ETSI TS 102 240 V11.1.0 (2011-12)

Smart Cards;
UICC Application Programming Interface and Loader Requirements;
Service description
(Release 11)

Reference
RTS/SCP-R0263vb10

Keywords
API, smart card

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Contents

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://ipr.etsi.org).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This Technical Specification (TS) has been produced by ETSI Technical Committee Smart Card Platform (SCP).

It is based on work originally done by the 3GPP group in "TSG-Terminals WG3" and by "ETSI Special Mobile Group (SMG)".

The present document details the stage 1 aspects (overall service description) for the support of a UICC Application Programming Interface (API).

The contents of the present document are subject to continuing work within ETSI SCP and may change following formal ETSI SCP approval. Should ETSI SCP modify the contents of the present document it will then be republished by ETSI with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

x the first digit:

- 0 early working draft;
- 1 presented to TC SCP for information;
- 2 presented to TC SCP for approval;
- 3 or greater indicates TC SCP approved document under change control.

y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.

z the third digit is incremented when editorial only changes have been incorporated in the document.

1 Scope

The present document defines the service description of the UICC Application Programming Interface (UICC API) internal to the UICC. Stage one is an overall service description, and does not deal with the implementation details of the API.

The present document includes information applicable to network operators, service providers and terminal, UICC, Network Access Application (NAA) providers, switch and database manufacturers.

The present document contains the core requirements, which are sufficient to provide a complete service.

It is highly desirable however, that technical solutions for a UICC API should be sufficiently flexible to allow for possible enhancements. Additional functionalities not documented in the present document may implement requirements which are considered outside the scope of the present document. This additional functionality may be on a network wide basis, nation-wide basis or particular to a group of users. Such additional functionality shall not compromise conformance to the core requirements of the service.

2 References

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the reference document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at http://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

- In the case of a reference to a TC SCP document, a non specific reference implicitly refers to the latest version of that document in the same Release as the present document.

2.1 Normative references

The following referenced documents are necessary for the application of the present document.

[1] ETSI TS 102 221: "Smart cards; UICC-Terminal interface; Physical and logical characteristics (Release 7)".

[2] ETSI TS 102 223: "Smart cards; Card Application Toolkit (CAT) (Release 7)".

[3] ISO/IEC 7816-4: "Identification cards - Integrated circuit cards Part 4: Organization, security and commands for interchange".

[4] ETSI TS 102 622: "Smart Cards; UICC - Contactless Front-end (CLF) Interface; Host Controller Interface (HCI)".

[5] ETSI TS 102 613: "Smart Cards; UICC - Contactless Front-end (CLF) Interface; Part 1: Physical and data link layer characteristics".

[6] ETSI TS 102 600: "Smart Cards; UICC-Terminal interface; Characteristics of the USB interface".

[7] ETSI TS 102 483: "Smart cards; UICC-Terminal interface; Internet Protocol connectivity between UICC and terminal".

[8] ETSI TS 102 484: "Smart Cards; Secure channel between a UICC and an end-point terminal".

[9] OMA: "Smartcard Web Server Enabler Architecture", OMA-AD-Smartcard-Web-Server-V1-0-20070209-C.

[10] ETSI TS 102 412: "Smart Cards; Smart Card Platform Requirements Stage 1".

[11] ETSI TS 102 127: "Smart Cards; Transport protocol for CAT applications; Stage 2".

[12] ETSI TS 102 225: "Smart Cards; Secured packet structure for UICC based applications".

[13] ETSI TS 102 226: "Smart Cards; Remote APDU structure for UICC based applications".

[14] ETSI TS 131 130: "Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); LTE; (U)SIM Application Programming Interface (API); (U)SIM API for Java Card (3GPP TS 31.130 Release 9)".

[15] ETSI TS 102 267: "Smart Cards; Connection Oriented Service API for the Java Card(TM) platform".

[16] ETSI TS 102 241: "Smart Cards; UICC Application Programming Interface (UICC API) for Java Card (TM)".

2.2 Informative references

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

Not applicable.

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the following terms and definitions apply:

applet: application built up using a number of modules which will run under the control of a virtual machine

application: in the scope of the present document either an applet or a web-application.

bytecode: machine independent code generated by a bytecode compiler and executed by a bytecode interpreter

data structure: collection of related data values such as the age, birth date and height of an individual

framework: defines a set of Application Programming Interface (API) functions and data structures for developing applications and for providing system services to those applications

function: callable and executable body of computer instructions which perform a specific computation or data processing task

module: collection of functions and data structures which implement an entire application or a particular application feature or capability

UICC API framework: part of the UICC responsible for the handling of applications (including triggering and loading)

NOTE: It also contains the library for the proactive API.

Servlet: application built up using a number of modules responding to incoming Internet protocol request (e.g. TCP, HTTP, HTTPS, etc.)

NOTE: A Servlet runs under the control of a Servlet engine.

Servlet engine: part of the enhanced UICC API framework, responsible for handling incoming requests via the TCP/IP protocol (e.g. HTTP/HTTPS) and dispatching them to the web-application

toolkit applet: applet loaded onto the UICC seen by the mobile as being part of the UICC toolkit application and containing only the code necessary to run the application

NOTE: These applets might be downloaded over the radio interface.

trusted party: entity trusted by the card issuer with respect to security related services and activities

virtual machine: part of the run-time environment responsible for interpreting the bytecode

web-application: at least one Servlet or a combination of one or more Servlets, additional modules, applets, and static content

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

AID	Applet IDentifier
APDU	Application Protocol Data Unit
API	Application Programming Interface
AVN	Applet Version Number
BER	Bit Error Rate
CAD	Card Acceptance Device
CAT	Card Application Toolkit
CLF	Contactless Front-end
EPOS	Electronic Point Of Sale
HCI	Host Controller Interface
HTTP	Hypertext Transfer Protocol
IFD	InterFace Device
IP	Internet Protocol
MExE	Mobile Execution Environment
NAA	Network Access Application
P2P	Peer to peer
RPC	Remote Procedure Call
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TLV	Tag, Length, Value
UDP	User Datagram Protocol
UICC	Universal Integrated Circuit Card
WAP	Wireless Application Protocol

4 Description

The present document describes the high level requirements for an API for the UICC. This API shall allow application programmers easy access to the functions and data described in TS 102 221 [1] and TS 102 223 [2], such that UICC based services can be developed and loaded onto UICCs, quickly and, if necessary, remotely, after the UICC has been issued.

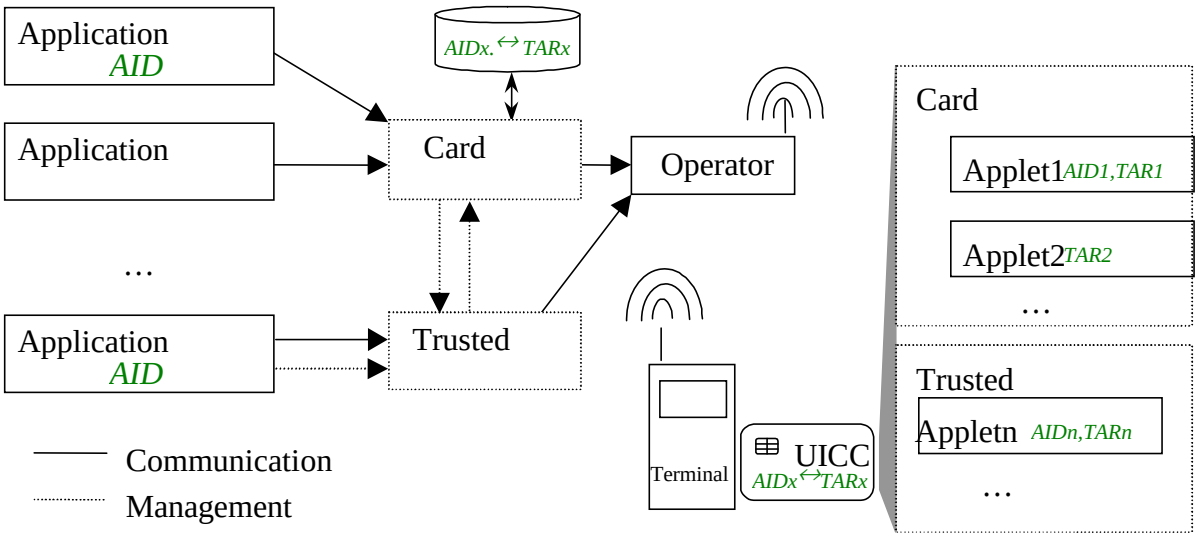


Figure 1: Toolkit applet management and communication

4.1 Design of UICC based applications using the UICC API

Figure 2 shows how UICC applications can be developed in a standard development environment and converted into an interpreted format, then loaded into the UICC.

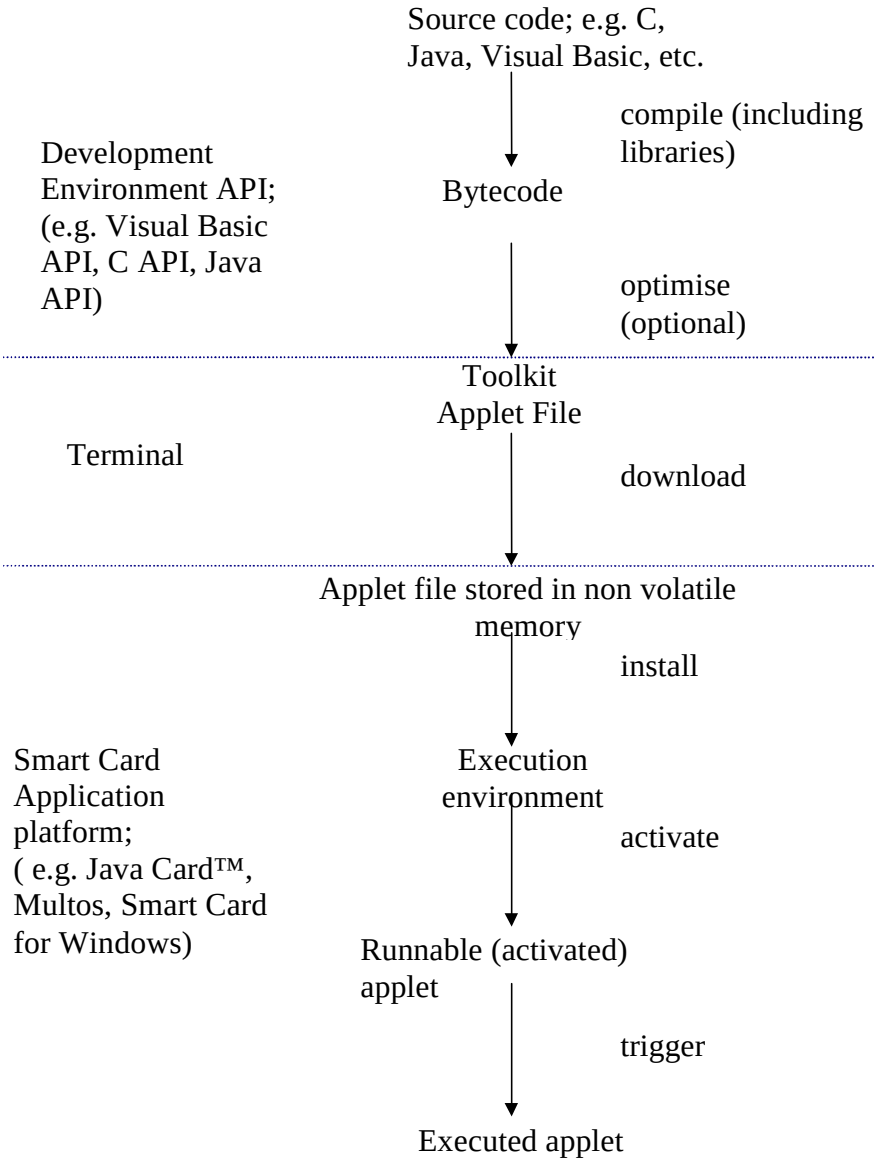
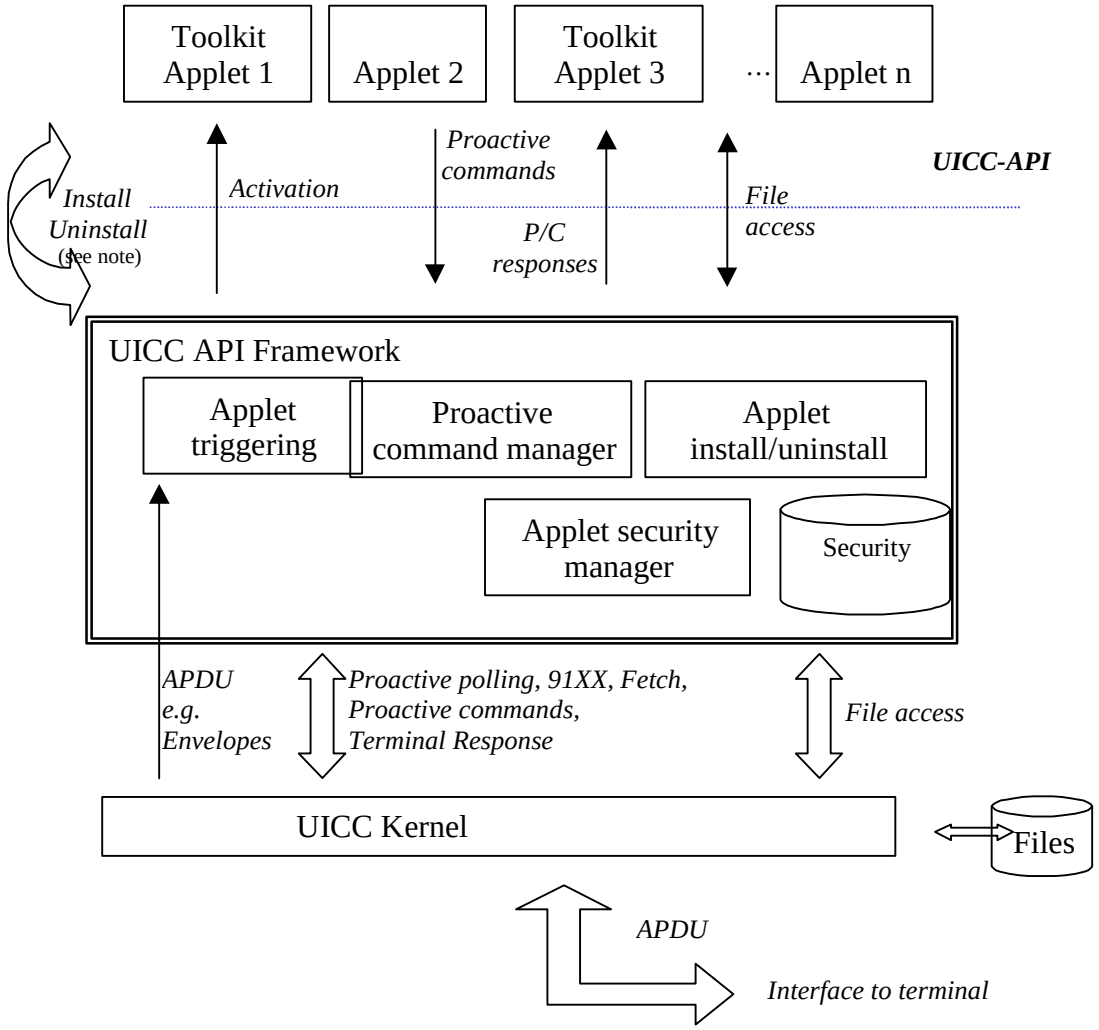


Figure 2: Flow diagram of the development of a UICC application

4.2 UICC API architecture

The UICC API shall consist of APIs for TS 102 223 [2] (pro-active functions) and TS 102 221 [1] (transport functions). Figure 3 illustrates the interactions between these APIs.



NOTE: The install / uninstall process does not form part of the API. Its requirements are outlined in clause 7.

Figure 3: UICC API architecture

In this model, the UICC data field structure is viewed as a series of data structures and data access functions to the API. In the physical model of course, they may still be stored in elementary files, but the functions will access these data as values within those data structures.

A general requirement of the UICC API is that applets should not interfere with the basic UICC services.

The UICC API framework shall prevent the toolkit applets from sending proactive commands which would interfere with the correct execution of the UICC operating system and/or other toolkit applets.

4.3 UICC file data access

The following methods shall be offered by the API to UICC applets, to allow access to the UICC data:

activateFile	This function reactivates a deactivated EF. In case of successful execution of the command, the EF on which the command was applied becomes the current EF. After an unsuccessful execution, the current EF and current DF shall remain the same as prior to the execution.
deactivateFile	This function initiates a reversible deactivation of an EF. In case of successful execution of the command, the EF on which the command was applied becomes the current EF. After an unsuccessful execution, the current EF and current DF shall remain the same as prior to the execution.
increase	This function adds the value given in an array of bytes to the value of the last increased/updated record of the current cyclic EF, and stores the result into the oldest record. The record pointer is set to this record and this record becomes record number 1. The function does not perform the increase if the result would exceed the maximum value of the record (represented by all bytes set to "FF").
readBinary	This function reads an array of bytes from the current transparent EF.
readRecord	This function reads one complete record in the current linear fixed or cyclic EF into an array of bytes.
SearchRecord	This function searches through a linear fixed or cyclic EF to find record(s) containing a specific pattern.
select	Select a file without changing the current file of any other applet or of the subscriber session.
status	This function returns information concerning the current directory.
updateBinary	This function updates the current transparent EF with an array of bytes.
updateRecord	This function updates one specific, complete record in the current linear fixed or cyclic EF with an array of bytes.

4.4 UICC BER-TLV file access

The following methods shall be offered by the API to UICC applets, to allow access to the data stored in BER-TLV files as defined in TS 102 221 [1]:

- Retrieve a list of objects stored in the BER-TLV file identified by the TAG values of the objects.
- Select a TLV object in the BER-TLV file.
- Read data from a TLV object in the BER-TLV file.
- Write data to a TLV object in a BER-TLV file.
- Delete a TLV object in a BER-TLV file.
- Add a TLV object in a BER-TLV file.

5 Card interoperability

5.1 Loader requirements

There are a number of requirements for the loader which are seen as being vital to the successful deployment of UICC API based UICCs.

- The Applet format shall be common to all compliant UICCs, such that a card issuer can deploy UICC API based service applets to any UICC API compliant UICC.

- The loader environment that allows the loading of applets to the UICC shall be common to all UICC API compliant UICCs. This loader shall be able to send applets to UICCs in three distinct ways:
  - During the personalization of the UICC, prior to the issue of the UICC to the user.
  - During the life of the UICC using the UICC Data Download mechanism defined in TS 102 223 [2] or using other standardized application dependent mechanisms.
  - During the life of the UICC using an IFD (Interface Device) or CAD (Card Accepting Device, e.g. an EPOS terminal).

5.2 Application transport

The transport of applications shall be transparent to the terminal. Applications may be transported via several different bearers.

6 Applet activation

6.1 Applet triggering

The application triggering portion of the UICC Framework is responsible for the activation of toolkit applets, based on the APDU received by the UICC. The inputs and outputs could be represented in figure 4.

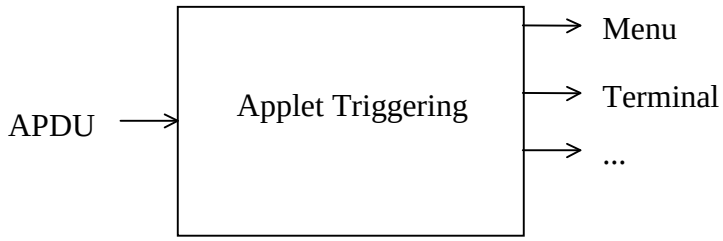


Figure 4: Applet triggering module

Entry points to the applet shall be provided in two ways:

- High level entry points, in order to have a simple programming of the UICC.
- Low level entry points to support the evolution of the TS 102 223 [2] specification.

Some of the high level entry points are listed below:

- Application loading.
- Application removal.
- Terminal profile.
- Menu selection.
- Events upon file system operations by the terminal or by application(s) in the card :
  - read file;
  - update file;
  - set data;
  - retrieve data;
  - search record;

- create file;
- delete file;
- resize file;
- terminate file;
- activate file;
- deactivate file.

6.2 Applet selection

Applet activation through selection shall follow the rules defined in TS 102 221 [1] for application selection.

7 Applet life cycle management

The applet life cycle management concerns the applet preparation, loading, installation, registration, configuration, execution and removal/deactivation.

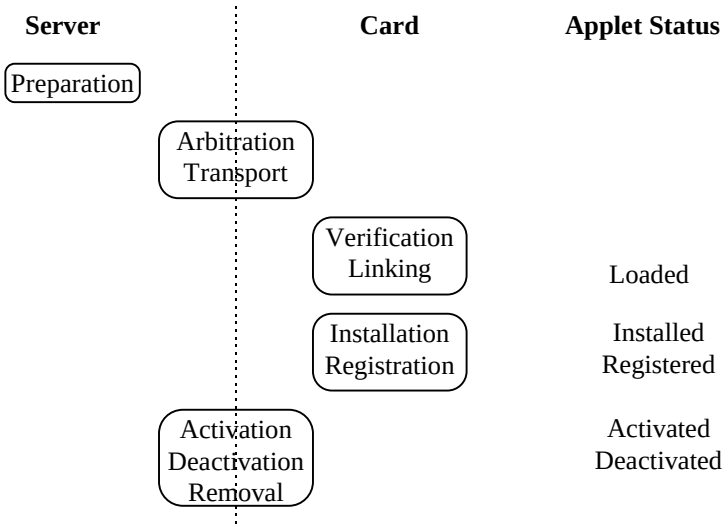


Figure 5: Applet life cycle

7.1 Applet preparation

"Applet preparation" refers to the optional phase of verifying the compliance of the applet code with card issuer or other standards.

The applet is to be identified through an Applet Identification Number (AID) which is assigned through the procedure detailed in ISO/IEC 7816-4 [3] and an Applet Version Number (AVN). Both AID and AVN are assigned during the applet preparation phase.

The minimum requirements for the applet (such as API versions, UICC capabilities, resource requirements) shall be specified.

7.2 Loading

"Loading" refers to the process of transporting the applet code from a load server to the UICC and generating the loaded code on the UICC.

The process shall be under the principle control of the card issuer, who may choose to delegate this responsibility to one or more trusted parties, possibly while imposing resource restrictions (e.g. maximum memory allowance) or access restrictions (e.g. limited or reduced functionality).

The loading process involves four distinct phases: Arbitration, Transport, Verification and Linking. The UICC shall provide acknowledgement of success or failure (including error identification code) to the load server if the load server requires this.

7.2.1 Arbitration

This phase is accomplished by mutual authentication between the UICC and the load server, and by establishing appropriate session keys for ensuring security during the data transfer, which is to follow.

The minimum applet requirements are verified with regard to the environment present on the UICC (e.g. API version, UICC capabilities and available memory). If this fails, the loading process shall be aborted.

The Applet IDentifiers (AIDs) and version numbers (AVNs) of any applets already installed on the UICC are compared to the AID and AVN of the applet, which is to be downloaded. If an identical applet is already installed on the UICC (i.e. both applet identifier and version number match), the phases Transport, Verification and Linking are skipped. If an applet with an identical applet identifier (AID) but different version number (AVN) is available on the UICC, that applet is removed (see clause 7.7).

7.2.2 Transport

This stage shall encompass the transport of the data packets from the load server to the UICC, and may be done with optionally additional encryption using session keys generated/exchanged during the arbitration phase.

7.2.3 Verification

This stage shall encompass the verification of the received data and may involve byte-code level or applet-specific verification. Should the verification stage fail, the applet shall be discarded.

7.2.4 Linking

This stage shall encompass the linking of the received code against the runtime environment present on the UICC.

7.3 Installation/registration/reactivation

This stage refers to the execution of applet-code regarding to the installation and registration of the applet with respect to the UICC runtime environment and is out of scope for the present document. It may require additional procedures depending on the UICC/terminal environment (e.g. this may involve the generation of an applet-specific menu entry in the terminal's user interface through the appropriate toolkit command, and the generation of applet-specific data structures in UICC memory).

If the applet already exists on the UICC and is deactivated (see clause 7.6), the installation request shall reactivate the applet. Other methods of reactivation are possible via a separate command.

7.4 Configuration

This stage may involve any necessary configuration of the applet code with regard to a particular user/set-up/environment. This stage is driven through code provided with the applet itself and may be executed repeatedly.

7.5 Execution

At this stage, providing the applet is activated, the applet is in a state where its execution may be triggered by any event as specified in clause 6.

7.6 Deactivation

This stage involves disabling the ability to execute applet code in the UICC and may be triggered by the user, the card issuer or any third party, providing sufficient access rights are granted to them. Deactivation may include the release of any applet reserved resources (e.g. memory resources, etc.).

7.7 Removal

This stage follows the deactivation of the applet and prevents the applet's reactivation. This may be followed by the release of the applet's memory. For security reasons, the memory may be overwritten by null data.

8 Security management

8.1 Management of applets

Security might be required during the loading of the applet from a load server onto the UICC, and the communications between the applet and any remote server during the execution of the applet code. In both cases security may involve the authentication of the communicating entities and the encryption of the data traffic between those entities.

A hierarchy of keys may be bootstrapped by initializing a set of keys by the card issuer during card personalization. Additional keys may be generated, distributed using existing keys, and equipped with limited authority. Such keys may be passed on to trusted parties and subsequently used for authentication and encryption.

8.2 Applet certification

The role of certification is to ensure that only the authorized entities are able to download an application on to the UICC. Based on this certificate, the UICC shall decide whether or not to accept the downloaded application.

9 API compatibility

9.1 Level of compatibility

The commands and features supported by the API shall be as specified in the same Release year of TS 102 221 [1] and TS 102 223 [2].

9.2 Compatibility at the interface

In order to provide compatibility with the UICC/terminal interface, a UICC using the UICC API shall provide full functional compatibility with the structure and content of TS 102 221 [1] and TS 102 223 [2] commands as specified in those documents.

9.3 Compatibility at the programming interface

All commands (at the functional level) shall be presented in a manner consistent with the customary or recommended use of the programming language at the programming level.

The UICC API shall be provided in two ways:

- an easy to use high level interface (proactive commands level); and
- a low level interface (i.e. the TLV parameters) to maximize scope without the need to extend the UICC API.

9.4 Accessibility of the programming interface

The UICC API shall be accessible by any UICC application (e.g. Java Card™ applet).

10 API extensibility

The UICC API shall support applications written for previous versions of the UICC API.

There shall be means to manage versions of the UICC API.

At installation of an applet the required UICC API version shall be checked as described in clause 7.

The ability to extend the UICC API to add functionality may be possible without reissuing the card.

10.1 Evolution of UICC/terminal interface (TS 102 221)

As the UICC/terminal interface is handled by the UICC kernel any evolution of the interface may require the introduction of a new UICC API version.

10.2 Evolution of CAT application toolkit (TS 102 223)

The UICC API shall provide a low-level interface to support any further releases of TS 102 223 [2].

The UICC API should provide a high level interface to support specific features.

10.3 Interworking with other systems

If interworking at APDU and UICC API level with other systems (e.g. MExE, WAP) require some specific functionality, it will first need to be defined either in the TS 102 221 [1] or TS 102 223 [2], and as a result it will be taken into account in the API specification.

10.4 Evolution of UICC/terminal contactless interface (TS 102 622 and TS 102 613)

The UICC API shall provide the following features:

- Transmit and receive messages to and from the CLF in card emulation mode according to TS 102 622 [4].
- Transmit and receive messages to and from the CLF in reader emulation mode according to TS 102 622 [4].
- Transmit and receive messages to and from the CLF in P2P mode according to TS 102 622 [4].
- Set and retrieve registry parameters defined in TS 102 622 [4] in the CLF.
- Subscribe and receive all the events defined in TS 102 622 [4] needed to operate in card emulation mode.

- Subscribe and receive all the events defined in TS 102 622 [4] needed to operate in reader mode.
- Subscribe and receive all the events defined in TS 102 622 [4] needed to operate in P2P mode.
- Provide a mechanism to access non-volatile memory content which is not part of the file-system or not addressable by the UICC API framework (in order to support memory writing and memory reading operations).
- Provide means to inform the UICC application of the phone status (availability of the end user interface, availability of the network interface).
- Initiate events defined for the connectivity gate as defined in TS 102 622 [4].
- Provide the availability of contactless interface power mode in the CLF according to TS 102 613 [5] and TS 102 622 [4]: i.e. low power mode/full power mode.

10.5 HCI low-level support

10.5.1 Use case

TS 102 622 [4] (HCI) supports the usage of proprietary gates that can be used to support application specific functionality. An application residing on the UICC creates a proprietary gate to provide application specific functionality, e.g. OTA services, to other hosts, e.g. an embedded secure element. An application residing on that embedded secure element can create and open a pipe to the proprietary gate provided by the application on the UICC to make use of services supported by the application.

This low-level API shall support this functionality and allow applications residing on the UICC to create proprietary gates, manage pipes for these gates and to receive and send data over these pipes accordingly.

This low-level API shall be separated from the functionality provided by the higher-level API covering the Card emulation mode, Reader mode, Connectivity functionality and P2P mode in that the gates and pipes used exclusively by these modes shall not be affected.

10.5.2 Requirements

- Access to proprietary gates shall be supported whereas access to the gates (e.g. card RF gates, card application gates. reader RF gates, reader application gates, connectivity gate, etc.) defined by the HCI specification in TS 102 622 [4] shall be excluded by the low-level API.
- Support a Host discovery, by retrieving a list of available hosts in the system from the CLF.
- Modification of the host's own white list shall be supported.
- Allow notifications upon changes of the host list on the CLF.
- Support creation and deletion of proprietary, i.e. application specific, gates.
- Support creating, opening, closing, and deleting of pipes.
- Transmit and receive messages over pipes the applet created itself.
- Allow notifications upon pipe state changes.
- Support access control to gates and pipes:
    - An application shall not be able to delete/close pipes it did not create/open.
    - An application shall not be able to delete or modify a gate it did not create.
    - An application shall not be able to create a gate with an already assigned gate ID.
    - An application shall not be able to create a pipe with an already assigned pipe ID.

10.6 Application API for Secure messaging over HTTPS

10.6.1 Use Cases (informative)

An application residing in the UICC has several means to communicate with a remote server:

i) traditional SMS technology;

ii) CAT-TP protocol as described in TS 102 127 [11];

iii) TCP/IP;

iv) Secure messages over HTTPS.

All those technologies and related security layers are described in the TS 102 225 [12] and TS 102 226 [13].

For the first two technologies, Java Card API is available for applications (for the SMS technology relying on a combination of TS 102 241 [16] and TS 131 130 [14] and for the CAT-TP technology TS 102 267 [15]) while no API is available for the secure messages over HTTPS.

When an application residing in the UICC would like to exchange data (opening a connection, receiving or sending data) with a remote server is a secure way, the Secure Message structure as described in TS 102 225 [12] (referencing the Amendment B of the Global Platform Card Specification v 2.2 with some clarifications) may be appropriate. Nevertheless, in order to ease the development of such services, it is expected that the application does not have to manage the security and HTTP related layers. As an example, an application managing some phonebook data may need to back up the data on a remote server in a confidential way, without having to implement the actual SSL/TLS and HTTP protocols.

10.6.2 Requirements (normative)

There shall be a means for an application residing in the UICC to receive and send messages according to the protocol specified in Amendment B of the Global Platform Card Specification v 2.2 (described as Secured Message for HTTPS in TS 102 225 [12]).

The secure messaging over HTTPS API shall allow an application to manage message content, without managing the HTTP and SSL/TLS protocols.

The secure messaging over HTTPS API shall allow an application to register and de-register the application to incoming messages formatted as Secured Message for HTTPS.

The secure messaging over HTTPS API shall allow an application to provide data for outgoing messages formatted as Secured Message for HTTPS.

10.7 Machine to machine (M2M) UICC applications

A UICC application shall be able to indicate via the UICC API that specific application data shall support a specific life time expectation based on the number of updates allowed for that specific data.

The UICC API shall provide a means allowing the notification of memory failure to UICC applications that a potential loss of data reliability is detected. The notification may be generated only if the memory required to handle the notification has itself not been affected.

10.8 Secure Channel between UICC and terminal

The API for UICC supporting the Secure Channel as defined in TS 102 484 [8] shall provide the following features:

- Management of the establishment and the closure of a secure channel.
- Communication over an established secure channel.
- Management of application settings such as the mandating of the use of the secure channel.

- Remote configuration of Secure Channel settings: e.g. load a PSK or a certificate in the UICC.
- Management of the UICC endpoints.
- Inform applications of their Secure Channel communication status.
- The API applies to secured APDU platform to platform and application to application.

There shall be a mechanism to restrict access to this API to authorized applications only.

11 Data and function sharing and access control

11.1 Sharing resources between applets

The API shall provide a secure data structure and function sharing mechanism between applets and with the UICC kernel.

The UICC kernel should be able to share with applets:

- files: to get file status, read and update data field;
- PIN1, PIN2: to get status.

A toolkit applet shall be able to share any kind of data with any other applet even a non-toolkit applet.

The data and function sharing mechanism and the access control management shall be common to all card issuers.

To ease the deployment, these requirements have the following priorities:

- high: UICC kernel data sharing;
- medium: inter industry sharing mechanism between applets.

11.2 Access to data

The UICC API shall provide a way to let each applet indicate:

- the shared data and functions;
- the associated access functions to these data and functions;
- the security or trust level required;
- the accepted certification authorities; and
- the identity of the applet provider.

The UICC API framework shall check all these parameters before granting an access to data.

12 Technology considerations

12.1 UICC hardware requirements

The UICC API requires a smart card device that is capable of implementing a virtual machine and the UICC API framework. It is seen as necessary that there is sufficient non volatile memory to contain UICC Applets alongside mandatory application specific files and potentially many (if not all) of optional application specific files.

12.2 Technology limitations

12.2.1 Memory recovery

Although there is a requirement for UICC API compliant devices to allow reconfiguration, termination and removal of Applets, it is recognized that UICC API devices may not be fully capable of reclaiming the memory freed up.

12.3 Evolution

12.3.1 Remote Procedure Call (RPC)

Some current technologies that meet the needs of the UICC API are not designed to allow RPC. Future alternative technologies may be able to support this. It is seen as a future requirement of UICC API when interacting with terminal based execution environments.

13 Enhanced Runtime Environment

For a UICC that supports multiple logical interfaces based on TS 102 600 [6], TS 102 622 [4] and TS 102 483 [7] a runtime environment, called the enhanced UICC API framework, allowing applications concurrent access to these multiple interfaces, may be implemented. This runtime environment shall have the following characteristics.

13.1 Interworking between multiple hardware and logical UICC/terminal interfaces

A UICC based on a combination of TS 102 221 [1], TS 102 600 [6], TS 102 622 [4], TS 102 483 [7] and TS 102 484 [8] can have multiple active logical interfaces based on different protocol layers (APDU, TCP, UDP, HCI) with the terminal. The enhanced UICC API framework shall manage all the communication over these multiple interfaces concurrently and independently from each other. In order to handle these logical interfaces independently, the UICC API framework shall be based on a technology allowing concurrent access to the interfaces. The enhanced UICC API framework is responsible to manage the communication between the terminal and the different applications via these logical interfaces. A reset of one of the interfaces, either hardware or logical, shall not affect the communication via the other logical interfaces or hardware interfaces. The enhanced UICC API framework shall provide information about the establishment as well as the reset of interfaces to registered applications in the form of events.

13.2 Support for TCP and UDP

The functionality for applications to communicate concurrently via TCP/IP or UDP/IP according to TS 102 483 [7] shall be provided by an enhanced UICC API framework. The enhanced UICC API framework shall provide the functionality to applications to communicate over TCP/IP in server and client mode.

The enhanced UICC API framework shall provide a generic and extendable way to retrieve APIs for the different types of the Internet protocol family as well as support any further releases of TS 102 483 [7].

The enhanced UICC API framework shall be able to concurrently activate applications upon incoming TCP/UDP requests that are listening on a specific TCP/UDP port. The UICC API framework shall be able to manage several incoming and outgoing requests via TCP/UDP concurrently. The UICC API framework shall provide the functionality to establish a secure communication on this specific TCP/UDP port according to TS 102 484 [8].

13.3 Support for HTTP

The functionality for applications to receive and respond to HTTP requests shall be provided by a servlet engine that is part of the enhanced UICC API framework. Applications that deal with the HTTP protocol are called Web applications. This servlet engine shall be able to activate concurrently Web applications upon request to a URI. Web applications shall be identified by a URI, see OMA [9]. The URI identifying a Web application must be unique for the UICC on which this Web application is loaded.

13.4 Support for Card Application Toolkit (CAT)

The enhanced UICC API framework shall provide the functionality to receive Toolkit events and send Proactive commands as described in TS 102 223 [2].

13.5 Secure communication

The enhanced UICC API framework shall provide the means to establish a secure channel, by implementing the TLS application to application secure channel, over TCP/IP according to TS 102 484 [8]. It shall be possible to assign at least one TCP port, as an end point for a secure channel, per application. It shall be possible for applications to open a secure channel in server-listen mode or in client mode. It shall be possible that several secure channel sessions are active concurrently at the same time.

The enhanced UICC API framework shall provide the means to perform the following operations:

- the key agreement for the secure channel;
- the secure channel setup.

The enhanced UICC API framework shall provide the means to manage secure communication for the APDU application to application protocol according to TS 102 484 [8].

13.6 Events

The enhanced UICC API framework shall support an extendable event framework, that allows the definition of events that can be raised by the platform (UICC, Interfaces, Frameworks) or by applications deployed in the UICC. The enhanced UICC API framework shall provide a mechanism to applications to subscribe and unsubscribe to these events.

13.7 Access to the enhanced UICC API framework

The enhanced UICC API framework shall support a permission based security mechanism as a means to restrict access to the features of the framework, as requested by TS 102 412 [10].

13.8 Inter-application communication

The enhanced UICC API framework shall provide a mechanism allowing applications to share data in a secure and authenticated way.

13.9 Backward compatibility

The enhanced UICC API framework shall provide the means allowing existing applications, based on the Rel-7 or earlier API specifications, to be deployed on new cards implementing the enhanced UICC API framework.

Annex A (informative): Change history

This annex lists all change requests approved for the present document by ETSI SCP.

Date	Meeting	TC SCP Doc.	CR	Rv	Cat	Subject/Comment	Old	New
2004-11	SCP#19	N4-040456	001			Clarification for non-specific references	6.0.0	6.1.0
2005-06	SCP#21	SCP-050157	002		F	ISO/IEC update	6.1.0	6.2.0
2005-06	SCP#21	SCP-050157			B	Requirements for large file support by the API	6.1.0	6.2.0
2005-12	SCP#23	SCP-050521	003		B	Requirements for system events	6.2.0	7.0.0
		SCP-050522	004	1	B	UICC API accessibility Requirement		
2008-07	SCP#38	SCP-080347	005		B	Addition of requirements for a Contactless API	7.0.0	8.0.0
2009-01	SCP#40	SCP-090055	006		B	Requirements for an enhanced Runtime Environment	8.0.0	9.0.0
2010-03	SCP#44	SCP(10)0047	007		B	Addition of contactless low-level API functionality	9.0.0	10.0.0
2011-03	SCP#48	SCP(11)0130	008		B	Definition of a new API allowing application to manage Secure messaging over HTTP(S)	10.0.0	11.0.0
2011-03	SCP#48	SCP(11)0131	009		B	Addition of requirements for M2M applications	10.0.0	11.0.0
2009-04	SCP#41	SCP-090176	-		B	Requirements for the Secure Channel API	11.0.0	11.1.0
2011-09	SCP#52	SCP(11)0325	010		C	Update of requirements for M2M applications	11.0.0	11.1.0

History

Document history		
V11.0.0	July 2011	Publication
V11.1.0	December 2011	Publication