Outil d'installation d'applications sur une grille de cartes à puce de type Java à API respectant GlobalPlatform - Partie 1

Manuel utilisation

Responsables scientifiques :
Serge CHAUMETTE
Achraf KARRAY
Responsable pédagogique :
Mohamed MOSBAH

ABDELKHALEK Rached BEN MBARKA Moez
DIYAB Rachid ESSABIR Ayoub
GATTI Christophe Renteria Morales Eder
TRIKI Hamza

27 avril 2006

Table des matières

Chapitre 1

Livrables

L'archive fournie aux clients contient le éléments suivants :
- Codes sources
- Scripts de compilation pour UNIX et Windows.
- Scripts permettant l'utilisation du gpShell pour UNIX et Windows.
- Documentation javadoc des packages
- Librairies nécessaires pour l'exécution.

1.1 Arborescence

Les éléments précédents sont fournis avec l'arborescence suivante :
- src : Dossier contenant tous les fichiers sources .java sous forme de packages.

- lib : Dossier contenant les bibliothèques externes nécessaires pour l'utilisation de l'API et l'exécution des classes du gpShell.
  Il contient les bibliothèques suivantes :
  - junit.jar : utilis pour l'exécution de la classe de tests ApplicationOnCard-Test du package globalPlatform.
  - mysql-connector-java-3.0.17-ga-bin.jar : Nécessaire pour la classe Database du package offCard permettant l' accès à la base de données.

  - jpcsc.dll : Bibiliothèque Windows utilisé par l'API jpcsc permettant la communication avec le lecteur.

  C'est aussi dans ce dossier que sera créée l'archive .jar de notre API.

- obj : C'est le dossier destination de la compilation. Il contient tous les fichier .class

- docs : Dossier contenant la javadoc généré par javadoc, ainsi que les documents suivants : Rapport projet, Rapport maintenance et manuel utilisation.

- sql : Dossier contenant le fichier Database.sql permettant de créer la table 'cartes' dans la base de données.

- licences : Les licences de l'utilisation de l'API sont dans ce dossier.
- compile : Script de compilation pour UNIX.
- compile.bat : Script de compilation pour Windows.
- run : Script pour UNIX facilitant l'exécution des classes du package gpShell.

– run.bat : Script pour Windows facilitant également l'utilisation du gpShell.

1.2 Dépendances

1.2.1 Kit de développement java

Un compilateur java est nécessaire pour la compilation des sources. Celui qui est utilisé par défaut dans les scripts de compilation est javac.
La version qui a été utilisée pour faire les tests est : 1.5.0_06.

1.2.2 JPCSC

Pour communiquer avec les lecteurs, nous utilisons l'API jpcsc. Pour cette raison, le package com.linuxnet.jpcs est incorporé dans le code source de notre API.

Cependant, jpcsc n'est qu'un Wrapper écrit en JAVA utilisant JNI(Java Native Interface) pour renvoyer les appels de méthodes et les arguments Java vers la bibliothèque client fournie avec Microsoft PC/SC ou pcsc-lite(dans notre cas pcsc-lite). Pour pouvoir utilisé l'API avec jpcsc, il faut installer sa bibliothèque.

Pour Windows, il suffit d'avoir la bibliothèque jpcsc.dll accessible par la variable java.library.path. Ce fichier est fournit dans notre archive et le script d'exécution run.bat pour Windows l'inclut avec l'option -Djava.library.path

L'installation de jpcsc sur UNIX est plus compliquée et sort du cadre de ce projet.

1.2.3 Base de données

L'API nécessite l'accès à une base de données MySQL pour récupérer des informations sur une carte. Donc, un serveur MySQL doit être installé et accessible.

L'adresse par défaut du serveur MySQL : 127.0.0.1
Le nom de la base : grille.

Le fichier Database.sql permet de créer la table cartes utilisée par l'API.

D'autre part, la classe Database.java nécessite la bibiliothèque JAVA mysql-connector-java-3.0.17-ga-bin.jar. Celle-ci est fournie dans le dossier lib.

1.2.4 L'outil Junit

Les tests unitaires utilisent l'outil junit. Celui-ci est gratuit et écrit en Java. L'exécution de la classe des tests ApplicationOnCardTest nécessite la bibiliothèque junit.jar fournie aussi dans le dossier lib.

1.3 Licences

Les licences pour l'utilisation de l'API sont dans le répertoire license.

Chapitre 2

Compilation

Deux scripts de compilation : compile et compile.bat sont fournit respectivement pour les systèmes UNIX et Windows.

Ces deux scripts offrent les mêmes services. Ils peuvent être utilisés avec ou sans paramètres.

Usage :

1. ```
   compile
   ```

   Sans paramètre le script compile simplement tous les packages. La destination des fichiers .class est obj.
2. ```
   compile clean
   ```

   Permet de supprimer les fichiers .class, la bibliothèque globalPlatform.jar ainsi que la documentation javadoc.
3. ```
   compile jar
   ```

   Crée la bibliothèque JAVA globalPlatform.jar dans le dossier lib.
4. ```
   compile doc
   ```

   Génère la documentation avec javadoc dans le dossier docs.
5. ```
   compile ?
   compile help
   ```

   Affiche une aide textuelle pour l'utilisation du script.
   ```

Chapitre 3

Utilisation de l'API

3.1 Package globalPlatform

Dans cette section nous allons décrire pour les différentes opérations les méthodes appelées.

3.1.1 Sélection

pour construire l'APDU qui sert à sélectionner une application sur la carte il faur faire appel à la méthode *selectApplication* voici une description de cette méthode.

```
public static byte[] selectApplication(byte[] aid, byte p2)
```

- Paramètres : byte[] aid : tableau de byte pour l'AID de l'application à selectionner sur la carte.

  byte p2 : C'est un byte qui ne peut prendre que deux valeurs qui ont la signification suivante :
  - '0' : Sélectionner la première occurrence.
  - '1' : Sélectionner l'occurrence suivante.
- Valeur retournée : La méthode retourne un tableau de byte qui contient l'APDU correspondante à la sélection.
- Les Exceptions :
  - SW1='68', SW2='82' : message sécurisé non soutenu.
  - SW1='6A', SW2='81' : fonction non soutenue ; ex si le Life Cycle State est CARD_LOCKED.
  - SW1='6A', SW2='82' : application déjà sélectionnée ou fichier introuvable.

3.1.2 Suppression

Pour construire l'APDU qui permettra de supprimer une application présente sur la carte il faur faire appel à la méthode *deleteApplication*. Voici une description de cette méthode

```
public static byte[] deleteApplication(byte[] aid, byte p2)
```

- Paramètres :
  byte[] aid : un tableau de byte qui représente l' AID de l'application à supprimer sur la carte.

  byte p2 : ne peut avoir que deux valeurs se distinguant par le dernier bit :
  - '0' : Supprimer seulement l'objet ayant l'AID envoyé.

  - '1' : Supprimer en plus les objets liés.
- Valeur retournée : Cette méthode retourne un tableau de byte qui contient l'APDU correspondante à la suppression.
- Les Exceptions :
  - SW1='65', SW2='81' : échec mémoire.
  - SW1='6A', SW2='88' : donnée référenciée introuvable.
  - SW1='6A', SW2='82' : application introuvable.
  - SW1='6A', SW2='80' : valeurs incorrectes entrées dans la commande.

3.1.3 Chargement

Pour construire l'APDU qui permettra de transférer un package sur la carte il faut appeler la commande *load* voici une description de cette méthode.

```
public static byte[] load(byte P1, byte P2,
int Lc, byte[]loadData)
```

- Paramètres byte P1 : Ce paramètre code sur son dernier bit
  - '0' : si la carte doit attendre un autre bloc.
  - '1' : s'il s'agit du dernier bloc.

  byte P2 : Ce paramètre permet le comptage des blocs envoyés à la carte. En effet, Chaque bloc load doit être envoyé avec son numéro d'ordre dans la totalité des blocs.

  int Lc : taille des données à charger.

  byte[] : données à charger.
- Valeur renvoyée Cette fonction renvoie un tableau de bytes.
- Les Exceptions :
  - SW1='65', SW2='81' : échec mémoire.
  - SW1='6A', SW2='84' : mémoire insuffisante.

3.1.4 Installation

Pour construire l' APDU qui permettra d'installer une application sur la carte à partir de son AID il faut faire appel à la méthode $installForInstall$, voici la description de cette méthode.

```
public static byte[] installForInstall(byte[] executableLoadFileAID,
byte[] executableModuleAID,
byte[] applicationAID,
byte applicationPrivileges,
byte[] installParameters)
```

- Paramètres : byte[] executableLoadFileAID : Tableau de bytes qui donne l'AID du package précédemment chargé.

  byte[] executableModuleAID : Tableau de bytes qui donne l'AID du module.

  byte[] applicationAID : Tableau de bytes qui donne l'AID de l'application à installer.

  byte applicationPrivileges : Donne les privilèges concernant l'application. Ce codage est spécifié dans les spécifications Global Platform.

  byte[] installParameters : Tableau de bytes qui contient les paramètres d'installation. Ce paramètre est optionnel

- Valeur Renvoyée : Cette fonction renvoie un tableau de byte.
- Les Exceptions :
  - SW1='65', SW2='81' : échec mémoire.
  - SW1='6A', SW2='80' : paramètres incorrects dans le champ de données.

- SW1='6A', SW2='84' : mémoire insuffisante.
- SW1='6A', SW2='88' : référence des données introuvable.

3.1.5 Etat

Pour construire l' APDU qui permettra d'obtenir des informations sur la carte, le CardManager ou les applications présentes sur la carte, il faut faire appel à la méthode *getStatus* voici la description de cette méthode.

```
public static byte[] getStatus(byte p1, byte p2, byte[] data)
```

- Paramètre : byte p1 : Code le domaine sur lequel la recherche doit être effectuée sur la carte.
  - '0x40' : On inclut seulement les applications et le domaine de sécurité.
  - '0x80' : On retourne seulement des informations sur le domaine de sécurité. Avec cette valeur, le critère de recherche est ignoré.

  byte p2 : Le premier bit code si on attend la première occurrence de la réponse ou la suivante. Le résultat de la recherche dans la carte peut ne pas tenir sur une seule réponse. Sur Le deuxième bit on code la forme souhaitée de la réponse.
  byte[] data : Ce tableau code le critère de recherche à utiliser (recherche sur une ou plusieurs applications, recherche sur la carte...).

- Valeur renvoyée : Cette fonction renvoie un tableau de bytes.
- Les Exceptions :
  - SW1='6A', SW2='88' : référence des données introuvable.
  - SW1='6A', SW2='80' : valeurs incorrectes entrées dans la commande.

3.2 Package offCard

3.2.1 Fonctionnalités

Connexion

la connexion au lecteur de cartes est faite par la méthode *connect*. Cette méthode permet d'ouvrir une session en effectuant la connexion au lecteur, la sélection du card manager et l'authentification mutuelle, ceci est indispensable pour toute action sur la carte ou les packages de la carte.

```
public void connect(String cardName) throws UserException ;

public void connect(String cardName, String userkeys) throws UserException ;

public void connect(String cardName, int keyVersion, int keyIndexNumber)
  throws UserException ;
```

Pour se connecter à la carte, on fait appel aux fonctions du package cardGridCom. Le card manager est le responsable côté carte de gérer les commandes de gestion, il doit être sélectionné avant l'envoi des commandes, ceci se fait en faisant appel à la méthode *selectCardManagement* L'authentification mutuelle est assurée par deux méthodes. La premiire initialise l'authentification, il s'agit de $mutual\_authentification$. La seconde finit l'authentification, il s'agit de la fonction $complete\_authentification$

Chargement et installation

Pour enregistrer un package sur la carte on fait appel à la méthode *install* celle-ci fait la préparation de la carte à recevoir les données en envoyant l'AID du fichier à charger, charge le fichier en question sur la carte et procède à l'installation du package.

On dispose de deux signatures de cette méthode la première suppose qu'une connexion a été initialisé avec connect, la deuxième prend tous les paramètres nécessaires pour initialiser la communication.

Suppression

La suppression d'un package sur la carte se fait par la méthode *delete*. Cette fonction nécessite la sélection du card manager au préalable. Pour cette méthode aussi on dispose de deux signatures selon que l'initialisation de la connexion a été faite ou pas.

Sélection

La sélection d'un package présent sur la carte se fait en appelant la méthode *select*. L'application sélectionnée est celle qui recevra les commandes suivantes, c'est la raison pour laquelle on met *openSelected* à false à la fin. On dispose également de deux signatures. On dispose aussi d'une autre méthode $selectNext$ aui permet de sélectionner l'occurence suivante de l'application spécifiée.

Prévilèges et états

La méthode *getPrivileges* retourne les prévilèges associés à une application, ces prévilèges sont divers et variés, et concernent la sécurité, le blocage, la terminaison, la vérification DAP et l'opération par défaut. Cette méthode initialise une connexion.

Listage des applications

Il s'agit d'avoir la liste de toutes les applications qui sont sur une carte donnée, on dispose pour ceci de la méthode *list_applications*. Cette méthode nécessite aussi l'authentification et la sélection du card manager. La réponse est un tableau qui contient la liste des AIDs des applications trouvées ainsi que d'autres informations comme l'état et les prévilèges de l'application.

Formatage

Pour supprimer toutes les applications de la carte on dispose de la méthode $format$. Elle fonctionne en faisant des appels en boucle de la fonction *deleteApplication* et ce sur la liste des applications que fournit la méthode précédente.

3.2.2 Exemple d'utilisation

Scénario : Une seule carte "carte1"
- installer package1 sur carte1
- installer package2 sur carte1
- selectionner le package1

Comment faire ?

```
...
/*Instance de ApplicationOffCard*/
ApplicationOffCard app =new ApplicationOffCard() ;
...

/* Connexion */
app.connect(carte1) ;
...

/* Installer package1*/
app.install(package1) ;

/* Installer package2*/
app.install(package2) ;

/* Sélectionner package1*/
app.select(package1) ;
```

Deuxième méthode :

```
...
/*Instance de ApplicationOffCard*/
ApplicationOffCard app =new ApplicationOffCard() ;
...

/* Installer package1*/
app.install(carte1, userKeys, package1) ;

/* Installer package2*/
app.install(carte1, userKeys, package2) ;

/* Sélectionner package1*/
app.select(carte1, userKeys, package1) ;
```

La différence entre les deux méthodes est que la deuxième fournit pour chaque appel les paramètres de connexion(nom de carte et clés) : La connexion et l'authentification sont réinitialisées à chaque opération.

On voit donc que la première méthode est plus efficace dans le cas où on veut lancer un grand nombre d'opérations sur une même carte, la connexion et l'authentification à celle-ci sont effectuées une seule fois avec la méthode connect().

Cependant la deuxième méthode est utile dans le cas où on manipule un certain nombre de cartes avec une seule instance de la classe ApplicationOffCard.

Chapitre 4

Utilisation du gpShell

Le package globalPlatform fournit une suite de programmes principaux permettant d'utiliser en ligne de commande les différentes fonctionnalités de l'API.

Pour faciliter l'utilisation de ces classes, deux scripts run et run.bat sont fournis respectivement pour UNIX et Windows.

4.1 Commandes gpShell

- **install :** Installe un package sur une carte.

```
install -help to see this help.

install cardName packagePath [userKeys]
Or
install cardName packagePath [keyVersion keyIndexNumber]
```

- **installInAll :** Installe un package sur toutes les cartes connectées.

```
installInAll -help to see this help.

installInAll packagePath
```

- **format :** Formate une carte .

```
format -help to see this help.

format cardName [userKeys]
Or
format cardName [keyVersion keyIndexNumber]
```

- **formatAll :** Formate toutes les cartes connectées .

```
formatAll -help to see this help.

formatAll
```

- **delete :** Supprime un package de la carte.

```
delete -help to see this help.
```

```
delete cardName packagePath [userKeys]
Or
delete cardName packagePath [keyVersion keyIndexNumber]
```

- **ls :** Affiche les AID des packages de la carte.

```
ls -help to see this help.

ls cardName [userKeys]
Or
ls cardName [keyVersion keyIndexNumber]
```

4.2 Usage

Les scripts run et run.bat offrent l'accès aux commandes précédentes de la manière suivante :

```
run commande [arguments]
```

Où commande est une des commandes gpShell et [arguments] est la suite des arguments de cette commande.

Chapitre 5

Les tests unitaires

Les scripts run et run.bat permettent aussi de lancer le module des tests unitaires ApplicationOnCardTests :

```
run tests
```

L'exéctuion de ces tests nécessite l'archive junit.jar fournit dans le dossier lib.

Chapitre 6

Fichier log

Notre API permet de générer un fichier log résumant les différentes opérations et communications avec les cartes.
Le fichier log généré est : "GP_log.log".

Cependant, il est possible de désactiver la génération de ce fichier ou diriger le flux du log vers la sortie standard. Ceci se fait avec la méthode setLog() de la classe ApplicationOffCard :

```
app = new ApplicationOffCard() ;
/* La destination par défaut du log est la sortie standard */
...
/* Pour rediriger vers le fichier de log:*/
app.setLog(1) ;


/* Pour désactiver le log */
app.setLog(0) ;
...
```