ETSI TS 102 588 V12.0.0 (2019-09)

Smart Cards;
Application invocation
Application Programming Interface (API)
by a UICC webserver for Java Card™ platform
(Release 12)

Reference
RTS/SCP-T102588v1200

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

Essential patents

IPRs essential or potentially essential to normative deliverables 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 (https://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.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

Foreword

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

The contents of the present document are subject to continuing work within TC SCP and may change following formal TC SCP approval. If TC SCP modifies 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.

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

1 Scope

The present document defines an API that allows a UICC based SCWS defined by OMA to forward HTTP requests to an Applet and to receive the response from the Applet. It also defines an API for the Applet to register and unregister to the SCWS.

2 References

2.1 Normative 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 referenced document (including any amendments) applies.

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

- 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.

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

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

[1] IETF RFC 2616: "Hypertext Transfer Protocol -- HTTP/1.1".

NOTE 1: Available at http://www.ietf.org/rfc/rfc2616.txt.

NOTE 2: IETF RFC 2616 has been obsoleted by IETF RFC 7230, IETF RFC 7231, IETF RFC 7232, IETF RFC 7233, IETF RFC 7234, IETF RFC 7235.

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

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

[4] OMA-TS-Smartcard-Web-Server-V1-0-20070209-C: "Smartcard-Web-Server".

[5] ORACLE: "Application Programming Interface, Java Card™ Platform, 3.0.1 Classic Edition".

[6] ORACLE: "Runtime Environment Specification, Java Card™ Platform, 3.0.1 Classic Edition".

[7] ORACLE: "Virtual Machine Specification Java Card™ Platform, 3.0.1 Classic Edition".

NOTE: SUN Java Card Specifications can be downloaded at http://java.sun.com/products/javacard.

[8] ETSI TS 101 220: "Smart Cards; ETSI numbering system for telecommunication application providers".

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

[10] ETSI TS 102 221: "Smart Cards; UICC-Terminal interface; Physical and logical characteristics".

2.2 Informative 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 referenced document (including any amendments) applies.

- 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.

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

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 Definition of terms, symbols and abbreviations

3.1 Terms

Void.

3.2 Symbols

Void.

3.3 Abbreviations

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

AID	Application IDentifier
API	Application Program Interface
CAT	Card Application Toolkit
CAT_TP	Card Application Toolkit Transport Protocol
FFS	For Further Study
HTTP	HyperText Transfer Protocol
JCRE	Java Card™ Run-time Environment
OMA	Open Mobile Alliance
SCWS	Smart Card based Web Server

NOTE: According to OMA specifications [3] and [4].

URI	Uniform Resource Identifier

4 Description

4.1 Architecture

The present document describes an API and a SCWS Runtime Environment that enables Java Card™ platform based applets, defined in [5], [6], [7], to register to and unregister from an SCWS implemented in the UICC, defined by OMA in [3] and [4].

The API enables a registered Applet to receive an incoming HTTP request that is forwarded by the SCWS. The API provides the necessary methods to allow registered Applets to respond with a correctly formatted HTTP response to the SCWS. The API provides means to the Applet to access the HTTP header data and the content of the HTTP request, to send specific HTTP status values, and to set the content of the HTTP response.

The HTTP request and response are defined in the Hypertext Transfer Protocol - HTTP/1.1 [1].

This API allows application programmers to extend the functionality of the SCWS defined by OMA in [3] and [4].

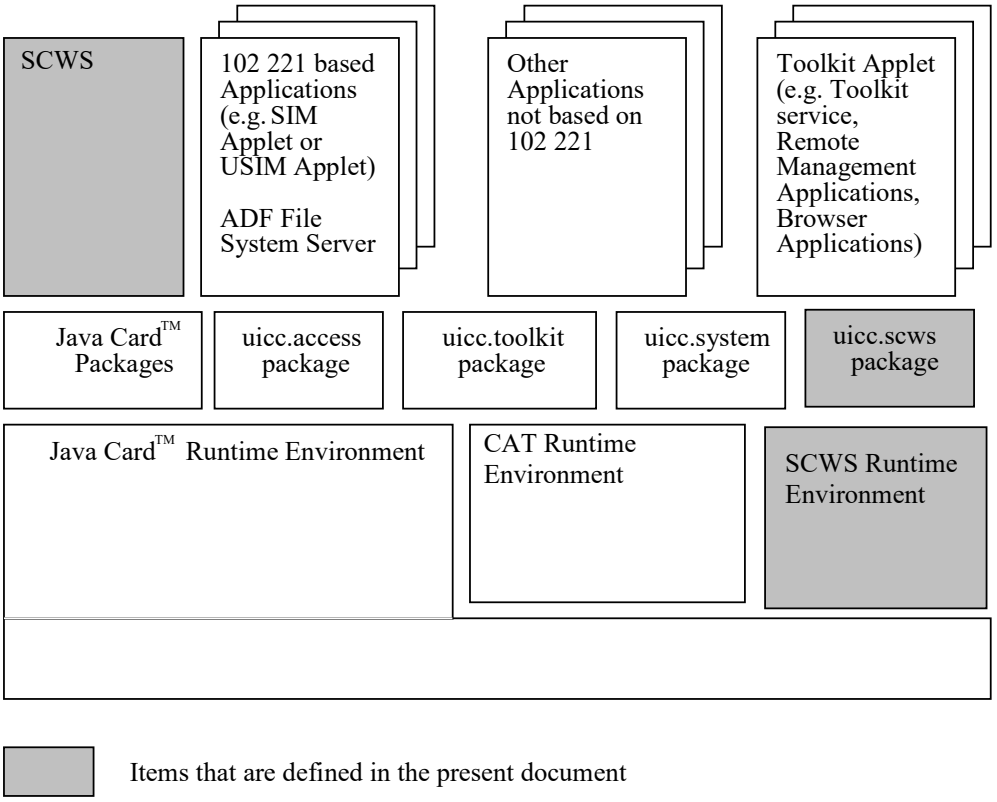

Figure 1

Smart Card based Web Server (SCWS): handles HTTP request as defined by OMA in [3] and [4] and provides a mechanism to the Applet for the registration.

SCWS Runtime Environment: extensions to the Java Card™ platform described in [5], [6], [7] and the CAT Runtime Environment described in ETSI TS 102 241 [2] to facilitate the communications between Applets and the SCWS.

Applet: these derive from *javacard.framework.Applet* and provide the entry points: *process, select, deselect, install* as defined in the " Runtime Environment Specification, Java Card™ Platform, 3.0.1 Classic Edition " [6].

Registry of the SCWS: is provided as a JCRE entry point object defined in [6], and provides an interface to the Applet to pass a name to the SCWS for registration and deregistration. The registry is part of the SCWS Runtime Environment.

SCWS API: consists of the package *uicc.scws*, provides the methods to register and deregister, to receive HTTP requests and to provide the content of the HTTP response.

4.2 Registration and deregistration

The registration of Applets to the SCWS enables the server to invoke a specific applet when it has received an HTTP request. Applet Instances can register with a name to the SCWS.

The mapping of the HTTP request to the name of the applet is described by OMA in [3] and [4] by the use of administrative commands {[FFS] other non-HTTP based mechanism.}. It is not possible to register several Applets under the same name to the SCWS. It is possible for an Applet to register several times with different names to the SCWS.

The Applet can also deregister from the SCWS. When the Applet deregisters from the SCWS the mapping information is deleted from the registry.

If an Applet is deleted then the registration information in the SCWS Registry is deleted by the SCWS Runtime Environment.

If the Applet is in a non-selectable state, its registration to the SCWS is still valid.

4.3 Invocation and Retrieval of Data

The SCWS invokes the Applets according to the mapping information.

If the complete request fits into the *HTTPRequest* buffer, the Applet is invoked when the complete HTTP request has been received by the SCWS.

If chunked transfer encoding is used and/or the HTTP request does not fit into the *HTTPRequest* buffer, the Applet is invoked when a certain amount of data was received by the SCWS and not yet retrieved by the applet. This amount may correspond with one chunk of data if it fits into the *HTTPRequest* buffer. The buffer management and invocation frequency of the Applet by the SCWS and the *HTTPRequest* buffer size are implementation specific.

The content of the buffer of the *HTTPRequest* is only changed between invocations of the Applet. That means the Applet can read from the buffer multiple times during the same invocation and get the same data. Only an Applet that is in selectable state can be invoked by the SCWS.

If processing of an HTTP request ends without any invocation of the flush() method and without throwing an exception the SCWS shall finalize the response and send it.

4.4 Transfer of response data

There are two transfer modes defined for the SCWS API: "fixed buffer size mode" and "chunked mode".

The API offers a method to switch between transfer modes. This method shall be called before calling *finalizeHeader()* and before the first call of *appendContent()*.

The default transfer mode is "fixed buffer size".

The header attributes ("Content-Length: xxx" and "Transfer-Encoding: chunked") will be set according to the active transfer mode by the SCWS runtime environment. The Application is not supposed to set these attributes.

The SCWS runtime environment is not required to enforce this policy. The behaviour of the SCWS runtime environment is undefined if the application manipulates the header attributes for content length and transfer encoding.

In "fixed buffer size mode" an exception will be thrown by *appendContent()* if the buffer size would be exceeded.

In "fixed buffer size mode" no data are sent out before the application has called the *flush()* method, subsequent calls are permitted but have no effect.

In "chunked mode" a call of *flush()* sends all data in the response buffer. If there are no data in the response buffer no data will be sent.

If a call of *appendContent()* exceeds the buffer size in "chunked mode" the data in the response buffer will be sent implicitly.

In chunked mode no response data shall be lost. Therefore, if the total size of the data in the response buffer and the data added to it in appendContent() exceeds the size of the response buffer, the SCWS framework shall send several chunks of data after invocation of appendContent() if necessary.

Exceptions thrown by the invoked Applet or any uncaught exception occurring during its processing shall not be propagated to the terminal, and the SCWS shall follow these rules:

- In case of "chunked mode", some response data could have already been sent by the SCWS. In this case, no error status code shall be sent by the SCWS and the chunked data transfer shall be terminated according to HTTP 1.1 [1].

- In any other case, an error status code according to HTTP 1.1 [1] is sent by the SCWS. In case of "fixed buffer size mode", any response data that may have been added to the HTTP response (regardless if in the response header or body) shall be discarded.

4.5 Response header management

Response header fields can be provided by applications by using the *appendHeaderVariable()* methods.

The following headers shall be added by the SCWS if not provided by the application:

- Status line, indicating success status (200 - OK or 204 - No Content).
- Server field, containing "OMA Smart Card Web Server" or a customized string.
- If status code 200 is returned, Content-type field, indicating "text/plain".
- If status code 200 is returned, Content-length and Transfer-Encoding, according to clause 4.4.

4.6 ProactiveHandler and ProactiveResponseHandler

The rules about the availability of the *ProactiveHandler* and the *ProactiveResponseHandler* as defined in ETSI TS 102 241 [2] are extended according to the following rules:

The *ProactiveHandler* shall be available for applets that are invoked by an incoming HTTP request (i.e. by one of the methods *doGet()*, *doPost()*, *doDelete()*, *doHead()*, *doOptions()*, *doPut()*, and *doTrace()*) in the same way as if it would be available for an applet which was triggered by the method *processToolkit()* in the current card state. If available, the *ProactiveHandler* shall be available for these applets until the termination of the method that was invoked by the incoming HTTP request.

The availability of the *ProactiveResponseHandler* shall depend on the availability of the *ProactiveHandler* as defined in ETSI TS 102 241 [2], except that it is available until the termination of the method that was invoked by the incoming HTTP request.

Applets implementing the ToolkitInterface as defined in ETSI TS 102 241 [2] in addition to the SCWS interface will be triggered in the *processToolkit()* method upon reception of any Toolkit event.

Implementation dependent on a central CAT_TP multiplexing application as defined in ETSI TS 102 225 [9] may be present in the card. It shall not block the *ProactiveHandler* when an applet is invoked by an incoming HTTP request.

Annex A (normative): Application invocation API by a UICC Webserver for the Java Card™ platform

The source files for the (102588_Annex_A_Java.zip and 102588_Annex_A_HTML.zip) are contained in ts_102588v120000p0.zip, which accompanies the present document.

Annex B (normative): Application invocation API by a UICC Webserver for the Java Card™ platform

The export files for the uicc.scws package (102588_Annex_B_Export_Files.zip) are contained in ts_102588v120000p0.zip, which accompanies the present document.

NOTE: See the "Virtual Machine Specification Java Card™ Platform, 3.0.1 Classic Edition" [7].

Annex C (normative): Application invocation API by a UICC Webserver for the Java Card™ platform package version management

Table C.1 describes the relationship between each ETSI TS 102 588 specification version and its packages AID and Major, Minor versions defined in the export files.

Table C.1

uicc.scws package		
ETSI TS 102 588	**Major, Minor**	**AID**
7.2.0	1.0	A000000009 0005 FFFF FFFF 89 14 000000
7.3.0	1.1	
11.0.0	2.0	

The package AID coding is defined in ETSI TS 101 220 [8]. The uicc.scws package AID is not modified by changes to Major or Minor Version.

The Major Version shall be incremented if a change to the specification introduces byte code incompatibility with the previous version.

Annex D (informative): Change history

Meeting	Plenary Tdoc	WG Tdoc	CR	REV	CAT	SUBJECT	Resulting Version
SCP#31	SCP-070276	SCPt070646	001		F	Correct constant values	7.1.0
SCP#33	SCP-070420	SCPt070861	002		F	Fix of missing exceptions and wrong constant values	7.2.0
SCP#33	SCP-070420	SCPt070862	003		F	Define a new constant to retrieve the query part of an URI	7.2.0
SCP#33	SCP-070420	SCPt070863	004		D	Editorial correction	7.2.0
SCP#33	SCP-070420	SCPt070864	005		F	Add an exception in the HTTPResponse.sendError() and HTTPResponse.setContentType() methods	7.2.0
SCP#33	SCP-070420	SCPt070867	006		F	Protected SCWS exception (constructor of exception made public instead of private)	7.2.0
SCP#33	SCP-070420	SCPt070874	007		F	Specify default headers fields added by the SCWS in HTTP responses	7.2.0
SCP#35	SCP-080030	SCPt071043	008		F	Adding Reason BUFFER OVERFLOW to some SCWS methods	7.2.0
SCP#35	SCP-080030	SCPt071045	009		F	Adding reason HTTP_RESPONSE_ALREADY_SENT to getRemainingResponseBufferSize method	7.2.0
SCP#35	SCP-080030	SCPt071046	010		F	Adding SCWS Exception to method getContentType	7.2.0
SCP#35	SCP-080030	SCPt071048	011		F	Change constant value CONTENT_TYPE_UNKNOWN to avoid value overlapping	7.2.0
SCP#35	SCP-080030	SCPt071049	012		F	Specification of chunked buffer mode behaviour in case of exception	7.2.0
SCP#35	SCP-080030	SCPt071053	015		F	Align values returned by copy method with the one defined in Java Card™ or UICC APIs	7.2.0
SCP#35	SCP-080030	SCPt071055	016		F	Introduction of a reset method to the HTTPResponse interface	7.2.0
SCP#35	SCP-080030	SCPt071059	014	1	F	Delete the URI FRAGMENT TAG constant	7.2.0
SCP#35	SCP-080030	SCPt071060	013	1	D	Fix the "OPTION" method documentation and name	7.2.0
SCP#42	SCP-090231	SCPt090276	018		F	Adding missing exceptions in method signatures	7.3.0
SCP#41	SCP-090125	SCPt090091	017		B	Availability of the ProactiveHandler and the ProactiveResponseHandler	8.0.0
SCP#45	SCP(10)0132	SCPTEC(10)0195	019		A	Corrections in the Java files	8.1.0
SCP#45	SCP(10)0183	-	020		F	Change reference from "Java Card™ 2.2.2" to "Java Card™ 3.0.1 Classic Edition"	9.0.0
SCP#46	SCP(10)0270	SCPTEC(10)0479	024		A	Addition of HTTP status code	9.1.0
SCP#46	SCP(10)0267	SCPTEC(10)0488	025		A	Clarification of availability of ProactiveHandler	9.1.0
SCP#47	SCP(11)0045	SCPTEC(10)0489r3	026		F	Clarify behaviour if an exception occurs after setting response header fields or response status code	7.5.0
SCP#47	SCP(11)0046	SCPTEC(10)0489r3	027		A	Clarify behaviour if an exception occurs after setting response header fields or response status code	8.3.0
SCP#47	SCP(11)0047	SCPTEC(10)0489r3	028		A	Clarify behaviour if an exception occurs after setting response header fields or response status code	9.2.0
SCP#55	SCP(12)000100r1		029	1	C	Clarification of HTTPResponse.appendContent() in chunked mode	11.0.0
SCP#57	SCP(12)000269r1		030	1	C	Support for HTTP request when request size exceeds buffer size	11.0.0
-	-		-	-	-	Automatic upgrade to Release 12	12.0.0

History

Document history		
V12.0.0	September 2019	Publication