



Java Card Management (JCM)  
Task Force

# JAVA CARD MANAGEMENT SPECIFICATION

Version 1.0b  
06 October 2000

# Table of Content

<b>1</b>	<b>EXECUTIVE SUMMARY</b>	<b>3</b>
<b>2</b>	<b>CONVENTIONS</b>	<b>4</b>
<b>3</b>	<b>TASK FORCE OVERVIEW</b>	<b>5</b>
3.1	HISTORY	5
3.2	OBJECTIVES	6
3.3	NON OBJECTIVES	8
3.4	METHODOLOGY	9
<b>4</b>	<b>CARD MANAGEMENT REQUIREMENTS</b>	<b>10</b>
4.1	ASSUMPTIONS	10
4.2	GENERIC REQUIREMENTS	11
4.3	JAVA CARD REQUIREMENTS	12
4.3.1	GENERAL TOPICS	12
4.3.2	SECURITY NEEDS	13
<b>5</b>	<b>GLOBAL SPECIFICATION</b>	<b>15</b>
5.1	USE CASES	15
5.1.1	GENERIC ACTORS	15
5.1.2	GENERIC USE CASES	16
5.1.3	JAVA CARD USE CASES	16
5.2	CLASS DIAGRAMS	25
5.2.1	PRELIMINARY RECOMMENDATIONS	25
5.2.2	GENERIC CLASSES	29
5.2.3	JAVA CLASS DIAGRAMS	30
5.3	JAVA CARD SEQUENCE DIAGRAMS	35
<b>6</b>	<b>JAVA CARD MANAGEMENT API</b>	<b>37</b>
<b>7</b>	<b>STANDING QUESTIONS, AND ANSWERS</b>	<b>38</b>
7.1	STANDING QUESTIONS	38
7.2	ANSWERS	38
<b>8</b>	<b>NEXT STEPS</b>	<b>39</b>
<b>9</b>	<b>GLOSSARY – DEFINITIONS - REFERENCES</b>	<b>40</b>
<b>10</b>	<b>ANNEX 1: DATA DICTIONARY</b>	<b>41</b>

# 1 Executive summary

*This chapter summarises all topics of the present document.*

A previous task force beginning September 98 delivered a first document "Java Card Management Framework version 1.0 of 05/01/2000". This document is available on the Java Card Forum website [www.javacardforum.org](http://www.javacardforum.org).

Following on from this deliverable, the present task force, beginning September 1999, has:

- Defined the core of a Java CMS, on top of which companies could build their own CMS with any specific value-added objects/functions.
- Proposed on-card APIs for Java Card management, used through an off-card CMS.

The methodology has been the following:

- Definition of JCMS requirements using plain text.
- Definition of UML Use Cases, describing pertinent examples of JCMS usage.
- Deduction of Specification elements, partially using UML notions when convenient, and plain text:
  - Class diagrams: static description of JCMS objects.
  - Sequence diagrams: dynamic description of JCMS functionality.
- Deduction of necessary APIs using Java conventions (javadoc/HTML).

Requirements addressed were:

- Functional: Generic, Java related.
- Security: some models were defined.

Some APIs have been defined according to common decisions from all the companies involved. These APIs could mainly be used to 'discover' a card that has been issued by a first entity, then updated post-issuance by other entities. They include ways to:

- Identify the card: GetCardGUID
- Discover the card configuration: GetFreeNonVolatileMemory, GetJVMVersion, ...
- Discover the card content: GetAID, GetAIDVersion, ...

Implementation has been partially addressed through high level architecture dealing with standard process() method, RMI, ...

Next steps are described in the present document and should be mainly to address the security mechanisms for this API, and to complete the API in order to address more complex scenarios.

## 2 Conventions

*This chapter specifies conventions used in the present document.*

The present document uses the following typographic and presentation conventions:

- At the beginning of each chapter, a sentence written in *italic* format specifies the chapter content.
- Specification diagrams follow the UML conventions.
- Code examples are in `Courier` font.

### 3 Task force overview

*This chapter describes the Java Card Management System task force.*

#### 3.1 History

The Java Card Management (JCM) Task Force was initiated by Christian Goire and Bertrand Du Castel during the JCF meeting in Munich, September 22<sup>nd</sup> to 24<sup>th</sup>, 1998.

The initial objective for the JCM task force members was to define the term Java Card Management in relation to existing smart card terminology and JCF activities.

The deliverable was the document "Java Card Management Framework version 1.0, 05/01/2000", delivered to the JCF Technical Committee in Berlin on 30<sup>th</sup> November – 2<sup>nd</sup> December 1999. This document is available on the Java Card Forum website [www.javacardforum.org](http://www.javacardforum.org).

#### Participants of the task force first part (until JCF in Boston: 22nd and 23rd of September 1999):

Walter Haenel - Chairman	IBM	<haenel@de.ibm.com>
Jean-Claude Huot	Oberthur	<jc.huot@oberthursc.com>
Blake Ryan	Motorola	<A13278@email.mot.com>
Tim Jurgensen	Schlumberger	<tmjurgensen@slb.com>
Nils Diezmann	G&D	<nils.diezmann@gdm.de>
Claus Dietze	G&D	<claus.dietze@gdm.de>
Al Chan	VISA	<alchan@visa.com>
Lisa Lobue	VISA	<llobue@visa.com>
Corinne Bonifas	Gemplus	<corinne.bonifas@gemplus.com>
Gigi Ankeny	Bull	<gigi.ankeny@bull.com>
Dipendra Chowdhary	Motorola	<dchowdha@asc.corp.mot.com>
Sebastian Hans	Sun	<Sebastian.Hans@Swiss.Sun.COM>
Martine Guignard	DeLaRue	<Martine.Guignard@fr.delarue.com>
Tony Jeffreys	KeyCorp	<tjeffreys@keycorp.com.au>
Marie-Noelle Lepareux	Oberthur	<mn.lepareux@oberthursc.com>
Boutheina Chetali	Bull	<boutheina.chetali@bull.net>
Pierre Sagnières	Gemplus	<pierre.sagnieres@gemplus.com>

#### Participants of the task force second part (from end of JCF in Boston: 22nd and 23th of September):

Pierre Sagnières - Chairman	Gemplus	<a href="mailto:pierre.sagnieres@gemplus.com">pierre.sagnieres@gemplus.com</a>
Tim Jurgensen	Schlumberger	<a href="mailto:tmjurgensen@slb.com">tmjurgensen@slb.com</a>
Boutheina Chetali	Bull	<a href="mailto:boutheina.chetali@bull.net">boutheina.chetali@bull.net</a>
Bill Reding	Datacard Platform7	<a href="mailto:bill.reding@platform7.com">bill.reding@platform7.com</a>
Sebastian Hans	Sun	<a href="mailto:Sebastian.Hans@Swiss.Sun.COM">Sebastian.Hans@Swiss.Sun.COM</a>
Thierry Gualini	Oberthur Card Systems	<a href="mailto:t.Gualini@oberthurcs.com">t.Gualini@oberthurcs.com</a>

With collaboration of:

Jack Pan	Citibank	<a href="mailto:jack.pan@citicorp.com">jack.pan@citicorp.com</a>
----------	----------	--

### 3.2 Objectives

*This chapter defines the objectives of the task force.*

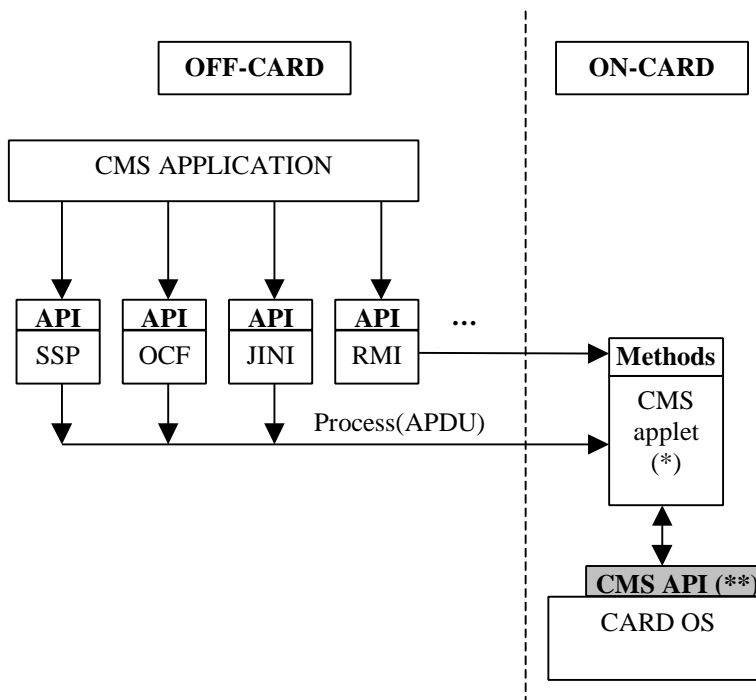
The objectives of the second task force are globally to complete the work of the previous task force with:

- The specification of the core of a Java CMS, on top of which companies could build their own CMS with any specific value-added objects/functions.
- The definition of proposed on-card APIs for Java Card management, used through an off-card CMS.
- Delivering specification documents:
  - Word document specifying the JCMS.
  - HTML files representing the proposed APIs.

In the present document, a CMS is a computer platform (backend, frontend, network access, software/hardware) for managing cards used by end-users in the filed: issue cards, remotely manage cards, re-issue cards, ... For a more detailed description of a CMS please refer to the first document of the task force on the website [www.javacardforum.org](http://www.javacardforum.org).

In a standard JCMS environment as described below, the actual work of the Task Force is focused on the basic On-Card CMS API. When defined, this on-Card API could be used by other actors in different ways:

- Proprietary implementations through high level framework (SSP, OCF, JINI, ...), or card applet implementation or standard Java Card methods (process()).
- Card applet may be implemented in emerging standard environments: OP, CMS vendors offerings, ...
- Future RMI interface to the Java Card.



SSP: Smart card Service Provider  
 RMI: Remote Method Invocation  
 (\*\*): May be included in the JCRE.

OCF: Open Card Framework  
 (\*): May be a specific applet, maybe OP Card Manager, ...

As the proposed on-card API may not completely fit on-card, the implementation could be partially on-card, partially off-card in the CMS. The present document also supplies proposals about mandatory/optional on-card information.

#### API implementation freedom:

Some card manufacturers could decide to implement, partially implement, or even not to implement the API responses. This on-card API can be used by any applet on the card, however according to specific/proprietary decisions or security policies, the API may not always answer.

#### Implementation phases in the future:

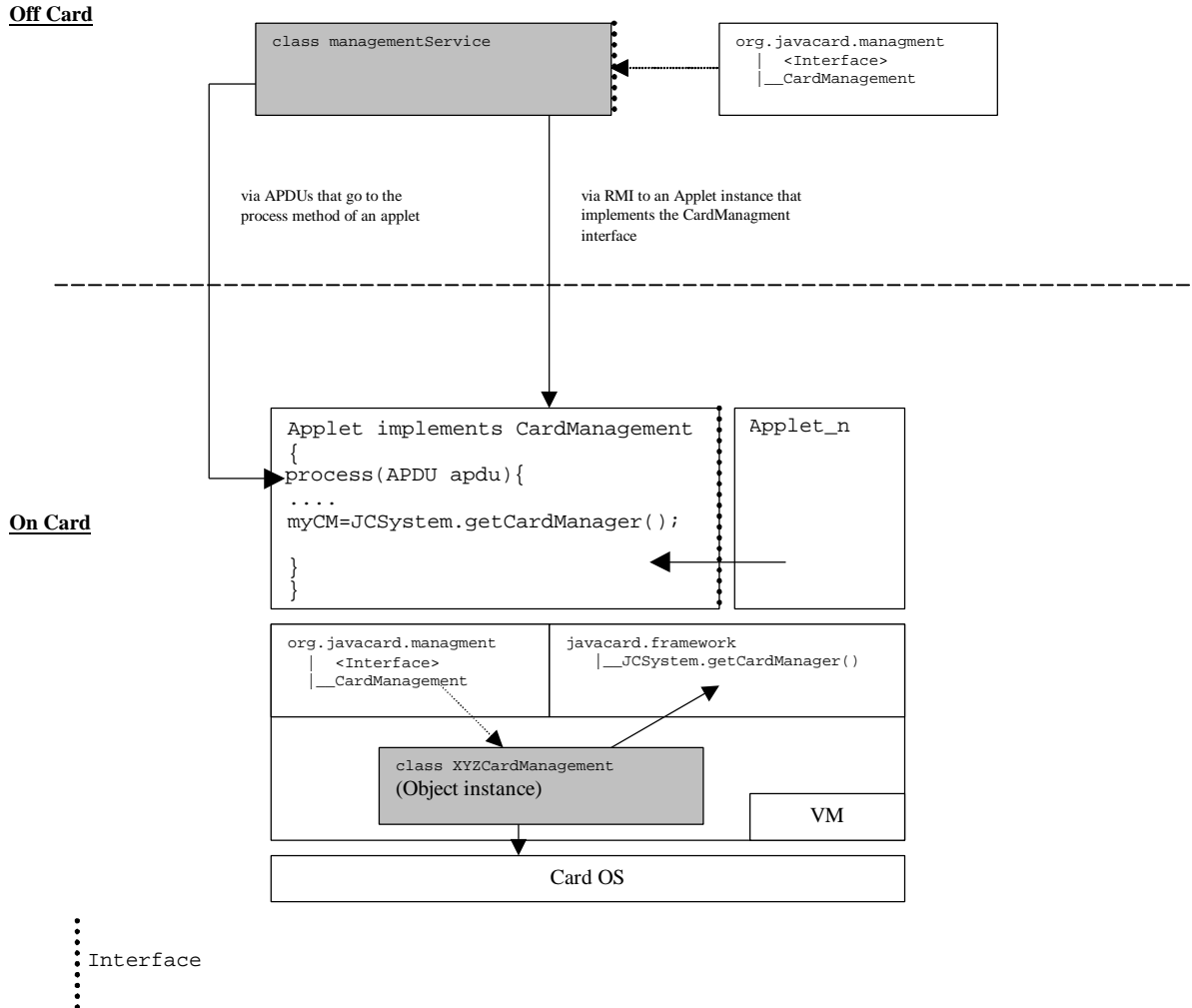
As the on-card API is not visible from outside the card, there must be a mechanism to open this API to the off-card world. Different mechanisms could be:

- A proprietary applet implementing the communication between the off-card world and the CMS API. This applet is developed/implemented by the company most interested in using the API, so implementing the required level of security: CMS developers, Card issuers, ...
- An implementation of OP Card Manager.
- API included in JCRE
- ...

The different phases in the near future could be:

- Phase A: RMI is not available: During this phase, APDUs exchanged between off-card and on-card CMS applet are not standardized. The security level is implemented by the company developing the applet.
- Phase B: RMI is available in the card manager. The CMS applet is card-independent.
- Phase C: The security could be standardized, or rationalized. At this point the CMS applet could be more or less independent of the Card Issuer.

The diagram below describes possible implementations:



The CardManagement interface is implemented by a concrete object in the JCRE. An applet gets access to this object by calling a method from the JCSYSTEM class. The CMApplet can then invoke all the methods defined in the CardManagement interface. The CMApplet can export this information to other applets in the card by implementing the CardManagement interface by itself. The off-card world can retrieve the information provided by the CardManagement object via the process method of the CMApplet. In future when we have RMI between the off-card VM and the on-card VM we can directly call the methods of the CardManagement interfaces implemented by the CMApplet.

### 3.3 Non Objectives

*This chapter defines the objectives which are outside the scope of the task force.*

The topics which are not included in our work are listed below.

- Our scope is not to define/standardize APDUs for card APIs.
- Our scope is not to define a complete and standardized CMS.
- Multi-chip cards (Cards with more than one chip on it).

### 3.4 Methodology

*This chapter defines the work methodology to achieve objectives.*

The methodology is the following:

- Definition of JCMS **requirements** using plain text, for functional requirements and non-functional requirements (openness, ...).
- Definition of **Use Cases**, describing pertinent examples of JCMS usage.
- Deduction of **Specification** elements, partially using UML notions when convenient, and plain text:
  - **Class diagrams**: static description of JCMS objects.
  - **Sequence diagrams**: dynamic description of JCMS functionality.
  - **Packages**: partitions of the JCMS specifications (if necessary).
- Deduction of necessary APIs using present Java conventions (javadoc/HTML).

The present specification addresses partially a generic CMS specification (considered as a pre-requisite), and focuses mainly on a Java CMS specification.

On the other hand we also used other approaches to derive those APIs:

- Bottom up approach: from a card point of view, and with no relation with management functions, what could be useful ?
- From interoperability points defined in the previous task force document, what could be added to ease Card Management?

## 4 Card management requirements

*This chapter defines the requirements of a Card Management System: generic requirements and Java Card specific requirements.*

The Generic requirements below are requirements that are supposed to be in all CMSs, and so will not be defined in this task force. On top of those requirements we will define Java Card requirements.

### 4.1 Assumptions

*This chapter defines the assumptions before defining the target requirements.*

Our assumptions before defining the present CMS APIs are the following:

- Card bootstrap: A booting mechanisms (like ATR, enhanced ATR, 'bootstrap' mechanism, ...) has been used to begin to identify the card. The 'bootstrap' information must be available before establishing the secure environment. APIs will just confirm this information 'afterwards' in a secure environment (Please refer to 5.2.1 What about card identification?).
- Some level of secure channel between card and CMS has been established: confidentiality and integrity are established
- Some level of mutual authentication between card and CMS has been established.
- The entity asking for an information on the card, through the CMS-API, has the privilege to do so.
- Legacy systems: Our scope is java cards. For previous java card versions, to be managed by the CMS, the CMS just tries with standard commands: select card manager, select applet, ...

On top of this we define the APIs.

Above, 'Some level' means different possibilities:

- 'No security available': non private channel, no authentication, ...
- Symmetric key cryptography.
- PKI.
- ...

The CMS and the card deal with the established level of security on their own, to determine what is possible to be done: no dialog at all, some dialog, strong interaction between card and CMS, proprietary actions, ...

We have considered that the security topics above were at this point outside the scope of the present task force. If necessary, the present task force could define those security aspects on a second round, or this could be done by another task force.

## 4.2 Generic requirements

*This chapter defines the generic requirements of a CMS even before managing java cards.*

The generic requirement is the management of physical cards, from manufacturing to deletion, including issuance and updates inside the card, done by the unique card issuer.

The requirements are the following:

- Card repository: To be able to manage cards in the field, there must be a kind of repository of cards describing cards with general attributes, like for example card identifier, card type, bill of materials, chip type, OS type, card security, ... This card repository could be very complete, or some information could reside on the card itself. However, as there is a unique card issuer (in this generic scope), the only actor interacting with the card, there is no discovery process of an unknown card, so there is little interest in having much on-card information.
  
- Card application repository: To be able to manage card applications in the field, there must be a kind of repository of card applications describing the application with general attributes, like for example application identifier, application version, security, ... This card application repository could be very complete, or some information could reside in the application itself, or in the card OS. However, as there is a unique card issuer, the only actor interacting with the card, there is no discovery process of an unknown card, so there is little interest in having much on-card information about installed applications.
  
- Management functions for every card state and transition of the card lifecycle: manage all card characteristics (create, retrieve, update, block, unblock, diagnose, delete) in the CMS and for the card itself, links between cards and card applications, transition logging, ...
  
- Post-issuance of card applications on cards already managed in the system: download applications, replace/delete existing card applications, ...
  
- Basic Non-functional requirements for the CMS itself are:
  - Technical constraints: portable onto different OSs and DBMSs, ...
  - Flexibility: easy to add or customize functions, ...
  - Openness: well defined interface, standards usage, ...
  - Performance: scalable in terms of performance to meet different performance requests, ...
  - Security: controlled access with security basics: user authentication, message integrity, confidentiality, non repudiation of transactions, ....
  
- ...

## 4.3 Java Card requirements

*This chapter defines topics closer to JCMS, so more precisely described.*

### 4.3.1 General topics

Card management of physical Java Cards, done by different organisations, one of them issuing the card, others updating the card content.

The main difference is that, if different actors are able to update the card content, there must be a way of knowing the card content in order to manage it. This could be:

- A mechanism between different JCMS in order to have information on different actions done on a card.
- An on-card mechanism in order to have those information stored in the card and ready to be used (solution we promote in this document).

The requirements are the following:

- Card repository must include specific Java characteristics: JVM characteristics, specific security mechanisms, ...
- Applet repository: applets/packages, applet issuers, ...
- Links between cards and applets from different issuers.
- Multi-management: applet incompatibilities, applet dependencies (packages, shareable interfaces), security issues, ...
- Post-issuance of applets on 'unknown' Java Cards (cards not already managed in the system).
- Resource management for installed applets: memory usage of applets, crypto capabilities, ...
- Management of different functional domains on the same java card: bank, GSM, loyalty, payment, ...
- Java card specific non-functional requirements for the CMS itself are:
  - Technical constraints: should have java access mechanisms in order to better interact with java cards (RMI, ...).
  - Flexibility: be able to manage different java cards and java card environments from different card issuers.

Mainly, these requirements mean that there must be a card discovery mechanism, in order to have information on a card used/updated by actors other than the card issuer.

### 4.3.2 Security needs

*This chapter defines security needs in the Java Card management scope.*

This section looks at security from a card management viewpoint, to see how the Java Card APIs need to be extended for security purposes. It is based on a 'bottom-up' check of Java Card 2.1 and Open Platform (OP) 2.0, and some understanding of the functions required in card management.

Application management (for example, sending application specific commands to perform functions) is included.

#### Establishing trust:

Authentication needs:

- The CMS (acting for the Card Issuer or Application Provider) may wish to authenticate the card (platform / chip, or applet) when it loads an application, or when it changes the state of a component.
- The card (platform / chip, or applet) authenticates the CMS when it receives a management request, to ensure that it (and associated data or applets) have come from an authentic source.

Other needs: In addition to authentication, there is an imperative need to ensure the integrity of requests and information in transit, and a possible need to ensure confidentiality of private information.

#### Card management models:

Four models of card management are suggested as needing to be supported. They have been addressed only to be able to specify the context of the Use Cases. They are not mandatory models.

##### 1. Centralised card management

In this model, the card (chip) is established by the Card Issuer, and Applet Issuers may only load applications with the approval of the Card Issuer.

##### 2. Devolved card management

In this model, the card (chip) is established by the Card Issuer, who then devolves card management privilege to Card Management Organisations. This might, for example, involve the Card Issuer signing CMO key certificates, so that Card Management Organisations can subsequently interact directly with an on-card Card Resources Manager.

##### 3. Delegated card management

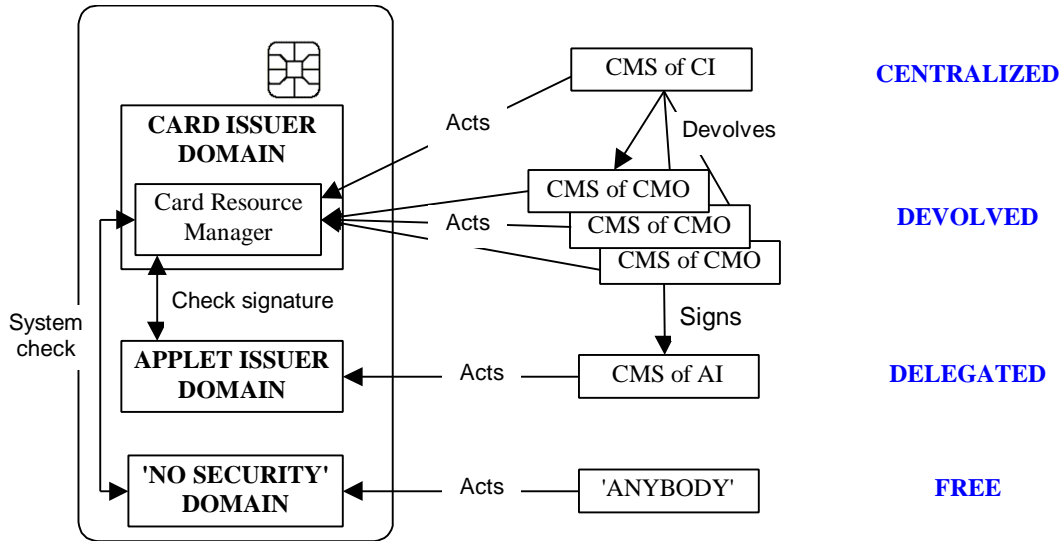
This is an option within Centralised Card Management, whereby the Card Issuer delegates some authority to the Applet Issuer. This allows the Applet Issuer to use an on-card Applet Issuer Domain (such as the OP Security Domain) to act on its behalf and feed card resource management requests through to the Card Resources Manager.

For the first three above models, an implication here is that security mechanisms have to be established to support card management, for example: PKI, limited operation using symmetric cryptography alone, ...

4. Free model

The card can be plugged into any reader of any CMS. In this 'extreme' model the card is established by the card issuer with no security mechanisms. Anybody (CardIssuer, CardManagementOrganizations, CardHolder, CardUser, ...) can use this card to add applets/files with no need of security. In this case the card is used like a little computer storing different files and programs. However the Card Resource Manager acts on the requests from outside the card.

These four models are illustrated in the diagram below.



Legend:

- CMS Card Management System
- CI Card Issuer
- CMO Card Management Organisation
- AI Applet Issuer

Dynamic model usage:

Centralised: The only way it works is that the card is inserted in a reader connected to the Card Issuer CMS. The card is known, its security is known. The CMS can do what it wants on the card, then record any action in the CMS. Card content and CMS card image are always coherent.

Devolved: Along with the first centralised approach, the card can also be plugged in a reader of a CMO that has been 'certified' by the card Issuer. The card is known, part of its security is known. The CMO CMS can do what it wants on the card, as far as the on-card resource manager identifies the CMO CMS. Card content and CMS card image are different. There should be a card discovery mechanism.

Delegated: Along with the previous approaches, the card can also be plugged in a reader of an AI that has been 'certified' by the card Issuer or the CMO. The card is known, part of its security is known, restricted to the AI domain. The AI CMS can do what it wants on the card, as far as it accesses to its AI domain. Card content and CMS card image are different. There should be a card discovery mechanism.

Free: Along with the first centralised approach, the card can also be plugged in any reader of any CMS. The card is unknown. Its security is non-existent. The CMS can do what it wants on the card, as far as the on-card resource manager accepts to let it interact with a free domain. Card content and CMS card image are different. There should be a card discovery mechanism.

## 5 Global specification

*This chapter specifies a Card Management System, partially for a generic CMS, mainly for a Java CMS.*

### 5.1 Use cases

*This chapter presents CMS use cases.*

#### 5.1.1 Generic actors

*This chapter lists and defines all actors related to a JCMS.*

##### Generic actors:

The **Card Manufacturer** (Embedder/Emboss) is the company that produces the cards: embed the processor, personalize the plastic, chip personalization (make the card ready for issuance), ...

The **Chip Manufacturer** is the semiconductor company which fabricates the processor and memory chip on which the multi-application smart card is based, according to specifications made by the Card Issuer.

The **Card Issuer** is the entity legally responsible for (owner of) the card when it is deployed. It generally personalizes the card with applets and personalizes applets to a specific cardholder.

The **Applet Issuer** is the entity responsible for the application delivery (code, data, post-issuance usage, support, ...).

The **Cardholder** is the final user of the card.

The **CMS** is a software system able to manage cards: register cards, access cards to perform management functions, log action on cards, ...

On-card actors: To be able to manage cards, some other on-card actors are needed.

The **Card Resources Manager** is an on-card software component. This accepts card management requests that affect card resources, verifies their authenticity, and performs the requested function. Examples of such functions are downloading an applet or updating a card-wide PIN. It could be accessed as a master file function or as a "management application" such as the OP Card Manager.

The Card Resources Manager may be configured so that it will handle off-card requests from the Card Issuer alone, or from Application Issuers under the devolved card management model.

This CRM could be implemented in different ways, possibly inside the JCRE.

An **Applet Domain Manager**. This can provide an on-card service to Applet Issuers under the delegated card management model. It provides authentication of the CMS, and data confidentiality, but leaves card resource management to the Card Resources Manager. It can also provide services to the application, as it may hold application cryptographic keys. This actor is similar to the Security Domain defined in OP.

An **Application** on the card. This accepts card management requests that affect the application, verifies their authenticity and performs the function. It may use an Applet Domain Manager to perform cryptographic functions. Examples of functions are personalizing the application using a Put Data command, blocking the application using an EMV script command, or changing an application PIN using an application proprietary command. The application is also an actor in some card management functions that affect card resources, under the Card Resource Manager control.

##### Other actors identified by the previous task force:

The **Platform Specifications Owner** is the entity which controls (owns) a specification on which interoperability of some aspect of the entire system is based. For example, it could be SUN for the Java Card standard, Visa for the Open Platform specification, or ETSI for GSM..

The **Compliance Authority** is an institution that proofs the cards and the applications needed according to specific requirements (ITSEC, national approvals...).

The **Government Authority** deals with national exportation rules and regulations associated with cryptography which have to be taken into account. Some legal rules on private data protection or security level evaluation should be applied.

The **Network Provider** is a company which provides services to transport data and applications.

### 5.1.2 Generic use cases

*This chapter presents generic CMS use cases.*

Examples of generic use cases are the following:

- Basic management (create, query, delete, ...): terminals (mobile phone, connected card terminal, ...), cards, card applications, cardholders, accounts, ...
- Issuance process management.
- Interface with external/legacy systems: billing, customer services, end-user services on Internet, traceability, ...
- Post-issuance proprietary download(s), by the Card issuer.
- Corrupted cards, blacklist management, ...
- ...

As they are generic (common to different CMSs), they are not described in the present document.

### 5.1.3 Java card use cases

*This chapter presents Java CMS use cases.*

Examples of Java card use cases are the following:

- Specific java card management: JVM, Applet issuers, security topics,...
- Specificity of Java Card issuance.
- Post-issuance download(s).
- Emergency card replacement (card lost/destroyed).
- Card Auditing.
- Blacklist cross information between applets providers.
- Functional specific uses cases: Telephony, Financial, IT, Identity, Access, Health, ...
- Download: , activation/deactivation, functional management (loyalty points, e-purse balance, ...) ...
- Memory(s) management: defragmentation, garbage collection, ...
- White cards.
- ...

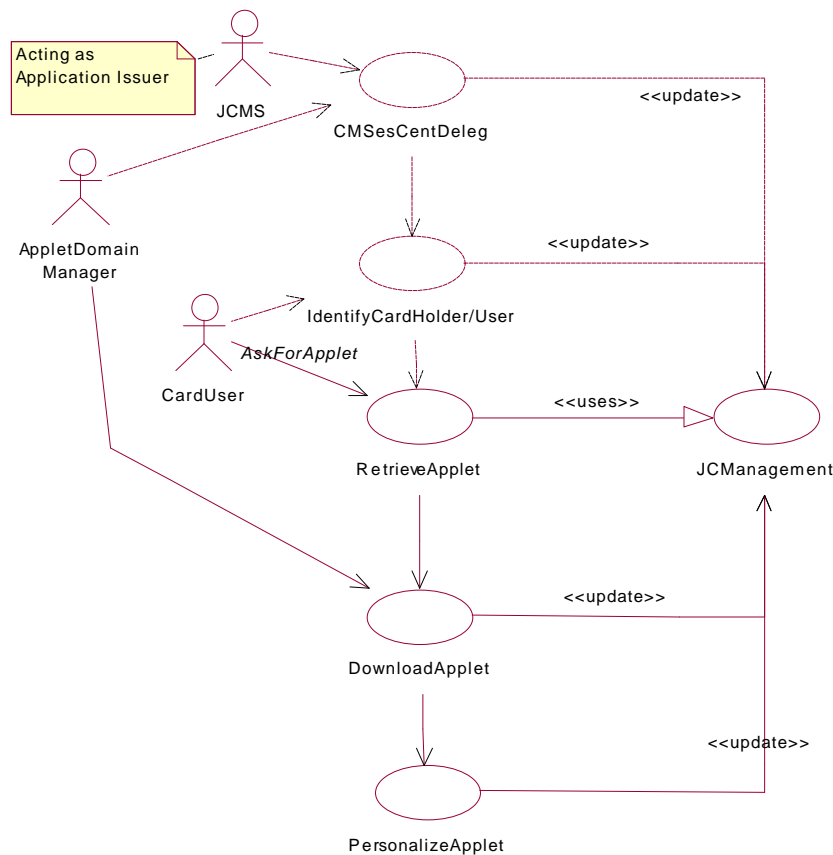
Among these use cases, some have been chosen by the task force to be more precisely described below.

5.1.3.1 Post-issuance download

Use case description:

Post-issuance download of a new applet, requested by the cardholder, performed by an applet provider different from the Card issuer (Delegated security model). Applet is already known and qualified, and in the JCMS.

Use case diagram:



Detailed description:

- CMSEScentDeleg: The CMS establishes a Delegated secure session with the card.
- IdentifyCardHolder: Identify the CardHolder of the card.
- RetrieveApplet: Retrieve applet according to CardHolder criteria.
- DownloadApplet: Download the applet into the card.
- PersonalizeApplet: Personalise the applet previously downloaded.
- JCManagement: Java Card management inside the JCMS: create, select, update, delete, ...

Conclusion:

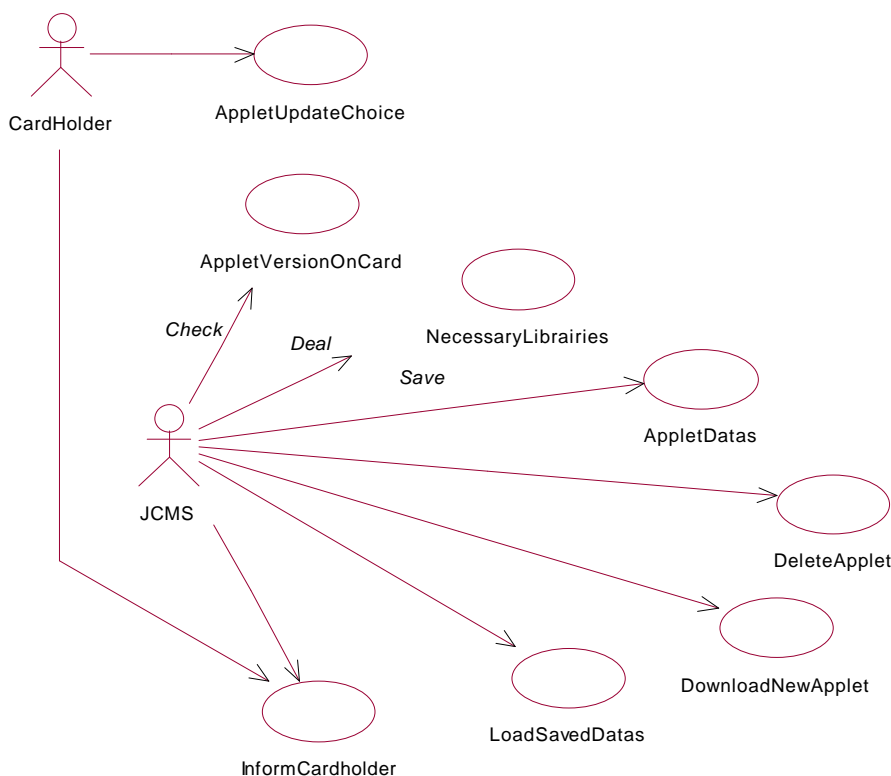
APIs needed are mainly to have information on card resources (free space, ...), applets already installed, ....

### 5.1.3.2 Applet update in a card

#### Use case description:

The cardholder initiates an update of an Applet available on his card. This results in the CMS updating an existing applet/package (which is in his responsibility scope) in the card with a new version of the applet, checking the jc version, the existence of libraries/packages inside the card (save functional data, then delete the applet, then install the applet, instantiate the applet then re-personalize). Take into account the interface with the applet provider, and security issues.

#### Use case diagram:



#### Detailed description:

- **AppletData:** Specific applet's data are saved for future use. Those data are considered as confidential so those data remain on the card and are never exposed out of the card. An access mechanism must be used in order for the new applet to access those data. The new applet must know how to access these data, how to make a copy of those data in its own structures and how to delete the previous data of the deleted applet
- **AppletUpdateChoice:** The CardUser chooses the applet version to be downloaded on his card.
- **AppletVersionOnCard:** The JCMS checks the version of the installed applet, and compare it with applets inside its database (most recent applets for example).
- **DeleteApplet:** The JCMS deletes the applet on card.
- **DownloadNewApplet:** Load, instantiate and personalize the chosen applet.
- **InformCardholder:** The JCMS informs the CardUser with the results of the update.
- **LoadSavedData:** Previously saved data are re-loaded on card.
- **NecessaryLibrairies:** The JCMS checks if the necessary libraries are present on card, with the right versions.

#### Conclusion:

At this time, this use case is not really implementable. No need to define new API from this Use Case.

5.1.3.3 Applets interactions

Use case description:

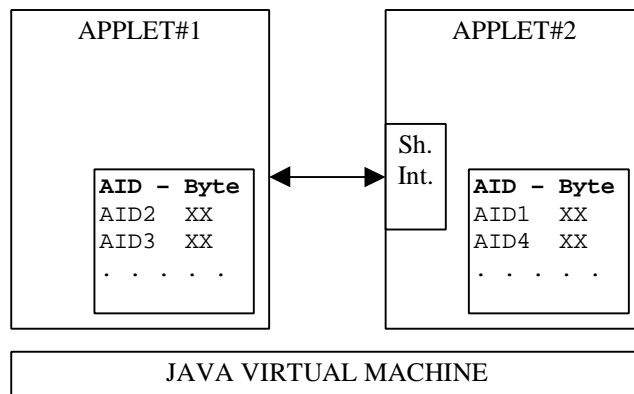
Interaction between applets: Exploration of the environment of the applet to be able to interact with. Exchange loyalty points from one applet to another one. Card management functions before the real exchange only, not the functional exchange of points

Detailed description:

Applet1 and Applet2 are linked loyalty/purse programs, may be installed at two different times in the card life. Applet1 sees if there is enough money on the purse to buy a hamburger (NO), so it looks at Applet2 to see if there are enough loyalty points to buy this hamburger (because there is loyalty program, and you can buy hamburgers with the points of the second applet).

- 1) Applet1 has a list of applets that it can deal with. Does Applet1 need to know if Applet2 is on-card? In fact no, because Applet1 has just to use GetAppletShareableInterfaceObject().
- 2) Applet1 gets an object reference of Applet2: done by GetAppletShareableInterfaceObject()
- 3) Applet1 invokes method on object in Applet2: Standard java card.
- 4) Applet2 checks if Applet1 is a registered applet with which it can deal.

Security is implemented through applicative matters between the two applets (ACL, ...), and is out of our scope.



This use case gives no new APIs, as it can be run with standard Java Card 2.1.

#### 5.1.3.4 Card and applet lifecycles management

Initial study:

List the use case where Card lifecycles and Applet Lifecycles must be used, deduce two state transition diagrams (not contrary with OP, GSM, ...). Then list the APIs for the Applet class and the Card Class to manage these lifecycles.

Conclusion:

Be open, and give the opportunity to define/use different lifecycles, and different related states. This can be done at issuance time, or during the life of the different entities: card, applet, package, ...  
See data model.

### 5.1.3.5 Security usage in CMS

#### Initial RFP:

Both bottom up and top down (VOP and jc2.1 analysis for security aspects only) approach, to list the security elementary use cases needed in a java card (if possible relations between these use cases: chronology, dependencies, ...). Examples: authentication of the card by the JCMS, secure channel (ciphering, signature of the applet), ...

Other use cases will just use these particular security aspects.

#### List of identified use cases:

Establish card management session, centralised, no delegation: Establish a secure session between CMS and Card Resources Manager (or 'platform') to load an application. Assume that cardholder authentication is not needed in this case.

Establish card management session, centralised, with delegation: Establish a secure session between CMS and an Applet Domain Manager to load an application. Assume that cardholder authentication is not needed in this case.

Approve devolved Application Issuer: Establish an Application Issuer for a card. In a PKI environment for example, this means sign the certificate of the Application Issuer.

Establish card management secure session, devolved: Establish a secure session between CMS and Card Resources Manager (or 'platform') to load an application. Assume that the cardholder requests the session, and cardholder authentication is needed in this case.

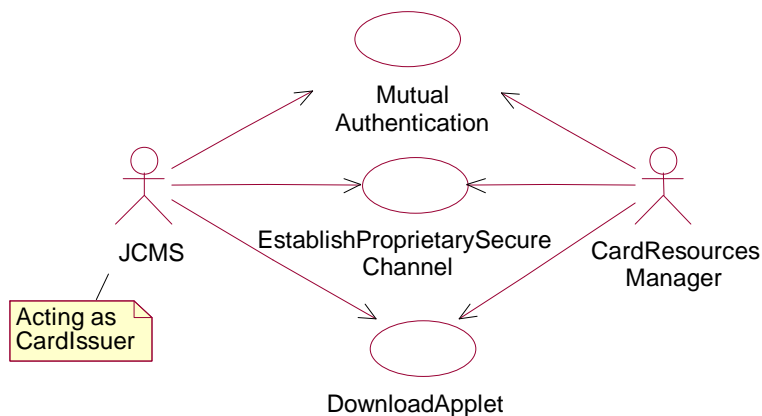
Establish application management secure session: Establish a secure session between CMS and an application, in order to perform management functions. These may be application personalisation, or proprietary functions.

**Use case: Card management session, centralised, no delegation:**

Establish a secure session between CMS and Card Resources Manager (or 'platform') to load an application. Assume that cardholder authentication is not needed in this case.

As it is a centralised CMS with no delegation, all the necessary information is somewhere 'proprietary'. The card and its security mechanisms, which can be very numerous, are known by the CMS. It seems not very relevant to define this use case in the present document, except if we choose standard mechanisms like PKI for example.

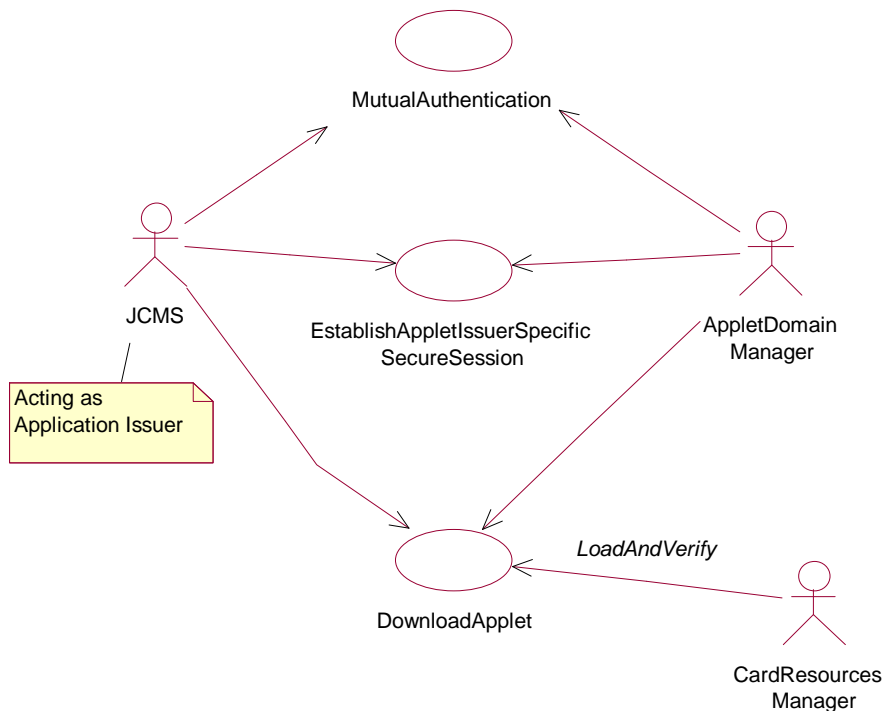
Use case diagram:



**Card management session, centralised, with delegation:**

Establish a secure session between CMS and an Applet Domain Manager to load an application, assuming that the JCMS (acting as the Card Issuer) has approved the applet loading beforehand. Assume that cardholder authentication is not needed in this case.

Use case diagram:

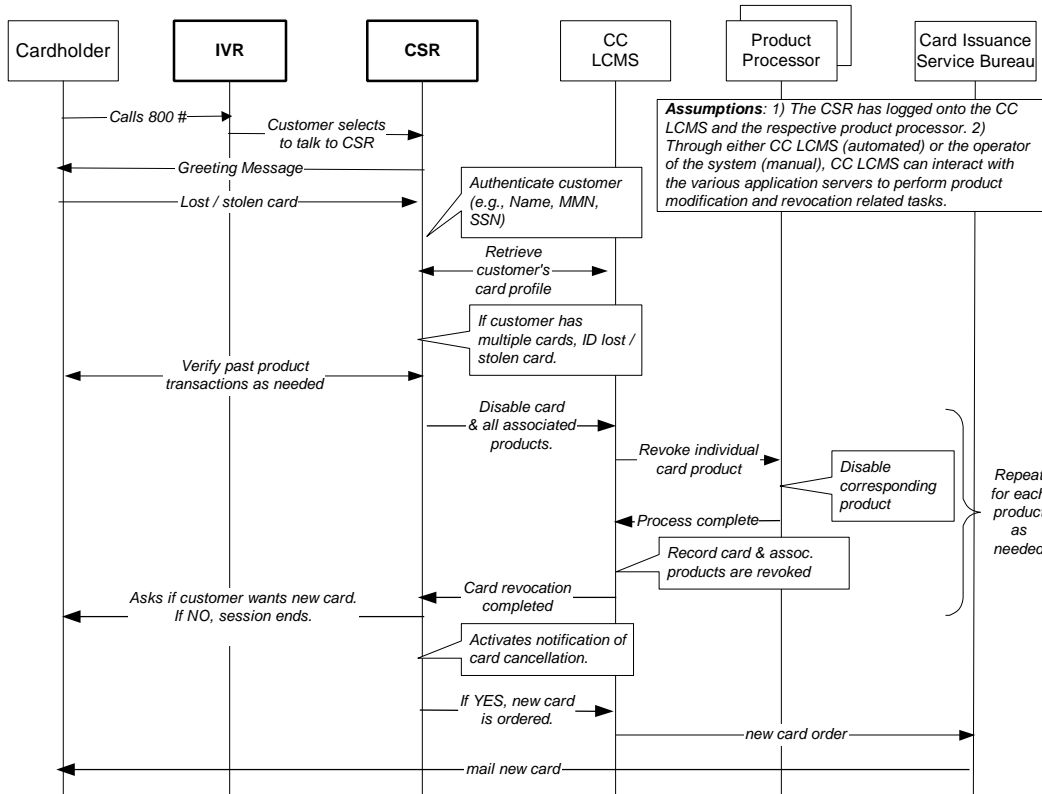


5.1.3.6 Lost/stolen card

Initial study:

Describe a use case when a card has been stolen or lost, and when the user wants to replace it.

Sequence diagram:



- IVR - Interactive Voice Response system
- CC - Chip Card
- LCMS - Life Cycle Management System (for managing card life cycle on the backend)
- Product Processor - also known as Application Server, the backend server for the individual application
- SSN - Social Security Number (in the US)
- CSR - Customer Service Representative
- MMN - Mother's Maiden Name

Conclusion:

As there is no card present (the card has been stolen/lost), there is no need for an API in this use case. An API is needed for the card reconstruction. However, it means to re-execute the different tasks that have been previously done, so no new API is needed for this use case.

Maybe it could be interesting to define an interface between CMSs, in order for a single actor (the Card Issuer) to reconstruct, if possible, all the card with the different applets that were previously installed by different CMSs. As the reconstructor CMS has the list of CMSs (from AIs) that may have interacted with the card to be replaced, possibilities are:

- ask the different CMSs to re-download the previous applets
- ask the AI CMS for the applet and download mechanisms (done by the reconstructor CMS).

In the free model there is no way to reconstruct a card 'automatically'. This part is considered out of our scope.

## 5.2 Class diagrams

*This chapter presents CMS class diagrams.*

### 5.2.1 Preliminary recommendations

#### 5.2.1.1 What about a card discovery mechanism?

##### Requirements

*This paragraph describes a proposal for “bootstrap data” and its access, and also a mechanism for accessing the Card Resources Manager*

A card can be identified in various ways. Existing systems, schemes and platforms have their own ‘local’ methods of identification, such as:

- identifiers in CPLC data (OP cards)
- ICCID (GSM cards)
- MCD Number (Multos cards)

In future there will need to be globally unique identifiers. A Globally Unique Identifier (GUID) can either be a new value:

- IP address (IPv6)
- An identification number based on the scheme defined in ISO/IEC 7812
- An ISO Object Identifier
- A large random number
- Another new value

or it can be a qualifier for the existing ‘local’ value, to make it globally unique. The approach discussed below allows for either approach but specifically proposes the ‘qualifier’ mechanism.

The use of ‘bootstrap’ data is proposed that can be accessed in a manner independent of the card type and which allows off-card systems to discover how to identify and handle a multi-application card. It should at least identify the type of card (Java Card etc), so that the external party will know how to access the card to get further information. It may optionally also provide any other information that is not otherwise available from the card. In this way it will allow the card identification information and card usage details to be brought up to a consistent level for all cards.

The data would be BER/TLV (tag, length, value) encoded according to ISO/IEC 7816-6, so that all data is self-identifying. Some of the data would be as defined in 7816-6, other data would be defined using application class tags as part of a ‘compatible tag allocation scheme’ as defined in 7816-6. For the purposes of this paper, it is assumed that there is a ‘Smart Card Authority’ that will take responsibility for tag registration. In practice this could be an existing organisation such as JCF.

Bootstrap Data would provide at least the following information:

- An identifier for the organization that has assigned the application class tag values. This should be an Object Identifier (OID) as defined in ISO/IEC 8825-1 and ISO/IEC 9834-1. (It should not be a RID, since there is no standard tag value for a RID.) This identifier also serves to distinguish the Bootstrap Data from any other data. In this example, it would indicate <SmartCardAuthority>.
- An identifier to specify the card type and (possibly) the major version. This might, for example, be an Object Identifier indicating <Visa OpenPlatform>, <Visa OpenPlatform version7.4> or <SmartCardAuthority VisaOpenPlatform version7.4>. This will in many cases indicate the general

card management and application selection regime supported by the card. It may be supplemented by additional (optional) data objects.

There would then be various optional information that could be included in Bootstrap Data if it were not available from the underlying card type. Examples of such information are:

- The specific card type, fully defining the chip and its ROM mask (part number?): XYZ31247912-13
- The specific application selection scheme supported: Java Card, MULTOS, VOP, ...
- The AID of the Card Resources Manager or Card Manager (but see the note below).
- A qualifier to make a 'local' Card ID unique.
- A global identifier for the card. There have been several suggestions for this, such as an IP v6 address. However, because of privacy concerns it may not be a suitable item for inclusion in Bootstrap Data, which is available to anyone in clear.

(Announcing and identifying a default or 'implicitly selected' application is not included in this list, as there is a separate ISO mechanism for this.)

There is also a need for a mechanism that allows the Card Resources Manager or Card Manager to be discovered, without knowing anything about the card. (This also supports the bootstrap data mechanism, since by selecting the Card Resources Manager the request for bootstrap data can avoid being misinterpreted by the currently selected application.)

### How the mechanisms work

To access the Card Resources Manager before its AID is known, the following steps can be used:

- (a) Issue a SELECT FILE command with no command data and with the 'first or only occurrence' option set. This selects the card management function on the card.
- (b) If step (a) gives an error response other than '6Cxx', use the SELECT FILE command with the AID <to be defined>. Ignore any error response other than '6Cxx'.

If either of these commands succeeds, the response gives the AID of the card resource manager.

Whether or not the SELECT FILE commands succeeded, an attempt can be made to access bootstrap data as described below.

Step (b) awaits the allocation of an AID for this purpose, which is an outstanding issue. Until then, step (b) should be omitted.

The main mechanism for a card management system to access bootstrap data is through the use of the GET DATA command, as follows:

- (c) The device or system issues a GET DATA command for Card Data (tag '66' hexadecimal). If the command fails, then trial-and-error is necessary.
- (d) If GET DATA is successful the card returns Card Data, which contains the bootstrap data in a constructed data object with tag '73' (hex).
- (e) The bootstrap data contains an OBJECT IDENTIFIER for the universally recognised tag allocation authority, which unambiguously indicates that this is bootstrap data.
- (f) The bootstrap data then contains bootstrap data objects, with application tags assigned by the tag allocation authority.

The tag allocation authority must publish its object identifier and details of the tags it assigns and their purpose.

Bootstrap data can be made available through the ATR by including a GET DATA command to perform in the historical bytes “Initial Access Data” object to read Card Data. This would be coded in the historical bytes as ‘45 00 CA 00 66 00’. However, this may not be useful since the card management system may not have access to the reset function.

### Tag Allocation Example

The following application tags might be assigned initially by the ‘tag allocation authority’; the details are as discussed above:

APPLICATION 0: card type, for management purposes (e.g. “JCOP Version 5”).

APPLICATION 3: a qualifier to make a local card identifier unique.

All these tags would indicate ‘constructed’ data objects, since they contain further self-defining data objects which are themselves tagged. Note that tag values 1, 2 and 15 are reserved for ISO use, as alternative methods of identifying the tag allocation authority.

### Example of Coding

The following is an example of how the bootstrap data could be coded (in hexadecimal):

- 73 1E - template, containing the rest of the data.
- 06 06 aaaaaaaaaa - the OBJECT IDENTIFIER (OID, tag ‘06’) for the tag allocation authority, shown here as 6 bytes ‘aaaaaaaaaa’.
- 60 0a 06 08 bbbbbbbbbbbbbbb - ‘60’ is APPLICATION 0 tag which contains an 8-byte OID, ‘bbbbbbbbbbbbbb’ used to specify the card type (and possibly the major version).
- 63 08 06 06 dddddddddddd - ‘63’ is APPLICATION 3 tag which contains an OID used as the qualifier to make the ‘local’ Card ID unique, shown here as 6 bytes ‘ddddddddddd’. This object identifier need not be parsed by systems, but can be treated as a data string that is appended to the local identifier. Note that this OID may belong to the organisation that controls the scheme of ‘local’ card identifiers used on this card. For example, it might be ‘CardsRus’ for cards whose identifiers are assigned under a scheme operated by CardsRus.

#### 5.2.1.2 AID class not to be final

The AID plays an important role in the management and the identification of applet in the cards. Most applications are using the AID to store additional information in the PIX part. The type and the format of this additional information vary from application to application. Due to the fact that the actual AID class is declared final it does not allow derivation of a new class to implement the handling of this information in your own class. At the moment it is only possible to write a wrapper class around the existing class.

Proposal: In an object oriented system it would be better to declare the class non final but the methods as final, and have some more methods in this class for better handling of the AID content (see example code in javadoc files).

#### 5.2.1.3 Specify the default applet

In the JCRE specification it is not clearly specified how to set the default applet, it is up to the JCRE implementation if and how this feature is implemented. That means that the indication of the default applet in the card is not interoperable, in some cards the default applet is already loaded, in other cards it is the first applet that is loaded into the card. However for several projects where the card issuer must know exactly which applet is the default applet in the card we need a precise definition of the default applet in the card.

- In the contactless environment, where there is no time to submit an application selection command
- In an environment where the terminals can not select an applet. For example a banking environment assumes that there is one applet on the card.

One way to indicate that an applet loaded into the card shall be the default applet is a flag-interface like:

```
public interface DefaultApplet {};
```

Proposal: The implementation for specifying the default applet has to be defined in the next Java Card specification.

## 5.2.2 Generic classes

*This chapter presents generic CMS class.*

Examples of generic classes are the following:

- Generic Card: CardID and characteristics.
- Generic Card application: CardApplicationID and characteristics:
  - Types: GSM, Loyalty, ...
  - ...
- Subscriptions: information concerning services/products acquired by a CardHolder.
- ...

As they are generic (common to different CMS), they are not described in the present document.

### 5.2.3 Java class diagrams

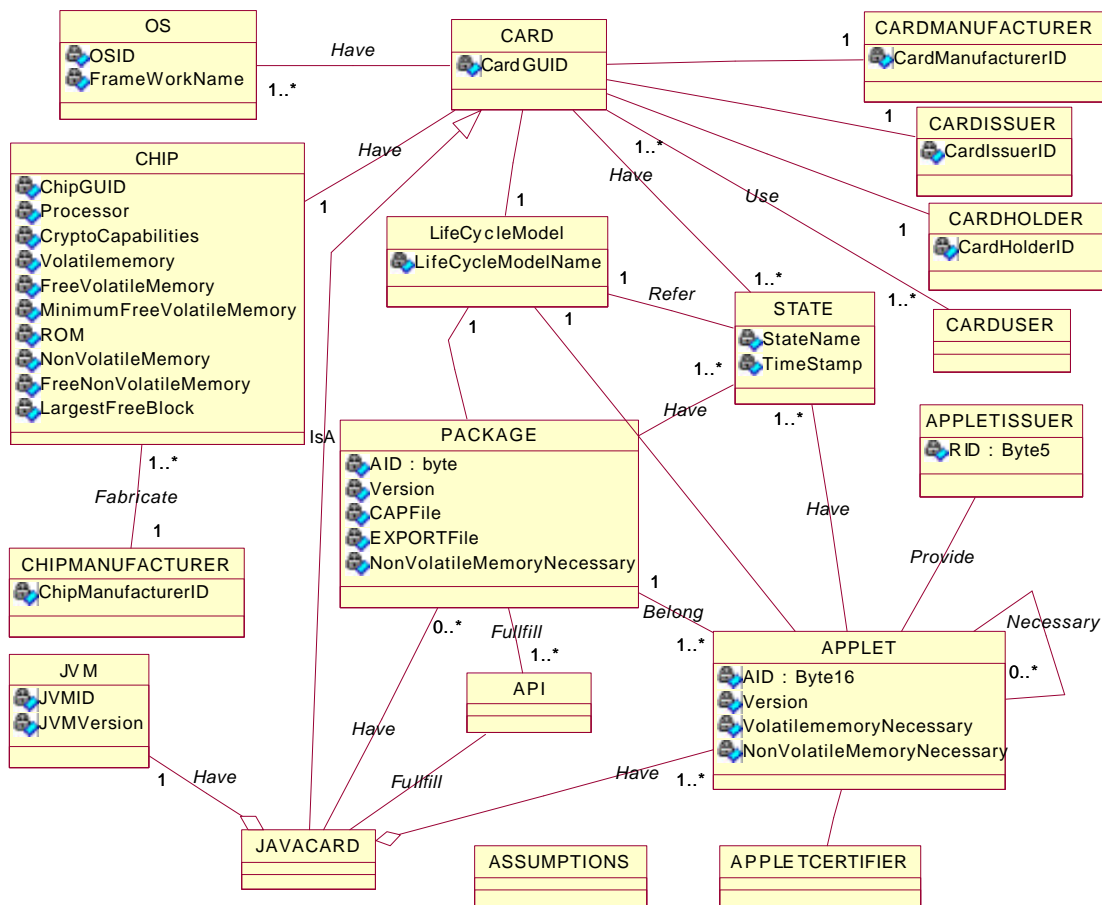
This chapter presents Java CMS class diagrams, and if necessary State/transition diagrams.

The class diagrams below have been set up using use cases definitions.

#### 5.2.3.1 CMS Class diagram

The following Class diagram represents the Java CMS class diagram of the core of a JCMS as it is defined for the present study.

It is mainly off-card. Part of this model could be implemented on-card as proposed in chapter 5.2.2.2.



Comments:

For detailed description, please refer to Annex 1 'Data dictionary'.

### 5.2.3.2 On-card information system

*This paragraph defines the management information system implemented on-card.*

The on-card information system is composed of all information concerning the card, to be able to manage it. That information may be really implemented in the card, or calculated by the card according to its internal structure.

#### Preliminary notes:

Not all of the following APIs are completely mandatory for implementation. Some vendors can implement only part of the APIs. But we recommend that every API call should be implemented, even if for some of those the API call just returns an exception. In this way all applets using the API could work at run-time.

#### Legend:

**Mandatory:** A backend system knowing the card is available (Card Issuer system for example), or the information is so dynamic that it cannot be trusted if only in a backend system.

**Recommended:** This card can be management by a CMS not aware of the card, or if the card passes from one CMS to another (thus the card is an information carrier).

**Optional:** Could be obtained through a backend system, but very useful if it's on-card.

**Useless:** Means that we didn't identify any use case using this information.

<b>Information name</b>	<b>On-Card implementation</b> <u>Mandatory</u> <u>Recommended</u> <u>Optional</u> <u>Useless</u>	<b>Comment</b>	<b>APIs types needed</b>
Javacardforum.management			
CardGUID	M	This is the minimum information to reside on-card, to be able to join with information inside a backend system. This identifier could be made of a real cardGUID, or a GUID made of a Qualifier and a local ID. Please see 'Preliminary recommendations and the Note below.	Get
FreeVolatileMemory	M	Free volatile memory on the card at the time it is requested to the card, for an applet to execute.	Get
FreeNonVolatileMemory	M	To check for space needed for specific applets (installation). As this information is dynamic, it cannot reasonably reside only in the backend system. This information is automatically maintained by the Card Resources Manager.	Get
Card: LifecycleModel StateName	M	To determine where the card is in its lifecycle, to know what transition is available (what can be done on this card).	Get Get-Set
AID	M	To be able to know which packages/applets are on the card, to reload a package/applet, to delete a package/applet, ...	Get

Information name	On-Card implementation <u>Mandatory</u> <u>Recommended</u> <u>Optional</u> <u>Useless</u>	Comment	APIs types needed
AID: LifeCycleModel StateName	M	To determine where is the applet/package (identified by the AID) in its lifecycle, to know what transition is available (what can be done on this applet/package: install it, process commands, ...).	Get-Set Get-Set
CID	M	A way to access to Card Identification Data (CID) after the ATR-time. Please see Note 1 below.	Get
Javacardforumx.management			
MinimumFreeVolatileMemory	R	Free volatile memory on the card in the worst situation (when all applets are selected, ...), for an applet to execute.	Get
LargestFreeBlock	R	Largest free block of non-volatile memory. To check for space needed for specific applets (installation). This information is automatically maintained by the Card Resources Manager.	Get
JVMID	R	Gives the build version of the VM in the card. To check for capabilities for specific applets (functionality, bugs, security holes, ...)	Get
JVMVersion	R	Returns the current major and minor version of the Java VM (Example: 2.1). To check for capabilities for specific applets.	Get
APIVersion	R	Return the current version of the Java Card API (Example 2.1.1). We already have an access to this information through the jc 2.1 getVersion() method, so it's easy to implement.	Get
CardIssuerID	R	With the CardIssuer identifier, the CMS is able to determine if the card is in its management scope, and also able to look for missing information about this card.	Get
CardHolderID	R	To be able to identify the CardHolder for that CardIssuer, for billing purposes for example. Set should be necessary for CardHolderID changes: wedding, security changes, ... But is not realistic now, because an applet could have used this CardHolderID, and the CMS has to change it in the applet itself.	Get
AppletIssuer (RID)	R	To be able to have information on the applet. No API is necessary because this information can be retrieved through AIDs. We suggest that there should be a new method in AID Class called getRID() – see javadoc files.	-

Information name	On-Card implementation <u>Mandatory</u> <u>Recommended</u> <u>Optional</u> <u>Useless</u>	Comment	APIs types needed
AIDVersion	R	To be able to determine which version of a specific applet/package is on-card.	Get
Processor	O	To check for performance and capabilities for specific applets. Performance depends on the applet implementation also, but processor is part of the performance.	Get
CryptoCapabilities	O	To check for performance and capabilities for specific applets.	Get
NonVolatileMemory	O	Just for information.	Get
VolatileMemory	U	To check for space needed for specific applets (execution), but the interesting information are more FreeVolatileMemory and MinimumFreeVolatileMemory.	-
CardManufacturerID	O	To be able to have information, e.g. for audit purposes.	Get
ChipManufacturerID	O	To be able to look for missing information (defects, ...) about this chip, and for audit purpose.	Get
ChipGUID	O	To check dependencies with specific Chips.	Get
OSID	O	To be able to deal with specificity of the OS: capabilities, known bugs, ...	Get
FrameworkName	O	To be able to determine the Application Selection Mechanism available on the card.	Get
Necessary applets	U	Under Applet Issuer responsibility only.	-
AppletCertifier	U	Under AppletIssuer responsibility.	-
NonVolatileMemoryNecessary (Package)	U	Under AppletIssuer responsibility.	-
CardUser	U	Application dependant.	-
ROM	U	No use.	-
NonVolatileMemoryNecessary (Applet)	U	Under AppletIssuer responsibility.	-
VolatileMemoryNecessary (Applet)	U	Under AppletIssuer responsibility.	-
<b>OTHERS</b>			
LifeCycleModel	U	To determine what lifecycle is referred by the StateName, but acquired through next APIs.	-

Information name	On-Card implementation <u>Mandatory</u> <u>Recommended</u> <u>Optional</u> <u>Useless</u>	Comment	APIs types needed
Assumptions	U	Definitely CMS-side.	-

Note : This note is intended to discuss the usage of the bootstrap data with a CardGUID. The bootstrap data may contain different information, including potentially a CardGUID. Different cases appears:

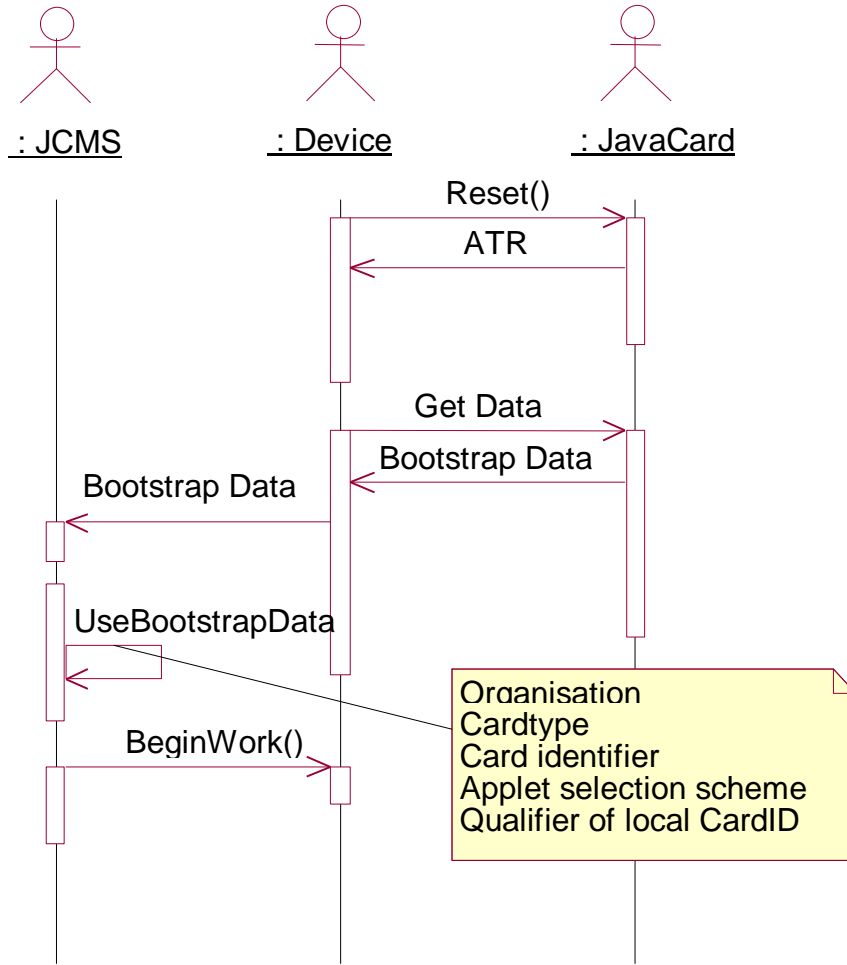
- The CardGUID is not required by the CardIssuer (He doesn't want to have a CardGUID): No problem
- A CardGUID is required by the Card issuer, but this CardGUID must not be revealed to everyone (It must be private). In such a case the CardGUID is not included in the bootstrap data, so it's not available at ATR-Time. The only way to have it is after security mechanisms using the GetCardGUID() CMS API.
- A CardGUID is required by the Card issuer, and this CardGUID can be revealed to everyone (It is public). In such a case the CardGUID is included in the bootstrap data, so it's available at ATR-Time. It can also be accessed through the use of the GetCID() CMS API

However there must be only one CardGUID.

### 5.3 Java Card Sequence diagrams

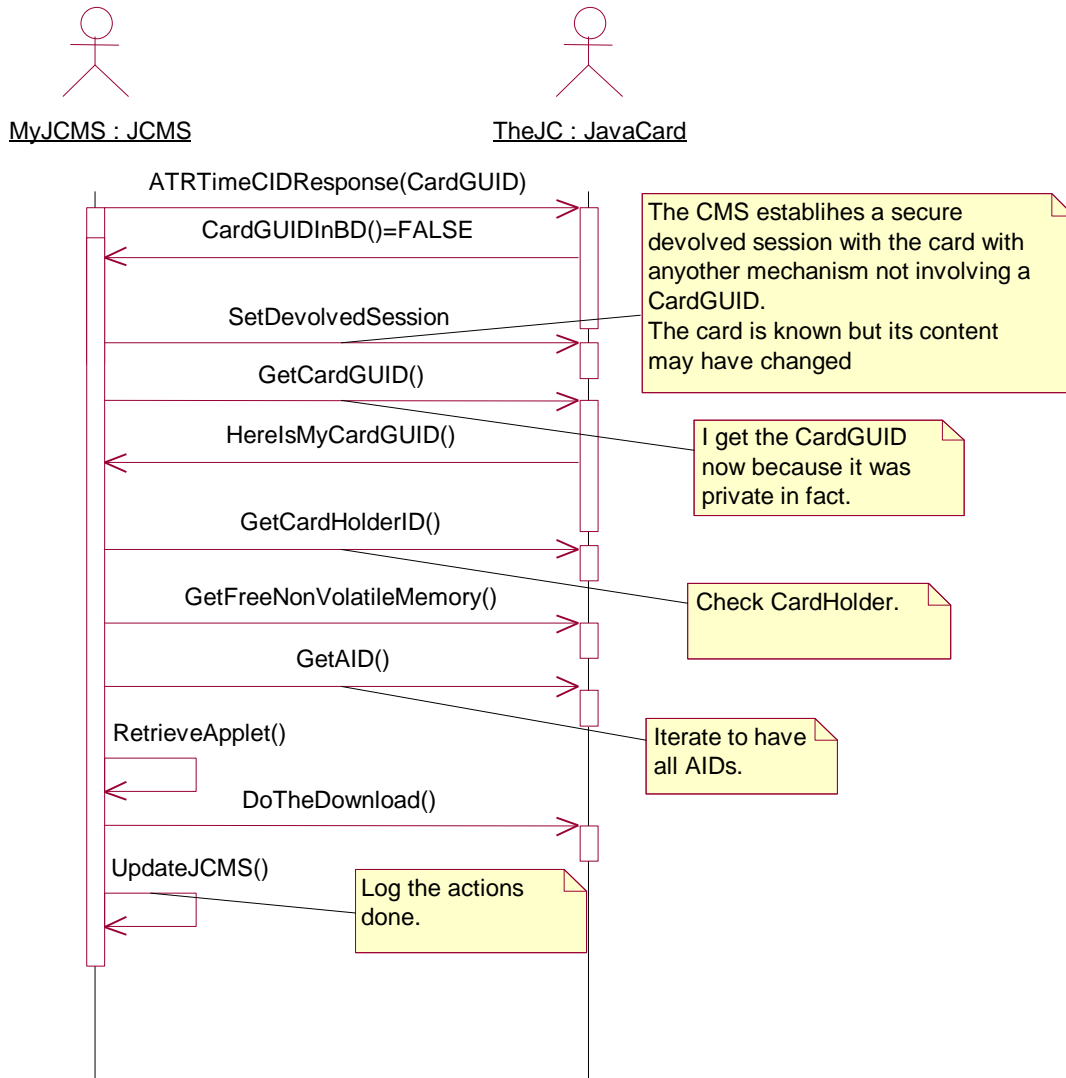
This chapter presents Java CMS sequence diagram using the proposed API.

#### 5.3.1.1 ATR-time bootstrap data retrieval mechanism



5.3.1.2 Successful post-issuance download

This sequence diagram describes the possible API usage for an applet download done by a devolved entity.



## 6 Java Card management API

*This chapter describes the Java Card Management API proposed by the Task Force.*

Please note that an HTML version of the API is in the following and attached file:

[First HTML Document \(doc\index.html\)](#)

## 7 Standing questions, and answers

*This chapter lists current topics/questions remaining to be studied by the task force members.*

### 7.1 Standing questions

- Standardisation of the JCMS itself defining JCMS APIs in order to have interoperable JCMS?

### 7.2 Answers

- Management standards (OP, VOP, GOP, ...) have a management API. What is our position towards this? Do we have to be critical about this spec (then taking part of it inside our API), do we have to take it as it is, ...? Other future standards may also be concerned.

Being a force of proposal, trying to be independent of card management standards: VOP and other similar standards.

We are defining a low-level on-card API only (used or not by those standards);

When defining our APIs we try to illustrate the API usage by those standards (and trying not to be in contradiction with those standards, or in contradiction but on purpose).

Being critical to those standards, adding APIs/functions if needed (recommendations).

- JINI: What is the impact on our work, and do we have to take it into account?

A Jini server cannot stand inside the card. When it will be able, then it will no longer be Java Card but standard java. What about a small Jini server in the card, just saying 'Hello I am here'. ....

- RMI: done by Technical Framework subgroup

- Key generation: lead taken by SUN Microsystems.

- As java card technology is more flexible in terms of products/services offered to the CardHolder not directly related to the card (insurance, loyalty combined with payment, ...), do we have to address in our task force the product/service management? Are there any existing/future service/products that need specific use cases and APIs?

Maybe we can pinpoint in the JCMS part that there is a need for the management of such global products: cross loyalty programs, ...

## 8 Next steps

*This chapter lists topics that have not been addressed yet, and that could be of some interest.*

Following the present work, we propose to address the following actions:

- Assumptions made have to be resolved: security mechanisms, ...
- Consider whether it would be better to join the previous document/content with the present one.
- Define the internal semantics of our API's parameters: semantics, internal representation (byte arrays now), ...
- Add APIs on specific or more complex actions: dynamic linking between an application and on-card APIs, download of application, ...
- Give notes for implementation.
- JCF positioning with Global Platform work.

## 9 Glossary – Definitions - References

*This chapter lists specific term and abbreviations, and occasionally defines these terms, or gives references for further information.*

ACL	Access Control List	
AID	Application Identifier as defined in ISO 7816-5. An application identifier is a byte string that is between 1 and 16 bytes long, identifying applets and packages. The identifiers are administered by ISO. The first 5 bytes represent the Registered application provider Identifier (RID), managed by ISO. The last 11 bytes represent the Proprietary application Identifier eXtension (PIX), managed by companies.	
API	Application Programming Interface	
BD	Bootstrap Data	§5.2.1
BER	Basic Encoding Rules	See ISO/IEC 8825-1.
CID	Card Identification Data: here referred to as 'bootstrap data'.	§5.2.1
CMS	Generic Card Management System	
CPLC	Card Production Life Cycle (OP)	
EMV	Europay-MasterCard-Visa - a joint industry working group created to facilitate the introduction of chip technology into the international payment systems environment	<a href="http://www.emvco.com">www.emvco.com</a>
Global Platform		<a href="http://www.GlobalPlatform.org">www.GlobalPlatform.org</a>
GUID	Globally Unique Identifier	
ICCID	Integrated Circuit Card Identifier	
ISO/IEC 8825-1	Information Technology - ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).	See also ITU-T X.690.
ISO/IEC 9834-1	Information Technology - Open Systems Interconnection Procedures for the operation of OSI Registration Authorities: General Procedures.	See also ITU-T X.660.
JCF	Java Card Forum	<a href="http://www.javacardforum.org">www.javacardforum.org</a>
JCMS	Java Card Management System, as opposed to CMS	
JINI	Sun Microsystems initiative called "spontaneous networking." Using the Jini architecture, users will be able to plug Jini-compliant-devices, speakers, directly into a network and every other computer, device, and user on the network will know that the new device has been added and is available.	<a href="http://www.sun.com/jini/">www.sun.com/jini/</a>
MCD	MULTOS Carrier Device MULTOS smart card.	<a href="http://www.multos.com">www.multos.com</a>
OCF	Open Card Framework. Specification of a terminal and card framework from IBM, for abstracting terminals and cards from the developer's point of view. It is mainly implemented through an OCF core (framework), CardTerminal (abstraction of terminal) and CardService (abstraction of card).	<a href="http://www.opencard.org">www.opencard.org</a>
OP	Open Platform	<a href="http://www.visa.com/openplatform">www.visa.com/openplatform</a>
PC/SC	Personnal Computer / Smart Card	<a href="http://www.pcscworkgroup.com">www.pcscworkgroup.com</a>
PIX	Proprietary application Identifier eXtension.	See also AID.
PKI	Public Key Infrastructure	
RID	Registered application provider Identifier as defined in ISO 7816-5.	See also AID.
RMI	Remote Method Invocation	
UML	Unified Modelling Language	<a href="http://www.rational.com">www.rational.com</a> <a href="http://www.omg.org">www.omg.org</a>

## 10 Annex 1: Data dictionary

*This chapter defines elementary information used in the UML diagrams.*

### **API**

API specification. For example, Java Card API, but other possible (OP, GSM, ...).

### **APPLET**

Java Card application.

#### ATTRIBUTES

- AID: Applet IDentifier
- Version
- VolatileMemoryNecessary
- NonVolatileMemoryNecessary

### **APPLETCERTIFIER**

Entity certifying an applet.

### **APPLETISSUER**

The Applet Issuer is the entity responsible for the applet code delivery (result of the development).

#### ATTRIBUTES

- RID: Registered application provider IDentifier (RID) managed by ISO.

### **ASSUMPTIONS**

List of assumptions for the current CMS: terminals supported, drivers, libraries, Environment (OCF, PC/SC, ...), loading protocols, ...

### **CARD**

Standard smart card.

#### ATTRIBUTES

- CardGUID: Identifier of the card.

### **CARDHOLDER**

The Cardholder is the final owner of the card, as defined by the CardIssuer. He will use the card, maybe with other CardUsers.

#### ATTRIBUTES

- CardHolderID: A name, a number, or a certificate containing different pertinent information: names, password, ... - identifying the person to which the card has been issued (uniquely set by the CardIssuer).

### **CARDISSUER**

The Card Issuer is the entity responsible for (owner of) the card when it is deployed.

#### ATTRIBUTES

- CardIssuerID

### **CARDMANUFACTURER**

The Card Manufacturer (Embedder/Embosser) is the company that produces/personalises the cards.

#### ATTRIBUTES

- CardManufacturerID

### **CARDUSER**

The CardUser is the person using the card at the time it is presented to the CMS. It may be the CardHolder or another person.

## **CHIP**

Integrated circuit able to run commands, store information, ...

### ATTRIBUTES

- ChipGUID: Global Unique Identifier of the chip.
- Processor: Type of processor.
- CryptoCapabilities: Cryptographic capabilities implemented in the card, as a cryptoprocessor, or in the main processor of the card.
- VolatileMemory: Size of the RAM.
- FreeVolatileMemory: Free RAM on the card at the time it is requested to the card, for an applet to execute.
- MinimumFreeVolatileMemory: Free RAM on the card in the worst situation (when all applets are selected, ...), for an applet to execute.
- ROM
- NonVolatileMemory: Overall size of the EEPROM.
- FreeNonVolatileMemory: EEPROM left that can be used.
- LargestFreeBlock: Largest free block of the non volatile memory.

## **CHIPMANUFACTURER**

The Chip Manufacturer is the semiconductor company which fabricates the processor and memory chip on which the multi-application smart card is based.

### ATTRIBUTES

- ChipManufacturerID

## **JAVACARD**

Smart card 'Java Card'

## **JVM**

Java Virtual Machine running on a card.

### ATTRIBUTES

- JVMID: Identifier of this JVM.
- JVMVersion: Java card reference: 2.1, ...

## **LifeCycleModel**

### ATTRIBUTES

- LifeCycleModelName

## **OS**

Operating System in the card, upon which the JVM runs.

### ATTRIBUTES

- OSID: Identifier of the OS.
- FrameworkName: Name of the framework for applet selection: GSM, EMV, ...

## **PACKAGE**

Java Card package.

### ATTRIBUTES

- AID: Applet Identifier
- Version: Package version.
- CAPFile
- EXPORTFile
- NonVolatileMemoryNecessary: Non volatile memory necessary for the package to run.

## **STATE**

State of a functional object managed by the CMS, as viewed by the card resource manager (not application dependant): Card, Applet, ...

This model can be used for OP.

### ATTRIBUTES

- StateName: For OP for example, it could be: SECURED, INITIALISED, ....
- Proprietary states: MANUFACTURED, PERSONALISED, ...
- For applets: LOADED, INSTALLED, READY, ...
- TimeStamp: Date and time of the state change.