Smart Card Web Server Stepping Stones

Version 1.0

Published by simalliance now Trusted Connectivity Alliance

December 2009

Index

Figure index

1 Introduction

It was 1999 when most Smart Card vendors started to provide Java Card specifications for supplying platforms to SIM Toolkit services. Since then, Java Card and SIM toolkit technologies continually evolved to define new services and new capabilities. And since then, SIMalliance ensured that every technological improvement was introduced preserving one of the most important values: interoperability.

Today, SIM card technologies are running fast: SCWS and NFC are changing the role of the SIM in the mobile ecosystem, but they are changing also and increasing the parties interacting with the SIM: web services and third party operator platform shall increase their interaction with the card.

But increased complexity and increased number of parties does not provide any value without interoperability: if companies are not safe to deploy a service on the SIM industry because they consider interoperability a risk, they will just avoid deploying services!

With this document, the *Smart Card Web Server Interoperability Stepping Stones*, SIMalliance intends to continue the successful series of *Java Card interoperability Stepping Stones* evolving it toward the Web paradigm.

Completing OMA and ETSI's work of releasing specifications and test suites, the purpose of this guide is to provide developers with information concerning SIM constraints and a common interpretation of the standards for the members of the SIMalliance that contributed to this document.

The target audience of this guide is Network Operators, Wireless Service Providers and anyone interested in interoperable Java Card applet development.

1.1 Acknowledgements

A lot of people have been contributing the document for years, but a special thank goes to the people active in the SM Alliance Interoperability Working Group or that have been very active in the past, including:
Laurence Bringer, Cyril Barras (Gemalto), René Huxol, Lars Schnake (Sagem Orga), Jimmy De Britto, Aurelian Raboisson (Oberthur Technology), Mariano Concilio, Amedeo Veneroso (ST Incard), Michael Schnellinger, Nils Nitsch (G&D).

Reference Documentation

Entity	Reference	Title
ETSI (www.etsi.org)	TS 101 220 Release 7	ETSI Numbering System for Telecommunications; Application Providers (AID)
	TS 102 221 Release 7	UICC-Terminal interface; Physical and logical characteristics
	TS 102 223 Release 7	Card Application Toolkit (CAT)
	TS 102 225 Release 6	Secured packet structure for UICC based applications
	TS 102 226 Release 6	Remote APDU structure for UICC based applications
	TS 102 241 Release 7	Java Card™ API for the UICC
3GPP (www.3gpp.org)	TS 31.115 Release 6	Secured packet structure for (Universal) Subscriber Identity Module (U)SIM Toolkit applications
	TS 31.116 Release 6	Remote APDU Structure for (Universal) Subscriber Identity Module (U)SIM Toolkit applications
3GPP2 (www.3gpp2.org)	C.S0078	Secured packet structure for CDMA Card application toolkit(CCAT) applications
	C.S0079	Remote APDU Structure for CDMA Card application toolkit (CCAT) applications
GlobalPlatform (www.globalplatform.org)		Global Open Platform Card Specification, Version 2.2 (with Amendment B)
Sun Microsystems (http://java.sun.com/products/javacard/)		Java Card 2.2.2 Virtual Machine Specification
		Java Card 2.2.2 Runtime Environment (JCRE) Specification
		Java Card 2.2.2 Application Programming Interface
		Java Card Applet Developer's Guide, Java Card Version 2.2.2
IETF (www.ietf.org)	RFC 2616	Hypertext Transfer Protocol – HTTP 1.1
	RFC 2818	Hypertext Transfer protocol over TLS protocol
	RFC 4279	Pre-shared key Cipher suites for Transport Layer Security (TLS)
	RFC 2246	The TLS Protocol
WWW Consortium (www.w3.org)		Hypertext mark-up language (HTML 4.0.1)

2 Abbreviations

ADN	Abbreviated Dialling Number
AES	Advanced Encryption Standard
AID	Application Identifier
APDU	Application Protocol Data Unit
API	Application Programming Interface
ASCII	American Standard Code For Information Interchange
BER	Basic Encoding Rules
BIP	Bearer Independent Protocol
CAT	Card Application Toolkit
CBC	Cipher Block Chaining
CR	Carriage Return
DES	Data Encryption Standard
DF	Dedicated File
EF	Elementary File
ETSI	European Telecommunications Standards Institute
FTP	File Transfer Protocol
GP	Global Platform
GPL	GNU General Public License
GPRS	General Packet Radio Service
GSM	Global System for Mobile communications
HSP	High Speed Protocol
HTML	Hypertext Markup Language
HTTP(S)	Hypertext Transfer Protocol (Secure)
ICC	Integrated Circuit Card
IETF	Internet Engineering Task Force
IP	Internet Protocol
JC	Java Card
LF	Linefeed
LPGL	GNU Lesser General Public License
LWP	Lightweight Protocol
MD5	Message Digest Algorithm 5
ME	Mobile Equipment
MIME	Multipurpose Internet Mail Extensions
OMA	Open Mobile Alliance
OTA	Over The Air
PoR	Proof of Receipt
PPS	Protocol Parameter Selection
PSK	Pre Shared Key
RAM	Remote Applet Management
RFC	Request For Comments
RFM	Remote File Management
SCWS	Smartcard Web server
SIM	Subscriber Identity Module
SHA	Secure Hash Algorithm
SMS	Short Message Service
SMTP	Simple Mail Transfer Protocol
SSL	Secure Sockets Layer
SW	Status Word
TAR	Toolkit Application Reference
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TLV	Tag Length Value
TS	Technical Specification
UICC	Universal Integrated Circuit Card
URI	Uniform Resource Identifier

URL	Uniform Resource Locator
USB	Universal Serial Bus
USIM	Universal Subscriber Identity Module
WWW	World Wide Web

3 Definitions

Chunked mode	HTTP transfer mode without knowing the overall length of the HTTP request/response in advance. HTTP message body is divided into multiple chunks having each chunk's length at it's beginning. End of message is indicated by special zero-length chunk and two CR LF.
Global Platform API	The Global Platform API provides services to Applications (e.g. cardholder verification, personalization, or security services).
GZIP	Compression algorithm for files under GPL
HTTP Pipelining	Sending multiple HTTP requests/responses without closing the client ←→ server connection. The HTTP responses must have the same sequence as the previously received HTTP requests.
HTTP Request	A binary request using the HTTP protocol issued by a client to a server
HTTP Response	A binary response using the HTTP protocol send from a server to a client due to a previous HTTP Request
Integrated Circuit Card	The most general term for a smart card is "ICC". It is always a physical and logical entity either a SIM or a UICC.
Issuer Security Domain	The representative entity of the card issuer. It provides support for control, security and communication requirements of the card issuer.
Over-The-Air	Technology which uses the mobile network features to download data to the UICC.
Remote Application Management	Remote Application Management applications are OTA interfaces to the Issuer Security Domain and other Security Domains.
Security Domain	A special application that supports a secure communication between an Application Provider's application and off-card entities during its personalization phase and runtime.
Subscriber Identity Module	"SIM" is the term that defines the ICC for a 2G card, there is no distinction between the physical and logical entity and the application itself. In a UICC, the "SIM" is an application. If it is active, the UICC is functionally identical to a 2G card.
Toolkit Application Reference	Unique identification for Toolkit applications when using Over-The-Air functionality.
Universal Integrated Circuit Card	The UICC is the physical and logical platform for the USIM. It can, at least, contain one USIM application and may additionally embed a SIM application.
Universal Subscriber Identity Module	The USIM is not a physical entity. It is a purely logical application on a UICC. It does only accept 3G commands and therefore it is not compatible with a 2G ME. The USIM may provide mechanisms to support 2G authentication and key agreement to allow a 3G ME to access to a 2G network. (see 3GPP TS 31 102)
Web Browser	Software for viewing documents and data received from web servers using the HTTP protocol
Web Server	Entity answering requests of one or more clients using the HTTP protocol

4 HTTP/1.1 (Hypertext Transfer Protocol)

4.1.1 HTTP overview

The SCWS implements HTTP 1.1 protocol defined by the IETF in the RFC 2616. HTTP is an application-level protocol for distributed, collaborative, hypermedia information systems.
There are many use of the HTTP protocol but the most famous one is the dialog between a HTTP browser and a HTTP server, HTTP is the language of internet:

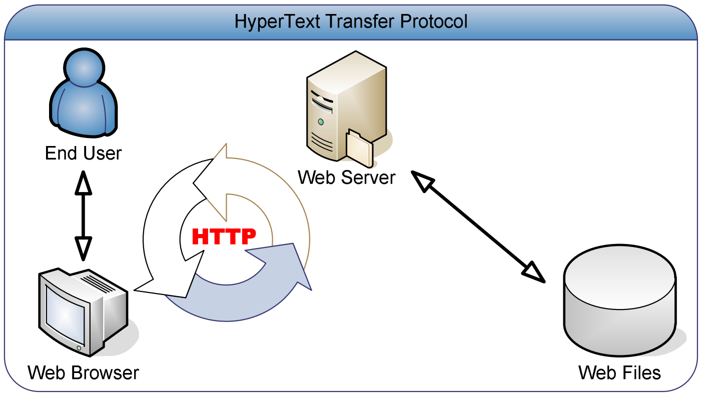


Figure 1 – The HTTP

Web content lives on web servers which are HTTP servers. Those servers store web resources such as HTML pages, images, video, music … and provide those resources to web clients.
Web resources are not only static files inside the web server file system they can also be applications which generate content to the web client:

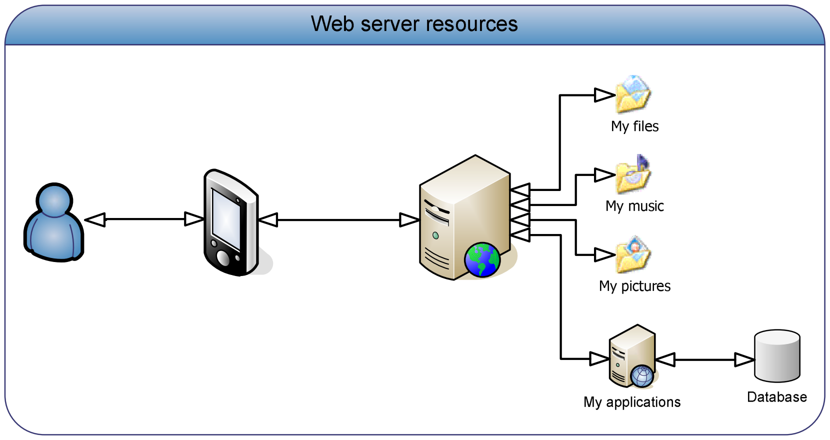


Figure 2 – A traditional web server

4.1.2 URIs and URLs

Resources on a web server are identified using Uniform Resource Identifiers (URI). In most typical uses a user will use a URI subset called Uniform Resource Locators (URL) which is the specific location of a resource on a web server. URLs are used to identify a page or a website. They are commonly referred as Web Address.

A URI is specific to the scheme (HTTP, FTP, SMTP …) used to access the resource. For hierarchical URIs the generic format is the following:

scheme : hier_part

For HTTP scheme the URI parts contains the following information:

– scheme: http
– hier_part: (net_path | abs_path) ["?" query]
– net_path: //<authority>[abs_path]
– abs_path: /<path_segments>
– authority: <host>[:<port>]

Leading a HTTP URI to have the following format:

http://<host>[:<port>][/<path_segments>][?query]

URI parameter	Description	BIP SCWS	TCP SCWS	Case sensitive
http	The HTTP scheme	-	-	No
host	The hostname or dotted IP address of the web server.	127.0.0.1	Smartcard IP address	No
port	Port of the HTTP protocol	3516	80 as this is HTTP default port it can be optional.	-
path_segment	Hierarchical path to the resource	-	-	Yes
query	Sequence of <name>=<value> pairs separated by the '&' character.	-	-	Yes/No depends on the resource interpretation

The SCWS shall support the following constraints:

- The minimum URI length to support is 1024 bytes
- The minimum abs_path to support is 128 bytes

URIs are using a restricted character set based on the US-ASCII characters set and are separated into 3 subsets:

URI characters = reserved | unreserved | escaped

- Reserved characters are used for URI format. Those characters cannot be used in a URI component.

Reserved character	Corresponding escaped character	Description
;	%3B	Reserved as the parameter delimiter
/	%2F	Reserved as path delimiter
?	%3F	Reserved as query delimiter
:	%3A	Reserved as scheme or port delimiter
@	%40	Reserved as a delimiter for some schemes
&	%26	Reserved as a delimiter for some schemes
=	%3D	Reserved as a delimiter for some schemes
+	%2B	Reserved
$	%24	Reserved
,	%2C	Reserved

– Unreserved characters can be used in any URI component.

Unreserved character	Description
A ➔Z	Upper case letters
a ➔ z	Lower case letters
0 ➔ 9	Digits
-	Punctuation mark
_	Punctuation mark
.	Punctuation mark
!	Punctuation mark
~	Punctuation mark
*	Punctuation mark
`	Punctuation mark
(	Punctuation mark
)	Punctuation mark

– Escaped characters are a set of special characters that cannot be used in a URI component as they could be misinterpreted by some specific scheme URI parser.

Escaped character	Escaped value	Description
ASCII control characters (0x00 ➔ 0x1F and 0x7F)	%00 ➔ %1F %7F	ASCII control characters
space	%20	Space characters
<	%3C	Scheme specific delimiter
>	%3E	Scheme specific delimiter
#	%23	Scheme specific delimiter
%	%25	Escape indicator
"	%22	Scheme specific delimiter
{	%7B	Unwise/Unsafe
}	%7D	Unwise/Unsafe
|	%7C	Unwise/Unsafe
\	%5C	Unwise/Unsafe
^	%5E	Unwise/Unsafe
[	%5B	Unwise/Unsafe
]	%5D	Unwise/Unsafe
`	%60	Unwise/Unsafe

– Characters from the *reserved* and *escaped* sets must be escaped when they are used in any URI component.
– Characters from the *unreserved* set are equivalent to their escaped encoding.

The escape encoding of an octet is a character triplet consisting of the % character followed by the 2 hexadecimal digits of the ASCII code.

The base URLs

The base URL of a resource is everything up to and including the last slash in its pathname

Absolute URL	Base URL
http://smartcardwebserver.com/	http://smartcardwebserver.com/
http://smartcardwebserver.com/html/	http://smartcardwebserver.com/html/
http://smartcardwebserver.com/html/index.html	http://smartcardwebserver.com/html/
http://smartcardwebserver.com/foo/form.html?test	http://smartcardwebserver.com/foo/

The relative URLs

Since the URLs are used to locate everything, they can get very long. There is a way to abbreviate them in the form of relative URLs. Relative URLs identify a resource relatively to its context.

For example the resource http://WebReference.com/html/about.html has to refer to the document http://WebReference.com/html/links.html. What can be used is the relative URL links.html. This relative URL will be added to the base URL.

Most of the time, the absolute URL can be rebuilt by concatenating the base URL and the relative URL. But there can be exceptions:

- A directory called .. (two periods) in a relative URL indicates the *parent directory*
- A directory called . (one period) refers to the current directory
- A relative URL that begins with / (a slash) always replace the entire pathname of the base URL
- A relative URL that begins with // (two slashes) always replaces everything from the hostname onwards

Relative URLs are much more important than just saving time typing the URL: they point to a resource *relative to their context*. This is the main reason why they are used. URIs may not always be used in the same context (for example when copying a website into another one).

The following table shows relative URLs which are all assumed to have the base URL http://smartcardwebserver.com/html/

Relative URL	**Absolute URL**
about.html	http://smartcardwebserver.com/html/about.html
example1/	http://smartcardwebserver.com/html/example1/
example1/1.html	http://smartcardwebserver.com/html/ example1/1.html
/	http://smartcardwebserver.com/
//www.internet.com/	http:// www.internet.com/
/example/	http://smartcardwebserver.com/example/
../	http://smartcardwebserver.com
../example	http://smartcardwebserver.com/example/
../../../	http://smartcardwebserver.com/
./	http://smartcardwebserver.com/html/
./about.html	http://smartcardwebserver.com/html/about.html

4.1.3 HTTP Message Format

HTTP messages are simple messages flowing downstream inbound to the origin server or outbound back to the client. Those messages are composed of 3 parts:

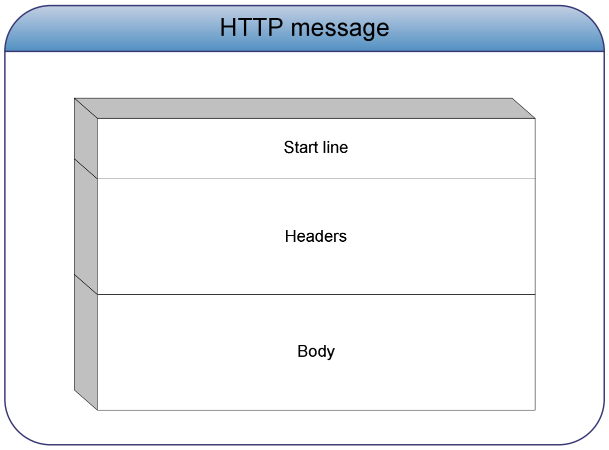


Figure 3 – HTTP message

HTTP considers 2 messages: HTTP requests and HTTP responses

Message part	Request	Response
Start line	Request line: indicates what to do	Response line: indicates what happened
Headers	General headers, request headers and entity headers	General headers, response headers and entity headers
Body	Entity body	Entity body

- Start line indicates what the server has to do or what happened on the server. The start line ends with a CRLF sequence.

- Headers contain meta-information about the request or the response. 4 subsets exist:

  - General headers: Appear in both the request and the response (cache control …)
  - Request headers: Provide meta-information on the request (content negotiation …)
  - Response headers: Provide meta-information on the response (ETAG …)
  - Entity headers: Provide meta-information on the entity (size, content …)

A header has the following formats:

Header name **:** **LWS** **Header value** **LWS** **CRLF**

The header part of a message must comply with:

- Each header ends with a CRLF sequence.
- The header part of the message ends also with a CRLF sequence.
- A header may be split on several lines, each sub-line beginning with at least a <sp> or <tab> character.
- A header may be present multiple times. The resulting header value is the concatenation of all header values separated with commas.
- A header name is case-insensitive.
- A header value is case-insensitive?

- The entity body or entity is an optional chunk of data. Its presence depends on the request method value.

4.1.4 HTTP Requests

HTTP requests have the following format:

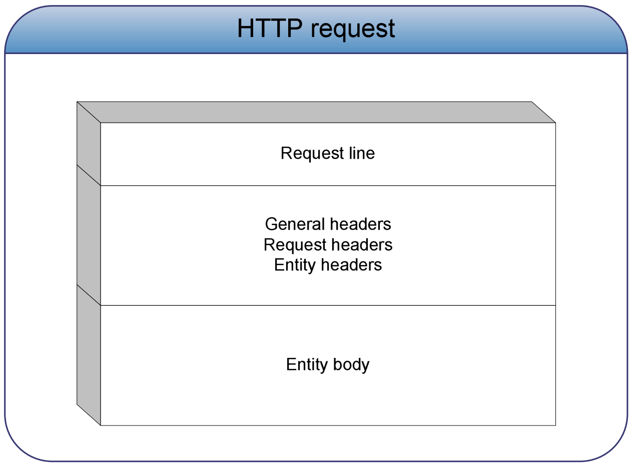


Figure 4 – The HTTP request

– The request line has the following format:

<Method> <Request-URI> <Version> CRLF

- HTTP defines eight methods indicating the desired action to be performed on the identified resource.

Supported methods:

Method	Description
GET	Requests a representation of the resource available at the given URI.
HEAD	Requests meta-information of the resource available at the given URI.
PUT	Uploads a representation of the resource at the given URI. This may result in the creation of a resource or the update of an existing one.
DELETE	Deletes the resource at the given URI.
POST	Submit data to the resource at the given URI.

Optional methods:

Method	Description
OPTIONS	Asks the server about its capabilities on the resource available at the given URI.
TRACE	Traces the request received by the server.
CONNECT	Requests a HTTP tunnel establishment.

Developer tip:
For security reasons *PUT* and *DELETE* methods may be restricted to administration purpose. If those methods are not supported for standard HTTP clients a Client Error 4xx status should be returned to the client.

Optional methods may not be supported by all SCWS implementations.

- The request-URI is an identifier for a resource.

- The version can be either HTTP/1.0 or HTTP/1.1.

– Headers in a HTTP request includes:

- General headers:

These are general purpose headers used by both servers and clients.

Header	Description	Used by SCWS
Cache-Control	Indicates cache parameter for the request	Yes
Connection	Indicates client connection state	Yes
Date	Indicates the request date	No
Pragma	Implementation specific directives	No
Trailer	Indicates that given headers are present in the trailer of a message encoded with a chunk encoding	No
Transfer-Encoding	Indicates the encodings used to transfer the request	Yes
Upgrade	Indicates the additional protocols supported by the client	No
Via	Indicates intermediaries between the client and the server	No
Warning	Indicates additional information about the status or transformation of the request	No

- Request headers:

Provide information about what the client is expecting to receive.

Header	Description	Used by SCWS
Accept	Indicates what content types the client accepts	No
Accept-Charset	Indicates what character sets the client accepts	No
Accept-Encoding	Indicates what content encoding the client accepts	No
Accept-Language	Indicates what languages the client accepts	No
Authorization	User credentials for HTTP authentication	Yes
Expect	Requests specific server behaviour (100-continue)	No
From	Indicates the user email address	No
Host	Indicates the authority	Yes
If-Match	Conditional request based on ETAGs	Yes
If-Modified-Since	Conditional request based on date	No
If-None-Match	Conditional request based on ETAGs	Yes
If-Range	Conditional request based on range and ETAGs	No
If-Unmodified-Since	Conditional request based on date	No
Max-Forwards	Proxy parameter	No
Proxy-Authorization	Proxy parameter	No
Range	Indicates what range the client request	No
Referrer	URI of the reference resource	No
TE	Indicated what transfer encoding the client accepts	No
User-Agent	Indicates the client user agent ID	No

- Entity headers:

Header	Description	Used by SCWS
Allow	List of methods supported for a resource	No
Content-Encoding	Content encoding of the entity	Yes
Content-Language	Content language of the entity	Yes
Content-Length	Content length of the entity	Yes
Content-Location	Content location of the entity	No
Content-MD5	MD5 of the entity	No
Content-Range	Range of data in the entity	No
Content-Type	Media type of the entity	Yes
Expires	Stale date of the entity	No
Last-Modified	Last modified date of the entity	No

– The entity body is optional and contains blocks of raw data. Its presence depends on the method of the HTTP request:

Method	Entity availability
GET	No
HEAD	No
PUT	Yes
DELETE	No
POST	Yes

Entity availability means that an entity may be present in the request. A null entity is allowed (Content-Length = 0).

4.1.5 HTTP Response

HTTP responses have the following format:

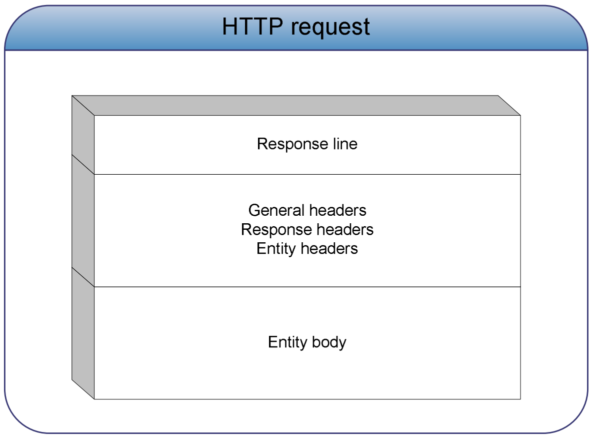


Figure 5 – The HTTP response

– The response line has the following format:

<Version> <Status> <Reason-phrase> CRLF

- Version may be HTTP/1.0 or HTTP/1.1
- The response includes a status and a reason phrase describing the result of the server operation. Status are separated in 5 groups:

Status group	Description
1XX	Informational
2XX	Successful operation
3XX	Redirection
4XX	Client error
5XX	Server error

The statuses are the following:

Group	Status	Reason phrase	Entity enclosed	Used by SCWS
Informational	100	Continue	No	May
	101	Switching protocols	No	May
Successful operation	200	OK	May	Yes
	201	Created	Should	Yes
	202	Accepted	Should	May
	203	Non-Authoritative Information	No	May
	204	No Content	No	Yes
	205	Reset Content	No	May
	206	Partial Content	Yes	May
Redirection	300	Multiple Choices	Should	May
	301	Moved Permanently	Should	May
	302	Found	Should	May
	303	See Other	Should	May
	304	Not Modified	No	Yes
	305	Use Proxy	No	May

Group	Status	Reason phrase	Entity enclosed	Used by SCWS
	307	Temporary Redirect	Should	May
Client error	400	Bad Request	No	Yes
	401	Unauthorized	No	Yes
	402	Payment Required	-	May
	403	Forbidden	No	Yes
	404	Not Found	No	Yes
	405	Method Not Allowed	No	Yes
	406	Not Acceptable	Should	May
	407	Proxy Authentication Required	No	May
	408	Request Timeout	No	Yes
	409	Conflict	Should	May
	410	Gone	No	May
	411	Length Required	No	Yes
	412	Precondition Failed	No	May
	413	Request Entity Too Large	No	Yes
	414	Request-URI Too Long	No	Yes
	415	Unsupported Media Type	No	May
	416	Request Range Not Satisfiable	No	May
	417	Expectation Failed	No	May
Server error	500	Internal Server Error	No	Yes
	501	Not Implemented	No	Yes
	502	Bad Gateway	No	May
	503	Service Unavailable	Should	Yes
	504	Gateway Timeout	No	May
	505	HTTP Version Not Supported	No	Yes

– Headers in a HTTP response includes:

- General headers: The general headers are the same as in the request.
- Response headers:

Provide information about what the client will receive:

Header	Description	Used by SCWS
Accept-Ranges	Acceptance of range request for a resource	May
Age	Age of the response	May
ETag	Current ETag linked to the resource	Yes
Location	Location of the resource	Yes
Proxy-Authenticate	Proxy authentication	May
Retry-After	Delay to retry the same request	May
Server	Server identification and version	Yes
Vary	Variance for server driven negotiation	May
WWW-Authenticate	HTTP authentication	Yes

- Entity headers:

Provide information about the entity enclosed in the response:

Header	Description	Used by SCWS
Allow	List of methods supported for a resource	Yes
Content-Encoding	Content encoding of the entity	Yes
Content-Language	Content language of the entity	Yes
Content-Length	Content length of the entity	Yes
Content-Location	Content location of the entity	May
Content-MD5	MD5 of the entity	May
Content-Range	Range of data in the entity	May
Content-Type	Media type of the entity	Yes
Expires	Stale date of the entity	May

Last-Modified	Last modified date of the entity	May

4.1.6 Cache Management

Caching is a new feature introduced in OMA SCWS 1.1 particular useful in SCWS technology as it allows to avoid transmitting a specific resource if it has already been stored inside the handset memory.

If the handset supports the cache management, every time a resource from the SCWS is retrieved it is stored in the cache and information (eTag) is associated to it.

When the handset needs again the same resource, it asks the card if it has changed by presenting the eTag of the previous retrieved resource. Then the SCWS can reply by saying that the resource hasn't been changed or that it has changed by providing the new eTag.

> **Developer tip:**
>
> For SCWSExtension usually the Response to the HTTP request is different every time the SCWSExtension is triggered. To force the handset to retrieve every time the response from the card, the SCWSExtension can add the header "Pragma: no-cache" to the HTTP Response.

4.1.7 Persistent Connection

HTTP persistent connection (also referred as HTTP keep Alive or HTTP connection reuse) is a process that uses the same TCP connection to send and receive multiple HHTP messages, in opposition to use one connection per single pair of request/response.

The main asset to use persistent connection is to improve the HTTP performances the advantages are:
- Less network traffic due to less TCP connection management (setting up and tearing down)
- Reduction of latency on requests, due to initial TCP handshake

4.1.8 Secure HTTP

There is currently one existent method of establishing a secure HTTP connection: the HTTPS URI scheme. The HTTPS URI scheme is still the dominant method of establishing a secure HTTP connection. Secure HTTP is notated by the prefix HTTPS:// instead of HTTP://.
"HTTPS:" is a URI scheme syntactically identical to the "http:" scheme used for normal HTTP connections, but which signals the browser to use an added encryption layer of SSL/TLS to protect the traffic.

4.1.9 Content types

The content type identifies the nature of a resource. It is coded according to the MIME types. The content types defined by MIME standards are also of importance outside of e-mail, such as in communication protocols like HTTP. MIME's use, however, has grown beyond describing the content of HTTP resource.

Multipurpose Internet Mail Extensions (MIME) is an Internet standard that extends the format of e-mail to support:

- Text in character sets other than ASCII
- Non-text attachments
- Message bodies with multiple parts
- Header information in non-ASCII character sets

This header indicates the Internet media type of the message content, consisting of a *type* and *subtype*, for example:
- Content-Type: text/plain

Through the use of the multipart type, MIME allows HTTP request to have different contents included in to body part. Such as:

- simple text messages using *text/plain* (the default value for "Content-type:")

• alternative content, such as a message sent in both plain text and another format such as HTML (*multipart/alternative* with the same content in *text/plain* and *text/html* forms)

• image, audio, video and application (for example, *image/jpg*, *audio/mp3*, *video/mp4*, and *application/msword* and so on)

Here some examples of MIME types supported .css, javascript...

In the SCWS the MIME type and the resource are just associated. Basically the SCWS 'supports' all MIME types. There is no check of the MIME type of a resource neither when a it is stored or returned.
Depending on the MIME type of an entity, the client (usually a browser) has to interpret it accordingly. When a resource is returned or stored on the SCWS there is no check on the MIME type.

4.1.10 Content encoding

The content encoding defines how the resource is stored on the web server. To be more precise it indicates the transformation (if any) that had been applied to an entity. This is used to compress, for example, a document without data loss.

The zip compression can be used to save space on the web server when storing a static resource. It also allows limiting the data exchange between the client and the server when retrieving the resource. It is the client responsibility to uncompress the content.

There are several ways to store a zipped resource:
- Either the resource is put already zipped in the card
- Or the card zips the content when receiving the resource

Usually the first solution is used, because it is less resources consuming. For GZIP compression there is no check on the "accept encoding" field of the client.

4.2 Transfer Encoding

The chunked transfer encoding allows splitting http messages in several parts. The body is sent in an unspecified number of blocks of data called chunks. Each chunk begins with the number of bytes of data. The last chunk must be an empty one (size set to 0).
This transfer encoding is used to send request slice by slice. That way the server may start computing the response before receiving the whole request. On the other side, the server can start sending the response before having processed it completely.

It is important to point that because the http message length may be not known at the sending of the first chunks, the length of the http message is not specified. The chunk size may be different one from another.

Example:

```
HTTP/1.1 200 OK
Content-Type: text/plain
Transfer-Encoding: chunked

28
This is the first chunk of the examples

24
and this is the second and last one

0
```

Interoperability trap:
The request chunked mode may not be supported by every implementation. It is recommended not to use it.

4.2.1 Recommendation of file size

To have a better user experience, it is recommended to keep the resources to display small. That way, it will reduce the wait before displaying the web page. Compression of the content can be a good alternative if supported by the client.

> **Interoperability trap**
> Depending on the implementation, the max file size can be limited. Some cards support the file up to 32KB. Try to keep the resources smaller than this size.

4.2.2 Authentication

The authentication process defined in the RFC 2617, allows a user to identify himself to a http server by providing a username and the corresponding password. If successful the server will grant the user access to his restricted resources.

Basically when a client tries to access protected resources, the web server will ask for a username and a password. If the pair username/password does not match, a HTTP 401 (*Unauthorized*) error code is sent.

Two authentication methods are defined in the RFC:

- The Basic mode: this method is simple and the username and password are sent practically unencrypted. It is recommended to use this mode in a secure session (https).

- The Digest mode: in this method is more complex because it involves more parameters. The password is never sent in plain text over the network. Indeed HA1 is sent, HA1 = MD5(username:realm:password)

The Digest mode was issued to compensate the lack of security of the basic. Nevertheless the security level is not strong enough to replace strong authentication protocols.

5 SIM / Handset communication

The UICC including SCWS and the handset device exchange HTTP data packet over TCP/IP or over BIP, depending on what is supported by both devices. Today, usually UICC cards and handset are not able to communicate over TCP/IP, so only the Bearer Independent Protocol (BIP) according to ETSI specification TS 102 223 release 7 is used to tunnel the data.
If a HTTP client on the handset (e.g. a browser) sends a request to the SCWS, the BIP in server mode is used to forward request and response data between the handset application and the SIM card. In case the SCWS initiates a HTTP connection to a remote administration server in the internet, the BIP in client mode is used and the handset will act as a gateway to enable the exchange of administration data between the remote server and the SCWS.
Despite the two BIP modalities seem very similar, in fact they show different behaviour and issues; in the following the peculiar behaviours will be described.

5.1.1 BIP for Browsing (Server mode)

The SCWS should send the Proactive Command OPEN CHANNEL in server mode directly after the handset has send the Terminal Profile command and only if the Terminal profile indicates support of BIP in server mode.

Interoperability Trap:
The HTTPS browsing is an optional feature and it is not supported by all SIMalliance Members

The following parameters for the OPEN CHANNEL command are recommended:

Parameter	Data
OPEN CHANNEL	• No background mode
Buffer Size	0x05DC (dec.: 1500) Buffer size can be subject of card profile configuration but the SIMalliance recommends this value.
Transport level	• Protocol = TCP • UICC in server mode • Port 3516 according to OMA

If the mobile phone requests a smaller buffer size it is accepted.
If the mobile phone requests a bigger buffer size and the card is unable to provide it, the handset can also choice to close the channel.
Mobile phones usually support access to SCWS using *localhost* and 127.0.0.1 .

5.1.2 Wrapping of HTTP stream in BIP packages

When using SCWS it is necessary to put the stream based HTTP(S) connection into the package based APDU connection. To achieve this goal the BIP communication is used.

The principle of BIP is to fill the BIP buffer with the HTTP(S) stream. Afterwards the content of the buffer is split into APDU matching parts and serially send to card (See Figure 6 – Streaming over BIP). The UICC has to convert the packages back into a stream. The original HTTP requests have to be taken out of the stream for processing afterwards.

Interoperability trap:
For HTTP specification the header size is not limited but current SCWS implementations from SIMalliance members can be unable of handling requests larger than a configured size.

Developer tip:
In general, the usage of the Alpha Identifier in BIP commands shall cause the mobile phone to display the BIP connection, as the connection is intended to remain in the background. For this reason it is advisable to set the Alpha Identifier for BIP commands to null.

Interoperability issue:
If the Alpha Identifier is used it depends on the implementation of the mobile phone if it is displayed.

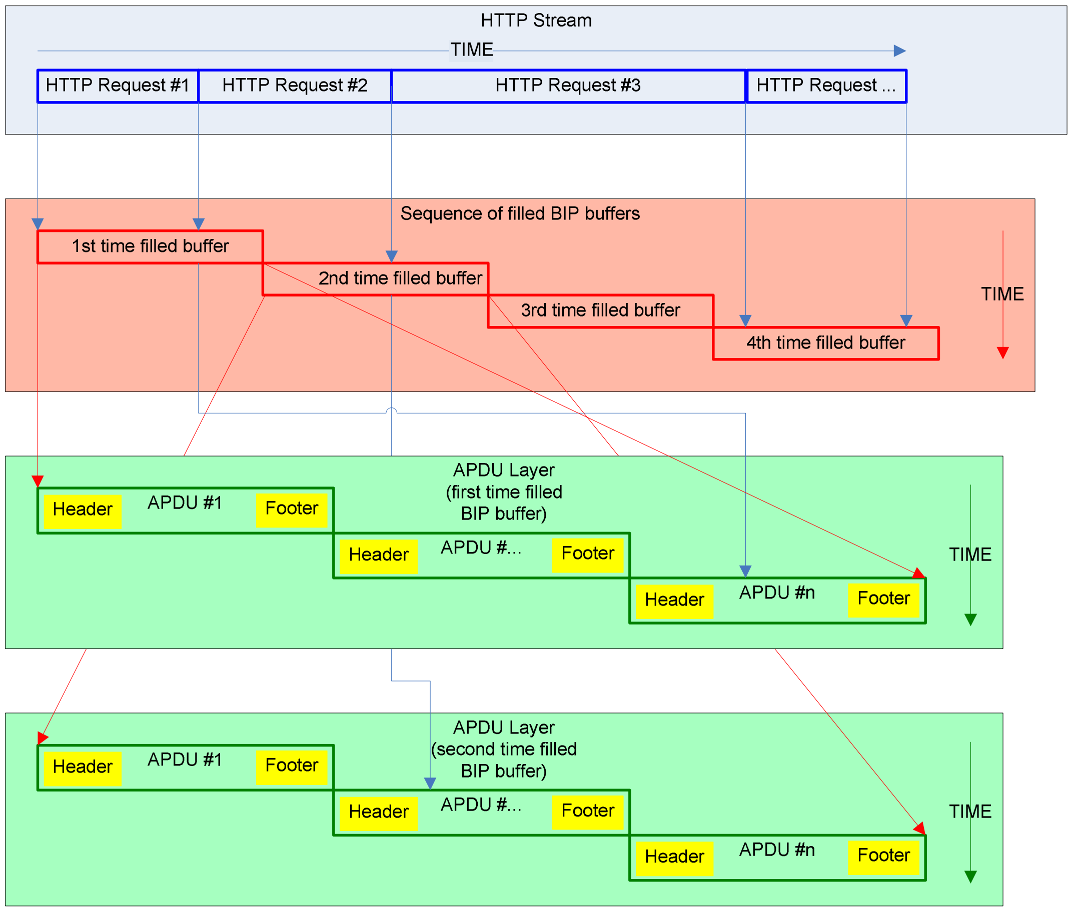


Figure 6 – Streaming over BIP

5.1.3 Occurrence of DATA_AVAILABLE during Browsing or Admin or generic Toolkit applet execution

It is usual behaviour of mobile phones that the EVENT_DATA_AVAILABLE is send during the transmission of a BIP packet. This is in general caused by the memory management in the mobile phone when parts of the previous BIP packets were send to card and are replaced by a new (HTTP) request which generates this new EVENT_DATA_AVAILABLE events.
In this case the UICC stores all incoming EVENT_DATA_AVAILABLE events and executes the events after the actual EVENT_DATA_AVAILABLE is processed completely.
This guarantees that no events or data can get lost.

5.1.4 Command OPEN_CHANNEL

The command OPEN_CHANNEL in server mode is performed by the card after the Terminal Profile is send or as soon as the ADMIN COMMAND HTTP=ON is received within an ADMIN session.

Interoperability Trap:
Some cards might open several BIP channels to the mobile device to allow parallel connections.
SIMalliance members guarantee that at least one channel is open for HTTP and one additional channel is open for HTTPS if supported.

Interoperability Trap:
All type of applications can connect to the SCWS. ON BIP concurrent connections are not guaranteed to work on all SIMalliance cards. This issue is not present when using SCWS over an USB connection.

5.1.5 Cancel request during browsing with SCWS

There are multiple possibilities to behave regarding to the mobile phone to cancel the browsing with SCWS.

After cancel is requested the last request will be proceeded in the background hidden from the user. Exchanged data will be discarded after receive.
This causes a normal ending of the browsing session.

Other possible ways are that the mobile phone reacts with:
1) EVENT_CHANNEL_STATUS(Listen State) or
2) a Terminal response BROWSING_TERMINATION
These possibilities are used to inform the card that no further data are requested by the user.
In both cases the UICC goes back into a state where the acceptance of a new browsing request is possible. Here it is recommended to clear the buffer and temporary data to enable the UICC for the new browsing request.

5.2 BIP for administration (client mode)

If the UICC starts a remote administration session for the SCWS, the BIP client mode is used to connect to the admin server.
The following parameters for the OPEN CHANNEL command are recommended:

Parameter	Data
OPEN CHANNEL	• No background mode
Buffer Size	0x05DC (dec.: 1500) Buffer size can be subject of card profile configuration but the SIMalliance recommends this value.
Bearer Description	• Type = GPRS or UMTS (choice can also be left to the handset using default bearer) • Example GPRS parameter = 000003000003
Transport level	• Protocol = TCP • UICC in client mode

5.2.1 Multi access to SCWS in Admin session

In the case that the SCWS has an established browsing connection and a request for an administration session arrives via SMS for the SCWS the UICC has two possibilities:

1. The UICC indicates with SW 9300 (Toolkit application busy) that the command can currently not be executed. The retry mechanism has to be managed by the network.
2. The UICC stores the incoming SMS event and responds SW1SW2 9000 (Normal ending of the command). The start of the administration session will be delayed until the UICC is able to process it.

In both cases no data are getting lost.

5.2.2 Behaviour in case of an error during administration sessions

In the case that an established administration session gets terminated (reasons can be the loss of communication or an abort from user) the UICC shall close the channel, clear the buffer and all temporary data. Afterwards the retry policy has to be executed as specified in OMA.

The figure is depicting the communication flow at start-up and connection time.

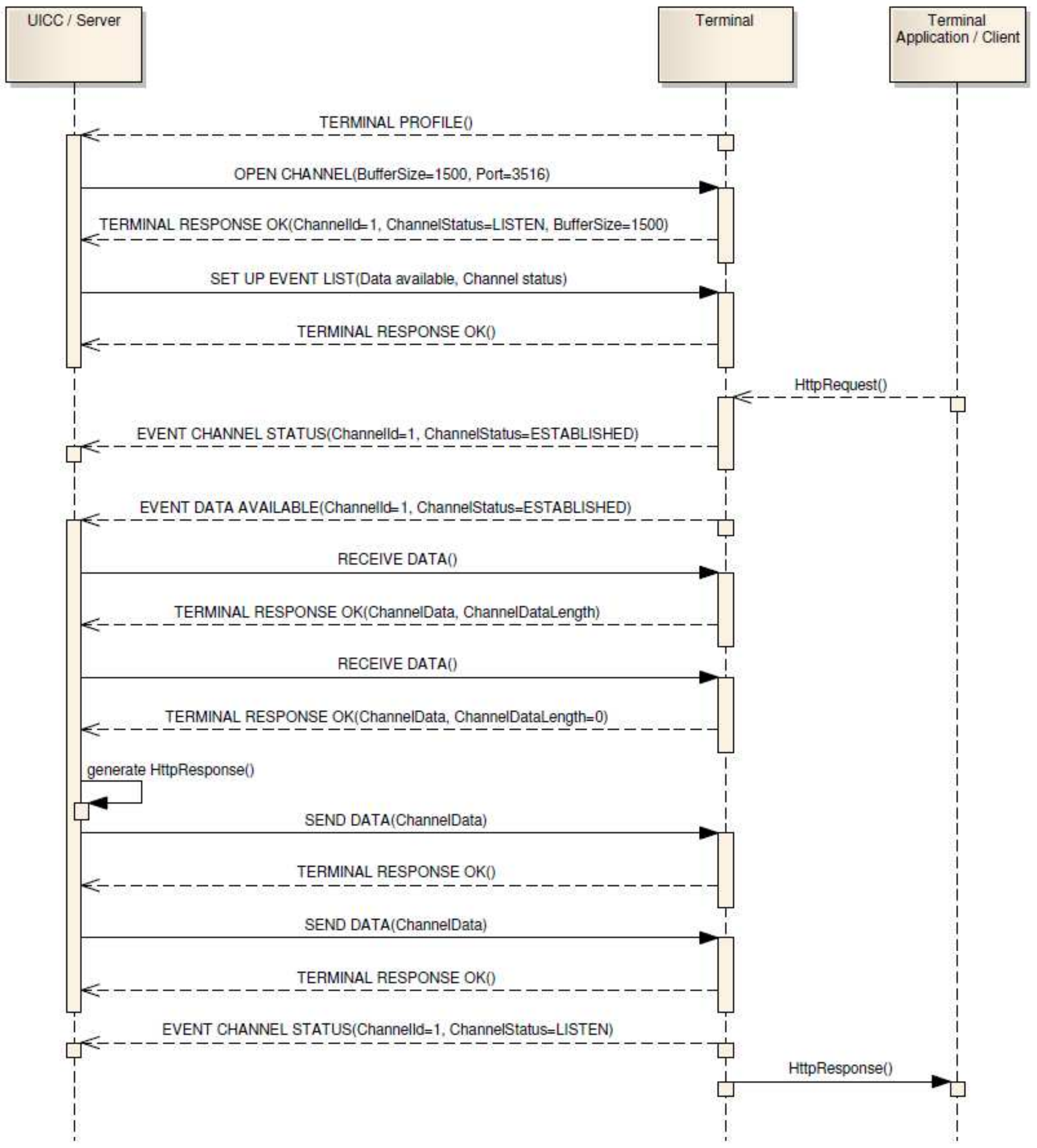

UICC / Server
Terminal
Terminal Application / Client
TERMINAL PROFILE()
OPEN CHANNEL(BufferSize=1500, Port=3516)
TERMINAL RESPONSE OK(ChannelId=1, ChannelStatus=LISTEN, BufferSize=1500)
SET UP EVENT LIST(Data available, Channel status)
TERMINAL RESPONSE OK()
HttpRequest()
EVENT CHANNEL STATUS(ChannelId=1, ChannelStatus=ESTABLISHED)
EVENT DATA AVAILABLE(ChannelId=1, ChannelStatus=ESTABLISHED)
RECEIVE DATA()
TERMINAL RESPONSE OK(ChannelData, ChannelDataLength)
RECEIVE DATA()
TERMINAL RESPONSE OK(ChannelData, ChannelDataLength=0)
generate HttpResponse()
SEND DATA(ChannelData)
TERMINAL RESPONSE OK()
SEND DATA(ChannelData)
TERMINAL RESPONSE OK()
EVENT CHANNEL STATUS(ChannelId=1, ChannelStatus=LISTEN)
HttpResponse()

Figure 7 – BIP Communication flow

6 SCWS - The SCWS Extensions

6.1 Introduction

A powerful mechanism inside the SCWS is the possibility of using Java Card based applications to create HTTP content as an answer to the request. Those Java Card applications, called *SCWSExtensions* allow the service developer to elaborate the HTTP Request when creating the HTTP Response.

The aim of this chapter is to give an overview of the *SCWSExtensions* technology, aiming to explore the potentiality, the programming model, the Application program interface and the most common mechanisms usually found.

For detailed information on Java Card applications, refer to Java Card StepStones .

6.2 Overview of SCWSExtensions

Being a Java Card application, the *SCWSExtension* can, upon serving a specific HTTP Request:

- Process data that have been inserted by the user in a form
- Access to other card resources, e.g. the card file system
- Maintain an internal state, e.g. the number of times that has been requested a specific page
- Perform internal elaboration before providing the response, e.g. encrypting data or providing random number
- Create dynamic content depending on input data and internal SCWS Extension state
- Interact with the terminal by means of proactive commands

A *SCWSExtension* is a Java Card application compliant to ETSI TS 102 588 and to ETSI TS 102 241 Release 6 or later. The application has one or more objects implementing the *SCWSExtension* interface that are registered to the *SCWSExtension* registry.

Actually, the registration is performed by a specific class of the ETSI TS 102 588, the *SCWSExtensionRegistry* class. In order to register an object, the method *register* of the *SCWSExtensionRegistry* is invoked specifying:

- The object that will be invoked when a request is directed to the *SCWSExtension*
- The application id

This second parameter is a specific name (contained in a byte array) that must be unique for the card, that is, no other *SCWSExtension* must be registered in the registry with the same name. Once the name is present, the *SCWSExtension* can be mapped to one or several URIs by using the *map* command.
It is possible to map several URIs both by using several map commands (adding one by one the URIs mapped to the *SCWSExtension*), both by mapping a sub tree, e.g. */bankApplication/**: requesting all the pages included under the sub tree results in an invoking of the *SCWSExtension.* In case several applications match the same URI, the one who matches more is invoked.

> **Example:**
> If a SCWSExtension is mapped to the address */bankApplication/** and another is mapped to the address */bankApplication/checkAccount/**, in case of accessing the resource */bankApplication/checkAccount/verifyPIN.html*, only the second application is invoked.

> **Developer tip:**
> The application id can be any sequence of byte, even a friendly name (e.g. "MyServlet"); however, to be sure the name is unique in the card, application developers can use the Instance AID as the application id.

Once a *SCWSExtension* is registered, when requests are directed to the object, the methods of the object are invoked, depending on the kind of request (*GET, POST …)*. However, all the methods have the same structure, with two parameters:

- the *HTTPRequest* object, containing methods to access the request that triggered the application
- The *HTTPResponse*, containing methods to create the Response to the Request

Any method can be used to invoke a *SCWSExtension*, the two most common are the *doGet* and the *doPost*. However, an object implementing a *SCWSExtension* interface must implement also the other methods in the interface (*doPut, doDelete…*), to indicate that those HTTP methods are not supported.
To avoid implementing explicitly the unused HTTP methods, it is possible for the *SCWSExtension* object to extend the *SCWSExtensionService* class, that is a class with a mere implementation of all methods consisting in throwing the METHOD_NOT_SUPPORTED *SCWSException*; so the developer can override only the methods that are actually supported.

As there are two ways to send an HTTP request to the card, there are actually two ways to forward an HTTP Request to a SCWSExtension: the browsing session (as specified in 4) and the full administration session (as specified in 7). In both cases, the *SCWSExtension* is triggered by the same methods (*doGet, doPost…*) and uses the same APIs to access the Request and the Response.
In the Full Administration Protocol, it is specified that the HTTP Request targeting the *SCWSExtension* is enveloped in an HTTP Response to preserve the protocol. The *SCWSExtension* has access only to the HTTP Request that is enveloped inside the Response.
In the same way, when creating the Response, the Full Administration Protocol will create the external Request and send it over the BIP or USB protocols.

6.3 Accessing the Request

The *HTTPRequest* object can be used to access the request that triggered the SCWSExtension; any part of the request can be accessed, both from the header and from the body.

Depending on the service provided by the *SCWSExtension*, it may be more important to access the header or the body.

In particular, in case of a *SCWSExtension* processing data from a form (e.g. username and password):

- If the form gives data by using the GET method, the GET Request does not have a body, so the SCWSExtension shall find form data in the header; the API *findAndCopyKeywordValue* can be used to access the URI Query
- If the service is based on the *doPost*, the variables are present in the body of the request and can be accessed by using the *readContent* method.

Developer tip:
To avoid Request buffer overflow and to speed up card communication, application developers should minimize the size and the number of form parameters.

Other information that can be found in the header and that can be used by the application developer are:

- The URI schema, to understand if the connection is HTTP or HTTPS
- The Host, to understand if it is called on browsing or on full administration operations
- ETag for caching, to avoid providing a large content that has already been cached by the handset

Interoperability issue:
It is not specified and hence not interoperable if a *SCWSExtension* is triggered once the full request has been received or if it is triggered just after receiving the request header and the request body is received when the application invokes the *readContent* API.

6.4 Creating the Response

The *HTTPResponse* object can be used to create an HTTP response in return to the incoming request.
The Response is created by the *SCWSExtension* with full control on any part of the Response: the Status Code, the header and the content. The Response can be created only inside the method invocation (e.g. inside the *doGet*); when the method returns, the SCWS sends out what has been created by the *SCWSExtension*.

Moreover, if the *SCWSExtension* forgets to complete some part, e.g. the status code, the SCWS completes the response with the missing information.

Developer tip:
SIMalliance members agree that the following information are added by the OS if not by the application.
In case a body has been added by the application:

- Status line, indicating success status (200 OK)
- Server field, containing a customized string (such as "OMA Smart Card Web Server")
- Content-type field, indicating "text/plain"
- Content-length or Transfer-Encoding, depending on the choice of the SCWSExtension

In case no body has been added by the application:

- Status line, indicating success status, no content (204 OK)
- Server field, containing a customized string (such as "OMA Smart Card Web Server")

Two modes to send out the Response are available, the *Chunked mode* and the *Fixed length* mode. The second one is the default until the *SCWSExtension* switches by using the *enableChunkedMode*; then, it is not possible for the *SCWSExtension* to come back to the previous mode.

The fixed length mode foresees that the application creates *all* the response before it is sent out. This has the advantage that any part of the Response can be modified at any moment: as an example, if during the creation of the Response an error is found and a specific Status Code wants to be sent out, this is allowed as no data has been yet sent out. On the other side, the SCWS internal Response buffer has a size that can be overflowed, causing an application exception.

Developer tip:
Size of Response buffer depends on card configuration and may greatly vary from a few hundreds of bytes to many kilobytes and more for USB UICC. Application developer can retrieve information about that by using the getRemainingResponseBufferSize method. In order to avoid buffer overflow, chunked mode can be used:

```
/* Chunked mode - usually for SCWSExtension this is the best choice.*/
response.enableChunkMode();
```

The Chunked Mode implies that the application creates the Response and while it is created the Response is sent out slice by slice. This implies that the Response buffer is virtually infinite but also that, once a chunk has been sent, there's no possibility to modify that specific chunk. As a consequence, some methods (such as *reset*) are forbidden once the *enableChunkedMode* is invoked. Applications are not in charge to explicitly handle the length of the single chunks as the operating system performs this operation.

6.5 Interceptors

In addition to *SCWSExtension* applications that create the Response upon processing a specific Request, an additional type of applications is defined in OMA SCWS specification: the Interceptor.

Those applications are different as they are not allowed to create the Response but they can only access the Request Header for reading; they can be used only to perform logging operations of the incoming requests.

All the interceptors matching a specific request are triggered, and the order is not relevant.

Example:
If an interceptor is mapped to the address */bankApplication/** and another interceptor is mapped to the address */bankApplication/checkAccount/**, in case of accessing the resource */bankApplication/checkAccount/verifyPIN.html*, both the applications are invoked before the resource is processed as a static or as a dynamic content.

Interoperability issue:
Despite not explicitly required by the ETSI TS 102 588, some card manufacturers provide support for interceptors applications; in this case, the *HTTPResponse* parameter of the *doGet, doPut*… methods is set to 'null'.
SIMalliance members don't guarantee that applications defined as interceptors are invoked on all cards.

Developer tip:
As specified in the OMA specification, as the interceptors execution can potentially happen before content providing execution, it is crucial that interceptors' execution is very fast to avoid delaying the user perception of the web server, that means that interceptors developer are discouraged to make time consuming operations such as massive persistent memory update, cryptographic operations, complex algorithms, etc.

6.6 Performing proactive commands

Several services based upon the SCWS require the interaction with the handset to perform proactive commands; as an example, a *SCWSExtension* processing might result in sending out an SMS. However, depending on card OS, it could be not possible for the *SCWSExtension* to send proactive commands directly from the *SCWSExtension* methods.
Definition of proactive commands and explanations on relevant behaviours can be found in Java Card StepStones.

To be more precise, in ETSI Release 8 it has been introduced the possibility for a *SCWSExtension* to directly send the proactive commands from within the *doGet, doPost...* methods. The outcome of the proactive command can then be used by the *SCWSExtension* to build the HTTP Response.

In ETSI Release 7, this is forbidden and card OS throws an exception in case the Proactive Handler is retrieved. However, there is a specific behaviour that allows to override this limitation: the usage of the toolkit mechanism

EVENT_PROACTIVE_HANDLER_AVAILABLE, as explained in ETSI TS 102 241. In this case, the application is triggered *twice*: once inside the *SCWSExtension* method and once inside the *processToolkit* method.

This gives the application the opportunity of creating a proactive command; however, as the *processToolkit* is invoked after the return from the *SCWSExtension* method, there is no possibility of managing the Response. A possible work around is the following:

- When *SCWSExtension* is invoked, it registers to the EVENT_PROACTIVE_HANDLER_AVAILABLE and sends an HTTP Response requesting the reload (303 See other).
- Then it is triggered in the *processToolkit* where it sends out the Proactive command
- Finally it is invoked again in the *SCWSExtension* method, where it can use the result of the Proactive Command to build the Response.

If the *SCWSExtension* has been triggered over the USB interface, SCWS protocol and toolkit protocol are completely independent.

Interoperability issue:
Some Release 7 SCWS implementation supports behaviour for toolkit commands as specified in Release 8.

Developer Tip:
Application developers should design their implementation according to Release 7 SCWS as applications designed for Release 7 work also on Release 8 cards, but applications designed according to Release 8 will not work over Release 7 cards.

7 Administrative commands

7.1 Overview

According to OMA SCWS specification the following administrative commands are available for both Lightweight Protocol and Full Admin Protocol:

- PUT
- GET
- DELETE
- POST

7.2 PUT

This command is used to store a resource on the SCWS or can be used to personalise a *SCWSExtension* mapped to the given URI. In the later case no resource will be updated but the request will be forwarded to the *SCWSExtension*.

Interoperability issue:
In case that the referenced resource is already available on the SCWS, the replace behaviour depends on the implementation. I.e. some cards may keep the old resource until the update has successfully finished, others may not. It is recommended to first delete the resource and then put it onto the SCWS.

Interoperability issue:
The final memory consumption of the loaded resource on the SCWS depends on the implementation

Interoperability issue:
On some special SCWS implementations (e.g. small NVM) the maximum size for one resource depends on the implementation, see chapter § 4 for more details about limits

Interoperability issue:
Usually no limits are given on a SCWS, but maximum URI length, number of resources etc. depends on the implementation, see chapter § 4 for more details about limits

7.3 GET

This command is used to retrieve a resource from the SCWS. The important thing is that this command is also available via the admin channel.

Developer tip:
To use the GET command via the admin channel is useful if some resources are protected by a protection set or to get information from a *SCWSExtension*

7.4 DELETE

This command is used to delete a resource on the SCWS. If the resource identifies a directory, the directory and its content shall be deleted.

7.5 POST

The POST request is used to send special administration commands to the SCWS as described below with the settings from OMA or to send data to a *SCWSExtension* mapped to the given URI. This chapter only the first case as the second one depends on the implementation of the addressed *SCWSExtension*.

All administrative commands must be encapsulated in a HTTP POST method with the fixed URI "/SCWS/admin".

Example:

```
POST /SCWS/admin HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: xx
Host: anything

admin-body ;the message-body of the POST request is called admin-body and
;contains one or several admin commands
```

7.5.1 Protection Sets

In order to secure some amount of content it is possible to define so called "Protections Sets". Therefore a protection set can be assigned to a URI. In that case the URI can only be accessed if the protection set access condition is fulfilled.

Setting protection sets is done by using the HTTP POST method including the OMA command "define protection set" ("dps").

Example:

```
POST /SCWS/admin HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 31
Host: anything

cmd=dps&psid=ps1&protocol=https
```

This defines a protection set named "ps1" and states that HTTPS must be used to access content protected by this set.

After defining a protection set, which is more or less its name and how to access it, multiple resources can be added to this protection set by using the OMA command "pr". Furthermore it is possible to protect a whole sub tree.

Example:
[header as above]

```
cmd=pr&uri=/bank/*&psid=ps1
```

This would link all resources located under "/bank/" to the given protection set, also including whole subdirectories.

Furthermore it is possible to assign one protection set for a sub tree and assign a different protection set to a child of the sub tree. Using the example described above and issuing the following command

Example:
[header as above]

```
cmd=pr&uri=/bank/secret/*&psid=ps2
```

then all files located under "/bank/" are still protected by protection set "ps1" but all files located under "/bank/secret" are protected by protection set "ps2", which has to be defined separately in advance.

In the same way of setting a protection set to URIs a protection set can be deleted by using the OMA command "delps". When a protection set is removed from a sub tree, the protection set from the upper URI sub tree is inherited. If no more protection sets are available the given sub tree has free access.

Example:
If we have:
A protection set "ps1" to protect the URI prefix "/bank"
A protection set "ps2" to protect the URI prefix "/bank/secret
A resource "/bank/secret/index.html" (so under "ps2" control)

If the following command is used:

"cmd=delps&psid=ps2"

Then the resource "/bank/secret/index.html" is now protected by "ps1".

In order to provide the ability to access URIs based on users, it is necessary to define a so called realm (see [RFC2617]). If a protection set is created with a realm, the users must authenticate themselves to the realm. Therefore it must also be specified how the users of this realm must be authenticated. OMA SCWS specification states two ways of authentication:

- Basic authentication (based on username and password)
- Digest authentication (optional, depending on SCWS implementation)

Protection sets could become useful e.g. if the content provider rolls out its smartcards even if the associated background server system is not yet available. If the background system is then available he removes the protection sets and the services become visible to the customer.

> **Developer tip:**
> As OMA SCWS specification states that a URI can be protected with a protection set before the resource is actually loaded onto the SCWS it is recommended to 1st protect the resource and then load the resource. Otherwise there would be a small security gap after the resource is loaded but not yet protected.

7.5.2 Users

It is possible to define users who must authenticate themselves to a protection set with a defined realm (see previous chapter). A user can also be an administrator to have full access to the realm. A user is defined by login and password and, if needed, an indication for admin rights (basic authentication). The administration of users is encapsulated in a HTTP POST method.

> **Example:**
> **[header as above]**
>
> cmd=dusr&user=James&pwd=XYZ675&user-admin=false

After defining a user it must be added to a specified protection set.

> **Example:**
> **[header as above]**
>
> cmd=addusr&user=James&psid=ps1

Of course a user can also be deleted. By deleting a user it is removed from ALL protection sets available in the SCWS.

Users are helpful if certain services shall only be available for paying customers. After a customer has subscribed to a certain service he can be added to the protection set and has access to the subscribed services.

According to OMA there is the optional feature that a user can change a password. This can be done via a HTML page as in the example given by the OMA SCWS specification.

> **Interoperability issue:**
> Some SCWS implementations may not support the feature that a user can change his password, as this feature is specified as an optional feature by OMA SCWS specification

In the same way as a user can be added to a protection set he can also be removed from a specific protection set.

> **Example:**
> **[header as above]**
>
> cmd=rmusr&user=James&psid=ps3

> **Interoperability issue:**
> The maximum number of users depends on the implementation

7.5.3 Map SCWSExtensions

SCWSExtensions can be used to bring dynamic content to the SCWS. A SCWSExtension is a JAVACard applet implementing a special interface. For more details on SCWSExtension programming see § 6.

To use a SCWSExtension in the SCWS it is addressed via a URI. By calling this URI e.g. in a GET request the linked SCWSExtension is triggered.

A SCWSExtension mapped to a certain URI can be protected by using the protection set mechanism as described above.

7.5.4 Get a SCWSExtension running on the SCWS

At first the SCWSExtension must be downloaded to the smartcard, e.g. via Remote Applet Management (RAM).

Before a SCWSExtension can be triggered with one of the defined methods of the SCWSExtension interface, it must be mapped to a URI. Therefore a SCWSExtension must register itself to the "SCWSExtensionRegistry"

Afterwards the SCWSExtension can be mapped to a specific URI.

> **Example:**
> **[header as above]**
>
> cmd=map&appid=application1&path=/app/application1

After issuing the command above and sending e.g. a HTTP GET request on the URI "/app/application1" to the SCWS the SCWSExtension will be triggered through the method

```
SCWSExtension.doGet()
```

It is also possible to map a SCWSExtension to a whole sub tree, which is useful for intercepting applets (see below).

In the map command it can be specified if the SCWSExtension is "intercepting" or "not intercepting".

SCWSExtensions can also be unmapped by using the appropriate command.

> **Example**:
> **[header as above]**
>
> cmd=unmap&appid=application1

> **Interoperability issue:**
> In case that a SCWSExtension applet is actually removed from the smartcard, it depends on the implementation of the SCWS if the mapping is removed or not

> **Developer tip:**
> If the SCWSExtension uses the `SCWSExtensionRegistry.deregister()` method, the mapping is automatically removed from the SCWS

8 Multiple Security Domains for SCWS card application

In order to provide maximum flexibility on customer side it is possible to have multiple security domains for the SCWS card application, each having a different TAR. Each of these security domains can be “sold” to third party content providers where they can administrate their content, independent from each other. By having different security domains for different SCWS card applications each third party has their own security settings, keys etc.

Interoperability issue:
The support of multiple admin agents is left to the implementation of the SCWS

Interoperability issue:
It is left to the implementation how multiple SCWS card applications are managed, if a “Master- SCWS card application” exists.

Interoperability issue:
It is left to the implementation if a certain URI can be assigned to a certain SCWS card application. Although if a resource is created by a certain SCWS card application then it is owned by this SCWS card application

9 Lightweight Protocol

The lightweight protocol (LWP) is a feature according to OMA SCWS specification in order to provide another way of administrating the SCWS. It uses common short messages as transport layer with encapsulated HTTP commands. It is recommended to use the LWP only for a small amount of data, e.g. administrating users, mapping SCWSExtensions etc. The short messages need to be formatted according to ETSI TS 102 225, holding the TAR of the SCWS card application. For more information about ETSI TS 101 225 see Interoperability Stepping Stones Release 7.

Interoperability issue:
As the LWP is mandated as optional by OMA it may not be supported by all implementations.

The TAR to be used for LWP according to ETSI TS 101 220 is: B2 01 01

According to OMA SCWS specification the total internal allocated size for incoming OTA messages, that contain administration commands, SHALL be at least 512 bytes, which results in at least 4 concatenated short messages for each direction.

Interoperability issue:
The values for minimum security and message buffer size depend on the customer requirements

Interoperability issue:
The security domain used for LWP can be the same as for the Full Admin Protocol (see appropriate chapter)

The maximum amount of data to be transferred via LWP is left to the implementation of the SCWS.

In case of errors (e.g. not enough buffer slots) common error handling according to TS 102 225 is recommended.

10 SCWS Security Protocol

10.1 Remote-Admin-Server / Card-Admin-Client

This chapter introduces how to establish a secure connection between the SmartCardWebServer remote-admin-server and the card-admin-client. This chapter is written in compliance with the standard:

- OMA-TS-Smartcard_Web_Server-V1_1
- RFC 2246 / TLS protocol 1.0
- RFC 4279 / PSK Ciphersuites for TLS
- RFC 3546 / TLS Extensions

10.2 TLS Description

Transport Layer Security (TLS) are cryptographic protocols that provide security for communication over networks such as the internet. TLS encrypt the segments of network connections at the Transport Layer end-to-end.

The TLS protocol allows client/server application to communicate across a network in a way designed to prevent eavesdropping, tampering, and message forgery. TLS provide endpoint authentication and communications confidentiality over the Internet using cryptography.

In application design, TLS is implemented on top of the Transport Layer protocols (BIP for the Card Admin Agent), encapsulating the application specific protocol HTTP.

10.3 PSK TLS Description

Transport layer security pre-shared key ciphersuites (TLS-PSK) is a set of cryptographic protocols that provide secure communication based on pre-shared keys (PSKs). These pre-shared keys are symmetric keys shared in advance among the communicating parties.
Usually, TLS uses public key certificates for authentication. TLS-PSK uses symmetric keys, shared in advance among the communicating parties, to establish a TLS connection. There are several reasons to use PSKs:
Using pre-shared, avoid the need for public key operations. This is useful if TLS is used in performance-constrained environments with limited CPU power.
Using pre-shared, provide strong mutual authentication without needing to deploy a PKI.
Pre-shared keys may be more convenient from a key management point of view. For instance, in closed environments where the connections are mostly configured manually in advance, it may be easier to configure a PSK than to use certificates. Another case is when the parties already have a mechanism for setting up a shared secret key.

10.4 TLS handshake in detail

Key Selection in PSK-TLS

- The client indicates its willingness to use PSK-TLS by including PSK cipher-suites in ClientHello.
- ServerKeyExchange is not used in SCWS key selection mechanism. A good implementation of the RemoteAdminServer is do not send this message.
- The client sends PSK-Identity to the server to inform about the key it chooses.

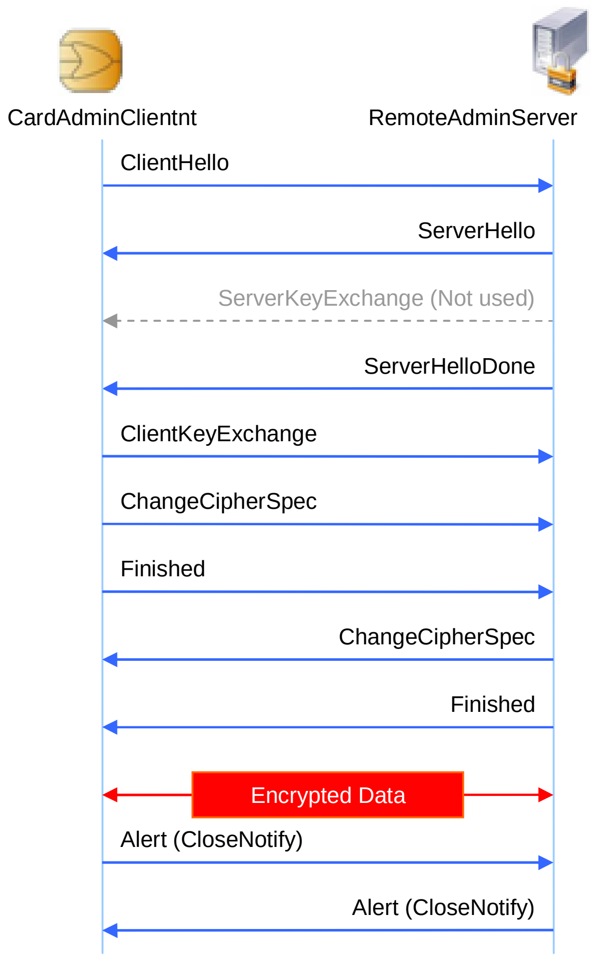


Figure 8 – The TLS handshake

10.5 PSK keys

The PSK keys are provisioned at personalisation level. GlobalPlaform amendment B (Remote Management over HTTP) specify who to provision this key in interoperable way, if not supported each card manufacturer propose there own mechanism to provision the key over the air.

The keys length is not fixed as for common DES and 3DES encryption algorithm, the key can be 1, 2, 3, …, 64, bytes length or larger.

Card Admin Agent can host several PSK keys referenced by the CardKeyIdentifier, but a common usage of the PSK TLS without third party is a single keys.

10.6 Cipher Suite Definition

There are several PSK cipher suites supported by Card Admin Client.
All PSK cipher suite provide strong mutual authentication.
The TLS_PSK_WITH_AES_128_CBC_SHA and TLS_PSK_WITH_3DES_EDE_CBC_SHA cipher suite provides mutual authentication, integrity and confidentiality of the message.
The TLS_PSK_WITH_NULL_SHA cipher suite provides mutual authentication and integrity of the message.
No mechanism exists (Minimum Security Level like) to configure cipher-suite selection on client side.

10.7 Software

- OpenSSL: a free implementation (BSD licence with some extensions).
- GnuTLS : a free implementation (LGPL licensed)
- JSSIE: a java implementation.
- CyaSSL : GPL licensed Open Source embedded SSL library that supports TLS 1.2

11 SCWS Full Administration Protocol

This chapter describes the OMA Full Administration Protocol. This chapter is written in compliance with the standard:

- OMA-TS-Smartcard_Web_Server-V1_1

11.1 Overview of Full admin protocol

This protocol has been defined to open a secure pipe between the SCWS in the card and the OTA platform to administrate SCWS

Full Admin protocol is based on the widely deployed and field proven HTTP protocol and architecture, providing good performance, reliability and scalability; moreover, deployment of platforms in operator network is eased as it can leverage on existing expertise, infrastructure, software components and tools.

OTA server is *really* a (HTTP) server, following Internet responsibilities:

- Give responsibility of session management to the card
- Give responsibility of retry management to the card

➔ Improve OTA server efficiency (less resource consumption, clear responsibilities for card and server) and considerably ease management of out-of-coverage problem

Use HTTP*S* with PSK-TLS

- SSL based on private keys known by both entities
- Ensure mutual authentication, integrity and privacy

➔ Clear separation between transport security and applicative protocol

Main principle of OTA over HTTP protocol is that the embedded element has an Admin Agent component which main features are:
Dialog with server is done in HTTP then Admin Agent is a HTTP client and remote server is a HTTP server. The Protocol with remote server is based on HTTP POST, with Next-URI mechanism

- Administration session:
    - o Connection is at the initiative of the Admin Agent
    - o Manages default connectivity and administration session information
    - o Retry policy is managed by the Admin Agent, and not the server

General dialog mechanism is based on the fact that the Admin Agent asks the OTA server for the next commands to be executed

11.2 Session description

If the remote administration server directly connects to the SCWS_Server: It does not work, both are server applications.

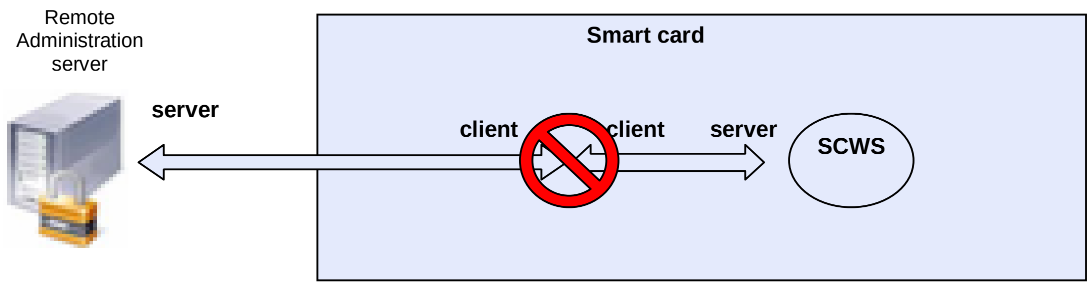


Figure 9 – Administration… how it DOES NOT work

Add a card administration agent: It is in charge to manage connection establishment between the remote administration server and the SCWS_Server.

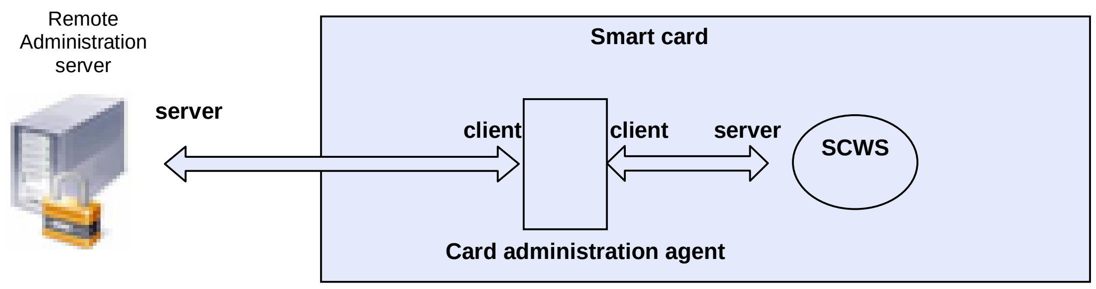


Figure 10 – Administration… how it works

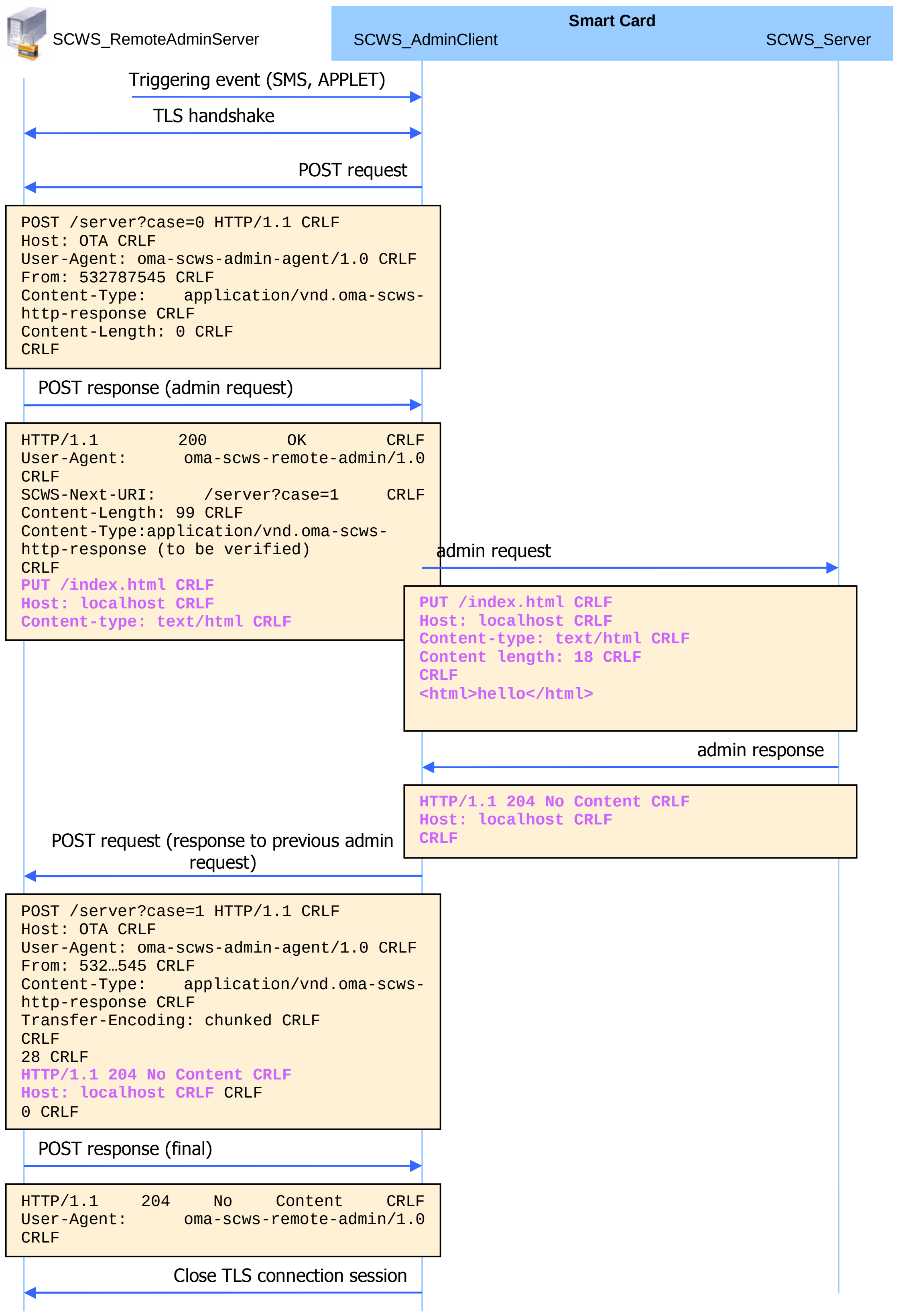
The protocol is based on the following main principles:

HTTP POST messages are sent by the Admin Agent to the OTA server to request for next commands to be executed by the Admin Agent. The same HTTP POST request is also used to return back the results of the command execution.

The Content-Type: specifies the type of message (possible in both the HTTP POST request and in its response). It enables to specify the nature of the commands that are transported (commands or responses; for SCWS, RAM, RFM).

The Body contains the data associated to the type of message (possible in both the HTTP POST request and in its response). The content of the Body is of course dependent on the Content-Type of the messages. E.g., for SCWS commands, the Body contains the HTTP request and HTTP response.

The next URI to be navigated by the Admin Agent (only in HTTP POST response) in order to send back the result of execution of the previous commands, and to ask for new commands to be executed.


SCWS_RemoteAdminServer
Smart Card
SCWS_AdminClient
SCWS_Server
Triggering event (SMS, APPLET)
TLS handshake
POST request
POST response (admin request)
admin request
admin response
POST request (response to previous admin request)
POST response (final)
Close TLS connection session

```
POST /server?case=0 HTTP/1.1 CRLF
Host: OTA CRLF
User-Agent: oma-scws-admin-agent/1.0 CRLF
From: 532787545 CRLF
Content-Type:    application/vnd.oma-scws-
http-response CRLF
Content-Length: 0 CRLF
CRLF
```

```
HTTP/1.1        200        OK        CRLF
User-Agent:      oma-scws-remote-admin/1.0
CRLF
SCWS-Next-URI:     /server?case=1     CRLF
Content-Length: 99 CRLF
Content-Type:application/vnd.oma-scws-
http-response (to be verified)
CRLF
PUT /index.html CRLF
Host: localhost CRLF
Content-type: text/html CRLF
```

```
PUT /index.html CRLF
Host: localhost CRLF
Content-type: text/html CRLF
Content length: 18 CRLF
CRLF
<html>hello</html>
```

```
HTTP/1.1 204 No Content CRLF
Host: localhost CRLF
CRLF
```

```
POST /server?case=1 HTTP/1.1 CRLF
Host: OTA CRLF
User-Agent: oma-scws-admin-agent/1.0 CRLF
From: 532…545 CRLF
Content-Type:    application/vnd.oma-scws-
http-response CRLF
Transfer-Encoding: chunked CRLF
CRLF
28 CRLF
HTTP/1.1 204 No Content CRLF
Host: localhost CRLF CRLF
0 CRLF
```

```
HTTP/1.1     204     No     Content     CRLF
User-Agent:      oma-scws-remote-admin/1.0
CRLF
```

Figure 11 – Administration, protocol example

11.3 Remote Admin Request

Two triggering modes exist to start an administration session: by polling mode (API) or pushing mode (SMS)

The behaviour if a triggering event is received during admin session execution is not clearly specified and depends on the implementation.

The triggering parameters are overload according to this initialization order:

- Parameter from administration request.
- Parameter from the file defined by the administration request
- Parameter from a default configuration

All administration parameters are defined using the same coding as the TS 101 220 COMPREHENSION-TLV data objects.

```
83 xx  [Admin Agent configuration parameters]
    84 xx  [Connection parameters]
    85 xx  [Security parameters]
    86 xx  [Retry Policy parameters]
        87 xx  [Retry failure SMS MO]
    89 xx  [Agent HTTP POST parameters]
       8A xx  [Administration Host parameter]
       8B xx  [Agent ID parameter]
       8C xx  [Administration URI parameter]
```

The connection parameters buffer size value should be set to a suitable value (ex 1500 Bytes).
SIMalliance members agree that the configuration file must contain all the parameters as indicated with the exceptions of the Command Details and Device Identities TLV in the Connection Parameter TLV.

The Security parameters Card Key Identifier format is not defined in oma-ts-scws.

11.4 HTTP POST request

```
POST <URI> HTTP/1.1 CRLF
Host: <Administration Host> CRLF
User-Agent: oma-scws-admin-agent/1.1 CRLF
From: <Agent ID (as defined in triggering event)> CRLF
[SCWS-Resume: true]
[Content-Type: application/vnd.oma-scws-http-response CRLF]
[Content-Length: xxxx CRLF] or [Transfer-Encoding: chunked CRLF]
CRLF
[body-with-previous-scws-commands-responses]
```

11.5 HTTP POST response

```
HTTP/1.1    200    OK    CRLF    [or    HTTP/1.1    204    No    Content    CRLF]
User-Agent:                 oma-scws-remote-admin/1.1                       CRLF
[SCWS-Next-URI:                       <next-URI>                            CRLF]
[Content-Type:            application/vnd.oma-scws-http-request             CRLF]
[Content-Length:                        xxxx                                CRLF]
CRLF
[body-with-scws-command-request]
```

Interoperability traps
Follow standardized HTTP Header Case (`SCWS-Next-URI`)

As a smartcard has limited resources it's recommended to the remote server to send the smallest http response possible. It's means:

- Small URI and small header SCWS-Next-URI.
- No additional HTTP Header ex Server: …

SIMalliance members agree that there is no limit of the POST response body size.
SIMalliance members propose too used chunk mode for large content.

11.6 Retry and Resume management

The Admin Agent is responsible for the connection to the remote administration server and for the accomplishment of the session

If a communication error occurs during the processing of the administration flow the Admin Agent should try to reconnect according to a card issuer specific retry policy.

- The Admin Agent makes several attempts for resuming the administration session. A waiting period between two attempts and the maximum number of attempt is specified by the retry policy.
- If the communication is re-established, the Admin Agent tries to resume the HTTP dialog by navigating the last URL of this administration session and setting a Resume HTTP header tag.
- At the opposite, if the maximum number of attempts has been reached the administration session request is then abandoned.
- Error failure SMS-MO could be sent in case of administration failure
- If the TLS session establishment fails for security/authorization reason the administration session SHALL be immediately discarded.

- Problems that may occur during a session:
    - The server does not respond
    - The server responds but is too busy to immediately perform the request
    - The handset shuts down or the network coverage is lost during an administrative session

11.7 Few handset requirements

Support of standard CAT version with the additional release 7 features:
Class E (Bearer independent protocol - BIP) must be implemented as in ETSI TS102 223 R7 in TCP mode
Empty or "NULL" Alpha Identifier of OPEN CHANNEL BIP command shall result in a transparent channel opening (no notification to the end user)
To increase protocol performance we encourage handset the handset to support of PPS (Fi;Di) = 96 (16 clock strokes per etu), or above

11.8 Pipelining

The Administration server can pipelines several requests embedded into one POST response, then the SCWS pipeline all the responses in the next administration agent POST request body. The order of the responses SHALL be the same than the requests (see HTTP/1.1 for details).

For admin session performance is recommended to support http pipelining. Refer to OMA-TS-Smartcard_Web_Server-V_1 Appendix H. Pipelining over full administration protocol.

We encourage OTA platform to use this mechanism for performance purpose.

SIMalliance members agree that there are no there is no limitation of the number of HTTP request pipelined.
SIMalliance members agree that there is no limitation of the size of the concatenated HTTP request.

12 Appendix A. SCWS Handshake message

12.1.1 ClientHello (client to server)

```
-----------------------------------------------
16 03 01 00 38 01 00 00 34 03 01 93 18 CE 09 B1
5E 56 A6 84 2F 50 B7 91 82 21 1E 05 68 ED 86 B1
FA B5 F4 53 8F C2 9F 17 33 47 02 00 00 06 00 8C
00 8B 00 2C 01 00 00 05 00 01 00 01 01
-----------------------------------------------
```

Field	**length**	**Value**	**Note**
protocol	1	0x16	Handshake protocol
Major version	1	0x03	TLS version 1.0
minor version	1	0x01	
length	2	0x00 0x38	TLS message length
handshake type	1	0x01	ClientHello
length	3	0x00 0x00 0x34	handshake message length
major version	1	0x03	
minor version	1	0x01	
random value	32	[0x93...0x02]	gmt_unix_time(4bytes) + random(28 bytes)
session id length	1	0x00	session-resume is not used
cipher suite length	2	0x00 0x06	
cipher suite	1	0x00 0x8C	TLS_PSK_WITH_AES_128_CBC_SHA
cipher suite	1	0x00 0x8B	TLS_PSK_WITH_3DES_EDE_CBC_SHA
cipher suite	1	0x00 0x2C	TLS_PSK_WITH_NULL_SHA
compression length	1	0x01	
compression method	1	0x00	compression is not used
length	2	0x05	Extension_message_length
Extension-type	2	0x01	max_fragment_length id(1)
length	2	0x01	Extention_data_length
enum of max-fragment-length	1	0x01	2^9(1)

12.1.2 ServerHello + ServerHelloDone (server to client)

```
-----------------------------------------------
16 03 01 00 55 02 00 00 4D 03 01 49 FE CD 57 15
E0 44 CD 99 32 4E B3 4B 6B 83 DD 1D 63 4C DB 8E
41 B0 EA 3F 9E A4 0A A8 16 39 96 20 C2 09 82 D5
75 DE 69 A0 14 64 53 6E C1 74 FB D6 1E 9D C8 F1
50 5C A6 81 90 1E 6B FA DC 7C B0 5B 00 8B 00 00
05 00 01 00 01 01 0E 00 00 00
-----------------------------------------------
```

field	**length**	**Value**	**Note**
protocol	1	0x16	Handshake protocol
major version	1	0x03	TLS version 1.0
minor version	1	0x01	
length	2	0x00 0x55	TLS message length
handshake type	1	0x02	ServerHello
length	3	0x00 0x00 0x4D	handshake message length
major version	1	0x03	
minor version	1	0x01	
random value	32	[0x49...0x96]	gmt_unix_time(4bytes) + random(28bytes)
session id length	1	0x20	
session id		[0xC2...0x5B]	
cipher suite	1	0x00 0x8B	TLS_PSK_WITH_3DES_EDE_CBC_SHA

compression method	1	0x00	compression is not used
length	2	0x00 0x05	Extension_message_length
Extension-type	2	0x00 0x01	max_fragment_length id (1)
length	2	0x00 0x01	Extention_data_length
enum of max-fragment-length	1	0x01	2^9(1)
handshake type	1	0x0E	ServerHelloDone
length	3	0x00 0x00 0x00	handshake message length

Here the messages ServerHello and ServerHelloDone are concatenated in a single TLS-Record. As define is TLS specification client shall support to received these messages in two TLS Record or concatenated in a single TLS Record.

12.1.3 ClientKeyExchange (client to server)

```
-----------------------------------------------
16 03 01 00 11 10 00 00 0D 00 0B 73 69 6D 61 6C
6C 69 61 6E 63 65
-----------------------------------------------
```

field	length	Value	Note
protocol	1	0x16	Handshake protocol
major version	1	0x03	TLS version 1.0
minor version	1	0x01	
length	2	0x00 0x11	TLS message length
handshake type	1	0x10	ClientKeyExchange
length	3	0x00 0x00 0x0D	handshake message length
psk identity length	1	0x0B	
psk identity	12	[0x73...0x65] "simalliance"	psk identity value of Security parameter TLV(see OMA 13.3.2.9.5) defined by OMA specification MUST be used as value of psk identity.

12.1.4 ChangeCiperSpec (client to server)

```
-----------------------------------------------
14 03 01 00 01 01
===============================================
```

field	length	Value	Note
protocol	1	0x14	ChangeCipherSpec protocol
major version	1	0x03	TLS version 1.0
minor version	1	0x01	
length	2	0x00 0x01	TLS message length
changeCipherSpec	1	0x01	

12.1.5 Session Key Generation (for 3DES_EDE)

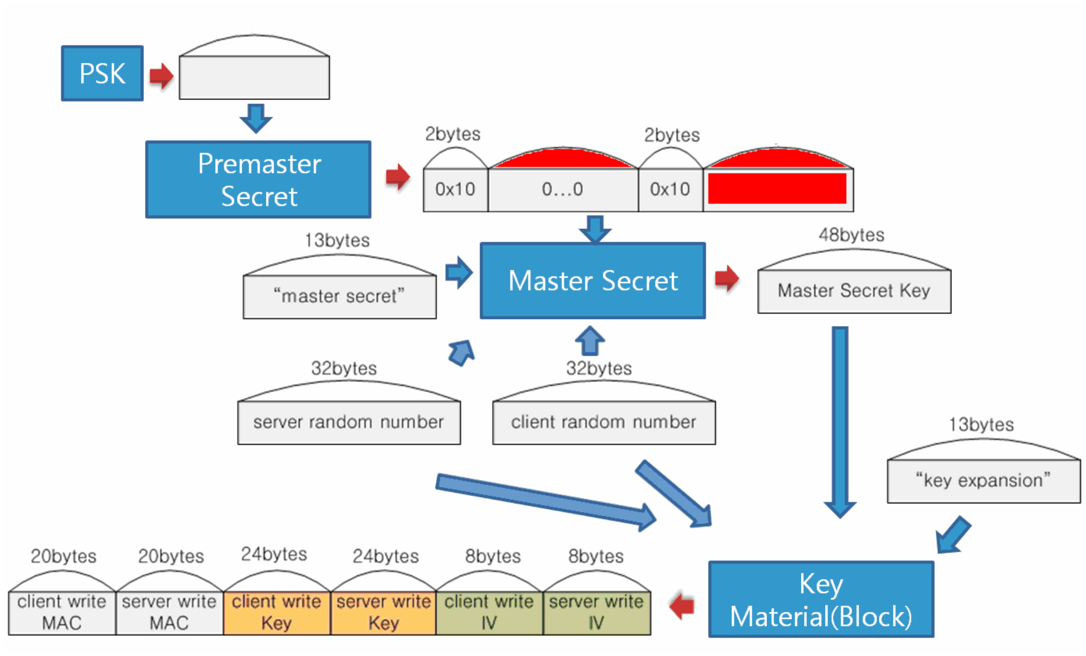


Figure 12 – Session key generation for TLS

```
PSK
-----------------------------------------------
4c:cf:06:d6:8e:00:91:57:e2:4e:71:02:7b:38:40:57
-----------------------------------------------

preMasterSecret
-----------------------------------------------
00:10:00:00:00:00:00:00:00:00:00:00:00:00:00:00
00:00:00:10:4c:cf:06:d6:8e:00:91:57:e2:4e:71:02
7b:38:40:57
-----------------------------------------------

client.random
-----------------------------------------------
93:18:ce:09:b1:5e:56:a6:84:2f:50:b7:91:82:21:1e
05:68:ed:86:b1:fa:b5:f4:53:8f:c2:9f:17:33:47:02
-----------------------------------------------

server.random
-----------------------------------------------
49:fe:cd:57:15:e0:44:cd:99:32:4e:b3:4b:6b:83:dd
1d:63:4c:db:8e:41:b0:ea:3f:9e:a4:0a:a8:16:39:96
-----------------------------------------------

masterSecret
-----------------------------------------------
25:93:6c:a7:6b:55:e3:b7:07:e3:58:85:55:59:75:ab
e6:70:4f:80:48:b4:06:17:7d:e9:d5:f0:a3:cc:48:26
8a:3d:ae:be:52:dc:64:86:c0:01:16:5f:22:0e:64:61
-----------------------------------------------
```

Generated keys:

```
CLIENT_WRITE_MAC_SECRET
-----------------------------------------------
a7:ee:f0:37:54:97:29:1c:44:90:9d:14:c2:d6:b4:f4
8a:ca:1d:44
-----------------------------------------------
```

```
SERVER_WRITE_MAC_SECRET
-----------------------------------------------
26:f3:77:ad:09:c7:05:3b:c9:e6:7f:7b:62:15:d1:2a
39:45:27:b8
-----------------------------------------------

CLIENT_WRITE_KEY
-----------------------------------------------
82:7f:7d:f8:a8:53:1e:2b:74:a4:72:f9:c3:6e:d7:06
9a:76:d6:34:27:b3:7b:24
-----------------------------------------------


SERVER_WRITE_KEY
-----------------------------------------------
12:4a:fc:58:44:e3:65:35:6e:7e:76:63:ba:4a:03:3e
c8:92:72:68:bc:5a:01:ee
-----------------------------------------------

CLIENT_WRITE_IV
-----------------------------------------------
00:2f:b2:4b:23:06:76:4b
-----------------------------------------------


SERVER_WRITE_IV

-----------------------------------------------
4a:63:00:a5:a1:9c:17:f6
-----------------------------------------------
```

12.1.6 Finished (client to server)

```
-----------------------------------------------
16 03 01 00 28 2A 55 41 50 54 D6 8D D5 A8 0A 57
81 F2 2F A5 0B F0 98 3E 84 5B 58 25 3E 85 B2 F5
4C F4 8A 28 E2 79 7F BE E4 B1 DE 97 B6
-----------------------------------------------
```

field	length	Value	Note
protocol	1	0x16	Handshake protocol
major version	1	0x03	TLS version 1.0
minor version	1	0x01	
length	2	0x00 0x28	TLS message length
Handshake type	1	0x14	Finished
length	3	0x00 0x00 0x0C	
Verify-data	12	...	PRF(master_secret, "client finished", MD5(handshake_message) + SHA-1 (handshake_message))[0..11]

[ ]: this data is encrypted

handshake_message

```
-----------------------------------------------
01 00 00 34 03 01 93 18 CE 09 B1 5E 56 A6 84 2F
50 B7 91 82 21 1E 05 68 ED 86 B1 FA B5 F4 53 8F
C2 9F 17 33 47 02 00 00 06 00 8C 00 8B 00 2C 01
00 00 05 00 01 00 01 01 02 00 00 4D 03 01 49 FE
CD 57 15 E0 44 CD 99 32 4E B3 4B 6B 83 DD 1D 63
4C DB 8E 41 B0 EA 3F 9E A4 0A A8 16 39 96 20 C2
09 82 D5 75 DE 69 A0 14 64 53 6E C1 74 FB D6 1E
9D C8 F1 50 5C A6 81 90 1E 6B FA DC 7C B0 5B 00
8B 00 00 05 00 01 00 01 01 0E 00 00 00 10 00 00
0D 00 0B 73 69 6D 61 6C 6C 69 61 6E 63 65
-----------------------------------------------
```

```
clientHello + ServerHello + ServerHelloDone + clientKeyExchange
```

```
MD5(handshake_message)
-----------------------------------------------
e2 f9 99 ed 8e 3f 19 d8 79 a4 77 80 50 10 79 e4
-----------------------------------------------

SHA-1 (handshake_message)
-----------------------------------------------
75 43 b2 96 92 38 82 21 02 be 6e a1 d0 8a 8d bb
ab fb f0 08
-----------------------------------------------

PRF(master_secret,    "client    finished",          MD5(handshake_message)    +    SHA-1
(handshake_message))[0..11]
-----------------------------------------------
af f0 de 3a 5c 99 4e f1 a0 7b c2 88
-----------------------------------------------
```

12.1.7 ChangeCiperSpec (server to client)

```
-----------------------------------------------
14 03 01 00 01 01
-----------------------------------------------
```

field	length	Value	Note
protocol	1	0x14	ChangeCipherSpec protocol
major version	1	0x03	TLS version 1.0
minor version	1	0x01	
length	2	0x00 0x01	TLS message length
ChangeCipherSpec	1	0x01	

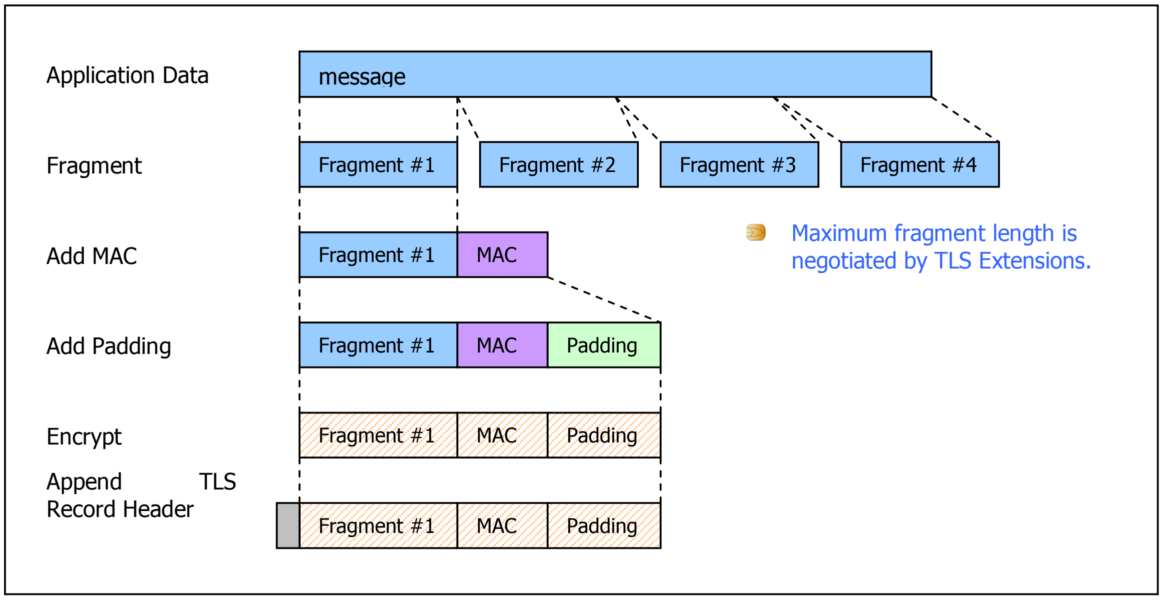
12.1.8 Finished (client to server)

```
-----------------------------------------------
16 03 01 00 58 0B 75 2F 20 9D DF 97 59 3D 52 7A
F1 35 24 58 34 39 70 83 21 8B 88 F5 43 2D 4F 85
1D 65 37 7D 14 B0 83 F7 EF 34 75 03 F8 35 08 AB
C9 90 B0 F0 05 E0 C3 3B 3E 4D 8A 27 50 78 01 DF
5D C3 F0 A9 99 E8 18 E1 80 38 16 9A 24 82 2C 05
86 9C D9 84 24 39 E2 F2 EE 1A B0 30 AA
-----------------------------------------------
```

field	length	Value	Note
protocol	1	0x16	Handshake protocol
major version	1	0x03	TLS version 1.0
minor version	1	0x01	
length	2	0x00 0x58	TLS message length
Handshake type	1	0x14	Finished
length	3	0x00 0x00 0x0C	
Verify-data	12	...	PRF(master_secret, "client finished", MD5(handshake_message) + SHA-1 (handshake_message))[0..11]

[ ]: This data is encrypted

12.1.9 Application data (client to Server)



Example of Application data

```
-----------------------------------------------
17 03 01 00 80 32 92 24 1F BA E8 63 6C 9B 0A 59
09 A9 F9 34 38 E0 B0 0A 6E 45 51 5A C4 CB 29 31
4E A2 F3 82 7E C7 CD A9 23 55 CC 81 4D 26 C3 E9
BA B6 53 7D AA 33 03 EE 1F DA 77 E1 20 03 64 54
32 88 43 E5 95 C5 46 BE AA 1C AF D3 AE 7C 70 97
BA 2F 66 F4 E1 DC 8D D7 F4 EE D2 92 31 FC 3F 0B
D8 3B E2 A0 2A 21 88 36 7A DA 36 E7 3B 2F 1E FD
D6 14 CE FC B1 72 F3 06 F7 20 00 82 91 33 24 BC
2E F3 33 0E 55
-----------------------------------------------
[ ]: TLS Record Header
[ ]: Encrypted message
```

Application data deciphering

```
--------------------------S-----------------------------------------------
00000000  50 4f 53 54 20 2f 43 57 53 2f 74 65 73 74 44 65     POST /CWS/testDe
00000010  66 61 75 6c 74 55 52 49 2f 20 48 54 54 50 2f 31     faultURI/ HTTP/1
00000020  2e 31 0d 0a 48 6f 73 74 3a 20 6c 6f 63 61 6c 68     .1..Host: localh
00000030  6f 73 74 0d 0a 55 73 65 72 2d 41 67 65 6e 74 3a     ost..User-Agent:
00000040  20 6f 6d 61 2d 73 63 77 73 2d 61 64 6d 69 6e 2d      oma-scws-admin-
00000050  61 67 65 6e 74 2f 31 2e 31 0d 0a 46 72 6f 6d 3a     agent/1.1..From:
00000060  20 73 63 77 73 0d 0a 0d 0a 08 26 48 65 da e1 75      scws.....&He..u
00000070  1e 13 dc 61 5f aa 18 0b df ba 52 e0 c1 02 02 02     ...a_.....R.....
---------------------------------------------------------------------------------

[ ]: Fragment#1
[ ]: MAC
[ ]: Padding
```

13 Annex B – Example of SCWSExtension

13.1 Introduction

The *Graphical Book Monitor* is a SCWSExtension able to show users phonebook contacts in a web fashion.

To each entry in the phonebook is a picture and a short menu including the possibility of performing a call associated.

This application is able to prove SCWSExtension basic development, implementation of the *doGet* method, interaction with the Toolkit commands, SCWSExtension using the file system, plus a set of toolkit features.

The application is made of two classes and of a set of static contents.

The two classes are:

- An application (PhoneBookSCWSApplet)
- SCWSExtension simAllianceSCWSExtension, able to process the initial request and of showing the enriched phonebook content

13.2 Card configuration and application installation procedure

The applet PhoneBookSCWSApplet can be created as a traditional Release 6 toolkit application, with no requirement on Toolkit install parameter.

Access specific parameter shall be specified providing access to the EF ADN under DF_TELECOM.

For each entry in the EF ADN that shall be shown by the SCWSExtension, a jpeg image resource_xy.png shall be stored as a static content of the web server.

An URI (e.g. /SCWSPhonebook.html) shall be mapped to the "SIMALL" resource, that URI shall be used to invoke the SCWSExtension.

13.3 Application behaviour

13.3.1 Applet installation

In addition to the normal installation procedures, the applet creates and initializes the two *SCWSExtension* objects. Moreover, the two objects are registered to the web server registry as follows:

```
ScwsExtensionRegistry.register(
      new simAllianceSCWSExtension( phoneBookView, recordLength, recordNumber ),
      phoneBookDisplayId, (short)0, (short)phoneBookDisplayId.length);
```

The two objects shall have different ids and names when registered to the SCWS Registry, so they can be targeted by different URIs.

Inside the constructors of the two SCWSExtension objects, object initialization is performed.

13.3.2 Operative behaviour

The application starts being triggered by the URI mapped to the *phonebookDisplay*, creating and returning an HTTP content containing several information; the information are analyzed in the following.
First part created by the SCWSExtension is the HTTP header, with the following fields:

- Status code set to "200 OK"
- Cache control set to "No cache"; this is required otherwise the handset could store in the cache the page without asking the SCWSExtension to recreate it. However the page is dynamic as phonebook contacts may change
- Transfer mode set to chunked mode; this is to avoid that a buffer overflow could happen in creating the response
- Content type set to HTML

```
/* OK - page has been found */
response.writeStatusCode(ScwsConstants.SC_OK);

/* Never cache it - usually for SCWSExtension this is the best choice.*/
response.appendHeaderVariable(cacheControl, (short)0, (short)cacheControl.length);

/* Chunked mode - usually for SCWSExtension this is the best choice.*/
response.enableChunkMode();

/* Content type = HTML */
response.setContentType(ScwsConstants.CONTENT_TYPE_TEXT_HTML);
```

Second part created by the SCWSExtension is the HTTP body. The body is divided into three parts:

- The HTML header
- The central part, consisting of Names, Pictures and phone number of people in the phonebook
- The HTML footer

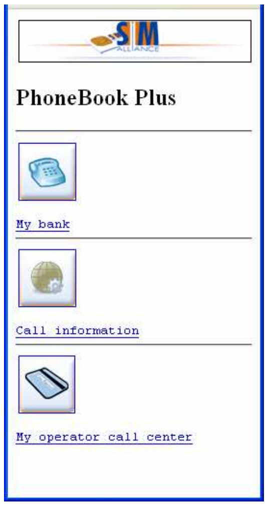


Figure 13 – The web page returned by the example SCWS Extension

The HTML header and the HTML footer are fixed, i.e. they don't depend on the central part.

To create the central part, records from the ADN file are read and put inside the page. If no record is found, a generic error page is returned.

13.3.3 Installation parameters

The following installation parameters are required:

UICC Toolkit Specific Parameters

- No explicit parameter is required; of course, the Toolkit Specific parameters must be present, otherwise the applet won't be able to perform outgoing calls.

UICC Access Application specific parameters field

- No special access is required; however, the UICC Access Application specific parameters must be present to get access to the file system.

UICC Administration Access Application specific parameters field

- No special access is required.

13.4 Example applet source code

```
/*************************************************************************************

  The Phonebook Caller

  Example applet developed by SIMalliance to show how to develop a SCWS Extension

  The SCWS Extension analyses the EF ADN file (6F3A) under 7F10 and builds an HTTP page
  where, for each name, assigns a specific icon id: resource___xy.jpg, where xy is the record
  number in the ADN decimally coded. Images are shown in addition to phone number and name.

  User can click on phone number to ask the handset to place the call.
  ********************************************************************************/

package org.simalliance.phoneBookCaller;

import javacard.framework.*;
import uicc.scws.*;
import uicc.toolkit.*;
import uicc.access.*;
import uicc.system.*;

public class PhoneBookSCWSApplet extends Applet implements ToolkitConstants {


    static short recordLength; // Length of the record of the ADN file
    static short recordNumber; // Number of records in ADN file

    // Resource name, as stored than in the SCWSRegistry
    static byte [] phoneBookDisplayId = { (byte)'S', (byte)'I', (byte)'M', (byte)'A', (byte)'L', (byte)'L'};

    static FileView phoneBookView;



   public static void install(byte bArray[], short bOffset, byte bLength)
    {
            // The Phonebook SCWS Applet is a Java card applet
     PhoneBookSCWSApplet phoneBookSCWSApplet = new PhoneBookSCWSApplet();
            phoneBookSCWSApplet.register();

            // Initializes: phoneBookView, recordLength, recordNumber
            readADN();

            ScwsExtensionRegistry.register(
                    new simAllianceSCWSExtension( phoneBookView, recordLength, recordNumber ),
                    phoneBookDisplayId, (short)0, (short)phoneBookDisplayId.length);

    }


   public void process(APDU apdu) {
            if (selectingApplet())
            {
                    return;
            }
    }

    /****************************
```

```
     readADN
     The method access the 2G ADN file and gets infos about the file dimensions
    ****************************/
    static void readADN()
    {
        // The getTheVolatileByteArray buffer is used to save memory space
        byte[] buffer = UICCPlatform.getTheVolatileByteArray();

        short fcp_len;

        phoneBookView=UICCSystem.getTheUICCView( JCSystem.NOT_A_TRANSIENT_OBJECT);

        phoneBookView.select((short)0x3F00 );

        phoneBookView.select((short)0x7F10 );

        fcp_len = phoneBookView.select((short)0x6F3A, buffer, (short)0, (short)buffer.length);

        // accessing record size and length - the Handler Builder class is used to access the TLV list
        ViewHandler  fcpResponse  =  HandlerBuilder.buildTLVHandler(HandlerBuilder.BER_EDIT_HANDLER,  (short)256,  buffer,
(short)2, (short)(fcp_len-2));

        fcp_len = fcpResponse.findAndCopyValue((byte)0x82, buffer, (short)0);

        recordLength = Util.getShort(buffer, (short)0x02);

        recordNumber = (short)((short)buffer[0x04] & 0x00FF);

    }
}

/*  Class simAlliance SCWS Extension

    This class is a SCWS Extension able to create, in the doGet method, the HTML page containing the PhoneBook.  */
class simAllianceSCWSExtension extends ScwsExtensionService {

        // Utility buffer is a temporary buffer filled with data
        private byte[] utilityBuffer;

        // DestinationAdderss contains the phone number connected to the
        private byte[] DestinationAddress;

        // Small (2 bytes) temporary buffer
        private byte[] temp;

        // List of references containing the EF ADN references, i.e. the record numbers.
        private byte[] References;

        FileView phoneBookView;
        short recordLength, recordNumber;

        // Constructor
        public simAllianceSCWSExtension(FileView _phoneBookView, short _recordLength, short _recordNumber)
        {
            phoneBookView = _phoneBookView;
            recordLength = _recordLength;
            recordNumber = _recordNumber;

            utilityBuffer = JCSystem.makeTransientByteArray((short) 254, JCSystem.CLEAR_ON_RESET);

            DestinationAddress     = JCSystem.makeTransientByteArray((short) 30, JCSystem.CLEAR_ON_RESET);

            References = JCSystem.makeTransientByteArray(recordNumber, JCSystem.CLEAR_ON_RESET);

            temp    = JCSystem.makeTransientByteArray((short) 2, JCSystem.CLEAR_ON_RESET);;
        }


    /****************************************************************************
     * The method returns an HTML page containing the ADN records in an HTML fashion
     * Selecting a specific number the SET UP CALL starts
     ****************************************************************************/
public void doGet(HttpRequest request, HttpResponse response)
    {
        short startOffset;
        short validRecords;
        short numberLen;


        /* OK - page has been found */
        response.writeStatusCode(ScwsConstants.SC_OK);

        /* Never cache it - usually for SCWSExtension this is the best choice.*/
        response.appendHeaderVariable(cacheControl, (short)0, (short)cacheControl.length);

        /* Chunked mode - usually for SCWSExtension this is the best choice.*/
        response.enableChunkMode();

        /* Content type = HTML */
        response.setContentType(ScwsConstants.CONTENT_TYPE_TEXT_HTML);

            /* The ADN is read and the counter of valid Records is stored in the PhoneBookSCWSApplet.References_index field */
            validRecords = checkADN();

            if(validRecords==0){
                  //ADN is empty
                  // Returning the error page.
                  response.appendContent(errorPage,(short)0,(short)errorPage.length);
            }else{

                  // Creating the HTML
                  // 1) HTML header

                  //     "<html><head><title>PhoneBook     Plus</title></head><body><img     src="resource://incard_logo.jpg"
hspace="3" vspace="3" border="1"><br><h2>PhoneBook Plus</h2>"
                  response.appendContent(headerPage,(short)0,(short)headerPage.length);

                  for(short i=0;i<validRecords;i++){


                        // 2) Hyperlink to a specifing image
                        // "<hr><img src="resource___
                        response.appendContent(resourceText,(short)0,(short)resourceText.length);

                        //putting the image number in the HTML in hex
                        temp[0]=(byte)(((References[i] / 10) & 0x0F)+(byte)0x30);
                        temp[1]=(byte)( (References[i] % 10)        +(byte)0x30);
                        response.appendContent(temp,(short)0,(short)2);

                        // ".jpg" hspace="3" vspace="3" border="1"><br><pre>"
                        response.appendContent(trailerText, (short)0, (short)trailerText.length);

                        // 3) Alfa id in text and number
                        phoneBookView.readRecord(References[i],(byte)0x04,(short)0,utilityBuffer,(short)0,recordLength);
                        numberLen = convertDA();

                        short len=getAlphaLen(utilityBuffer);

                        Util.arrayFillNonAtomic(utilityBuffer,len,(short)(recordLength-14-len),(byte)0x20);

                        // Name, for user eyes
                        response.appendContent(utilityBuffer,(short)0,(short)(recordLength-14));

                        // "<a href="platformRequest://tel:"
                        response.appendContent(platformReqText, (short)0, (short)platformReqText.length);

                        // Phone number, for "tel://" URI
                        response.appendContent(DestinationAddress,(short)0,numberLen);

                        // "">"
                        response.appendContent(tagTrailer,(short)0, (short)tagTrailer.length);

                  }

                  // </body></html>
```

```
                    response.appendContent(httpTrailer,(short)0, (short)httpTrailer.length);
                }

        }

        /* Function converting the Destination address from the ADN format to the HTML Ascii format */
        private short convertDA(){

                byte length = 0;
                byte i=0;

                short startOffset=(short)(recordLength-14);
                // If the first digit is a '+' then it has to add 0x2B
                if(utilityBuffer[(short)(startOffset+1)] == (byte)0x91)
                {
                        DestinationAddress[length++] = (byte) 0x2B;
                }

                // swap of ADN bytes and converting to SMS format
                for (i= 1; i < utilityBuffer[(short)startOffset]; i++)
                {
                        DestinationAddress[length++]  =  (byte)  (  (byte)  (utilityBuffer[(short)(startOffset+i  +1)]  &  (byte)0x0F)  +
(byte)0x30 );

                        DestinationAddress[length++]  =  (byte)  (  (byte)  (  (byte)(utilityBuffer[(short)(startOffset+i  +1)]  >>4  )  &
(byte)0x0F) + (byte)0x30 );
                }

                // Is there an odd number of digits??
                if( (byte) ((byte)DestinationAddress [(short)(length-1)] & (byte) 0x0F) == (byte)0x0F )
                {
                        length--;
                }

                return length;

        }

        private short getAlphaLen(byte[] utilityBuffer){

                short len=0;

                while((utilityBuffer[len]!=(byte)0xFF)&(len<(short)(recordLength-14))){
                        len++;
                }
                return len;
        }

        /********************************************************
          The method reads the ADN records and checks which are empty
          The "non-empty" are stored in the utilityBuffer array
        ********************************************************/
        private short checkADN()
        {

                short validRecords=0;

                for(short i=1;i<=recordNumber;i++)
                {
                        phoneBookView.readRecord(i,(byte)0x04,(short)0,utilityBuffer,(short)0,(short)1);

                        //Only name length is checked
                        if(utilityBuffer[0]!=(byte)0xFF)
                                References[validRecords++]=(byte)i;

                }

                return validRecords;
        }


        /*********************************
                HTML constant strings
```

```
       **********************************/

      private static final byte[] errorPage=
      {
            //Sorry, the PhoneBook is empty
            //<html><head><title>PhoneBook Plus</title></head><body><img src="resource://simAlliance.jpg" hspace="3"
vspace="3" border="1"><br><h3>Sorry, the PhoneBook is empty!</h3></body></html>

(byte)'<',(byte)'h',(byte)'t',(byte)'m',(byte)'l',(byte)'>',(byte)'<',(byte)'h',(byte)'e',(byte)'a',(byte)'d',(byte)'>',(byte)'<',(byt
e)'t',(byte)'i',(byte)'t',(byte)'l',(byte)'e',(byte)'>',(byte)'P',(byte)'h',(byte)'o',(byte)'n',(byte)'e',(byte)'B',(byte)'o',(byte)'o',(b
yte)'k',(byte)'
',(byte)'P',(byte)'l',(byte)'u',(byte)'s',(byte)'<',(byte)'/',(byte)'t',(byte)'i',(byte)'t',(byte)'l',(byte)'e',(byte)'>',(byte)'<',(byte)'
/',(byte)'h',(byte)'e',(byte)'a',(byte)'d',(byte)'>',(byte)'<',(byte)'b',(byte)'o',(byte)'d',(byte)'y',(byte)'>',(byte)'<',(byte)'i',(by
te)'m',(byte)'g',(byte)'
',(byte)'s',(byte)'r',(byte)'c',(byte)'=',(byte)'"',(byte)'r',(byte)'e',(byte)'s',(byte)'o',(byte)'u',(byte)'r',(byte)'c',(byte)'e',(byte)'
:',(byte)'/',(byte)'/',(byte)'s',(byte)'i',(byte)'m',(byte)'A',(byte)'l',(byte)'l',(byte)'i',(byte)'a',(byte)'n',(byte)'c',(byte)'e',(byte)'
.',(byte)'j',(byte)'p',(byte)'g',(byte)'"',(byte)'
',(byte)'h',(byte)'s',(byte)'p',(byte)'a',(byte)'c',(byte)'e',(byte)'=',(byte)'"',(byte)'3',(byte)'"',(byte)'
',(byte)'v',(byte)'s',(byte)'p',(byte)'a',(byte)'c',(byte)'e',(byte)'=',(byte)'"',(byte)'3',(byte)'"',(byte)'
',(byte)'b',(byte)'o',(byte)'r',(byte)'d',(byte)'e',(byte)'r',(byte)'=',(byte)'"',(byte)'1',(byte)'"',(byte)'>',(byte)'<',(byte)'b',(byt
e)'r',(byte)'>',(byte)'<',(byte)'h',(byte)'3',(byte)'>',(byte)'S',(byte)'o',(byte)'r',(byte)'r',(byte)'y',(byte)',',(byte)'
',(byte)'t',(byte)'h',(byte)'e',(byte)'
',(byte)'P',(byte)'h',(byte)'o',(byte)'n',(byte)'e',(byte)'B',(byte)'o',(byte)'o',(byte)'k',(byte)' ',(byte)'i',(byte)'s',(byte)'
',(byte)'e',(byte)'m',(byte)'p',(byte)'t',(byte)'y',(byte)'!',(byte)'<',(byte)'/',(byte)'h',(byte)'3',(byte)'>',(byte)'<',(byte)'/',(byt
e)'b',(byte)'o',(byte)'d',(byte)'y',(byte)'>',(byte)'<',(byte)'/',(byte)'h',(byte)'t',(byte)'m',(byte)'l',(byte)'>'
      };

      private static final byte[] headerPage=
      {
            //<html><head><title>PhoneBook Plus</title></head><body><img src="simAlliance.jpg" hspace="3" vspace="3"
border="1"><br><h2>PhoneBook Plus</h2>

(byte)'<',(byte)'h',(byte)'t',(byte)'m',(byte)'l',(byte)'>',(byte)'<',(byte)'h',(byte)'e',(byte)'a',(byte)'d',(byte)'>',(byte)'<',(byt
e)'t',(byte)'i',(byte)'t',(byte)'l',(byte)'e',(byte)'>',(byte)'P',(byte)'h',(byte)'o',(byte)'n',(byte)'e',(byte)'B',(byte)'o',(byte)'o',(b
yte)'k',(byte)'
',(byte)'P',(byte)'l',(byte)'u',(byte)'s',(byte)'<',(byte)'/',(byte)'t',(byte)'i',(byte)'t',(byte)'l',(byte)'e',(byte)'>',(byte)'<',(byte)'
/',(byte)'h',(byte)'e',(byte)'a',(byte)'d',(byte)'>',(byte)'<',(byte)'b',(byte)'o',(byte)'d',(byte)'y',(byte)'>',(byte)'<',(byte)'i',(by
te)'m',(byte)'g',(byte)'
',(byte)'s',(byte)'r',(byte)'c',(byte)'=',(byte)'"',(byte)'s',(byte)'i',(byte)'m',(byte)'A',(byte)'l',(byte)'l',(byte)'i',(byte)'a',(byte)'
n',(byte)'c',(byte)'e',(byte)'.',(byte)'p',(byte)'n',(byte)'g',(byte)'"',(byte)'
',(byte)'h',(byte)'s',(byte)'p',(byte)'a',(byte)'c',(byte)'e',(byte)'=',(byte)'"',(byte)'3',(byte)'"',(byte)'
',(byte)'v',(byte)'s',(byte)'p',(byte)'a',(byte)'c',(byte)'e',(byte)'=',(byte)'"',(byte)'3',(byte)'"',(byte)'
',(byte)'b',(byte)'o',(byte)'r',(byte)'d',(byte)'e',(byte)'r',(byte)'=',(byte)'"',(byte)'1',(byte)'"',(byte)'>',(byte)'<',(byte)'b',(byt
e)'r',(byte)'>',(byte)'<',(byte)'h',(byte)'2',(byte)'>',(byte)'P',(byte)'h',(byte)'o',(byte)'n',(byte)'e',(byte)'B',(byte)'o',(byte)'o',
(byte)'k',(byte)' ',(byte)'P',(byte)'l',(byte)'u',(byte)'s',(byte)'<',(byte)'/',(byte)'h',(byte)'2',(byte)'>'
      };

      private static final byte[] resourceText=
      {
            //<hr><img src="resource___
            (byte)'<',(byte)'h',(byte)'r',(byte)'>',(byte)'<',(byte)'i',(byte)'m',(byte)'g',(byte)'
',(byte)'s',(byte)'r',(byte)'c',(byte)'=',(byte)'"',(byte)'r',(byte)'e',(byte)'s',(byte)'o',(byte)'u',(byte)'r',(byte)'c',(byte)'e',(byte)'
_'
      };

      private static final byte[] trailerText=
      {
            (byte)'.',(byte)'p',(byte)'n',(byte)'g',(byte)'"',(byte)'
',(byte)'h',(byte)'s',(byte)'p',(byte)'a',(byte)'c',(byte)'e',(byte)'=',(byte)'"',(byte)'3',(byte)'"',(byte)'
',(byte)'v',(byte)'s',(byte)'p',(byte)'a',(byte)'c',(byte)'e',(byte)'=',(byte)'"',(byte)'3',(byte)'"',(byte)'
',(byte)'b',(byte)'o',(byte)'r',(byte)'d',(byte)'e',(byte)'r',(byte)'=',(byte)'"',(byte)'1',(byte)'"',(byte)'>',(byte)'<',(byte)'b',(byt
e)'r',(byte)'>',(byte)'<',(byte)'p',(byte)'r',(byte)'e',(byte)'>'
      };

      private static final byte[] platformReqText =
      {
            //<a href="platformRequest://tel:
            (byte)'<',(byte)'a',(byte)'
',(byte)'h',(byte)'r',(byte)'e',(byte)'f',(byte)'=',(byte)'"',(byte)'p',(byte)'l',(byte)'a',(byte)'t',(byte)'f',(byte)'o',(byte)'r',(byte)'
m',(byte)'R',(byte)'e',(byte)'q',(byte)'u',(byte)'e',(byte)'s',(byte)'t',(byte)':',(byte)'/',(byte)'/',(byte)'t',(byte)'e',(byte)'l',(byte
)':'
      };
```

```
	private static final byte[] tagTrailer =
	{
		(byte)'"', (byte)'>'
	};


	private static final byte[] httpTrailer =
	{

(byte)'<',(byte)'/',(byte)'b',(byte)'o',(byte)'d',(byte)'y',(byte)'>',(byte)'<',(byte)'/',(byte)'h',(byte)'t',(byte)'m',(byte)'l',(byte
)'>'
	};


	// Cache control pragma
	// "Pragma: no cache"
	public static final byte[] cacheControl =
	{
		(byte)'P',(byte)'r',(byte)'a',(byte)'g',(byte)'m',(byte)'a',(byte)':',(byte)'		',(byte)'n',(byte)'o',(byte)'-
',(byte)'c',(byte)'a',(byte)'c',(byte)'h',(byte)'e'
	};

 }
```

14 Document history

Version	**Published**	**Main changes**
1.0	**9/12/2009**	**First issue**