needed for profile download and installation. All types specified in section 4.3.3 may be returned.

Parameters

- `sctx`: A session context of type `UICC_SessCtx_t`.
- `type`: Indicates the requested data. Section 4.3.3 lists the supported data types.
- `buffer_p`: A pointer to a `UICC_Buffer_t` structure where the returned data is stored. If the allocated buffer is not large enough to fit the complete data, an error is returned. Upon function return, the buffer length is updated to indicate the length of the returned data. A call to this function with the buffer pointer set to NULL returns the required minimum length of the buffer to query the particular data.

Return

- `UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_INVALID_CONTEXT, UICC_INVALID_TYPE, UICC_TOO_SMALL_BUFFER, UICC_UNDEFINED_ERROR, UICC_UNSUPPORTED_TYPE`

Programmer Error

None

Implementer Notes

None

4.4.10 UICC_SetUICCData

```
UICC_Result_t UICC_SetUICCData(
   UICC_SessCtx_t sctx,
   UICC_DataType_t type,
   UICC_Buffer_t *buffer_p)
```

Description

This function sets UICC data needed for profile download and installation. The following data/structures may be set:

- Default SM-DP+ address

Parameters

- `sctx`: A session context of type `UICC_SessCtx_t`.
- `type`: Indicates the type of data to be written (see section 4.3.3).
- `buffer_p`: A pointer to a `UICC_Buffer_t` structure that contains the data to be written.

Return

- `UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_INVALID_CONTEXT, UICC_INVALID_TYPE, UICC_NOT_ENOUGH_MEMORY, UICC_UNDEFINED_ERROR, UICC_UNSUPPORTED_TYPE`

Programmer Error

None

Implementer Notes

None

4.4.11 UICC_InitiateAuthentication

```
UICC_Result_t UICC_InitiateAuthentication(
   UICC_SessCtx_t sctx,
   UICC_Buffer_t *buffer_p)
```

Description

This function initiates authentication as part of the common mutual authentication procedure specified in [SGP.22]. The function returns the 16-byte UICC challenge.

Parameters

- `sctx`: A session context of type `UICC_SessCtx_t`.
- `buffer_p`: A pointer to a `UICC_Buffer_t` structure where the UICC challenge of length 16 bytes is to be stored. If the allocated buffer is not large enough to fit the complete UICC challenge, an error is returned. Upon function return, the buffer length is updated to indicate the length of the UICC challenge.

Return

- `UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_INVALID_CONTEXT, UICC_TOO_SMALL_BUFFER, UICC_UNDEFINED_ERROR`

Programmer Error

None

Implementer Notes

None

4.4.12 UICC_AuthenticateServer

```
UICC_Result_t UICC_AuthenticateServer(
   UICC_SessCtx_t sctx,
   UICC_Buffer_t *buffer_p)
```

Description

This function authenticates the server as part of the common mutual authentication procedure specified in [SGP.22].

Parameters

- `sctx`: A session context of type `UICC_SessCtx_t`.
- `buffer_p`: A pointer to a `UICC_Buffer_t` structure containing the `AuthenticateServerRequest` structure (see [SGP.22]).
    - Upon function return, the buffer contains `the AuthenticateServerResponse` structure (see [SGP.22]) or, in case of error, the transaction ID of `AuthenticateServerRequest` (if available).
    - If the buffer is not large enough to include the complete `AuthenticateServerResponse` structure, the buffer is filled and the length of the buffer indicates the remaining bytes; note that the buffer `len` parameter may in this case be larger than the `allocated` parameter. In addition, result code status bit `UICC_STATUS_MASK_REPLY_DATA_REMAINING` is set to indicate that remaining data is available. Further calls to this function may then be done to retrieve the remaining bytes.

Return

- `UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_CHALLENGE_MISMATCH, UICC_INVALID_CERTIFICATE, UICC_INVALID_CONTEXT, UICC_INVALID_OID, UICC_INVALID_SIGNATURE, UICC_NO_RSP_SESSION_CONTEXT, UICC_NOT_ENOUGH_MEMORY, UICC_TOO_SMALL_BUFFER, UICC_UNDEFINED_ERROR, UICC_UNSUPPORTED_CIPKID, UICC_UNSUPPORTED_CURVE`
- If remaining reply data is available, the result code values above are ORed with `UICC_STATUS_MASK_REPLY_DATA_REMAINING.`

Programmer Error

None

Implementer Notes

None

4.4.13 UICC_PrepareDownload

```
UICC_Result_t UICC_PrepareDownload(
   UICC_SessCtx_t sctx,
   UICC_Buffer_t *buffer_p)
```

Description

This function prepares profile download as part of the common mutual authentication procedure specified in [SGP.22].

Parameters

- `sctx`: A session context of type `UICC_SessCtx_t`.
- `buffer_p`: A pointer to a `UICC_Buffer_t` structure containing the `PrepareDownloadRequest` structure (see [SGP.22]).
  - Upon function return, the buffer contains the `PrepareDownloadResponse` structure (see [SGP.22]) or, in case of error, the transaction ID (internally stored from previously parsed `AuthenticateServerRequest`).
  - If the buffer is not large enough to include the complete `PrepareDownloadResponse` structure, the buffer is filled and the length of the buffer indicates the remaining bytes; note that the buffer `len` parameter may in this case be larger than the `allocated` parameter. In addition, result code status bit `UICC_STATUS_MASK_REPLY_DATA_REMAINING` is set to indicate that remaining data is available. Further calls to this function may then be done to retrieve the remaining bytes.

Return

- `UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_INVALID_CERTIFICATE, UICC_INVALID_CONTEXT, UICC_INVALID_SIGNATURE, UICC_INVALID_TRANS_ID, UICC_NO_RSP_SESSION_CONTEXT, UICC_NOT_ENOUGH_MEMORY, UICC_TOO_SMALL_BUFFER, UICC_UNDEFINED_ERROR, UICC_UNSUPPORTED_CURVE`
- If remaining reply data is available, the result code values above are ORed with `UICC_STATUS_MASK_REPLY_DATA_REMAINING`.

Programmer Error

None

Implementer Notes

None

4.4.14 UICC_LoadBoundProfilePackage

```
UICC_Result_t UICC_LoadBoundProfilePackage(
   UICC_SessCtx_t sctx,
   UICC_Buffer_t *buffer_p)
```

Description

This function loads and installs the Bound Profile Package (BPP) as specified in [SGP.22]. Since the BPP may be large and delivered in chunks, the function may be called several times. The UICC processes as much as it can from the received BPP part. The length of the buffer is updated to mark how many bytes of the supplied buffer were processed. In the next call to the function, the calling application is responsible to provide the remaining unprocessed bytes concatenated with the next BPP part. The UICC implementation may buffer already decrypted and verified data (e.g. part of the UPP) that cannot yet be processed until more data has been received.

Parameters

- `sctx`: A session context of type `UICC_SessCtx_t`.
- `buffer_p`: A pointer to a `UICC_Buffer_t` structure containing the BPP structure (see [SGP.22]).
  - Upon complete processing of the BPP, the buffer contains the `ProfileInstallationResult` structure (see [SGP.22]) at function return.
  - If the full BPP has not yet been provided and non-processed BPP data remains from the BPP chunk provided in this call, the length of the buffer is updated with the number of BPP bytes processed in this call and result code status bit `UICC_STATUS_MASK_NONPROCESSED_DATA_REMAINING` is set.
  - In case of error the buffer may or may not, depending on the type of error, contain the `ProfileInstallationResult` structure.
  - If the buffer is not large enough to include the complete `ProfileInstallationResult` structure, the buffer is filled and the length of the buffer indicates the remaining bytes; note that the buffer `len` parameter may in this case be larger than the `allocated` parameter. In addition, result code status bit `UICC_STATUS_MASK_REPLY_DATA_REMAINING` is set to indicate that remaining data is available. Further calls to this function may then be done to retrieve the remaining bytes.

Return

- `UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_ICCID_ALREADY_EXISTS, UICC_INCORRECT_INPUT_VALUES, UICC_INSTALL_FAILED_DATA_MISMATCH, UICC_INSTALL_FAILED_INSUFFICIENT_MEMORY, UICC_INSTALL_FAILED_INVALID_NAA_KEY, UICC_INSTALL_FAILED_PE_PROCESSING_ERROR, UICC_INSTALL_FAILED_PPR_NOT_ALLOWED, UICC_INSTALL_INTERRUPTED, UICC_INVALID_CONTEXT, UICC_INVALID_SIGNATURE, UICC_INVALID_TRANS_ID, UICC_LOAD_PROFILE_COMPLETED, UICC_NOT_ENOUGH_MEMORY, UICC_SCP03T_SEC_ERROR, UICC_SCP03T_STRUCT_ERROR, UICC_TOO_SMALL_BUFFER, UICC_UNDEFINED_ERROR, UICC_UNSUPPORTED_CRT_VALUES, UICC_UNSUPPORTED_PROFILE_CLASS, UICC_UNSUPPORTED_REMOTE_OPERATION_TYPE`
- If remaining reply data is available, the result code values above are ORed with `UICC_STATUS_MASK_REPLY_DATA_REMAINING`.

Programmer Error

None

Implementer Notes

None

4.4.15 UICC_CancelSession

```
UICC_Result_t UICC_CancelSession(
   UICC_SessCtx_t sctx,
   UICC_CancelSessionReason_t reason,
   UICC_Buffer_t *buffer_p)
```

Description

This function cancels an ongoing RSP session (see [SGP.22]).

Parameters

- `sctx`: A session context of type `UICC_SessCtx_t`.
- `reason`: Indicates the reason for cancelling the session. Section 4.3.5 lists the supported reasons.
- `buffer_p`: A pointer to a `UICC_Buffer_t` structure containing the transaction ID (see [SGP.22]).
    - Upon successful function return, the buffer contains the `CancelSessionResponse` structure (see [SGP.22]).
    - If the buffer is not large enough to include the complete `CancelSessionResponse` structure, the buffer is filled and the length of the buffer indicates the remaining bytes; note that the buffer `len` parameter may in this case be larger than the `allocated` parameter. In addition, result code status bit `UICC_STATUS_MASK_REPLY_DATA_REMAINING` is set to indicate that remaining data is available. Further calls to this function may then be done to retrieve the remaining bytes.

Return

- `UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_INVALID_CONTEXT, UICC_INVALID_TRANS_ID, UICC_NOT_ENOUGH_MEMORY, UICC_TOO_SMALL_BUFFER, UICC_UNDEFINED_ERROR`
- If remaining reply data is available, the result code values above are ORed with `UICC_STATUS_MASK_REPLY_DATA_REMAINING`.

Programmer Error

None

Implementer Notes

None

4.4.16 UICC_ManageNotifications

```
UICC_Result_t UICC_ManageNotifications(
   UICC_SessCtx_t sctx,
   UICC_NotificationsParams_t *params_p,
   UICC_Buffer_t *buffer_p)
```

Description

This function is used to manage notifications. The following operations are supported:

- Get a list of one or more metadata for pending notifications.
- Retrieve one or more pending notifications.
- Remove a particular notification from the list of notifications.

If notifications are not supported by the UICC Device, this function returns `UICC_UNSUPPORTED_FUNCTION.`

Parameters

- `sctx`: A session context of type `UICC_SessCtx_t`.
- `params_p`: A pointer to a `UICC_NotificationsParams_t` structure containing parameters related to management of notifications. This structure is described in section 4.2.14.
- `buffer_p`: A pointer to a `UICC_Buffer_t` structure where the returned data is stored.
  - o If the operation is to get a list of notification metadata, the returned data is a sequence of one or more `NotificationMetadata` structures (see [SGP.22]).
  - o If the operation is to retrieve notifications, the returned data is a sequence of one or more `PendingNotification` structures (see [SGP.22]).
  - o For the delete operation, no data is returned.

  If the allocated buffer is not large enough to fit the complete data, an error is returned. Upon function return, the buffer length is updated to indicate the length of the returned data. A call to this function with the buffer pointer set to NULL returns the required minimum length of the buffer to query the particular data.

Return

- `UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_INVALID_CONTEXT, UICC_INVALID_OPERATION, UICC_NOT_ENOUGH_MEMORY, UICC_NOTHING_TO_DELETE, UICC_TOO_SMALL_BUFFER, UICC_UNDEFINED_ERROR, UICC_UNSUPPORTED_FUNCTION, UICC_UNSUPPORTED_OPERATION`

Programmer Error

None

Implementer Notes

None

4.4.17 UICC_ManageProfile

```
UICC_Result_t UICC_ManageProfile(
   UICC_SessCtx_t sctx,
   UICC_ProfileMgmtOperation_t *mgmt_p)
```

Description

This function is used for management of profiles. The following operations are supported: enable profile, disable profile, and delete profile.

Parameters

- `sctx`: A session context of type `UICC_SessCtx_t`.
- `mgmt_p`: A pointer to a `UICC_ProfileMgmtOperation_t` structure containing parameters related to management of profiles. This structure is described in section 4.2.16.

Return

- `UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_ERROR_BUSY, UICC_ICCID_OR_AID_NOT_FOUND, UICC_INVALID_CONTEXT, UICC_INVALID_OPERATION, UICC_INVALID_TYPE, UICC_NOT_ALLOWED_BY_POLICY, UICC_PROFILE_NOT_DISABLED, UICC_PROFILE_NOT_ENABLED, UICC_TOO_SMALL_BUFFER, UICC_UNDEFINED_ERROR, UICC_UNSUPPORTED_OPERATION, UICC_UNSUPPORTED_TYPE, UICC_WRONG_PROFILE_REENABLE`

Programmer Error

None

Implementer Notes

None

4.4.18 UICC_GetProfilesInfo

```
UICC_Result_t UICC_GetProfilesInfo(
   UICC_SessCtx_t sctx,
   UICC_SearchFilter_t *filter_p,
   UICC_Buffer_t *buffer_p)
```

Description

This function is used to retrieve information about available profiles. Information may be requested for all available profiles, all profiles of a given class, or a single profile identified by an ICCID or ISD-P AID. The set of information requested per profile is specified using the `filter_p` parameter.

Parameters

- `sctx`: A session context of type `UICC_SessCtx_t`.
- `filter_p`: A pointer to a `UICC_SearchFilter_t` structure specifying the information to be returned for each profile, and identifying the profile(s) for which the information is to be returned. This structure is described in section 4.2.18.
- `buffer_p`: A pointer to a `UICC_Buffer_t` structure that contains the `ProfileInfoListResponse` structure (see [SGP.22]). If the allocated buffer is not large enough to fit the complete data, an error is returned. Upon function return, the buffer length is updated to indicate the length of the returned data. A call to this function with the buffer pointer set to NULL returns the required minimum length of the buffer to query the particular data.

Return

- `UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_INVALID_CONTEXT, UICC_INVALID_OPERATION, UICC_NOT_ENOUGH_MEMORY, UICC_TOO_SMALL_BUFFER, UICC_UNDEFINED_ERROR, UICC_UNSUPPORTED_OPERATION`

Programmer Error

None

Implementer Notes

None

4.4.19 UICC_SetProfileMetadata

```
UICC_Result_t UICC_SetProfileMetadata(
   UICC_SessCtx_t sctx,
   UICC_Buffer_t *iccid_p,
   UICC_ProfileMetadataType_t type,
   UICC_Buffer_t *buffer_p)
```

Description

This function sets profile metadata.

Parameters

- `sctx`: A session context of type `UICC_SessCtx_t`.
- `iccid_p`: A pointer to a `UICC_Buffer_t` structure that contains the ICCID of the profile for which the profile metadata is to be set.
- `type`: Indicates the type of profile metadata to be written (see section 4.3.4).
- `buffer_p`: A pointer to a `UICC_Buffer_t` structure that contains the data to be written.

Return

- `UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_ICCID_OR_AID_NOT_FOUND, UICC_INVALID_CONTEXT, UICC_INVALID_TYPE, UICC_NOT_ENOUGH_MEMORY, UICC_UNDEFINED_ERROR, UICC_UNSUPPORTED_TYPE`

Programmer Error

None

Implementer Notes

None

4.4.20 UICC_MemoryReset

```
UICC_Result_t UICC_MemoryReset(
   UICC_SessCtx_t sctx,
   UICC_MemoryReset_t options)
```

Description

This function is used to delete several profiles (e.g. all operational profiles) according to the input parameter `options`.

Parameters

- `sctx`: A session context of type `UICC_SessCtx_t`.
- `options`: This parameter of type `UICC_MemoryReset_t` (see section 4.2.19) indicates what profiles and other data structures should be deleted. It is a bit mask; section 4.3.12 describes the specification of the individual bits.

Return

- `UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_ERROR_BUSY, UICC_INVALID_CONTEXT, UICC_NOTHING_TO_DELETE, UICC_UNDEFINED_ERROR`

Programmer Error

None

Implementer Notes

None

4.4.21 UICC_Authenticate

```
UICC_Result_t UICC_Authenticate(
   UICC_SessCtx_t sctx,
   UICC_AkaChallengeParams_t *challenge_p)
```

Description

This function performs the 3GPP Authentication and Key Agreement (AKA) algorithm. It authenticates the network and computes the response (RES) on the received challenge used in authentication of the device. The function also returns the ciphering key (CK) and integrity key (IK) for use in protection of network signaling and user data between the device and the network. For more information on 3GPP AKA, refer to [33.102] and [33.401].

Parameters

- `sctx`: A session context of type `UICC_SessCtx_t`.
- `challenge_p`: A pointer to the `UICC_AkaChallengeParams_t` structure (see section 4.2.20) containing input and output parameters for the AKA algorithm.

Return

- `UICC_NO_ERROR`, `UICC_AKA_MAC_ERROR`, `UICC_AKA_SYNC_ERROR`, `UICC_ARGUMENT_ERROR`, `UICC_INVALID_CONTEXT`, `UICC_UNDEFINED_ERROR`

Programmer Error

None

Implementer Notes

None

4.4.22 UICC_ReadFile

```
UICC_Result_t UICC_ReadFile(
   UICC_SessCtx_t sctx,
   UICC_FileId_t id,
   UICC_Buffer_t *buffer_p)
```

Description

This function reads an elementary file containing configuration data relevant for the modem/device. This function only permits reading elementary files of the USIM application of the active profile.

Parameters

- `sctx`: A session context of type `UICC_SessCtx_t`.
- `id`: File identifier according to `UICC_FileId_t` structure (see section 4.2.8). Section 4.3.6 lists the file identifiers of the minimum set of files required to be supported by a UICC device.
- `buffer_p`: A pointer to a `UICC_Buffer_t` structure where the returned file data is stored. If the allocated buffer is not large enough to fit the complete file data, an error is returned. Upon function return, the buffer length is updated to indicate the length of the returned data. A call to this function with the buffer pointer set to NULL returns the required minimum length of the buffer to query the particular data.

Return

- `UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_FILE_NOT_FOUND, UICC_INVALID_CONTEXT, UICC_INVALID_FILE, UICC_TOO_SMALL_BUFFER, UICC_UNDEFINED_ERROR`

Programmer Error

None

Implementer Notes

None

Annex A UICC API Header File (Informative)

This annex presents an example UICC API header file "`uicc_iot_api.h`" based on the descriptions in Section 4. The following example code is informative. In the event of any discrepancy between this annex and the prior sections of this specification, the prior sections shall prevail.

```
/* UICC API for IoT version 1.0 */
#ifndef _HEADER_UICC_IOT_API_H
#define _HEADER_UICC_IOT_API_H

#include “stddef.h”
#include “stdint.h”

/* Defines */
/* Bit Masks for Extracting Parts of the Result Code */
#define UICC_VALUE_MASK 0x0000FFFF
#define UICC_STATUS_MASK 0xFF000000
#define UICC_STATUS_MASK_REPLY_DATA_REMAINING 0x01000000
#define UICC_STATUS_MASK_NONPROCESSED_DATA_REMAINING 0x02000000

/* Result Code Constants */
#define UICC_NO_ERROR 0
#define UICC_ARGUMENT_ERROR 1
#define UICC_INVALID_DEVICE_NBR 2
#define UICC_TOO_SMALL_BUFFER 3
#define UICC_UNSUPPORTED_TYPE 4
#define UICC_INVALID_TYPE 5
#define UICC_INVALID_CONTEXT 6
#define UICC_UNSUPPORTED_OPERATION 7
#define UICC_INVALID_OPERATION 8
#define UICC_MAX_SESSIONS_REACHED 9
#define UICC_UNSUPPORTED_FUNCTION 10
#define UICC_NOT_ENOUGH_MEMORY 11
#define UICC_AKA_MAC_ERROR 12
#define UICC_AKA_SYNC_ERROR 13
#define UICC_FILE_NOT_FOUND 14
#define UICC_INVALID_FILE 15
#define UICC_INVALID_TRANS_ID 16
#define UICC_INVALID_OID 17
#define UICC_CHALLENGE_MISMATCH 18
#define UICC_INVALID_CERTIFICATE 19
#define UICC_UNSUPPORTED_CURVE 20
#define UICC_UNSUPPORTED_CIPKID 21
#define UICC_NO_RSP_SESSION_CONTEXT 22
#define UICC_INVALID_SIGNATURE 23
#define UICC_INCORRECT_INPUT_VALUES 24
#define UICC_UNSUPPORTED_REMOTE_OPERATION_TYPE 25
#define UICC_UNSUPPORTED_CRT_VALUES 26
#define UICC_ICCID_ALREADY_EXISTS 27
#define UICC_INSTALL_FAILED_PPR_NOT_ALLOWED 28
#define UICC_INSTALL_FAILED_DATA_MISMATCH 29
#define UICC_INSTALL_FAILED_INSUFFICIENT_MEMORY 30
```

```
#define UICC_INSTALL_INTERRUPTED 31
#define UICC_INSTALL_FAILED_PE_PROCESSING_ERROR 32
#define UICC_INSTALL_FAILED_INVALID_NAA_KEY 33
#define UICC_UNSUPPORTED_PROFILE_CLASS 34
#define UICC_SCP03T_STRUCT_ERROR 35
#define UICC_SCP03T_SEC_ERROR 36
#define UICC_NOTHING_TO_DELETE 37
#define UICC_ERROR_BUSY 38
#define UICC_ICCID_OR_AID_NOT_FOUND 39
#define UICC_PROFILE_NOT_DISABLED 40
#define UICC_PROFILE_NOT_ENABLED 41
#define UICC_NOT_ALLOWED_BY_POLICY 42
#define UICC_WRONG_PROFILE_REENABLE 43
#define UICC_LOAD_PROFILE_COMPLETED 100
#define UICC_UNDEFINED_ERROR 127

/* UICC Data Types */
#define UICC_DATATYPE_DEFAULT_SMDP_ADDRESS 0
#define UICC_DATATYPE_INFO1 1
#define UICC_DATATYPE_INFO2 2
#define UICC_DATATYPE_EID 3
#define UICC_DATATYPE_ROOT_SMDS_ADDRESS 4
#define UICC_DATATYPE_RAT 5
#define UICC_DATATYPE_CERTIFICATE_CHAIN 6
#define UICC_DATATYPE_UICC_CERTIFICATE 7

/* Profile Metadata Types */
#define UICC_PROFILEDATATYPE_NICKNAME 0

/* Cancel Session Reasons */
#define UICC_CSREASON_ENDUSER_REJECTION 0
#define UICC_CSREASON_POSTPONED 1
#define UICC_CSREASON_TIMEOUT 2
#define UICC_CSREASON_PPR_NOT_ALLOWED 3
#define UICC_CSREASON_METADATA_MISMATCH 4
#define UICC_CSREASON_LOAD_BPP_EXECUTION_ERROR 5
#define UICC_CSREASON_UNDEFINED 127

/* Notification Management Operations */
#define UICC_NOTIFICATION_MANAGEMENT_OP_LIST 0
#define UICC_NOTIFICATION_MANAGEMENT_OP_RETRIEVE 1
#define UICC_NOTIFICATION_MANAGEMENT_OP_DELETE 2

/* Bit Mask Definition for Notifications Selection */
#define UICC_NOTIFICATION_MASK_PROFILE_INSTALL 0x0001
#define UICC_NOTIFICATION_MASK_PROFILE_ENABLE 0x0002
#define UICC_NOTIFICATION_MASK_PROFILE_DISABLE 0x0004
#define UICC_NOTIFICATION_MASK_PROFILE_DELETE 0x0008
#define UICC_NOTIFICATION_MASK_SEQNUMB 0x8000

/* Profile Search Types */
#define UICC_PROFILE_SEARCHTYPE_ALL 0
#define UICC_PROFILE_SEARCHTYPE_ICCID 1
```

```
#define UICC_PROFILE_SEARCHTYPE_ISDP_AID 2
#define UICC_PROFILE_SEARCHTYPE_CLASS 3

/* Profile Management Operations */
#define UICC_PROFILE_MANAGEMENT_OP_ENABLE 0
#define UICC_PROFILE_MANAGEMENT_OP_DISABLE 1
#define UICC_PROFILE_MANAGEMENT_OP_DELETE 2

/* Bit Mask Definition for Profile Information List */
#define UICC_PROFILE_INFOLIST_MASK_ICCID 0x0001
#define UICC_PROFILE_INFOLIST_MASK_ISDP_AID 0x0002
#define UICC_PROFILE_INFOLIST_MASK_PROFILE_STATE 0x0004
#define UICC_PROFILE_INFOLIST_MASK_PROFILE_NICKNAME 0x0008
#define UICC_PROFILE_INFOLIST_MASK_SP_NAME 0x0010
#define UICC_PROFILE_INFOLIST_MASK_PROFILE_NAME 0x0020
#define UICC_PROFILE_INFOLIST_MASK_ICON_TYPE 0x0040
#define UICC_PROFILE_INFOLIST_MASK_ICON 0x0080
#define UICC_PROFILE_INFOLIST_MASK_PROFILE_CLASS 0x0100
#define UICC_PROFILE_INFOLIST_MASK_NOTIFICATION_CI 0x0200
#define UICC_PROFILE_INFOLIST_MASK_PROFILE_OWNER 0x0400
#define UICC_PROFILE_INFOLIST_MASK_SMDP_PROP_DATA 0x0800
#define UICC_PROFILE_INFOLIST_MASK_PPR 0x1000

/* Bit Mask Definition for Memory Reset */
#define UICC_MEMRST_MASK_DELETE_OPERATIONAL_PROFILES 0x0001
#define UICC_MEMRST_MASK_DELETE_FIELDLOADED_TEST_PROFILES 0x0002
#define UICC_MEMRST_MASK_RESET_DEFAULT_SMDP_ADDRESS 0x0004
#define UICC_MEMRST_MASK_DELETE_ALL_PROFILES \
(UICC_MEMRST_MASK_DELETE_FIELDLOADED_TEST_PROFILES | \
UICC_MEMRST_MASK_DELETE_OPERATIONAL_PROFILES)

/* AKA Parameters Length Definitions */
#define AKA_AUTN_LEN 16
#define AKA_AUTS_LEN 14
#define AKA_CK_LEN 16
#define AKA_IK_LEN 16
#define AKA_RAND_LEN 16
#define AKA_RES_MAX 16

/* Type definitions */
typedef uint8_t UICC_DataType_t;
typedef uint8_t UICC_ProfileMetadataType_t;
typedef uint8_t UICC_CancelSessionReason_t;
typedef uint8_t UICC_Operation_t;
typedef uint8_t UICC_ProfileSearchType_t;
typedef uint16_t UICC_FileId_t;
typedef uint16_t UICC_NotificationsSelection_t;
typedef uint16_t UICC_InfoList_t;
typedef uint16_t UICC_MemoryReset_t;
typedef uint32_t UICC_Result_t;
typedef uint32_t UICC_SequenceNumber_t;
typedef void* UICC_DevCtx_t;
typedef void* UICC_SessCtx_t;
typedef struct {
  uint8_t major; /* major version value */
  uint8_t minor; /* minor version value */
} UICC_Version_t;

typedef struct {
  uint8_t *p;       /* pointer to the first byte of the buffer */
  size_t len;       /* number of bytes with valid data */
  size_t allocated; /* number of allocated bytes */
} UICC_Buffer_t;

typedef struct {
  UICC_Operation_t op; /* notification management operation to be performed */
  UICC_NotificationsSelection_t selection; /* selects notification(s) */
  UICC_SequenceNumber_t seq_nbr; /* sequence number of a particular notification */
} UICC_NotificationsParams_t;

typedef struct {
  UICC_Operation_t op; /* profile management operation to be performed */
  UICC_ProfileSearchType_t type; /* profile located by ICCID or ISD-P AID */
  UICC_Buffer_t buf; /* buffer containing the value of the ICCID or ISD-P AID */
} UICC_ProfileMgmtOperation_t;

typedef struct {
  UICC_ProfileSearchType_t type; /* indicates  a selection of profiles */
  UICC_InfoList_t info_list; /* indicates information to be returned per profile */
  UICC_Buffer_t buf; /* buffer containing data for selecting one or more profiles */
} UICC_SearchFilter_t;

typedef struct {
  uint8_t rand[AKA_RAND_LEN]; /* RAND – input to AKA computation */
  uint8_t autn[AKA_AUTN_LEN]; /* AUTN – input to AKA computation */
  uint8_t res[AKA_RES_MAX];   /* RES – output from AKA computation */
  size_t res_len;             /* Length of RES (in bytes) */
  uint8_t ck[AKA_CK_LEN];     /* Ciphering Key – output from AKA computation */
  uint8_t ik[AKA_IK_LEN];     /* Integrity Key – output from AKA computation */
  uint8_t auts[AKA_AUTS_LEN]; /* AUTS – output in case of synchronization error */
} UICC_AkaChallengeParams_t;

/* Functions */
/**
 * \brief This function returns the version value of the API.
 *
 * \param version_p Pointer to parameter where the version of the API is returned
 *
 * \return UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_UNDEFINED_ERROR
 */
UICC_Result_t UICC_GetVersion(UICC_Version_t *version_p);

/**
 * \brief This function returns the number of available UICC devices.
 *
 * \param nbr_p Pointer to parameter where the number of available devices is
 *              returned
 *
 * \return UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_UNDEFINED_ERROR
 */
UICC_Result_t UICC_GetNumberOfDevices(uint8_t *nbr_p);

/**
 * \brief This function returns information about UICC device i, 1 <= i <= n,
 *        where n is the number of devices as returned by UICC_GetNumberOfDevices.
 *
 * \param nbr       Number of the device for which information is requested
 * \param vendor_p  Pointer to buffer where the UICC vendor (and possibly
 *                  also model) information is given as a UTF8 string. The
 *                  maximum length of the string is 64 bytes. If the allocated
 *                  buffer is not large enough to fit the complete string,
 *                  an error is returned. Upon function return, the buffer length
 *                  is updated to indicate the length of the string. A call to
 *                  this function with the buffer pointer set to NULL returns
 *                  the required minimum length of the buffer to query the
 *                  information string.
 * \param version_p Pointer to a parameter where the version of the API as
 *                  supported by the UICC device is returned.
 *
 * \return UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_INVALID_DEVICE_NBR,
 *         UICC_TOO_SMALL_BUFFER, UICC_UNDEFINED_ERROR
 */
UICC_Result_t UICC_GetDeviceInfo(uint8_t nbr, UICC_Buffer_t *vendor_p,
                                 UICC_Version_t *version_p);

/**
 * \brief This function initializes a device context with the selected UICC device.
 *        After this function, the context will contain information that supports
 *        mapping to the correct device.
 *
 * \param nbr    Number of the device for which initialization is requested,
 *               1 <= nbr <= n, where n is the number of devices as returned by
 *               UICC_GetNumberOfDevices.

 * \param dctx_p Pointer to a UICC device context
 *
 * \return UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_INVALID_DEVICE_NBR,
 *         UICC_UNDEFINED_ERROR
 */
UICC_Result_t UICC_InitializeDeviceContext(uint8_t nbr, UICC_DevCtx_t *dctx_p);

/**
 * \brief This function ends communication between a UICC client and the UICC device
 *        identified by the provided UICC device context.
 *
 * \param dctx UICC device context
 *
 * \return UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_INVALID_CONTEXT,
 *         UICC_UNDEFINED_ERROR
 */
UICC_Result_t UICC_FinalizeDeviceContext(UICC_DevCtx_t dctx);

/**
 * \brief This function opens a session between a UICC client and the UICC device
 *        identified by the provided UICC device context.
 *
 * \param dctx   UICC device context
 * \param sctx_p Pointer to a session context
 *
 * \return UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_INVALID_CONTEXT,
 *         UICC_MAX_SESSIONS_REACHED, UICC_UNDEFINED_ERROR
 */
UICC_Result_t UICC_OpenSession(UICC_DevCtx_t dctx, UICC_SessCtx_t *sctx_p);

/**
 * \brief This function closes an open session between a UICC client and the UICC
 *        device identified by the provided session context.
 *
 * \param sctx Session context
 *
 * \return UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_INVALID_CONTEXT,
 *         UICC_UNDEFINED_ERROR
 */
UICC_Result_t UICC_CloseSession(UICC_SessCtx_t sctx);

/**
 * \brief This function returns UICC data needed for profile download and
 *        installation.
 *
 * \param sctx     Session context
 * \param type     Type of UICC data
 * \param buffer_p Pointer to buffer where the UICC data is to be stored. If the
 *                 allocated buffer is not large enough to fit the complete data,
 *                 an error is returned. Upon function return, the buffer length
 *                 is updated to indicate the length of the returned data. A call
 *                 to this function with the buffer pointer set to NULL returns
 *                 the required minimum length of the buffer to query the
 *                 particular data.
 *
 * \return UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_INVALID_CONTEXT,
 *         UICC_INVALID_TYPE, UICC_TOO_SMALL_BUFFER, UICC_UNDEFINED_ERROR,
 *         UICC_UNSUPPORTED_TYPE
*/
UICC_Result_t UICC_GetUICCData(UICC_SessCtx_t sctx, UICC_DataType_t type,
                               UICC_Buffer_t *buffer_p);

/**
 * \brief This function sets UICC data needed for profile download and
 *        installation. The following data/structures may be set:
 *        - default SM-DP+ address
 *
 * \param sctx     Session context
 * \param type     Type of UICC data
 * \param buffer_p Pointer to a buffer that contains the data to be written.
 *
 * \return UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_INVALID_CONTEXT,
 *         UICC_INVALID_TYPE, UICC_NOT_ENOUGH_MEMORY, UICC_UNDEFINED_ERROR,
 *         UICC_UNSUPPORTED_TYPE
*/
UICC_Result_t UICC_SetUICCData(UICC_SessCtx_t sctx, UICC_DataType_t type,
                               UICC_Buffer_t *buffer_p);

/**
 * \brief This function initiates authentication as part of the common mutual
 *        authentication procedure specified in GSMA SGP.22. The function returns
 *        the 16-byte UICC challenge.
 *
 * \param sctx     Session context
 * \param buffer_p Pointer to buffer where the UICC challenge of length 16 bytes
 *                 is to be stored. If the allocated buffer is not large enough to
 *                 fit the complete UICC challenge, an error is returned. Upon
 *                 function return, the buffer length is updated to indicate the
 *                 length of the UICC challenge.
 *
 * \return UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_INVALID_CONTEXT,
 *         UICC_TOO_SMALL_BUFFER, UICC_UNDEFINED_ERROR
 */
UICC_Result_t UICC_InitiateAuthentication(UICC_SessCtx_t sctx,
                                          UICC_Buffer_t *buffer_p);

/**
 * \brief This function authenticates the server as part of the common mutual
 *        authentication procedure specified in GSMA SGP.22.
 *
 * \param sctx     Session context
 * \param buffer_p Pointer to buffer containing the AuthenticateServerRequest
 *                 structure. Upon function return, the buffer contains the
 *                 AuthenticateServerResponse structure or, in case of error,
 *                 the transaction ID of AuthenticateServerRequest (if available).
 *                 If the buffer is not large enough to include the complete
 *                 AuthenticateServerResponse structure, the buffer is filled and
 *                 the length of the buffer indicates the remaining bytes; note that
 *                 the buffer len parameter may in this case be larger than the
 *                 allocated parameter. In addition, result code status bit
 *                 UICC_STATUS_MASK_REPLY_DATA_REMAINING is set to indicate that
 *                 remaining data is available. Further calls to this function may
 *                 then be done to retrieve the remaining bytes.
 *
 * \return UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_CHALLENGE_MISMATCH,
 *         UICC_INVALID_CERTIFICATE, UICC_INVALID_CONTEXT, UICC_INVALID_OID,
 *         UICC_INVALID_SIGNATURE, UICC_NO_RSP_SESSION_CONTEXT,
 *         UICC_NOT_ENOUGH_MEMORY, UICC_TOO_SMALL_BUFFER, UICC_UNDEFINED_ERROR,
 *         UICC_UNSUPPORTED_CIPKID, UICC_UNSUPPORTED_CURVE
 *
 *         If remaining reply data is available, the result code values above
 *         are ORed with UICC_STATUS_MASK_REPLY_DATA_REMAINING.
 */
UICC_Result_t UICC_AuthenticateServer(UICC_SessCtx_t sctx,
                                      UICC_Buffer_t *buffer_p);

/**
 * \brief This function prepares profile download as part of the common mutual
 *        authentication procedure specified in GSMA SGP.22.
 *
 * \param sctx     Session context
 * \param buffer_p Pointer to buffer containing the PrepareDownloadRequest
 *                 structure. Upon function return, the buffer contains the
 *                 PrepareDownloadResponse structure or, in case of error,
 *                 the transaction ID (internally stored from previously parsed
 *                 AuthenticateServerRequest).
 *                 If the buffer is not large enough to include the complete
 *                 PrepareDownloadResponse structure, the buffer is filled and
 *                 the length of the buffer indicates the remaining bytes; note that
 *                 the buffer len parameter may in this case be larger than the
 *                 allocated parameter. In addition, result code status bit
 *                 UICC_STATUS_MASK_REPLY_DATA_REMAINING is set to indicate that
 *                 remaining data is available. Further calls to this function may
 *                 then be done to retrieve the remaining bytes.
 *
 * \return UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_INVALID_CERTIFICATE,
 *         UICC_INVALID_CONTEXT, UICC_INVALID_SIGNATURE, UICC_INVALID_TRANS_ID,
 *         UICC_NO_RSP_SESSION_CONTEXT, UICC_NOT_ENOUGH_MEMORY,
 *         UICC_TOO_SMALL_BUFFER, UICC_UNDEFINED_ERROR, UICC_UNSUPPORTED_CURVE
 *
 *         If remaining reply data is available, the result code values above
 *         are ORed with UICC_STATUS_MASK_REPLY_DATA_REMAINING.
 */
UICC_Result_t UICC_PrepareDownload(UICC_SessCtx_t sctx,
                                   UICC_Buffer_t *buffer_p);

/**
 * \brief This function loads and installs the Bound Profile Package (BPP) as
 *        specified in GSMA SGP.22. Since the BPP may be large and delivered in
 *        chunks, the function may be called several times. The UICC process as
 *        much as it can from the received BPP part. The length of the buffer is
 *        updated to mark how many bytes of the supplied buffer were processed.
 *        In the next call to the function, the calling application is responsible
 *        to provide the remaining unprocessed bytes concatenated with the next BPP
 *        part. The UICC implementation may buffer already decrypted and verified
 *        data (e.g. part of the UPP) that cannot yet be processed until more data
 *        has been received.
 *
 * \param sctx     Session context
 * \param buffer_p Pointer to buffer containing the BPP structure. Upon complete
 *                 processing of the BPP, the buffer contains the
 *                 ProfileInstallationResult structure at function return.
 *                 If the full BPP has not yet been provided and non-processed BPP
 *                 data remains from the BPP chunk provided in this call, the
 *                 length of the buffer is updated with the number of BPP bytes
 *                 processed in this call and result code status bit
 *                 UICC_STATUS_MASK_NONPROCESSED_DATA_REMAINING is set.
 *                 In case of error the buffer may or may not, depending on the
 *                 type of error, contain the ProfileInstallationResult structure.
 *                 If the buffer is not large enough to include the complete
 *                 ProfileInstallationResult structure, the buffer is filled and
 *                 the length of the buffer indicates the remaining bytes; note that
 *                 the buffer len parameter may in this case be larger than the
 *                 allocated parameter. In addition, result code status bit
 *                 UICC_STATUS_MASK_REPLY_DATA_REMAINING is set to indicate that
 *                 remaining data is available. Further calls to this function may
 *                 then be done to retrieve the remaining bytes.
 *
 * \return UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_ICCID_ALREADY_EXISTS,
 *         UICC_INCORRECT_INPUT_VALUES, UICC_INSTALL_FAILED_DATA_MISMATCH,
 *         UICC_INSTALL_FAILED_INSUFFICIENT_MEMORY,
 *         UICC_INSTALL_FAILED_INVALID_NAA_KEY,
 *         UICC_INSTALL_FAILED_PE_PROCESSING_ERROR,
 *         UICC_INSTALL_FAILED_PPR_NOT_ALLOWED, UICC_INSTALL_INTERRUPTED,
 *         UICC_INVALID_CONTEXT, UICC_INVALID_SIGNATURE, UICC_INVALID_TRANS_ID,
 *         UICC_LOAD_PROFILE_COMPLETED, UICC_NOT_ENOUGH_MEMORY,
 *         UICC_SCP03T_SEC_ERROR, UICC_SCP03T_STRUCT_ERROR, UICC_TOO_SMALL_BUFFER,
 *         UICC_UNDEFINED_ERROR, UICC_UNSUPPORTED_CRT_VALUES,
 *         UICC_UNSUPPORTED_PROFILE_CLASS, UICC_UNSUPPORTED_REMOTE_OPERATION_TYPE
 *
 *         If remaining reply data is available the result codes above are
 *         ORed with UICC_STATUS_MASK_REPLY_DATA_REMAINING.
 */
UICC_Result_t UICC_LoadBoundProfilePackage(UICC_SessCtx_t sctx,
                                           UICC_Buffer_t *buffer_p);

/**
 * \brief This function cancels an ongoing RSP session (see GSMA SGP.22).
 *
 * \param sctx     Session context
 * \param reason   Indicates the reason for cancelling the session.
 * \param buffer_p Pointer to buffer containing the transaction ID. Upon
 *                 successful function return, the buffer contains the
 *                 CancelSessionResponse structure.
 *                 If the buffer is not large enough to include the complete
 *                 CancelSessionResponse structure, the buffer is filled and
 *                 the length of the buffer indicates the remaining bytes; note that
 *                 the buffer len parameter may in this case be larger than the
 *                 allocated parameter. In addition, result code status bit
 *                 UICC_STATUS_MASK_REPLY_DATA_REMAINING is set to indicate that
 *                 remaining data is available. Further calls to this function may
 *                 then be done to retrieve the remaining bytes.
 *
 * \return UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_INVALID_CONTEXT,
 *         UICC_INVALID_TRANS_ID, UICC_NOT_ENOUGH_MEMORY, UICC_TOO_SMALL_BUFFER,
 *         UICC_UNDEFINED_ERROR
 *
 *         If remaining reply data is available the result codes above are
 *         ORed with UICC_STATUS_MASK_REPLY_DATA_REMAINING.
 */
UICC_Result_t UICC_CancelSession(UICC_SessCtx_t sctx,
                                 UICC_CancelSessionReason_t reason,
                                 UICC_Buffer_t *buffer_p);

/**
 * \brief This function is used to manage notifications. The following operations
 *        are supported:
 *        - Get a list of one or more metadata for pending notifications.
 *        - Retrieve one or more pending notifications.
 *        - Remove a particular notification from the list of notifications.
 *
 * \param sctx     Session context
 * \param params_p Pointer to structure containing parameters related to
 *                 management of notifications
 * \param buffer_p Pointer to buffer where the returned data is stored. If the
 *                 operation is to get a list of notification metadata, the returned
 *                 data is a sequence of one or more NotificationMetadata
 *                 structures. If the operation is to retrieve notifications, the
 *                 returned data is a sequence of one or more PendingNotification
 *                 structures. For the delete operation, no data is returned.
 *                 If the allocated buffer is not large enough to fit the complete
 *                 data, an error is returned. Upon function return, the buffer
 *                 length is updated to indicate the length of the returned data.
 *                 A call to this function with the buffer pointer set to NULL
 *                 returns the required minimum length of the buffer to query the
 *                 particular data.
*
 * \return UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_INVALID_CONTEXT,
 *         UICC_INVALID_OPERATION, UICC_NOT_ENOUGH_MEMORY, UICC_NOTHING_TO_DELETE,
 *         UICC_TOO_SMALL_BUFFER, UICC_UNDEFINED_ERROR, UICC_UNSUPPORTED_FUNCTION,
 *         UICC_UNSUPPORTED_OPERATION
 */
UICC_Result_t UICC_ManageNotifications(UICC_SessCtx_t sctx,
                                       UICC_NotificationsParams_t *params_p,
                                       UICC_Buffer_t *buffer_p);

/**
 * \brief This function is used for management of profiles. The following
 *        operations are supported: enable profile, disable profile, and delete
 *        profile.
 *
 * \param sctx   Session context
 * \param mgmt_p Pointer to structure containing parameters related to management
 *               of profiles
 *
 * \return UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_ERROR_BUSY,
 *         UICC_ICCID_OR_AID_NOT_FOUND, UICC_INVALID_CONTEXT,
 *         UICC_INVALID_OPERATION, UICC_INVALID_TYPE, UICC_NOT_ALLOWED_BY_POLICY,
 *         UICC_PROFILE_NOT_DISABLED, UICC_PROFILE_NOT_ENABLED,
 *          UICC_TOO_SMALL_BUFFER, UICC_UNDEFINED_ERROR, UICC_UNSUPPORTED_OPERATION,
 *          UICC_UNSUPPORTED_TYPE, UICC_WRONG_PROFILE_REENABLE
 */
UICC_Result_t UICC_ManageProfile(UICC_SessCtx_t sctx,
                                 UICC_ProfileMgmtOperation_t *mgmt_p);

/**
 * \brief This function is used to retrieve information about available profiles.
 *        Information may be requested for all available profiles, all profiles of
 *        a given class, or a single profile identified by an ICCID or ISD P AID.
 *        The set of information requested per profile is specified using the
 *        filter_p parameter.
*
 * \param sctx     Session context
 * \param filter_p A pointer to a structure specifying the information to be
 *                 returned for each profile, and identifying the profile(s) for
 *                 which the information is to be returned.
 * \param buffer_p Pointer to buffer that contains the ProfileInfoListResponse
 *                 structure (see GSMA SGP.22). If the allocated buffer is not
 *                 large enough to fit the complete data, an error is returned.
 *                 Upon function return, the buffer length is updated to indicate
 *                 the length of the returned data. A call to this function with
 *                 the buffer pointer set to NULL returns the required minimum
 *                 length of the buffer to query the particular data.
 *
 * \return UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_INVALID_CONTEXT,
 *          UICC_INVALID_OPERATION, UICC_NOT_ENOUGH_MEMORY, UICC_TOO_SMALL_BUFFER,
 *          UICC_UNDEFINED_ERROR, UICC_UNSUPPORTED_OPERATION
 */
UICC_Result_t UICC_GetProfilesInfo(UICC_SessCtx_t sctx,
                                   UICC_SearchFilter_t *filter_p,
                                   UICC_Buffer_t *buffer_p);

/**
 * \brief This function sets profile metadata.
 *
 * \param sctx     Session context
 * \param iccid_p  Pointer to buffer containing the ICCID of the profile for which
 *                 the profile metadata is to be set
 * \param type     Type of profile metadata
 * \param buffer_p Pointer to buffer containing the profile metadata.
 *
 * \return UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_ICCID_OR_AID_NOT_FOUND,
 *          UICC_INVALID_CONTEXT, UICC_INVALID_TYPE, UICC_NOT_ENOUGH_MEMORY,
 *          UICC_UNDEFINED_ERROR, UICC_UNSUPPORTED_TYPE
 */
UICC_Result_t UICC_SetProfileMetadata(UICC_SessCtx_t sctx,
                                      UICC_Buffer_t *iccid_p,
                                      UICC_ProfileMetadataType_t type,
                                      UICC_Buffer_t *buffer_p);

/**
 * \brief This function is used to delete several profiles (e.g. all operational
 *        profiles) according to the input parameter options.
 *
 * \param sctx    Session context
 * \param options This parameter indicates what profiles and other data structures
 *                should be deleted.
 *
 * \return UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_ERROR_BUSY,
 *         UICC_INVALID_CONTEXT, UICC_NOTHING_TO_DELETE, UICC_UNDEFINED_ERROR
 */
UICC_Result_t UICC_MemoryReset(UICC_SessCtx_t sctx,
                               UICC_MemoryReset_t options);

/**
 * \brief This function performs the 3GPP Authentication and Key Agreement (AKA)
 *        algorithm. It authenticates the network and computes the response (RES)
 *        on the received challenge used in authentication of the device. The
 *        function also returns the ciphering key (CK) and integrity key (IK) for
 *        use in protection of network signaling and user data between the device
 *        and the network.
 *
 * \param sctx        Session context
 * \param challenge_p Pointer to a structure containing input and output parameters
 *                    for the AKA algorithm.
 *
 * \return UICC_NO_ERROR, UICC_AKA_MAC_ERROR, UICC_AKA_SYNC_ERROR,
 *         UICC_ARGUMENT_ERROR, UICC_INVALID_CONTEXT, UICC_UNDEFINED_ERROR
 */
UICC_Result_t UICC_Authenticate(UICC_SessCtx_t sctx,
                                UICC_AkaChallengeParams_t *challenge_p);

/**
 * \brief This function reads an elementary file containing configuration data
 * relevant for the modem/device. This function only permits reading elementary
 * files of the USIM application of the active profile.
 *
 * \param sctx     Session context
 * \param id       File identifier
 * \param buffer_p Pointer to buffer where the returned file data is stored.
 *                 If the allocated buffer is not large enough to fit the complete
 *                 file data, an error is returned. Upon function return, the buffer
 *                 length is updated to indicate the length of the returned data.
 *                 A call to this function with the buffer pointer set to NULL
 *                 returns the required minimum length of the buffer to query the
 *                 particular data.
 *
 * \return UICC_NO_ERROR, UICC_ARGUMENT_ERROR, UICC_FILE_NOT_FOUND,
 *         UICC_INVALID_CONTEXT, UICC_INVALID_FILE, UICC_TOO_SMALL_BUFFER,
 *         UICC_UNDEFINED_ERROR
 */
UICC_Result_t UICC_ReadFile(UICC_SessCtx_t sctx, UICC_FileId_t id,
                            UICC_Buffer_t *buffer_p);

#endif /*_HEADER_UICC_IOT_API_H*/
```