

## JAVA CARD APPLICATION MEMORY FOOTPRINT OPTIMIZATION

### TECHNICAL FIELD

The present invention relates to a method for Java Card application memory footprint optimization.

It finds applications, in particular, in smart card products for which there is a trend of storing ever growing numbers of applications (applets) despite limited memory resources.

### RELATED ART

The approaches described in this section could be pursued, but are not necessarily approaches that have been previously conceived or pursued. Therefore, unless otherwise indicated herein, the approaches described in this section are not prior art to the claims in this application and are not admitted to be prior art by inclusion in this section.

Java Card technology was introduced in 1996 and is now widely used in the smart card domain (notably for SIM cards or ATM cards). It allows java-based applications (applets) to run on smart cards and other similar devices which have limited memory resources. These applications usually do not have large memory footprint when taken individually. However, since the market tends to demand storing more and more of these applications on single java cards, it urges researchers to seek after solutions for reducing the memory space consumed by each application.

The different approaches considered so far all involve modifying the application code itself to reduce its memory footprint. The optimization of the code can be performed by hand, by refactoring byte code or by using a proprietary byte code in order to reduce the memory footprint. However, such optimization is set to be limited in terms of efficiency, and the memory footprint reduction currently achieved this way is non substantial. This calls for new ways of implementing a reduction of the memory space required for storing applications.

An alternative approach can be based, according to embodiments of the present invention, on the observation that once an application has been configured, the binary code implementing the personalization of this application becomes useless and yet is still using some memory space on the card. The idea behind the invention is then to eventually remove at least part of the binary code associated with the personalization of a Java Card applet to free memory space.

## SUMMARY

To achieve this goal, the present invention proposes a method for the optimization of the memory footprint of a Java Card application which allows to reduce an applet memory footprint by removing some or all of the code associated to its personalization once said personalization is complete.

More specifically, according to a first aspect there is proposed a method of reducing the memory footprint of a Java Card Application comprising loading (31a,31b) two distinct packages into the memory of a Java Card, namely a first, main package (11) containing a main application and dedicated to the installation and transaction phase of said main application in the Java Card, and a second package, capable of sharing resources with the main package and intended to be used during execution of the personalization phase of the main application and to be deleted after said execution of said main application.

The above method ultimately allows removing the binary code associated with the personalization of a Java Card applet once all the steps associated to its personalization have been completed. Consequently, it allows reducing the memory footprint associated to this application on the card.

The proposed method thus relies on the separation in advance of the code related to the personalization from the rest of the code. It allows this code to perform the personalization of an application installed from a main package while being itself included and installed from a separated package dedicated to the personalization, namely the Card Personalization Specifications (CPS) package, for instance. This way, the CPS package and all the code inherent to the personalization (referred to by "the personalization code" in what follows) can be removed once all the personalization steps have been completed.

Stated otherwise, the method makes it possible to overcome limitations set by runtime rules of Java Card which are implemented by a firewall to protect applets by preventing an applet instance of accessing objects created in a different applet package. To do so, embodiments of the method of the present invention can comprise modifying the firewall context of an object created by an instance, namely the CPS applet, contained in the CPS package. By assigning to the object created from the CPS applet (called the CPS object in the following) the same firewall context as the main applet, embodiments of the method can allow the personalization code contained in the CPS object to access the main applet and to execute the code required by the personalization of the application without being blocked by the firewall.

Additional to this, embodiments of the method can further comprise making the objects created during the personalization process belong to the main applet. These objects would otherwise belong to the CPS applet since they have been created from it. In such configuration they would then be removed along with said  
5 CPS package if this package were to be removed. By introducing some modifications of the ownership of the CPS object created from the CPS applet of the CPS package, embodiments of the method overcome this limitation. The ownership of the CPS object can be changed back and forth from CPS applet to main applet and reciprocally so that once the personalization is complete, the  
10 CPS object belongs to the CPS applet instance and can be removed along with said CPS package whereas all objects personalized within the main applet still belong to the main applet.

In embodiments of the method, taken either alone or in combination:

- the main package and the second package can use Java Card Shareable  
15 Interface to be capable of sharing resources with each other;
- the main package can be associated to a transaction phase and the second package is associated to a personalization phase, respectively;
- the second package is a Card Personalization Specification, CPS,  
package associated to the personalization phase, said CPS package containing a  
20 CPS applet dedicated to the creation of a CPS object into the memory of the Java Card;
- the CPS package can have a firewall context and an ownership, and be configured to execute a personalization code used for the personalization of the main application;
- 25 - the method can further comprise changing the firewall context of the CPS object by assigning thereto a firewall context of the main applet and changing the ownership of the CPS object from the CPS applet instance ownership to a main applet instance ownership;
- the method can further comprise handling the personalization of the main  
30 application by having the main application access the CPS object and execute the personalization code which initializes main application data.
- the method can further comprise:
  - changing back the ownership of the CPS object from the main applet ownership to the CPS applet ownership; and,
  - 35 - deleting the CPS package;

- the firewall context of the CPS object is changed by a first Application Programming Interface, API;
- the ownership of the CPS object is changed and/or changed back by the first API, or by a second API distinct from said first API;
- 5       - the first API and/or the second API, is used for changing the ownership and/or the firewall context of the CPS object, respectively, are provided by the Java Card platform or are created specifically;
- the first API and/or the second API used for changing the ownership and the firewall context of the CPS object can be deleted from the Java Card memory
- 10       after the change back of the CPS object ownership from the main applet to the CPS applet has been performed;
- in a variant, the first API and/or the second API used for changing the ownership and the firewall context of the CPS object can be rendered unusable after having performed the change back of the CPS object ownership from the
- 15       main applet to the CPS applet;
- the firewall context of the CPS object is changed by implementing a specific parameter in the standard install command used to install the CPS applet and defined by the Global Platform Card specification;
- the main application that is installed from the main package is a Europay
- 20       Mastercard Visa (EMV) application;
- the main package is loaded into the Read-Only Memory (ROM) of the Java Card;
- in a variant, the CPS package is loaded into the Non Volatile Memory (NVM) of the Java Card; and, finally,
- 25       - the method can further comprise dividing the Java Card application into the main package and the second package.

30       A second aspect of the invention relates to a computer program product comprising one or more stored sequences of instructions that are accessible to a processor and which, when executed by the processor, cause the processor to perform the steps of a method according to the first aspect.

      Finally, a third aspect of the invention proposes a device for reducing the memory footprint of a Java Card Application, comprising means for carrying out all the steps of the method according to the first aspect or any one of the above embodiments of said method.

### BRIEF DESCRIPTION OF THE DRAWINGS

Embodiments of the present invention are illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings, in which like reference numerals refer to similar elements and in which:

- 5       - Figure 1 is a schematic diagram showing a basic configuration for the installation of a Java Card applet structure according to the prior art;
- Figure 2 is a schematic diagram illustrating the technical problem solved by the invention; and,
- Figure 3 is a composite block diagram and flow chart illustrating steps
- 10     carried out for the implementation of embodiments of the proposed method for Java Card application memory footprint optimization.

### DESCRIPTION OF PREFERRED EMBODIMENTS

Referring first to Figure 1, there is shown therein a schematic description of

15     a basic configuration for the installation of a Java Card applet structure according to the prior art.

A single package, which shall be referred to as the main package 11 in what follows, contains all the elements required for the installation and the functionalities of an application, hereinafter referred to as the main applet 12. This

20     main applet 12 can be, for instance, a Europay Mastercard Visa (EMV) application. In the context of the invention, such an application is intended to be created/installed on the Java Card. To that end, the main package 11 is loaded onto the Java Card and is executed therein.

Some code 13 contained in the main applet 12 is dedicated to the

25     personalization of said main applet 12. This personalization code 13 is then also contained in the main package. Said code 13 is adapted to create the main applet data objects (such as instances of classes or array types) and it is further adapted to initialize the values of each field of each data object during the personalization process of the main applet. This is referred to as the "personalization phase" in

30     what follows.

Referring now to Figure 2, there is shown therein a schematic description of a configuration according to embodiments, wherein the package for the installation of the Java Card applet is split into two distinct applet packages: a main package 11 and a Card Personalization Specification (CPS) package 21.

35     Embodiments can, for instance, rely on the implementation of a multi-package design with different packages able to share and link their resources. A

way to achieve this functional association between the main package 11 containing the main applet 12 and the CPS package 21 containing a CPS applet 25 so as to allow the above resource sharing, can be to use Java Card Shareable Interface. This way, Shareable Interface Objects (SIO) of an applet instance in a package can be accessed from another applet instance in another package. In particular, a CPS object instance 32 created by the CPS applet 25 and containing the personalization code 13 can be requested by the main applet instance 12 once said main applet instance 12 has been created.

The arrow 22 which is to be seen in Figure 2 illustrates that last fact. The main applet instance 12 created from the main package calls for the personalization code 13 included in the CPS object 32 in the CPS package. The arrow 23 and the cross 24 illustrate that the personalization code cannot access the main applet instance 12 due to a Java Card firewall. Indeed, the Java Card firewall prohibits objects created in one applet package to access objects created in another applet package. Therefore, it prohibits any object created in the CPS package to access any object of the main package. It will be appreciated by the one with ordinary skills in the art that, in that situation, the personalization code cannot be executed and the personalization phase cannot be achieved properly.

According to actual Java Card rules, by default, the object instances created from one applet instance belong to this applet instance, meaning that the ownership of these objects is the one of the applet instance. As a consequence, the objects personalized by executing the personalization code included in the CPS object created from the CPS applet, have the same ownership as the one of the CPS applet. Since the CPS applet is in the CPS package, it follows that, if the CPS package would be removed, all the personalized objects that were created from the CPS object instance would be removed altogether. This effect is cumulated with the fact that, in that case, the main applet is not able to access objects owned by the CPS applet due to the firewall.

The method proposes a solution to overcome these limitations by changing not only the firewall context, but also the ownership of the CPS object created by the CPS applet contained in the CPS package.

Embodiments of the proposed method shall now be described with reference to Figure 3.

Referring to Figure 3, there is shown therein a composite block diagram and flow chart illustrating different tasks performed according to embodiments of the proposed method for Java Card application memory footprint optimization.

The method basically relies on a pre-separation of the Java Card applet into a main package and a CPS package.

More specifically, the method comprises the following.

At 31a, there is loaded into the memory of a Java Card a main package 11  
5 containing a main applet 12, which is intended to be installed on the card and personalized. In parallel, at 31b, a Card Personalization Specification (CPS) package 21, containing the CPS applet 25 within which the personalization code 13 required by the personalization of said main application is included is further loaded into the memory of the same Java Card. In Figure 3, these steps of  
10 loading are illustrated by arrows 31a and 31b, respectively. In one example, the main package 11 can be loaded into the Read-Only Memory (ROM) of the Java Card and the CPS package can be loaded into the Non Volatile Memory (NVM) of said Java Card. This example is advantageous since, as will be described in further detailed below, the code of the CPS package can eventually be deleted,  
15 but it is not intended to limit the scope of the invention. In other examples, both packages can be loaded in respective memory space of the same physical memory.

The creation of the CPS object 32 by the CPS applet 25 in the CPS package 21 loaded on the memory of the card is illustrated by the arrow 33 in  
20 Figure 3. It shall be noted that, the CPS object 32 has the ownership of the CPS applet 25 and the firewall context of the CPS package 21. This is so because the CPS object 32 has been created from the CPS applet instance 25 which, itself, was created from the CPS package 21. Once it has been created, the main applet 12 calls the CPS object to handle its personalization. This is possible since, as  
25 previously mentioned, the main package 11 and the CPS package 21 are associated (for instance through the appropriate use of Java Card Shareable Interface), and can thus share resources.

However, the personalization code 13 within the CPS object 32 cannot access the main applet 12. Indeed, as previously described with reference to  
30 Figure 2, should the CPS object 32 try to access the main applet 12 as illustrated by the arrow 37, then it would be blocked by the firewall 41 as illustrated by the cross 24.

In order to circumvent this situation, the method then continues with the changing of the firewall context and of the ownership of the CPS object 32 as  
35 illustrated by the arrow 34. The ownership change can be for instance performed in the CPS applet before the CPS object is obtained by the main applet or in the

main applet once the CPS object is obtained by said main applet. In both cases this change occurs before the execution of the personalization code hereafter mentioned.

5 In Figure 3, the CPS object with changed firewall context and ownership is represented by block 32'. This change can be achieved, for instance, by executing one or more Application Programming Interface (API) that is/are dedicated to changing the firewall context and/or the ownership of an object within the Java Card. For instance, the firewall context can be changed by a first API, and the ownership can be changed by this first API, or by a second API distinct from said first API. In one example, this(these) API(s) can be provided, for instance, by the Java Card platform or can be created specifically.

10 The process then goes on with executing the personalization code, from the CPS object 32', to personalize the data objects of the main application 12 as illustrated by the arrow 35. It shall be noted that the CPS object 32' can access the main applet 12 for that purpose, because the firewall context change illustrated by the arrow 34 has occurred. In Figure 3, main application 12 with personalized data objects is represented by block 12'.

15 After the personalization of the main applet is complete the ownership of the CPS object 32' is changed back to the CPS applet instance as illustrated by the arrow 39. In Figure 3, the CPS object with ownership changed back to the CPS applet instance is represented by block 32''.

20 Eventually, once the ownership of the CPS object has been changed back to the CPS applet, the CPS package 21 can be deleted as illustrated by the arrow 40. As a result, the personalization code is also deleted, since it is a part of CPS package code.

25 To summarize, embodiments of the method as described above allow to personalize a Java Card application while eventually removing all the code elements related to its personalization. Consequently, this advantageously provides memory footprint reduction for the main applet.

30 The method thus allows the personalization of an application created/installed from a package to be performed from the object created/installed from a different package. Further, the later package can be removed after the completion of the personalization process without removing any of the elements required by the main applet for its functioning.

35 The method is expected to allow reducing application memory footprint significantly and presents several additional advantages:

- It is easy to implement in already existing applications;
- In terms of card security it is no longer possible to execute "parasitic" personalization code to modify the application because the personalization code no longer exists;

5       • The reduction of the memory space required for an application offers the possibility to use java cards with less memory hence cheaper products;

- Reciprocally, it is possible to install more applications into the memory of a single java card thus increasing the productivity of a single card.

The present invention can also be embedded in a computer program  
10 product, which comprises all the features enabling the implementation of the methods described herein, and which - when loaded in an information processing system - is able to carry out these methods. Computer program means or computer program in the present context mean any expression, in any language, code or notation, of a set of instructions intended to cause a system having an  
15 information processing capability to perform a particular function either directly or after conversion to another language. Such a computer program can be stored on a computer or machine readable medium allowing data, instructions, messages or message packets, and other machine readable information to be read from the medium. The computer or machine readable medium may include non-volatile  
20 memory, such as ROM, Flash memory, Disk drive memory, CD-ROM, and other permanent storage. Additionally, a computer or machine readable medium may include, for example, volatile storage such as RAM, buffers, cache memory, and network circuits. Furthermore, the computer or machine readable medium may comprise computer or machine readable information in a transitory state medium  
25 such as a network link and/or a network interface, including a wired network or a wireless network, that allow a device to read such computer or machine readable information.

Expressions such as "comprise", "include", "incorporate", "contain", "is" and  
30 "have" are to be construed in a non-exclusive manner when interpreting the description and its associated claims, namely construed to allow for other items or components which are not explicitly defined also to be present. Reference to the singular is also to be construed in be a reference to the plural and vice versa.

While there has been illustrated and described what are presently  
35 considered to be the preferred embodiments of the present invention, it will be understood by those skilled in the art that various other modifications may be made, and equivalents may be substituted, without departing from the true scope

of the present invention. Additionally, many modifications may be made to adapt a particular situation to the teachings of the present invention without departing from the central inventive concept described herein. Furthermore, an embodiment of the present invention may not include all of the features described above.

- 5 Therefore, it is intended that the present invention not be limited to the particular embodiments disclosed, but that the invention include all embodiments falling within the scope of the appended claims.

- 10 A person skilled in the art will appreciate that various parameters disclosed in the description may be modified and that various embodiments disclosed and/or claimed may be combined without departing from the scope of the invention.

It is stipulated that the reference signs in the claims do not limit the scope of the claims, but are merely inserted to enhance the legibility of the claims.

**CLAIMS**

1. A method of reducing the memory footprint of a Java Card application, the method comprising loading (31a, 31b) two distinct packages into the memory of a Java Card, namely a first, main package (11) containing a main application and dedicated to the installation and transaction phase of said main application in the Java Card, and a second package, capable of sharing resources with the main package and intended to be used during execution of the personalization phase of the main application and to be deleted after said execution of said main application.
2. The method of Claim 1, wherein the main package and the second package use Java Card Shareable Interface to be capable of sharing resources with each other.
3. The method of any one of Claims 1 and 2, wherein the main package is associated to a transaction phase and the second package is associated to a personalization phase, respectively.
4. The method of Claim 3, wherein the second package is a Card Personalization Specification, CPS, package (21 ) associated to the personalization phase, said CPS package (21 ) containing a CPS applet (25) dedicated to the creation of a CPS object (32) into the memory of the Java Card.
5. The method of Claim 4, wherein the CPS package has a firewall context and an ownership, and is configured to execute a personalization code used for the personalization of the main application.
6. The method of any one of Claims 4 and 5, further comprising changing (34) the firewall context of the CPS object by assigning thereto a firewall context of the main applet and changing the ownership of the CPS object from the CPS applet instance ownership to a main applet instance ownership.
7. The method of any one of Claims 4 to 6, further comprising handling (35) the personalization of the main application by having the main application access the CPS object and execute the personalization code which initializes main application data.

8. The method of any one of Claims 6 and 7, further comprising:
- changing back (39) the ownership of the CPS object from the main applet ownership to the CPS applet ownership; and,
  - deleting (40) the CPS package.

5           9. The method of any one of Claims 6 to 8, wherein the firewall context of the CPS object is changed by a first Application Programming Interface, API.

10           10. The method of any one of Claims 6 to 9, wherein the ownership of the CPS object is changed and/or changed back by the first API, or by a second API distinct from said first API.

10           11. The method of any one of Claims 9 and 10, wherein the first API and/or the second API, used for changing the ownership and/or the firewall context of the CPS object, respectively, are provided by the Java Card platform or are created specifically.

15           12. The method of any one of Claims 10 and 11, wherein the first API and/or the second API used for changing the ownership and the firewall context of the CPS object is/are deleted from the Java Card memory after the change back of the CPS object ownership from the main applet to the CPS applet has been performed.

20           13. The method of any one of Claims 9 to 11, wherein the first API and/or the second API used for changing the ownership and the firewall context of the CPS object is/are rendered unusable after having performed the change back of the CPS object ownership from the main applet to the CPS applet.

25           14. The method of any one of Claims 6 to 13, wherein the firewall context of the CPS object is changed by implementing a specific parameter in the standard install command used to install the CPS applet and defined by the Global Platform Card specification.

15           15. The method of any one of Claims 1 to 14, wherein the main application installed from the main package is a Europay Mastercard Visa (EMV) application.

16. The method of any one of Claims 1 to 15, wherein the main package is loaded into the Read-Only Memory (ROM) of the Java Card.

17. The method of any one of Claims 4 to 16, wherein the CPS package is loaded into the Non Volatile Memory (NVM) of the Java Card.

5           18. The method of any one of Claims 1 to 17, further comprising dividing the Java Card application into the main package and the second package.

10           19. A computer program product comprising one or more stored sequences of instructions that are accessible to a processor and which, when executed by the processor, cause the processor to perform the steps of any of Claims 1 to 18.

20. A device for reducing the memory footprint of a Java Card Application, comprising means for carried out all the steps of the method of any one of Claims 1 to 18.