PMD2271-BC
- PUBLIC

SpringCard PC/SC Couplers - H663 group

Manuel de référence développeurs

www.springcard.com

Document Identification

Category	Developer's manual		
Family/Customer	PC/SC Couplers		
Reference	PMD2271	Version	BC
Status		Classification	Public
Keywords	H663, CrazyWriter HSP, TwistyWriter HSP, CSB HSP, Prox'N'Roll HSP, FunkyGate HSP, PC/SC, NFC P2P, contactless cards, RFID labels, NFC tags		
Abstract			

File name	V:\Dossiers\SpringCard\A-Notices\H663 Group\Developpement\[PMD2271-BD] H663 Developer's Reference Manual FR.odt		
Date saved	27/08/18	Date printed	05/12/12

Revision History

Ver.	Date	Author	Valid. by		Approv. by	Details
			Tech.	Qual.		
AA	29/08/12	JDA				Created from PMD841P-FA and PMD2176-AD
AB	18/03/13	JDA				Separated chapter 2 from chapter 1 Renumbered the chapters (moved up “contactless hints” and “NFC initiator role”), named last chapters “annex”. New drawings Documented all the configuration registers Removed the preliminary Watermark
AC	21/05/13	JDA				Added the capability to stop the contactless slot when a contact smartcard is inserted
AD	23/07/13	JDA				Documented new behaviour of firmware ≥ 1.75 for NFC Forum Type 1 tags, including Innovision (now Broadcom) Topaz and alike
AE	17/06/14	JDA				Added time-out extensions for T=0 and T=1 Improved chapter 7
AF	27/08/14	JDA				A few rewordings Order codes and related documents added
AG	07/11/14	JDA				New features from FW 1.81: - 'RFID memory control' instruction (INS=F6) - Support of EM5134 - Recognition of Mifare Plus cards, updated the PIX.NN listing
BA	13/04/15	JDA				New features from FW 2.00: - buzzer - name of reader could be the serial number Added FreeRTOS licence information Added Prox'N'Roll HSP
BB	06/06/16	JDA				Description of register C8 was not correct
BC	14/03/17	LCO				Kovio → ThinFilm
BD	18/05/17	JIZ				Corrected APDU in § 3.3.4.d (Read Multiple Blocks)

Contents

1. Introduction

1.1. Résumé

SpringCard H663 est un module coupleur PC/SC RFID et NFC, composé d'interfaces optionelles 0 à 5 T=0/T=1 pour les cartes à puces à contact ou SIM/SAM. Le module **H663** est le coeur de nombreux coupleurs PC/SC proposés par **SpringCard**, et de lecteurs spécifiques créés en OEM.

Ce document fournit toutes les informations nécessaires pour développer un logiciel qui utilisera le coeur **H663**.

1.2. Produits supportés

Au moment de son écriture, ce document s'applique à tous les **Coupleurs PC/SC SpringCard** dans le groupe **H663**:

- Le **H663S** et **H663A**: Modules OEM sans antenne,
- Le **H663-USB**: Coupleur OEM avec une antenne intégrée basée sur le **H663S**,
- Le **CrazyWriter HSP**: Coupleur OEM multi-interface basé sur le **H663A**,
- Le **TwistyWriter HSP**: Coupleur OEM avec antenne déportée basé sur le **H663S**,
- Le **Prox'N'Roll HSP**: coupleur de bureau,
- Le **CSB HSP LT**: coupleur de bureau large basé sur le **H663S**,
- Le **CSB HSP**: coupleur de bureau multi-interface basé sur le **H663S.**

Liste des produits avec leurs codes de commande

Product name	Order code	Description
H663S	SC14182	Contactless PC/SC OEM module for balanced antenna
H663SC	SC14183	Contact & contactless PC/SC OEM module for balanced antenna
H663A	SC14184	Contactless PC/SC OEM module for unbalanced antenna
H663AC	SC14185	Contact & contactless PC/SC OEM module for unbalanced antenna
H663-USB	SC3016	Contactless PC/SC OEM coupler with integrated antenna
CrazyWriter HSP	SC0168	Contact & contactless PC/SC OEM coupler with 1 x SAM + 1 x remote antenna (50Ω) *(expansion board for 3 more SAM available as an option)*
CrazyWriter HSP Dual	SC14148	Contact & contactless PC/SC OEM coupler with 1 x SAM + 2 x remote antennas (50Ω) *(expansion board for 3 more SAM available as an option)*
TwistyWriter HSP	SC14190	Contactless PC/SC OEM coupler with remote antenna (TP)
Prox'N'Roll HSP PC/SC	SC15131	Desktop contactless PC/SC coupler
CSB HSP	SC0177	Desktop contact & contactless PC/SC coupler with 3 x SAM + 1 x ID-1 slot
CSB HSP LT	SC14048	Desktop contactless PC/SC coupler

1.3. Documents liés

a. *Utilisateur final*

Editor	Doc #	Description
SpringCard	PMU14186	CSB HSP, CSB HSP LT QuickStart Guide
SpringCard	PMU14092	CrazyWriter HSP QuickStart Guide
SpringCard	PMU14189	TwistyWriter HSP QuickStart Guide

b. *Intégrateurs*

Editor	Doc #	Description
SpringCard	PMD2236	H663 Hardware Integration guide
SpringCard	PNA14187	CrazyWriter HSP Hardware Integration guide
SpringCard	PNA14188	TwistyWriter HSP Hardware Integration guide

1.4. Public

Ce manuel est créé pour les développeurs d'application. Il suppose que le lecteur a des connaissances expertes en développement informatique et des connaissances basique du PC/SC, du standard ISO 7816-4 pour les cartes à puce, et des spécifications du NFC Forum.

La chapitre 2 fournit une courte introduction à ces technologies et concepts, mais ne peut pas couvrir tous les aspects, comme le ferait un livre ou une session d'entraînement.

1.5. Support et mises à jour

Les documents liés (fiche technique de produits, notes d'application, échantillon logiciel, HOWTO, FAQ…) sont disponibles sur le site web SpringCard:

www.springcard.com

Les versions mises à jour de ce document et des autres seront postées sur le site web dès qu'elles seront disponibles.

Pour des demandes de support technique, merci de vous référer à la page support

www.springcard.com/support

1.6. Liens utiles

- La documentation de référence Microsoft PC/SC est incluses dans le système d'aide Visual Studio et disponible en ligne à http://msdn.microsoft.com . Entrer les mots clés “winscard” ou “SCardTransmit” dans la boîte de recherche.
- Projet MUSCLE PCSC-Lite : http://www.musclecard.com (lien direct pour le stack PC/SC: http://pcsclite.alioth.debian.org)
- PC/SC workgroup: http://www.pcscworkgroup.com
- NFC Forum: http://www.nfc-forum.org

2. PC/SC, cartes à puce et NFC: courte introduction et glossaire

2.1. Standards des cartes à puce et cartes à puce sans-contact

Une **carte à puce** est un microprocesseur (faisant fonctionner un logiciel bien sûr) monté sur une carte en plastique.

La famille de standards **ISO 7816** défini tout pour les cartes à puce à contact:

- **ISO 7816-1** et **ISO 7816-2** définissent les facteurs de forme et les caractéristiques électriques,
- **ISO 7816-3** introduit les protocoles des deux niveaux de transport entre un coupleur et la carte: “T=0” and “T=1”,
- **ISO 7816-4** mandate un ensemble de fonctions communes. Cet ensemble de fonctions montre la carte à puce comme un système de petits fichiers, avec des répertoires et des fichiers dans lesquels les données sont stockées. Les frames du niveau applicatif sont appellées des **APDU**.

La famille **ISO 14443** est la référence normative des cartes à puce sans-contact:

- **ISO 14443-1** et **ISO 14443-2** définissent les facteurs de forme, les caractéristiques RF et la comunication au niveau bit,
- **ISO 14443-3** précise les parties de communication aux niveaux byte et frames[1],
- **ISO 14443-4** introduit un protocole au niveau du transport qui ressemble plus ou moins à T=1,il est souvent appelé “T=CL” (mais ce nom n'apparaît jamais comme un standard).

En plus du T=CL, la **carte à puce sans-contact** est censée avoir le même ensemble de fonctions et de règles formattant les APDU que les **cartes à puce à contact,** ex: cela doit être “ISO 7816-4 en plus de ISO 14443”.

Dans ce contexte travailler avec une carte à puce (avec ou sans-contact) est aussi simple que d'envoyer une commande (C-APDU) à la carten et recevoir sa réponse (R-APDU). Le “lecteur de carte à puce” est simplement une gateway qui implémente cet **échange APDU** (avec une relative abstraction des protocoles du niveau transport).

[1] ISO 14443-2 and -3 are divided into 2 technologies: ISO 14443 type A and ISO 14443 type B. They use different codings and low-level protocols, but the transport protocol defined in ISO 14443-4 is type-agnostic: it makes no difference whether the card is type A or type B.

2.2. PC/SC

PC/SC est le standard pour interfacer les *ordinateurs personnels* avec les *cartes à puce* (et les lecteurs de cartes à puce évidemment). Les **coupleurs PC/SC SpringCard** sont compatibles avec ce standard. Ce qui permet d'utiliser ces produits sur la plupart des systèmes d'exploitation, en utilisant des API standardisées de pointe.

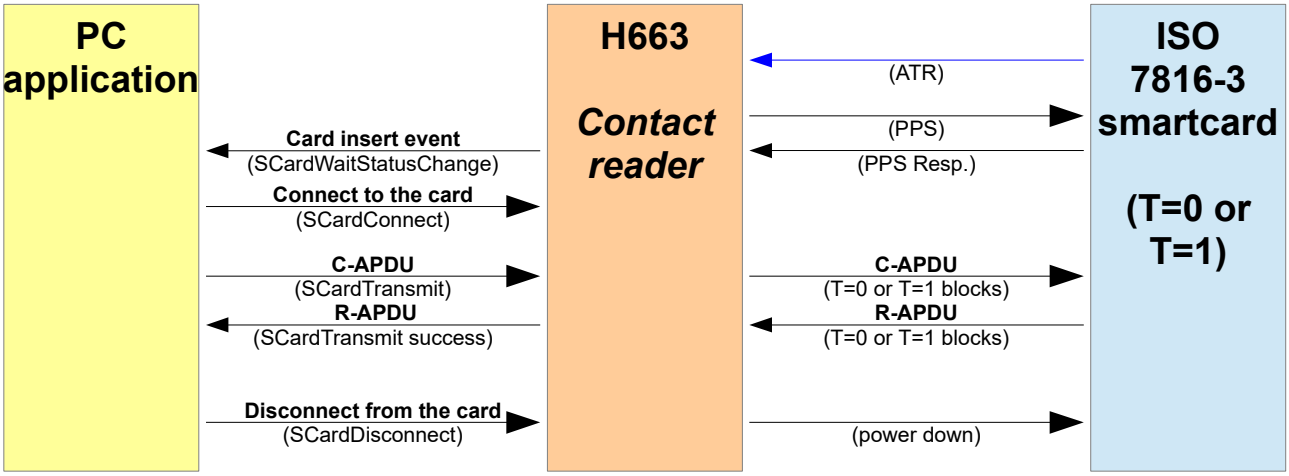
Pour commencer avec le PC/SC, merci de lire notre Introduction au développement PC/SC et documentation simplifiée de l'API, disponible en ligne à

http://www.springcard.com/download/find.php?file=pmdz061

Le coeur du PC/SC est la fonction *ScardTransmit,* qui est l'implémentation dans l'ordinateur des **échanges APDU**.

> *Si la carte à puce avec laquelle vous travaillez n'est pas compatible avec ISO 7816-4, il n'y a rien de plus à ajouter!*
> *Référez-vous au standard ISO 7816-4 et/ou à la documentation de la carte[2] pour connaître les APDU à envoyer, et comment comprendre les réponses. Ensuite implémenter votre processus connaissance de carte grâce à un lot d'appels SCardTransmit. Que la carte à puce soit avec ou sans-contact ne fait pas une grande différence[3].*

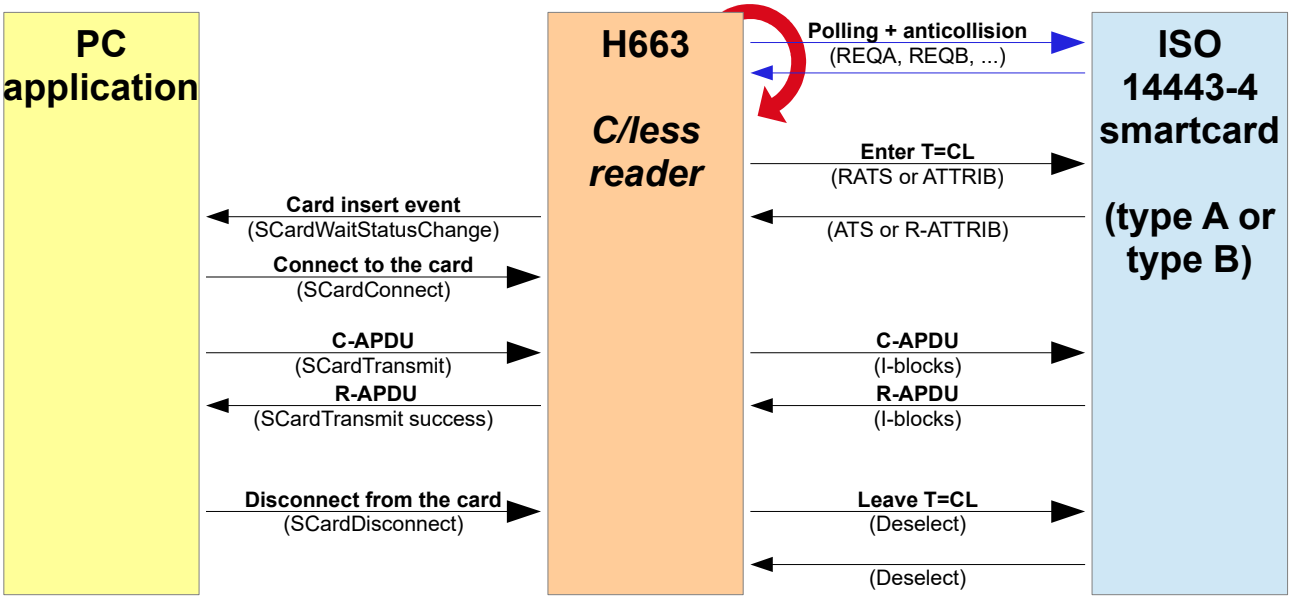
a. PC/SC et une carte à puce à contact



[2] Note that a microprocessor-based smartcard is a chip plus some application running in it. It could be a monolithic application, without an operating system, or an operating system (for instance JavaCard) with some applications added. You need the documentation of the application(s) and in some situations the documentation of the operating system, not the chip's.

[3] Actually there's more differences between contact protocols T=0 and T=1 than between contactless protocol "T=CL" and T=1. When developing an application for a contactless smartcard, read the ISO 7816-4 standard and the documentation of the smartcard as if it were running T=1.

b. *PC/SC et une carte à puce sans-contact*



2.3. Cartes sans-contact non-7816-4 – introduire l'interprète APDU embarqué

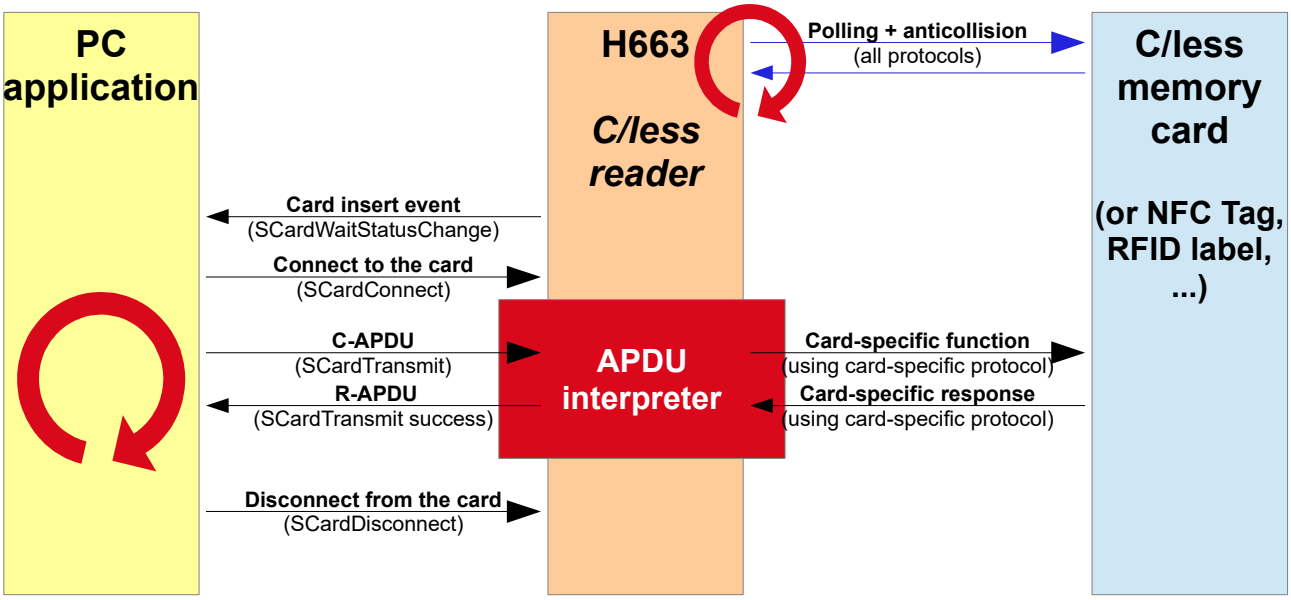
Beaucoup de cartes sans-contact ne sont pas des “cartes à puces” car elles ne sont pas compatibles avec ISO 7816-4. Elles ne sont pas compatibles avec le protocole du niveau transport, et l'ensemble de leurs fonctions spécifiques vendeurs ne correspondent pas à une fonction “échange” seule. Elles ne sont donc pas supportées naturellement par le système stack PC/SC. C'est le cas de:

- **Cartes mémoires à logique câblée** (Mifare, CTS, SR... families),
- **NFC Tags** (type 1, type 2, type 3),
- Certaines des cartes propriétaires basées sur le microprocesseur qui utilisent un **protocole de communication spécifique** avec un format de frame incompatible avec ISO 7816-4 (Desfire EV0...).

Le rôle de l'**interprète APDU embarqué**, fonctionnant dans le **H663**, est d''émuler' un standard de carte à puce dans ces cas. En faisant cela, le stack PC/SC (et en conséquence votre application) n'a pas besoin de gérer les protocoles sous-jacent et les commandes spécifiques à la puce.

En fait l'**interprète d'APDU embarqué** expose la carte comme étant une carte à puce T=1 compatible, et fournit deux fonctions prise de l'ISO 7816-4: READ BINARY et UPDATE BINARY. En ISO 7816-4, ses fonctions sont désignées pour accéder aux données dans un fichier (dans le système de fichiers de la carte), mais sur les cartes mémoire elles donnent accès à un stockage “brut”, utilisant un accès basé sur le byte, le bloc ou la page selon la technologie de la carte et ses options.

a. *Rôle de l'interprète APDU embarqué*



2.4. NFC ?

NFC signifie **Near Field Communication** (communication en champ proche), ce qui est le cas de tous les systèmes de communication utilisant des basses fréquences ou fonctionnant sur des distances très courtes. Mais le NFC est maintenant compris comme

- **NFCIP-1** (Near Field Communication Interface et Protocole), i.e. le standard ISO 18092, qui définit un nouveau protocole de niveau transport parfois appelé “peer-to-peer” (mais ce nom n'apparaît jamais comme standard),
- **NFC Forum**, une association qui promeut l'utilisation du NFC et publie dans standards au “niveau applicatif” (lorsque l'ISO est basé sur les niveaux techniques).

SpringCard H663 et les produits dérivés sont partiellement compatibles avec NFCIP-1 (rôle initiateur et mode de communication passive seulement). Merci de vous référer au chapitre 5 pour plus de détails. Ces produits doivent également supporter les applications NFC Forum, mais aucune compatibilité avec les critères bas du NFC Forum n'est demandée.

> *Noter que dans la littérature NFC Forum,*
> *- **ISO 14443 type A** et **ISO 18092 @ 106kbit/s** est appelé **NFC-A**,*
> *- **ISO 14443 type B** est appelé **NFC-B**,*
> *- **JIS:X6319-4** et **ISO 18092 @ 212/424kbit/s** est appelé **NFC-F**.*

2.5. Options vendeur spécifiques – Contrôle direct du coupleur

La fonction PC/SC *SCardTransmit* implémente une chaîne de communication entre votre application et la carte. Mais parfois l'application veut accéder à certaines options du **H663** lui-même: contrôler les LED ou le buzzer, connaître le numéro de série… En d'autres termes, l'application veut parler au coupleur et non pas à la carte.

La fonction PC/SC *SCardControl* a été créée pour cela. Le chapitre 6 détaille les commandes supportées par le **H663** en utilisant la chaîne de communication directe. Mais ouvrir une chaîne *SCardControl* signifie avoir un accès direct (et exclusif) au coupleur, et en conséquence bloque les autres chaînes de communication.

Pour surmonter cet inconvénient, l'**interprète APDU embarqué** peut également être utilisé pour transporter les commandes au coupleur avec une chaîne de carte existante et en utilisant les appels *SCardTransmit* (voir § 3.4.1 pour les détails).

2.6. Glossaire – termes utiles

La liste suivante présente les termes directement liés au sujet de ce document. C'est un extrait de notre glossaire technique, disponible en ligne à:

http://www.springcard.com/blog/technical-glossary/

- **ICC:** *carte à circuit intégré*. C'est le nom standard pour une carte en plastique avec une puce silicone (un circuit intégré) compatible avec les standards ISO 7816. Le nom commun est *carte à puce*.
- **CD:** *appareil coupleur* ou **coupleur.** Un appareil capable de communiquer avec une ICC. C'est pour cela que tout le monde l'appelle un *lecteur de carte à puce*. Techniquement, il peut être vu comme une gateway entre l'ordinateur et la carte.
- **Carte basée sur un microprocesseur:** une ICC (ou une PICC) dont la puce est un petit ordinateur. C'est le cas des cartes haut de gamme utilisées dans le paiement, les transports, les eID/passeports, le contrôle d'accès... Les points clés sont la sécurité, la possibilité de stocker beaucoup de données et de faire fonctionner l'application dans la puce. La plupart du temps ils implémentent l'ensemble de commandes définit par ISO 7816-4.
- **Carte mémoire** ou **carte à logique câblée:** une ICC (ou une PICC) dont la puce est uniquement capable de stocker certaines données, avec un schéma de sécurité limité (ou aucun schéma de sécurité). Elles sont moins chères que les cartes basées sur un microprocesseur et seront donc utilisées pour la traçabilité RFID, la fidélité, le contrôle d'accès...
- **PICC:** *carte de proximité à circuit intégré*. C'est le nom standard de n'importe quelle carte sans-contact compatible avec les standards ISO 14443 (proximité: moins de 10cm). Cela peut être une carte à puce ou une carte mémoire ou bien un objet NFC fonctionnant en mode émulation. Les noms communs sont *cartes sans-contact*, ou *carte RFID, tag NFC*.
- **PCD:** *appareil coupleur de proximité.* Appareil capable de communiquer avec un PICC, un coupleur sans-contact compatible avec ISO 14443.
- **RFID:** *identification par radio-fréquence*. C'est le nom général de n'importe quel système utilisant les ondes radios pour la communication M2M (machine à machine, dans notre cas PCD à PICC).
- **VICC:** *carte de proximité à circuit intégré*. C'est le nom standard pour n'importe quelle carte sans-contact compatible avec les standards ISO 15693 (proximité: mois de 150cm). Les noms communs sont *tag RFID*, *étiquette RFID*.
- **VCD:** *appareil coupleur de proximité.* Appareil capable de communiquer avec une VICC, un coupleur sans-contact compatible avec ISO 15693.
- **NFC:** *communication en champ proche*. Un sous-ensemble de la RFID, où la portée est plus courte que la longueur des ondes radio impliquées. C'est le cas pour ISO 14443: la

fréquence est de 13.56MHz, avec une longueur d'onde de of 22m. Les gammes de proximité sont plus courtes que cette longueur d'onde.

- **NFC Forum:** une association internationale qui a pour but de standardiser les applications du NFC dans la zone 13,56MHz. Leur contribution principale est le tag NFC, qui n'est rien de plus que les PICC dont les données sont formatées en accord avec leurs spécifications, pour que l'information qu'ils contiennent soit compatible avec n'importe quelle application compatible.
- **NDEF:** *NFC Data Exchange Format.* Le format des données dans les tags NFC spécifié par le NFC Forum.
- **ISO 7816-1** et **ISO 7816-2:** Ce standard international définit les caractéristiques hardware de l'ICC. Le format de carte à puce standard (84x54mm) est appelé ID-1. Un plus petit facteur de forme est utilisé pour les cartes SIM (utilisées dans les téléphones mobiles) ou SAM (module d'authentification sécurisé, utilisé pour les applications de transport ou de paiement) il est appelé ID-000.
- **ISO 7816-3:** Ce standard international définit deux protocoles de communication pour les ICCs: T=0 et T=1. Un coupleur compatible doit supporter les deux.
- **ISO 7816-4:** Ce standard international définit un schéma de communication et un ensemble de commandes. Le schéma de communication est fait d'APDU. L'ensemble de commandes suppose que la carte est structurée de la même manière que le disque dur d'un ordinateur: les répertoires et les fichiers peuvent être sélectionnés (instruction SELECT) etaccessibles pour la lecture ou l'écriture (instructions READ BINARY, UPDATE BINARY). Plus de 40 instructions sont définies par le standard, mais la plupart des cartes intègrent seulement un petit sous-ensemble, et ajoutent souvent leur instructions (spécifiques au vendeur).
- **APDU:** *unité datagramme du protocole d'application*. Ce sont les frames échangées au niveau applicatif entre une application fonctionnant sur l'ordinateur et une carte à puce. Le format de ses frames sont définies par ISO 7816-4 et vérifiées par le stack PC/SC du système. La commande (application vers la carte) est appelée C-APDU, la réponse (carte vers application) est R-APDU. Noter que c'est un schéma demande/réponse: la carte à puce n'a aucun moyen d'envoyer quelque chose à l'application sauf si l'application le demande.
- **ISO 14443:** Ce standard international définit le schéma de communication PCD/PICC. Il est divisé en 4 couches:
    1. Définit les caractéristiques hardware du PICC,
    2. Définit la fréquence et le schéma de communication au niveau bit,
    3. Définit le schéma de communication au niveau frame et la séquence d'ouverture de session (anti-collision),
    4. Définit le schéma de communication du niveau transport (parfois appelé “T=CL”).

Le niveau applicatif est hors du cadre de ISO 14443. La plupart des PICC basés sur des microprocesseurs implémentent ISO 7816-4 en plus de ISO 14443-4.

Beaucoup de PICC à logique câblée (la famille NXP Mifare, les familles ST Micro Electronics ST/SR, pour en nommer certaines) implémentent seulement un sous-ensemble de ISO 14443, et ont leur propre ensemble de fonctions en plus de ISO 14443-2 ou ISO 14443-3.

Noter que ISO 14443-2 et ISO 14443-3 sont divisés en 2 protocoles appelés 'A' et 'B' Un PCD doit implémenter les deux, mais le PICC implémente seulement l'un des deux[4]. Quatre débit sont possibles: 106 kbit/s est obligatoire, les débits plus élevés (212, 424 ou 848 kbit/s) sont optionnels.

- **ISO 15693:** Ce standard international définit le schéma de communication VDC/VICC. Il est divisé en 3 couches:

  1. Définit les caractéristiques hardware du VICC,
  2. Définit la fréquence et le schéma de communication au niveau bit,
  3. Définit le schéma de communication au niveau frame, la séquence d'ouverture de session (anti-collision/inventaire), et l'ensemble de commande du VICC.

  Toutes les VICC sont des puces mémoire. La zone de stockage de leurs données est divisée en blocs. La taille des blocs et leur nombre dépend de la VICC.

  Noter que ISO 18000-3 mode 1 est la même chose que ISO 15693[5].

- **ISO 18092** ou **NFCIP-1:** Ce standard international définit un schéma de communication (souvent appelé “peer to peer mode”) où deux “objets” pairs sont capables de communiquer ensemble (et pas seulement un PCD et un PICC). Le protocole sous-jacent est ISO 14443-A à 106 kbit/s et JIS:X6319-4 (aka protocole Sony Felica) à 212 et 424 kbit/s.
- **Initiateur:** selon NFCIP-1, l'objet NFC qui est le “maître” de la communication avec un pair connu comme la cible. Un PCD est une sorte d'initiateur.
- **Cible:** selon NFCIP-1, l'objet NFC qui est “l'esclave” de la communication avec un pair connu comme l'initiateur. Un PICC est une sorte de cible.
- **NFC-DEP:** *NFC Data Exchange Protocol.* C'est le nom utilisé par le NFC Forum pour le protocole ISO 18092 “haut de gamme”. Après une poignée de mains initiale (ATR_REQ/ATR_RES), l'initiateur et la cible échangent des blocs du niveau transport. (DEP_REQ/DEP_RES).
- **LLCP:** *Logical Link Control Protocol.* Un protocole réseau spécifié dans le NFC Forum en plus du NFC-DEP.
- **SNEP:** *Simple NDEF Exchange Protocol.* Un protocole d'application spécifié par le NFC Forum pour échanger des messages NDEF en plus du LLCP.

[4] Yet some NFC objects may emulate both an ISO 14443-A and an ISO 14443-B card.

[5] ISO 15693 has been written by the workgroup in charge of smartcards, and then copied by the workgroup in charge of RFID into ISO 18000, the large family of RFID standards.

- **ISO 21481** ou **NFCIP-2:** Le standard international définit comment un objet NFC doit également être capable de communiquer en utilisant les standards ISO 14443 et ISO 15693.
- **Mifare:** Cette marque de NXP (précedemment Philips Semiconductors) est la marque générique de leurs produits PICC. Des milliards de cartes Mifare Classic ont été déployées depuis les années 90. C'est une famille de PICC à logique câblée où le stockage des données est divisé en secteurs et protégés par un flux crypté propriétaire[6] appelé CRYPTO1. Chaque secteur est protégé par 2 clé d'accès appelées “clé A” et “clé B”[7]. NXP offre également une autre famille de PICC à logique câblée appellée Mifare UltraLight (adoptée par le NFC Forum comme les Tags NFC Type 2). Mifare SmartMX (et anciens Pro/ProX) est une famille de PICC basés sur des microprocesseurs qui peuvent virtuellement faire fonctionner n'importe quelle application de carte à puce, sur un système d'explotation JavaCard. Mifare Desfire est un PICC basé sur un microprocesseur particulier qui fait fonctionner une application à but unique.
- **Felica:** Cette marque de Sony est la marque générique de leurs produits PICC. Le protocole sous-jacent a été standardisé au Japon (JIS:X6319-4) et est utilisé par ISO 18092 à 212 et 424 kbit/s. Le standard Felica inclut un schéma de sécurité dont Sony est propriétaire qui n'est pas implémenté dans les produits SpingCard. Seules les puces Felica configurées pour fonctionner sans sécurité (“Felica Lite”, “Felica Lite-S”, ou NFC Type 3 Tags) sont supportées.

[6] And totally broken. Do not rely on this scheme in security-sensitive applications!

[7] A typical formatting would define key A as the key for reading, and key B as the key for reading+writing.

3. Interprète APDU embarqué

3.1. Base

Dans l'architecture PC/SC, la fonction ***SCardTransmit*** implémente le dialogue entre une application et une carte à puce, le coupleur est seulement une gateway “passive”. Le coupleur transmet les frames dans les deux directions sans traitement spécifique. Le dialogue suit les règles ISO 7816-4 APDU :

- Application à carte à puce **C-APDU** est *CLA, INS, P1, P2, Data In (optionnel)*
- Carte à puce à application **R-APDU** est *Data Out (optionnel), SW1, SW2*

Pour fonctionner avec les cartes non ISO 7816-4 comme si elles étaient des cartes sans contact, l'interprète APDU embarqué obéit aux mêmes règles, offrant sa liste d'instructions sous la classe réservée **CLA=$_h$FF**. Il est donc disponible via des appels *SCardTransmit* réguliers.

3.1.1. CLA byte de l'interprète APDU embarqué

La classe par défaut est $_h$FF. Cela signifie que chaque APDU commençant par CLA= $_h$FF seront interprété par le **H663**, et non transmis par la carte.

a. Changer le CLA byte de l'interprète APDU embarqué

Le CLA byte de l'interprète APDU embarqué est stocké dans le registre $_h$B2 de la mémoire non volatile du **H663** (voir § 7.4.2).

Noter: dans les paragraphes suivants, la documentatoin des APDU est écrite avec CLA= $_h$FF. Changez cela pour correspondre à votre CLA si nécessaire.

b. Désactiver l'interprète APDU embarqué

Définir le CLA byte = $_h$00 (registre $_h$B2= $_h$00, voir § 7.4.2) pour désactiver l'interprète APDU embarqué.

3.1.2. Statuts renvoyés par l'interprète APDU embarqué

SW1	SW2	Meaning
$_h$90	$_h$00	Success
$_h$67	$_h$00	Wrong length (Lc incoherent with Data In)
$_h$68	$_h$00	CLA byte is not correct
$_h$6A	$_h$81	Function not supported (INS byte is not correct), or not available for the selected PICC/VICC
$_h$6B	$_h$00	Wrong parameter P1-P2
$_h$6F	$_h$01	PICC mute or removed during the transfer

Certaines fonctions fournies par l'interprète APDU embarqué peuvent renvoyer des statuts spécifiques. Cette attitude est documentée dans le paragraphe dédié à chaque fonction.

3.1.3. Liste des instructions de l'interprète APDU embarqué

Instruction	INS	Contactless	Contact	Notes (see below)
LOAD KEY	$_h$82	✓		C
GENERAL AUTHENTICATE	$_h$86	✓		C
READ BINARY	$_h$B0	✓		A
ENVELOPE	$_h$C2	✓		B
GET DATA	$_h$CA	✓	✓	C
UPDATE BINARY	$_h$D6	✓		A
READER CONTROL	$_h$F0	✓	✓	D
MICORE CONTROL	$_h$F1	✓		D
ABCCORE CONTROL	$_h$F1		✓	D
MIFARE CLASSIC READ	$_h$F3	✓		D
MIFARE CLASSIC WRITE	$_h$F4	✓		D
MIFARE CLASSIC VALUE	$_h$F5	✓		D
RFID MEMORY CONTROL	$_h$F6	✓		D – starting from FW 1.81
CONTACTLESS SLOT CONTROL	$_h$FB	✓		D
TEST	$_h$FD	✓	✓	D
ENCAPSULATE	$_h$FE	✓	✓	D

Notes:

A Function fully implemented according to PC/SC standard

B Function implemented according to PC/SC standard, but some feature are not supported

C Function implemented according to PC/SC standard, but also provides vendor-specific options

D Vendor-specific function

3.2. Instructions définies par le standard PC/SC (v2 partie 3)

3.2.1. Instruction GET DATA

L'instruction **GET DATA** retrouve les informations selon le PICC inséré. Il peut être utilisé avec n'importe quel type de PICC, mais le contenu renvoyé variera avec le type de PICC actuellement dans le port.

GET DATA commande APDU

CLA	INS	P1	P2	Lc	Data In	Le
$_h$FF	$_h$CA	See below	See below	-	-	$_h$00

GET DATA paramètres de commande

P1	P2	Action	Fw
		Standard PC/SC-defined values	
$_h$00	$_h$00	Serial number of the PICC - ISO 14443-A : UID (4, 7 or 11 bytes) - ISO 14443-B : PUPI (4 bytes) - ISO 15693: UID (8 bytes) - Innovatron : DIV (4 bytes) - JIS:X6319-4 : IDm (8 bytes) - others: see chapter 4 for details	
		SpringCard specific values	
$_h$01	$_h$00	- ISO 14443-A : historical bytes from the ATS - ISO 14443-B : INF field in ATTRIB response - JIS:X6319-4 : PMm (8 bytes) - ISO 18092 : G_T bytes in ATR_RES - others: see chapter 4 for details	
$_h$F0	$_h$00	Complete identifier of the PICC: - ISO 14443-A: ATQA (2 bytes) + SAK (1 byte) + UID - ISO 14443-B: complete ATQB (11 or 12 bytes)[8] - ISO 15693: answer to GET SYSTEM INFORMATION command[9] - Innovatron: REPGEN - Innovision/Broadcom/NFC Forum Type 1 Tag: HR0, HR1 - JIS:X6319-4 : IDm and PMm (16 bytes) - ISO 18092 : complete ATR_RES	≥ 1.75

[8] SpringCard PC/SC Couplers are ready to support the extended ATQB (12 bytes), but since a lot of PICC currently in circulation don't reply to the REQB command with the “extended” bit set, this feature is not enabled by default.

[9] If the card doesn’t support the GET SYSTEM INFORMATION COMMAND, a valid SYSTEM INFORMATION value is constructed, including the UID and the DSFID byte.

P1	P2	Action	
$_h$F1	$_h$00	Type of the PICC/VICC, according to PC/SC part 3 supplemental document: PIX.SS (standard, 1 byte) + PIX.NN (card name, 2 bytes) See chapter 4.1 for details	
$_h$F1	$_h$01	NFC Tag[10] compliance: - $_h$01 if the PICC is recognized as a NFC Type 1 Tag - $_h$02 if the PICC is recognized as a NFC Type 2 Tag - $_h$03 if the PICC is recognized as a NFC Type 3 Tag - $_h$00 otherwise	
$_h$F2	$_h$00	"Short" serial number of the PICC - ISO 14443-A: UID truncated to 4 bytes, in "classical" order - others: same as P1,P2=$_h$00,$_h$00	
$_h$FA	$_h$00	Card's ATR	
$_h$FC	$_h$00	ISO 14443 communication indexes on 2 bytes (DSI, DRI)	
$_h$FC	$_h$01	PICC/VICC → H663 baudrate (DS in kbit/s, 2 bytes, MSB first)	
$_h$FC	$_h$02	H663 → PICC/VICC baudrate (DR in kbit/s, 2 bytes, MSB first)	
$_h$FC	$_h$03	Index of the active antenna on 1 byte	
$_h$FF	$_h$00	Product serial number (raw value on 4 bytes)	
$_h$FF	$_h$01	*Not available for H663*	
$_h$FF	$_h$02	Name of the RF interface component ("RC663")	
$_h$FF	$_h$81	Vendor name in ASCII ("SpringCard")	
$_h$FF	$_h$82	Product name in ASCII	
$_h$FF	$_h$83	Product serial number in ASCII	
$_h$FF	$_h$84	Product USB identifier (VID/PID) in ASCII	
$_h$FF	$_h$85	Product version ("x.xx") in ASCII	

GET DATA réponse

Data Out	SW1	SW2
XX … XX	See below	

GET DATA Statut

SW1	SW2	Meaning
$_h$90	$_h$00	Success
$_h$62	$_h$82	End of data reached before Le bytes (Le is greater than data length)
$_h$6C	XX	Wrong length (Le is shorter than data length, XX in SW2 gives the correct value)

[10] Please refer to NFC Forum's specifications for details. Note that NFC Forum Type 4 Tags are "standard" contactless smartcards; it is up to the application level to send the proper SELECT APPLICATION to recognize them.

3.2.2. Instruction LOAD KEY

L'instruction **LOAD KEY** charge une clé d'accès 6-byte Mifare Classic (CRYPTO1) dans la mémoire du **H663**.

LOAD KEY commande APDU

CLA	INS	P1	P2	Lc	Data In	Le
$_h$FF	$_h$82	Key location	Key index	$_h$06	Key value	-

LOAD KEY paramètre de commande P1 (clé de localisation)

P1	
$_h$00	The key is to be loaded in H663's volatile memory
$_h$20	The key is to be loaded in H663's non-volatile memory (secure E2PROM inside the RC chipset)

LOAD KEY paramètre de commande P2 (clé d'index)

Quand P1 = $_h$00, P2 est l'identifiant de la clé dans la mémoire volatile de **H663**. La mémoire a la capacité de stocker 4 clés de chaque type (A ou B).

P2 = $_h$00 à P2 = $_h$03 sont des clés "type A",

P2 = $_h$10 to P2 = $_h$13 sont des clés "type B".

Quand P1 = $_h$20, P2 est l'identifiant de la clé dans la mémoire non-volatile de **H663** (si disponible). Cette mémoire peut stocker plus de 16 clés de chaque type (A ou B).

P2 = $_h$00 à P2 = $_h$0F sont des clés "type A",

P2 = $_h$10 à P2 = $_h$1F sont des clés "type B".

Noter qu'il n'y a pas de moyen de relire les clés stockées dans les mémoires volatiles et non volatiles.

LOAD KEY réponse

SW1	SW2
See below	

LOAD KEY Statut

SW1	SW2	Meaning
$_h$90	$_h$00	Success
$_h$69	$_h$86	Volatile memory is not available
$_h$69	$_h$87	Non-volatile memory is not available
$_h$69	$_h$88	Key index (P2) is not in the allowed range
$_h$69	$_h$89	Key length (Lc) is not valid

3.2.3. Instruction GENERAL AUTHENTICATE

L'instruction **GENERAL AUTHENTICATE** réalise une authentification Mifare Classic (CRYPTO1). L'application doit fournir l'index des clés qui seront utilisées; cette clé doit avoir été chargée dans le **H663** avec une instruction LOAD KEY.

Ne pas invoquer cette fonction si le PICC activé actuellement n'est pas une Mifare Classic !

GENERAL AUTHENTICATE commande APDU

CLA	INS	P1	P2	Lc	Data In	Le
$_h$FF	$_h$86	$_h$00	$_h$00	$_h$05	See below	-

GENERAL AUTHENTICATE Data en bytes

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4
$_h$01	$_h$00	Block number	Key location or Key type	Key index

Le **numéro de bloc** (byte 2) est l'adresse de la carte Mifare, où l'application essaie d'être authentifiée (*noter: c'est le numéro du bloc, pas le numéro du secteur*).

La ***localisation de la clé ou type de clé*** (byte 3) doit être soit :

- $_h$60 pour authentification avec une clé CRYPTO1 "A" *(valeur définie par le standard PC/SC)*,
- $_h$61 pour authentification avec une clé CRYPTO1 "B" key *(valeur définie par le standard PC/SC),*
- Même valeur que le paramètre P1 utilisé dans l'instruction LOAD KEY : $_h$00 ou $_h$20 *(valeur spécifique SpringCard)*.

L'***index de clé*** (byte 4) est définit comme suit:

- Si le *type de clé* (byte 3) est $_h$60, utiliser les valeurs $_h$00 à $_h$03 pour sélectionner l'une des clés "A" stockées dans la mémoire volatile de **H663,** et les valeurs $_h$20 à $_h$2F pour sélectionner l'une des clés "A" stockées dans la mémoire non volatile de **H663 (**si disponible),
- Si le *type de clé* (byte 3) est $_h$61, utiliser les valeurs $_h$00 à $_h$03 pour sélectionner l'une des clés "B" stockées dans la mémoire volatile de **H663,** et les valeurs $_h$20 à $_h$2F pour sélectionner l'une des clés "B" stockées dans la mémoire non-volatile de **H663** (si disponible),
- Si le *type de clé* (byte 3) est $_h$00 ou $_h$20 (même valeurs que pour le paramètre P1 utilisé dans l'instruction LOAD key), choisir l'une des valeurs autorisées pour le paramètre P2 dans la même instruction LOAD KEY *(valeur spécifique SpringCard).*

GENERAL AUTHENTICATE réponse

SW1	SW2
See below	

GENERAL AUTHENTICATE Statut

SW1	SW2	Meaning
$_h$90	$_h$00	Success
$_h$69	$_h$82	CRYPTO1 authentication failed
$_h$69	$_h$86	Key location or type (byte 3) is not valid (or not available for this coupler)
$_h$69	$_h$88	Key index (byte 4) is not in the allowed range

3.2.4. L'instruction READ BINARY

L'instruction **READ BINARY** retrouve les données d'une carte mémoire (à logique câblée PICC ou VICC). Se référer au chapitre 4 pour plus de détails.

Pour tout PICC/VICC sauf Mifare Classic, cette instruction est réalisée sans prérequis.
Pour que Mifare Classic, soit capable de lire les données du secteur, l'application doit être authentifiée dans le secteur de la carte. L'application doit donc invoquer l'instruction GENERAL AUTHENTICATE (avec une clé A ou B valide pour ce secteur) avant d'invoquer l'instruction READ BINARY. Utiliser l'instruction MIFARE CLASSIC READ (§ 3.3.1) peut être plus simple et pourrait raccourcir votre temps de transaction.

READ BINARY commande APDU

CLA	INS	P1	P2	Lc	Data In	Le
$_h$FF	$_h$B0	Address MSB	Address LSB	-	-	XX

P1 et P2 forment l'**addresse** qui sera envoyée au PICC/VICC dans sa commande lecture spécifique. La plupart des PICC/VICC sont divisés en petits blocs (parfois appelés des pages). L'adresse est le numéro du bloc, et non pas un byte absolu contrebalancé dans la mémoire.

La gamme autorisée pour l'**adresse** et la valeur de **Le** dépendent des capacités du PICC/VICC. Merci de toujours vous référer à sa fiche technique pour plus de détails. Noter que Le = $_h$00 doit toujours fonctionner, si l'adresse est valide.

Pour Mifare Classic, P1,P2 est l'adresse du nouveau bloc ($_h$0000 à $_h$00FF), mais souvenez-vous que l'authentification est faite sur une base par secteur. Une nouvelle authentification doit être réalisée chaque fois que vous avez accès à un autre secteur.
Pour les tag NFC Type 2, P2 est le numéro du bloc, et P1 est le numéro du secteur si le PICC supporte cette option. Mettre P1 à $_h$00 si ce n'est pas le cas.

READ BINARY réponse

Data Out	SW1	SW2
XX … XX	See below	

READ BINARY Statut

SW1	SW2	Will return in Data Out
$_h$90	$_h$00	Success
$_h$62	$_h$82	End of data reached before Le bytes (Le is greater than data length)
$_h$69	$_h$81	Command incompatible
$_h$69	$_h$82	Security status not satisfied
$_h$6A	$_h$82	Wrong address (no such block or no such offset in the PICC/VICC)
$_h$6C	XX	Wrong length (Le is shorter than data length, XX in SW2 gives the correct value)

3.2.5. Instruction UPDATE BINARY

L'instruction **UPDATE BINARY** écrit les données dans une carte mémoire (à logique câblée PICC ou VICC). Se référer au chapitre 4 pour plus de détails.

Pour tous les PICC/VICC sauf Mifare Classic, cette instruction est éxécutée sans prérequis.
Pour que Mifare Classic, soit capable de lire les données du secteur, l'application doit être authentifiée dans le secteur de la carte. Votre application doit toujours invoquer l'instruction GENERAL AUTHENTICATE (avec une clé A ou B valide pour ce secteur) avant d'invoquer l'instruction UPDATE BINARY. Utiliser l'instruction MIFARE CLASSIC WRITE (§ 3.3.2) peut être plus facile et peut réduire le temps de transaction.

UPDATE BINARY commande APDU

CLA	INS	P1	P2	Lc	Data In	Le
$_h$FF	$_h$D6	Address MSB	Address LSB	XX	Data	-

P1 et P2 forment l'**adresse** qui sera envoyée au PICC/VICC dans sa commande écrire spécifique. La plupart des PICC/VICC sont divisés en petits blocs (parfois appelés pages). L'adresse est un numéro de bloc, et non pas un byte absolu compensé dans la mémoire.

Les gammes autorisées pour l'**adresse** et la valeur de **Lc** dépendent des capacités du PICC. Merci de toujours vous référer à sa fiche technique pour plus de détails.

Pour Mifare Classic, P1,P2 est l'adresse du bloc ($_h$0000 à $_h$00FF), mais rappelez-vous que l'authentification est faite sur une base par secteur. Une nouvelle authentification doit être réalisée chaque fois que vous avez à accéder à un autre secteur. Lc doit être $_h$10 (un bloc est long de 16-B).
Pour un tag NFC de Type 2, P2 est un numéro de bloc, et P1 est le numéro de secteur si le PICC ne supporte pas cette option. Mettre P1 à $_h$00 si ce n'est pas le cas. Lc doit être $_h$04 (un bloc est long de 4-B).

UPDATE BINARY réponse

SW1	SW2
See below	

UPDATE BINARY statut

SW1	SW2	Will return in Data Out
$_h$90	$_h$00	Success
$_h$69	$_h$82	Security status not satisfied
$_h$6A	$_h$82	Wrong address (no such block or no such offset in the PICC)
$_h$6A	$_h$84	Wrong length (trying to write too much data at once)

Avertissement important

La plupart des PICC/VICC ont des zones spécifiques:
*- qui ne peuvent être écrites **qu'une fois** (OTP: one time programming ou fuse bits),*
*- et/ou doivent être écrit avec **attention** car ils sont impliqués dans le schéma de sécurité de la puce (lock bits),*
- et/ou car écrire une valeur invalide rendra la carte inutilisable (remorque de secteur d'une Mifare Classic par exemple).

Avant d'invoquer UPDATE BINARY, toujours vérifier où vous écrivez, et pour les adresses sensibles, ce que vous écrivez !

3.3. Instructions spécifiques à SpringCard pour les ports sans-contact

3.3.1. Instruction MIFARE CLASSIC READ

L'instruction **MIFARE CLASSIC READ** retrouve les données d'une PICC Mifare Classic (ex. Mifare 1K ou Mifare 4K, ou Mifare Plus en niveau 1).

La différence avec READ BINARY réside dans le schéma d'authentification:

- Avec l'instruction READ BINARY, l'authentification doit être réalisée avant en utilisant l'instruction GENERAL AUTHENTICATE,
- Avec l'instruction MIFARE CLASSIC READ, l'authentification est réalisée automatiquement par le **H663**, qui essaie chaque clé l'une après l'autre, jusqu'à ce que l'une des deux réussisse.

Cette authentification “automatique” fait de l'instruction **MIFARE CLASSIC READ** une aide intéressante pour lire les données Mifare facilement.

Ne pas invoquer cette fonction the le PICC activé actuellement n'est pas un Mifare Classic!

a. MIFARE CLASSIC READ en utilisant les clés coupleurs

Dans ce mode, l'application ne spécifie rien. Le **H663** essaie chaque clé qu'il connaît (les clés permanentes dans E2PROM et les clés temporaires chargées dans la mémoire volatile - utiliser **LOAD KEY** pour le faire) jusqu'à ce que l'une réussisse.

> *Puisque le coupleur doit essayer toutes les clés, cette méthode peut durer jusqu'à 1000ms. L'ordre des clés dans la mémoire du coupleur est très importante pour accélérer le processus : plus la bonne clé est en haut, plus tôt l'authentification sera réussie.*
> *Noter que le coupleur essaie toutes les clés de type “A” en premier, puis ensuite les clé de type “B”. Cette attitude a été choisie car dans 95% des applications Mifare, le type de clé “A” est le type préféré pour la lecture (là où le type de clé “B” est utilisé pour l'écriture).*

MIFARE CLASSIC READ commande APDU

CLA	INS	P1	P2	Lc	Data In	Le
$_h$FF	$_h$F3	$_h$00	Block Number	-	-	XX

Se référer à la commande READ BINARY (§ 3.2.4) pour les réponses et statuts.

b. *MIFARE CLASSIC READ sélectionner une clé dans le coupleur*

Dans ce mode, l'application choisit l'une des clés préalablement chargée dans **H663** grâce à l'instructin **LOAD KEY**.

MIFARE CLASSIC READ commande APDU, sélectionner une clé

CLA	INS	P1	P2	Lc	Data In		Le
hFF	hF3	h00	Block Number	h02	Key Location or Type	Key Index	XX

La compréhension et valeur pour bytes ***Key location ou Key type*** et ***Key index*** sont documentées dans § 3.2.3 (GENERAL AUTHENTICATE instruction).

Se référer à l'instruction READ BINARY (§ 3.2.4) pour les réponses et statuts.

c. *MIFARE CLASSIC READ avec clé spécifiée*

Dans ce mode, l'application fournit une valeur 6-B de la clé à **H663**.

Le coupleur essaie les clés de type "A" en premier et seulement ensuite les clés de type "B".

MIFARE CLASSIC READ commande APDU, avec clé spécifique

CLA	INS	P1	P2	Lc	Data In	Le
hFF	hF3	h00	Block Number	h06	Key value (6 bytes)	XX

Se référer à l'instruction READ BINARY (§ 3.2.4) pour les réponses et les statuts.

3.3.2. Instruction MIFARE CLASSIC WRITE

L'instruction **MIFARE CLASSIC WRITE** écrit les données dans une PICC Mifare Classic (ex Mifare 1K ou Mifare 4K, ou Mifare Plus en niveau 1).

La différence avec UPDATE BINARY réside dans le schéma d'authentification:

- Avec les instructions UPDATE BINARY, l'authentification doit être réalisée avant en utilisant l'instruction GENERAL AUTHENTICATE,
- Avec l'instruction MIFARE CLASSIC WRITE, l'authentification est réalisée automatiquement par **H663**, qui essaie chaque clé l'une après l'autre jusqu'à ce que l'une réussisse.

Cette authentification “automatique” fait de l'instruction MIFARE CLASSIC WRITE une aide intéressante pour écrire les données Mifare facilement.

Ne pas invoquer cette fonction si la PICC activée n'est pas une Mifare Classic!

Avertissement important

Écrire les annonces secteurs (bloc sécurité) est possible si les conditions d'accès au secteur le permettent, mais les annonces secteur Mifare doivent suivre des règles spécifiques formattées (un mélance des bits de conditions d'accès) pour être valides. Sinon le secteur devient inutilisable en permanence.
Avant d'invoquer MIFARE CLASSIC WRITE, vérifiez toujours que vous n'êtes pas entrain d'écrire une annonce secteur, et si vous devez vraiment le faire, vérifiez que le nouveau contenu est formaté comme spécifié dans la fiche technique de la PICC.

a. MIFARE CLASSIC WRITE en utilisant les clés coupleurs

Dans ce mode, l'application ne spécifie rien. Le **H663** essaie toutes les clés qu'il connaît (les clés permanentes dans E2PROM et les clés temporaires préalablement chargées dans la mémoire volatile) jusqu'à ce que l'une réussisse.

Puisque le coupleur doit essayer toutes les clés la méthode peut prendre jusqu'à 1000ms. L'ordre des clés dans la mémoire du coupleur est donc très important pour accèlérer le processus: plus haute sera la bonne clé dans la mémoire du coupleur, plus tôt l'authentification sera réussie.
Noter que le coupleur essaie toutes les clés de type “B” en premier et seulement après les clés de type “A”. Cette attitude a été choisie car dans 95% des applications Mifare le type de clé B est la clé préférée pour l'écriture[11].

[11] Mifare Classic cards issued by NXP are delivered in “transport configuration”, with no “B” key and an “A” key allowed for both reading and writing. This “transport configuration” gives poorest writing performance ; card issuer must start the card personalisation process by enabling a “B” key for writing.

MIFARE CLASSIC WRITE commande APDU

CLA	INS	P1	P2	Lc	Data In	Le
hFF	hF4	h00	Block Number	XX	XX … XX	-

Lc doit être un multiple de 16.

Se référer à l'instruction UPDATE BINARY (§ 3.2.5) pour les réponses et statuts.

b. *MIFARE CLASSIC WRITE sélectionner une clé dans un coupleur*

Dans ce mode, l'application choisit une des clés préalablement chargée dans le **H663** grâce à l'instruction **LOAD KEY**.

MIFARE CLASSIC WRITE commande APDU, sélectionner une clé

CLA	INS	P1	P2	Lc	Data In	Le
hFF	hF4	h00	Block Number	XX	See below	-

MIFARE CLASSIC WRITE commande APDU, sélectionner une clé: Données en bytes

Bytes 0 to Lc-3	Byte Lc-2	Byte Lc-1
Data to be written (multiple of 16 bytes)	Key Location or Type	Key Index

La compréhension et les valeurs pour bytes ***Key location ou Key type*** et ***Key index*** sont documentées dans § 3.2.3 (GENERAL AUTHENTICATE instruction).

Se référer à l'instruction UPDATE BINARY (§ 3.2.5) pour les réponses et les statuts.

c. *MIFARE CLASSIC WRITE avec clé spécifiques*

Dans ce mode, l'application fournit la clé à **H663**.

Le coupeur essaie les clés de type “B” d'abord puis celles de type “A”.

MIFARE CLASSIC WRITE commande APDU, avec clé spécifiée

CLA	INS	P1	P2	Lc	Data In	Le
hFF	hF4	h00	Block Number	XX	See below	-

MIFARE CLASSIC WRITE commande APDU, avec clé spécifiée: Donnée en Bytes

Bytes 0 to Lc-7	Bytes Lc-6 to Lc-1
Data to be written (multiple of 16 bytes)	Key value (6 bytes)

Lc = 6 + 16 x (nombre de bloc à écrire).

Se référer à l'instruction UPDATE BINARY (§ 3.2.5) pour les réponses et les statuts.

3.3.3. Instruction MIFARE CLASSIC VALUE

L'instruction **MIFARE CLASSIC VALUE** permet d'invoquer les fonctions DECREMENT, INCREMENT, et RESTORE de la PICC Mifare Classic PICC (ex Mifare 1K ou Mifare 4K, ou Mifare Plus en niveau 1), suivi par une fonction TRANSFER.

Les fonctions DECREMENT, INCREMENT, RESTORE (et TRANSFER) peuvent être réalisées seulement sur les blocs qui ont été formatés en bloc VALEUR dans l'annonce secteur (bits de condition d'accès). Ne pas invoquer cette fonction sur des blocs DONNÉES, et ne pas invoquer cette fonction si la PICC activée n'est pas une Mifare Classic !

MIFARE CLASSIC VALUE opcode, operand, et adresse de transfert

Le paramètre P1 dans la commande **MIFARE CLASSIC VALUE** APDU dans le code opérationel (*opcode*) PICC, comme définit dans la spécification Mifare Classic. Les valeurs autorisées sont

- $_h$C1 pour INCREMENT
- $_h$C0 pour DECREMENT
- $_h$C2 pour RESTORE

Les trois opérations nécessitent un opérande. L'opérande est un 4-byte signé entier.

- INCREMENT opération: l'opérande doit être > 0 (entre $_h$00000001 et $_h$7FFFFFFF). L'opérande est ajoutée à la valeur actuelle du bloc source, et le résultat est gardé par la PICC dans un registre,
- DECREMENT opération: l'opérande doit être > 0 (entre $_h$00000001 et $_h$7FFFFFFF). L'opérande est soustrait de la valeur actuelle du bloc source, et le résultat est gardé par la PICC dans un registre,
- RESTORE opération: l'opérande doit être 0 ($_h$00000000). La PICC copie la valeur actuelle du bloc source dans un registre.

Après que l'opération INCREMENT, DECREMENT ou RESTORE ai été réalisée par la PICC, le **H663** invoque l'opération TRANSFER: la valeur du registre est écrite dans un bloc cible.

- Si le numéro du bloc de destination n'est pas le même que le numéro du bloc source, la valeur originale reste inchangée dans le bloc source (c'est une sorte d'option “backup”),
- Si le numéro du bloc de destination est le même que celui du bloc source, ou que le bloc de destination n'a pas de numéro défini, alors le bloc source est écrit avec la nouvelle valeur.

a. MIFARE CLASSIC VALUE en utilisant les clés du coupleur

Dans ce mode, l'application ne spécifie rien. Le **H663** essaie toutes les clés qu'il connait (les clés permanentes dans E2PROM et les clés temporaires préalablement chargées dans la mémoire volatile) jusqu'à ce qu'une clé réussisse.

Comme le coupleur doit essayer toutes les clés, cette méthode peut prendre jusqu'à 1000ms. L'ordre des clés dans la mémoire du coupleur est très importante pour accèlérer le processus: plus haut sera la bonne clé dans la mémoire du coupleur plus vite l'authentification sera réussie.
Pour les opérations DECREMENT et RESTORE, le coupleur essaie toutes les clés de type "A" d'abord et seulement après les clés de type "B".
Pour l'opération INCREMENT, le coupleur essaie toutes les clés de type "B" d'abord et seulement après les clés de type "A".

Le bloc de destination peut être spécifié à la fin de la commande APDU. S'il ne l'est pas le bloc source sera écrasé par l'opération TRANSFER.

MIFARE CLASSIC VALUE commande APDU, en utilisant la clé coupleur, sans sauvegarde

CLA	INS	P1	P2	Lc	Data In	Le
$_h$FF	$_h$F5	Opcode	Source block	$_h$04	Operand (4B – MSB first)	-

MIFARE CLASSIC VALUE commande APDU, en utilisant la clé coupleur, avec sauvegarde

CLA	INS	P1	P2	Lc	Data In		Le
$_h$FF	$_h$F5	Opcode	Source block	$_h$05	Operand (4B – MSB first)	Dest. block	-

Se référer à l'instructon UPDATE BINARY (§ 3.2.5) pour les réponses et statuts.

b. MIFARE CLASSIC VALUE sélectionner une clé dans le coupleur

Dans ce mode, l'application choisit l'une des clés préalablement chargées dans **H663** via l'instruction **LOAD KEY**.

Le bloc destination peut être spécifié à la fin de la commande APDU. S'il ne l'est pas, le bloc source sera écrasé par l'opération **TRANSFER**.

MIFARE CLASSIC VALUE commande APDU, sélectionner une clé, sans sauvegarde

CLA	INS	P1	P2	Lc	Data In			Le
$_h$FF	$_h$F5	Opcode	Source block	$_h$06	Operand (4B – MSB first)	Key location or Type	Key index	-

MIFARE CLASSIC VALUE commande APDU, sélectionner une clé, avec sauvegarde

CLA	INS	P1	P2	Lc	Data In				Le
$_h$FF	$_h$F5	Opcode	Source block	$_h$07	Operand (4B – MSB first)	Key location or Type	Key index	Dest. block	-

La compréhension et les valeurs pour for bytes ***Key location ou Key type*** et ***Key index*** sont documentées dans § 3.2.3 (instruction GENERAL AUTHENTICATE).

Se référer à l'instruction UPDATE BINARY (§ 3.2.5) pour les réponses et les statuts.

c. MIFARE CLASSIC VALUE avec clé spécifiée

Dans ce mode, l'application fournit la clé au **H663**.

Pour les opérations DECREMENT et RESTORE, le coupleur essaie les clés de type "A" en premier et seulement après les clés de type "B".
Pour l'opération INCREMENT, le coupleur essaie les clés de type "B" en premier et seulement après celles de type "A".

Le bloc de destination peut être spécifié à la fin de la commande APDU. S'il ne l'est pas le bloc source sera écrasé par l'opération TRANSFER.

MIFARE CLASSIC VALUE commande APDU, clé spécifiée, sans sauvegarde

CLA	INS	P1	P2	Lc	Data In		Le
$_h$FF	$_h$F5	Opcode	Source block	$_h$0A	Operand (4B – MSB first)	Key value (6B)	-

MIFARE CLASSIC VALUE commande APDU, clé spécifiée, avec sauvegarde

CLA	INS	P1	P2	Lc	Data In			Le
$_h$FF	$_h$F5	Opcode	Source block	$_h$0B	Operand (4B – MSB first)	Key value (6B)	Dest. block	-

Se référer à l'instruction UPDATE BINARY (§ 3.2.5) pour les réponses et les statuts.

3.3.4. Instruction RFID MEMORY CONTROL

L'instruction **RFID MEMORY CONTROL** donne accès à certaines fonctions des PICC ou VICC RFID à logique câblée qui n'ont pas d'équivalent dans le monde de la carte à puce.

Par exemple, lire ou écrire d'une carte mémoire RFID à READ BINARY / UPDATE BINARY qui sont des instructions “standards” définies par ISO 7816. Mais ISO 7816 n'a pas d'équivalent pour de nombreuses fonctions définies dans ISO 15693, comme “Write DSFID”, “Lock AFI”, et plein d'autres.

L'instruction **RFID MEMORY CONTROL** est donc une fonction définie par **SpringCard** qui facilites le fonctionnement de ISO 15693 est des VICC liés comme EM4134.

a. Lire un bloc unique

Cette fonction est disponible pour les VICC ISO 15693 et EM4134.

Cette fonction est une alternative bas-niveau à READ BINARY.

RFID MEMORY CONTROL : Lire un bloc unique commande APDU

CLA	INS	P1	P2	Lc	Data In	Le
$_h$FF	$_h$F6	$_h$20	$_h$00	$_h$01	Address (1B)	$_h$00

b. Écrire un bloc unique

Cette fonction est disponible pour les VICC ISO 15693 et EM4134.

Cette fonction est une alternative bas-niveau à UPDATE BINARY.

RFID MEMORY CONTROL : Écrire un bloc unique commande APDU

CLA	INS	P1	P2	Lc	Data In		Le
$_h$FF	$_h$F6	$_h$21	$_h$00	$_h$01 +Len	Address (1B)	Data (Len)	-

c. Bloc Lock

Cette fonction est disponible pour les VICC ISO 15693 et EM4134.

RFID MEMORY CONTROL : Bloc Lock commande APDU

CLA	INS	P1	P2	Lc	Data In	Le
$_h$FF	$_h$F6	$_h$22	$_h$00	$_h$01	Address (1B)	-

d. Lire des blocs multiples

Cette fonction est disponible pour les VICC ISO 15693 seulement.

Cette fonction est une alternative bas-niveau à READ BINARY.

RFID MEMORY CONTROL : Lire des blocs multiples commandes APDU

CLA	INS	P1	P2	Lc	Data In		Le
$_h$FF	$_h$F6	$_h$23	$_h$00	$_h$02	Address (1B)	Count (1B)	$_h$00

e. Écrire des blocs multiples

Cette fonction est disponible pour les VICC ISO 15693 seulement.

Cette fonction est une alternative bas-niveau à UPDATE BINARY.

RFID MEMORY CONTROL : Écrire des blocs multiples commande APDU

CLA	INS	P1	P2	Lc	Data In			Le
$_h$FF	$_h$F6	$_h$24	$_h$00	$_h$02 +Len	Address (1B)	Count (1B)	Data (Len)	-

f. Écrire AFI

Cette fonction est disponible pour les VICC ISO 15693 seulement.

RFID MEMORY CONTROL : Écrire AFI commande APDU

CLA	INS	P1	P2	Lc	Data In	Le
$_h$FF	$_h$F6	$_h$27	$_h$00	$_h$01	AFI (1B)	-

g. Lock AFI

Cette fonction est disponible pour les VICC ISO 15693 seulement.

RFID MEMORY CONTROL : Lock AFI commande APDU

CLA	INS	P1	P2	Lc	Le
$_h$FF	$_h$F6	$_h$28	$_h$00	$_h$00	-

h. Écrire DSFID

Cette fonction est disponible pour les VICC ISO 15693 seulement.

RFID MEMORY CONTROL : Write DSFID command APDU

CLA	INS	P1	P2	Lc	Data In	Le
$_h$FF	$_h$F6	$_h$29	$_h$00	$_h$01	DSFID (1B)	-

i. Lock DSFID

Cette fonction est disponible pour les VICC ISO 15693 seulement.

RFID MEMORY CONTROL : Lock DSFID commande APDU

CLA	INS	P1	P2	Lc	Le
$_h$FF	$_h$F6	$_h$2A	$_h$00	$_h$00	-

j. Connaître les informations système

Cette fonction est disponible pour les VICC ISO 15693 seulement.

RFID MEMORY CONTROL : Connaître les informations système commande APDU

CLA	INS	P1	P2	Lc	Le
$_h$FF	$_h$F6	$_h$2A	$_h$00	-	$_h$00

Note: le **H663** envoie toujours la commande *connaître les informations système* au VICC, dans le cadre du processus de découverte. Invoquer l'instruction GET DATA (§ 3.2.1) pour retrouver la valeur déjà renvoyée par le VICC au **H663**.

k. Connaître la sécurité des blocs multiples

Cette fonction est disponible pour les VICC ISO 15693 seulement.

RFID MEMORY CONTROL : Connaître la sécurité des blocs multiples commande APDU

CLA	INS	P1	P2	Lc	Data In		Le
$_h$FF	$_h$F6	$_h$24	$_h$00	$_h$02	Address (1B)	Count (1B)	-

3.3.5. Instruction CONTACTLESS SLOT CONTROL

L'instruction **CONTACTLESS SLOT CONTROL** permet de mettre en pause et de résumer les mécanismes de tracking de carte du **port sans-contact.**

C'est utile par le tracking de la carte implique envoyer des commandes au PICC périodiquement (et surveiller sa réponse). Ces commandes peuvent avoir des effets secondaires non-désirés, comme casser l'atomicité entre une paire de commandes. Mettre le mécanisme de tracking de carte sur OFF pendant la transaction résoudra ce problème.

SLOT CONTROL commande APDU

CLA	INS	P1	P2	Lc	Data In	Le
hFF	hFB	See below	See below	-	-	-

SLOT CONTROL paramètres de commande

P1	P2	Action	
h00	h00	Resume the card tracking mechanism	
h01	h00	Suspend the card tracking mechanism	
h10	h00	Stop the RF field	
h10	h01	Start the RF field	
h10	h02	Reset the RF field (10ms pause)	
h20	h00	T=CL de-activation (DESELECT[12])	
h20	h01	T=CL activation of ISO 14443-A card (RATS)	
h20	h02	T=CL activation of ISO 14443-B card (Attrib)	
h20	h04	Disable the next T=CL activation[13]	
h20	h05	Disable every T=CL activation (until reset of the H663)	
h20	h06	Enable T=CL activation again	
h20	h07	Disable the next T=CL activation and force a RF reset	
hFC	xx	Felica runtime parameters, see § 3.3.6 below	
hDE	hAD	Stop the slot NOTE: a stopped slot is not available to *SCardConnect* any more. It may be restarted only through an *SCardControl* command.	

[12] Or DISC for Innovatron cards. This makes it possible to operate ISO 14443-4 compliant cards at ISO 14443-3 level. No CARD INSERTED event is triggered, so the ATR of the card stays unchanged.

[13] Upon DISCONNECT, the CARD REMOVED event fires, then the CARD INSERTED event. A new ATR is computed, and reflects that the card runs at ISO 14443-3 level.

SLOT CONTROL réponse

Data Out	SW1	SW2
-	See below	

SLOT CONTROL Statut

SW1	SW2	Meaning
$_h$90	$_h$00	Success

3.3.6. Intruction SET FELICA RUNTIME PARAMETERS

Travailler avec les cartes Felica (Lite) cards ou les Tags NFC Type 3 implique 4 paramètres:

- Le *SYSTEM CODE* est envoyé par le **H663** pendant la boucle d'attente active JIS:X6319-4 (SENSF_REQ) pour préciser quelle famille de cartes peut répondre. La valeur $_h$FFFF permet à n'importe quelle carte de répondre,
- Le *REQUEST CODE* est envoyée par le **H663** pendant la boucle d'attente active JIS:X6319-4 (SENSF_REQ) pour obtenir les données techniques des cartes, et pas seulement leurs IDm/PPm. La valeur $_h$00 empêche la carte d'envoyer ses données techniques,
- Un premier *SERVICE CODE* est un paramètre obligatoire utilisé pendant les opérations de lecture (READ BINARY instruction) pour dire à la carte quel “service” est consulté. La valeur $_h$000B a été assignée par le NFC Forum pour donner (lire) accès aux dossiers des tags NDEF Type 3,
- Un autre *SERVICE CODE* est un paramètre obligatoire utilisé pendant les opérations d'écriture (UPDATE BINARY instruction) pour dire à la carte quel “service” est consulté. La valeur $_h$0009 a été assignée par le NFC Forum pour donner accès à l'écriture dans les dossiers des Tags NDEF de Type 3.

Les valeurs soulignées dans le paragraphe précédent sont les valeurs par défaut de **H663**. Elles peuvent être mises à jour en permanence grâce à la commande *WRITE REGISTER* (§ 6.3.6) appliquée aux registres de configurations $_h$B4 (§ 7.5.2) et $_h$CF (§ 7.6.1).

Ces valeurs peuvent être également changées dynamiquement en utilisant une simple commande APDU dans le flux *ScardTransmit,* comme décrit dans les paragraphes précédents.

a. SERVICE CODE pour l'instruction READ BINARY

SET FELICA SERVICE READ commande APDU

CLA	INS	P1	P2	Lc	Data In	Le
$_h$FF	$_h$FB	$_h$FC	$_h$01	$_h$02	Service Code to be used by the	-

					READ BINARY instruction (2 bytes, MSB first)	

b. SERVICE CODE pour l'instruction UPDATE BINARY

SET FELICA SERVICE WRITE commande APDU

CLA	INS	P1	P2	Lc	Data In	Le
$_h$FF	$_h$FB	$_h$FC	$_h$02	$_h$02	Service Code to be used by the UPDATE BINARY instruction (2 bytes, MSB first)	-

c. SERVICE CODE pour les instructions READ BINARY et UPDATE BINARY

SET FELICA SERVICES commande APDU

CLA	INS	P1	P2	Lc	Data In	Le
$_h$FF	$_h$FB	$_h$FC	$_h$03	$_h$02	Service Code to be used both by the READ BINARY and UPDATE BINARY instructions (2 bytes, MSB first)	-

d. SYSTEM CODE et REQUEST code pour l'attente active Felica

SET FELICA SYSTEM CODE commande APDU

CLA	INS	P1	P2	Lc	Data In	Le
$_h$FF	$_h$FB	$_h$FC	$_h$10	$_h$02	System Code to be used during JIS:X6319-4 polling (SC in SENS_REQ) (2 bytes, MSB first)	-

SET FELICA REQUEST CODE commande APDU

CLA	INS	P1	P2	Lc	Data In	Le
$_h$FF	$_h$FB	$_h$FC	$_h$11	$_h$01	Request Code to be used during JIS:X6319-4 polling (RC in SENS_REQ) (1 byte)	-

3.3.7. ENCAPSULATE instruction pour le port sans-contact

L'instruction **ENCAPSULATE** a été créée pour aider les applications à communiquer avec les PICC/VICC qui ne sont pas compatibles avec ISO 7816-4.

ENCAPSULATE commande APDU pour le port sans-contact

CLA	INS	P1	P2	Lc	Data In	Le
$_h$FF	$_h$FE	See below	See below	XX	Frame to send to the PICC/VICC	XX

ENCAPSULATE commande paramètre P1 pour le port sans-contact

P1	Standard communication protocols
$_h$00	For ISO 14443-4 (A or B) PICCs : send the frame in the T=CL stream[14]. Data In shall not include PCB, CID, NAD nor CRC fields For ISO 18092 targets : send the frame DEP_REQ/DEP_RES stream. Data In shall not include PFB, DID, NAD nor CRC fields
$_h$01	Send the frame “as is” using the ISO 14443-3 A protocol @ 106 kbit/s. The standard parity bits are added (and checked in return) by the H663. The standard CRC is added (and checked in return) by the H663.
$_h$02	Send the frame “as is” using the ISO 14443-3 B protocol @ 106 kbit/s. The standard CRC is added (and checked in return) by the H663.
$_h$03	Send the frame “as is” using the JIS:X6319-4 protocol @ 212 kbit/s. The standard CRC is added (and checked in return) by the H663.
$_h$04	Send the frame “as is” using the ISO 15693 protocol. The standard CRC is added (and checked in return) by the H663.
$_h$05	Send the frame “as is” using the ISO 15693 protocol. The UID of the VICC is added to the frame *(unselected access mode)*. The standard CRC is added (and checked in return) by the H663.
$_h$07	Send the frame “as is” using the JIS:X6319-4 protocol @ 424 kbit/s. The standard CRC is added (and checked in return) by the H663.

…/…

[14] This is the only way to send commands to a T=CL PICC that doesn't comply with the ISO 7816-4 APDU formatting, for instance a Desfire 0.4.

P1	Non-standard communication
$_h$09	Send the frame “as is” using the ISO 14443-3 A modulation @ 106 kbit/s. The standard parity bits are added (and checked in return) by the H663, but the CRC is not added (and not checked) by the H663 → the application must append the CRC to Data In and check it in Data Out.
$_h$0A	Send the frame “as is” using the ISO 14443-3 B modulation @ 106 kbit/s. The CRC is not added (and not checked) by the H663 → the application must append the CRC to Data In and check it in Data Out.
$_h$0C	Send the frame “as is” using the ISO 15693 modulation. The CRC is not added (and not checked) by the H663 → the application must append the CRC to Data In and check it in Data Out.
P1	Mifare low level communication[15]
$_h$0F	Send the frame “as is” using the ISO 14443-3 A modulation. The CRC is not added (and not checked) by the H663 → the application must append the CRC to Data In and check it in Data Out. The parity bits are not added (and not checked) by the H663 → the application must provide a valid stream, including the parity bits). The last byte is complete (8 bits will be sent)
$_h$1F	Same as $_h$0F, but only 1 bit of the last byte will be sent
$_h$2F	Same as $_h$0F, but only 2 bits of the last byte will be sent
$_h$3F	Same as $_h$0F, but only 3 bits of the last byte will be sent
$_h$4F	Same as $_h$0F, but only 4 bits of the last byte will be sent
$_h$5F	Same as $_h$0F, but only 5 bits of the last byte will be sent
$_h$6F	Same as $_h$0F, but only 6 bits of the last byte will be sent
$_h$7F	Same as $_h$0F, but only 7 bits of the last byte will be sent

[15] The above values allow an application to transmit “ciphered” Mifare frames (the CRYPTO1 stream cipher makes a non-standard use of the parity bits and CRC). The number of valid bits in the last byte of card’s answer will be reported in SW2.

P1	Redirection to another slot[16]
$_h$80	Redirection to the main contact slot (if present)
$_h$81	Redirection to the 1st SIM/SAM slot (if present)
$_h$82	Redirection to the 2nd SIM/SAM slot (if present)
$_h$83	Redirection to the 3rd SIM/SAM slot (if present)
$_h$84	Redirection to the 4th SIM/SAM slot (if present)

ENCAPSULATE commande paramètre P2 pour le port sans-contact

P2 encode la frame délai.

P2	Timeout value
$_h$-0	If P1 = $_h$00, use the default time-out defined by the PICC or the target (T=CL: card's FWT) If P1 ≠ $_h$00, this value shall not be used
$_h$-1	Timeout = 106 ETU ≈ 1ms
$_h$-2	Timeout = 212 ETU ≈ 2ms
$_h$-3	Timeout = 424 ETU ≈ 4ms
$_h$-4	Timeout = 848 ETU ≈ 8ms
$_h$-5	Timeout = 1696 ETU ≈ 16ms
$_h$-6	Timeout = 3392 ETU ≈ 32ms
$_h$-7	Timeout = 6784 ETU ≈ 65ms
$_h$-8	Timeout = 13568 ETU ≈ 0,125s
$_h$-9	Timeout = 27136 ETU ≈ 0,250s
$_h$-A	Timeout = 54272 ETU ≈ 0,500s
$_h$-B	Timeout = 108544 ETU ≈ 1s
$_h$-C	Timeout = 217088 ETU ≈ 2s
$_h$-D	Timeout = 434176 ETU ≈ 4s
$_h$0-	Set status word = $_h$6F XX, XX being the contactless specific error
$_h$8-	Set status word = $_h$63 00 on any contactless specific error

[16] Those values allow an application to transmit APDUs to a SAM or an auxiliary card through the PC/SC handle of the main card.

ENCAPSULATE réponse pour le port sans-contact

Data Out	SW1	SW2
Frame received from the PICC/VICC	See below	

ENCAPSULATE Statut pour le port sans-contact

SW1	SW2	Meaning
$_h$90	$_h$00	Success – last byte of Data Out has 8 valid bits
$_h$90	$_h$01	Success – last byte of Data Out has 1 valid bits
$_h$90	$_h$02	Success – last byte of Data Out has 2 valid bits
$_h$90	$_h$03	Success – last byte of Data Out has 3 valid bits
$_h$90	$_h$04	Success – last byte of Data Out has 4 valid bits
$_h$90	$_h$05	Success – last byte of Data Out has 5 valid bits
$_h$90	$_h$06	Success – last byte of Data Out has 6 valid bits
$_h$90	$_h$07	Success – last byte of Data Out has 7 valid bits
$_h$6F	XX	Error reported by the contactless interface (only allowed if high-order bit of P2 is 0). See chapter 8 for the list of possible values and their meaning.
$_h$63	$_h$00	Error reported by the contactless interface (when high-order bit of P2 is 1).
$_h$62	$_h$82	Le is greater than actual response from PICC/VICC
$_h$6C	XX	Le is shorter than actual response from PICC/VICC

3.3.8. ENCAPSULATE instruction pour l'un des ports Contact

L'instruction **ENCAPSULATE** a été créée pour aider les applications à communiquer avec les PICC/VICC qui ne sont pas compatibles avec ISO 7816-4.

ENCAPSULATE commande APDU pour un port contact

CLA	INS	P1	P2	Lc	Data In	Le
$_h$FF	$_h$FE	$_h$00	$_h$00	XX	Frame to send to the card	XX

ENCAPSULATE réponse pour un port contact

Data Out	SW1	SW2
Frame received from the card	See below	

ENCAPSULATE statut pour un port contact

SW1	SW2	Meaning
$_h$90	$_h$00	Success
$_h$6F	XX	Error reported by the contactless interface (only allowed if high-order bit of P2 is 0). See chapter 8 for the list of possible values and their meaning.
$_h$62	$_h$82	Le is greater than actual response from card
$_h$6C	XX	Le is shorter than actual response from card

3.4. Autres instructions spécifiques à SpringCard

3.4.1. Instruction READER CONTROL

L'instruction **READER CONTROL** permet de gérer l'attitude globale du **H663** (LEDs, buzzer, etc. selon les caractéristiques physiques du produit).

Pour les opérations ou si vous souhaitez intéragir avec le **H663** même s'il n'y a pas de carte insérée, utiliser *SCardControl* à la place (voir chapitre 6).

Si votre coupleur est multi-port (sans-contact + contact ou SAM), l'instruction READER CONTROL instruction est envoyée à un port (un coupleur logique), mais il aura sans doute un impact global sur l'intégralité du coupleur physique.
En d'autres termes, envoyer une instruction READER CONTROL à une chaîne de carte pourra avoir un impact sur une autre chaîne de carte.
Il est hautement recommandé d'utiliser un objet de synchronisation dans vos applications (mutex, section critique,…) pour éviter un accès concurrent au même coupleur physique lorsque l'instruction READER CONTROL est appellée.

READER CONTROL commande APDU

CLA	INS	P1	P2	Lc	Data In	Le
$_h$FF	$_h$F0	$_h$00	$_h$00	See below	See below	See below

a. Diriger les LED du coupleur

Pour un coupleur avec des LEDs rouges et vertes uniquement, envoyer l'APDU:

```
FF F0 00 00 03 1E <red> <green>
```

Pour u coupleur avec des LEDs rouges, vertes et jaunes/bleues, envoyer l'APDU:

```
FF F0 00 00 04 1E <red> <green> <yellow/blue>
```

Choisir les valeurs pour rouge, vert et jaune/bleu dans ce tableau:

$_h$00	LED is switched OFF
$_h$01	LED is switched ON
$_h$02	LED blinks slowly
$_h$03	LED is driven automatically by the H663's firmware *(default behaviour)*
$_h$04	LED blinks quickly
$_h$05	LED performs the "heart-beat" sequence

Pour retourner aux paramètres par défaut (LED gérées par le firmware **H663** automatiquement), envoyer l'APDU:

```
FF F0 00 00 01 1E
```

b. Gérer le buzzer du coupleur

Certains hardware disposent d'un beeper à tonalité simple. Pour démarrer le buzzer, envoyer l'APDU:

```
FF F0 00 00 03 1C <duration MSB> <duration LSB>
```

où *duration* précise la longueur de la tonalité en millisecondes (maximum 60000ms).

Mettre duration à 0000 si vous avez besoin d'arrêter le buzzer avant que la durée n'ai commencé dans un précédent appel.

Pour retourner aux paramètres par défaut (buzzer géré par le firmware **H663** automatiquement), envoyer l'APDU:

```
FF F0 00 00 01 1C
```

c. Autres

Le bloc donnée de l'instruction **READER CONTROL** est transmise tel quel à l'interprète du **reader control,** comme documenté dans le chapitre 6.

Chaque commande documentée dans § 6.3 et commençant par le code $_h$58 peut être transmit en lien *SCardTransmit* en utilisant l'instruction **READER CONTROL,** exactement comme s'il était transmit dans un lien *SCardControl*.

Ne pas utiliser cette option sauf si vous savez exactement ce que vous faites.

3.4.2. Instruction TEST

L'instruction **TEST** a été créée pour tester le driver et/ou les applications, avec une longueur arbitraire de données (entrée et sortie).

TEST commande APDU

CLA	INS	P1	P2	Lc	Data In	Le
$_h$FF	$_h$FD	See below	See below	XX	XX … XX	XX

TEST commande paramètres

Paramètres P1 précisent la longueur des données sorties que l'applicaton veut recevoir de **H663**:

$_h$00: Sortie de données vides,seul SW est renvoyé

$_h$FF: 255 bytes de données + SW

Toutes les valeurs entre $_h$00 et $_h$FF sont autorisées

6 bits d'ordre bas de P2 précisent le délai entre la commande et la réponse.

$_h$00: aucun délai, les réponses viennent immédiatement

$_h$3F: 63 secondes entre la comande et la réponse

Toutes les valeurs entre 0 et 63 sont autorisées

2 bits d'ordre ahut de P2 sont des RFU et doivent être mises à 0.

TEST réponses

Data Out	SW1	SW2
XX … XX	See below	

Le contenu des sorties de données n'est pas précisé, et doivent contenir des données fixées ou “au hasard”, selon la version **H663** et le statut actuel.

TEST statut

Si les 2 bits de haut-ordre de P2 sont 0, l'interprète APDU embarqué analyse le format de l'APDU et renvoient le statut approprié. Si l'un des ces bits est 1, le statut est fixé qu'importe le format de l'APDU.

SW1	SW2	Meaning
$_h$90	$_h$00	Success, APDU correctly formatted
$_h$67	$_h$00	APDU is badly formatted (total length incoherent with Lc value)
$_h$6A	$_h$82	Le is greater than data length specified in P1
$_h$6C	P1	Le is shorter than data length specified in P1

4. Travailler avec les cartes sans-contact – astuces utiles

4.1. Reconnaître et identifier des PICC/VICC dans un environnement PC/SC

4.1.1. ATR d'une carte à puce compatible avec ISO 14443-4

Si la PICC est avec le 14443 au niveau 4 ("**T=CL**"), le **H663** construit un pseudo-ATR en utilisant le format standard définit dans les spécifications PC/SC:

a. Pour ISO 14443-A:

Offset	Name	Value	Meaning (according to 7816-3)
0	TS	$_h$3B	Direct convention
1	T0	$_h$8…	Higher nibble 8 means: no TA1, no TB1, no TC1. TD1 to follow Lower nibble is the number of historical bytes (0 to 15)
2	TD1	$_h$80	Higher nibble 8 means: no TA2, no TB2, no TC2. TD2 to follow Lower nibble 0 means: protocol T=0
3	TD2	$_h$01	Higher nibble 8 means: no TA3, no TB3, no TC3, no TD3 Lower nibble 1 means: protocol T=1
4	H1	…	Historical bytes from ATS response
…	…		
3+k	Hk		
4+k	TCK	XX	Checksum (XOR of bytes 1 to 3+k)

b. Pour ISO 14443-B:

Offset	Name	Value	Meaning (according to 7816-3)
0	TS	$_h$3B	Direct convention
1	T0	$_h$88	Higher nibble 8 means: no TA1, no TB1, no TC1. TD1 to follow Lower nibble is the number of historical bytes (8)
2	TD1	$_h$80	Higher nibble 8 means: no TA2, no TB2, no TC2. TD2 to follow Lower nibble 0 means: protocol T=0
3	TD2	$_h$01	Higher nibble 8 means: no TA3, no TB3, no TC3, no TD3 Lower nibble 1 means: protocol T=1
4	H1	…	Application data from ATQB
5	H2		
6	H3		
7	H4		

8	H5	…	Protocol info byte from ATQB
9	H6		
10	H7		
11	H8	XX	MBLI from ATTRIB command
12	TCK	XX	Checksum (XOR of bytes 1 to 11)

c. Pour Innovatron (héritage des cartes Calypso)[17]:

Offset	Name	Value	Meaning (according to 7816-3)
0	TS	$_h$3B	Direct convention
1	T0	$_h$8…	Higher nibble 8 means: no TA1, no TB1, no TC1. TD1 to follow Lower nibble is the number of historical bytes (0 to 15)
2	TD1	$_h$80	Higher nibble 8 means: no TA2, no TB2, no TC2. TD2 to follow Lower nibble 0 means: protocol T=0
3	TD2	$_h$01	Higher nibble 8 means: no TA3, no TB3, no TC3, no TD3 Lower nibble 1 means: protocol T=1
4	H1	…	Historical bytes from REPGEN. This is the last part of the card's T=0 ATR, including its serial number[18].
…	…		
3+k	Hk		
4+k	TCK	XX	Checksum (XOR of bytes 1 to 3+k)

La plupart des cartes Calypso sont capables de communiquer selon la norme ISO 14443-B ou le protocole Innovatron. Le choix entre les deux protocoles est imprévisible.
La même carte aura deux ATR différents (un si ISO 14443-B est sélectionné, l'autre si le protocole Innovatron est sélectionné). L'application hôte doit connaître et vérifier le numéro de série de la carte[19] pour être sûr qu'il ne commencera pas une nouvelle transaction sur la même carte que plus tôt.

[17] When bit 7 of register $_h$B3 is 0. Otherwise, the "real" card ATR (found in REPGEN) is returned. This ATR reports that the card supports T=0 only, but the card behaves as it were T=1. This behaviour is not compliant with Microsoft's CCID driver.

[18] As a consequence, all the cards have a different ATR.

[19] Provided in the historical bytes of the ATR when the Innovatron protocol is selected, or available through the Calypso "Select Application" command.

4.1.2. ATR des PICC/VICC à logique câblée

Pour les cartes sans-contact et les tags RFID (Mifare, CTS, etc.), le **H663** construit un pseudo-ATR en utilisant le format normalisé décrit dans la spécification PC/SC:

Offset	Name	Value	
0	TS	h3B	Direct convention
1	T0	h8F	Higher nibble 8 means: no TA1, no TB1, no TC1. TD1 to follow Lower nibble is the number of historical bytes (15)
2	TD1	h80	Higher nibble 8 means: no TA2, no TB2, no TC2. TD2 to follow Lower nibble 0 means: protocol T=0
3	TD2	h01	Higher nibble 8 means: no TA3, no TB3, no TC3, no TD3 Lower nibble 1 means: protocol T=1
4	H1	h80	
5	H2	h4F	Application identifier presence indicator
6	H3	h0C	Length to follow (12 bytes)
7	H4	hA0	Registered Application Provider Identifier **A0 00 00 03 06** is for PC/SC workgroup
8	H5	h00	
9	H6	h00	
10	H7	h03	
11	H8	h06	
12	H9	PIX.SS	Protocol (see 4.1.4)
13	H10	PIX.NN	Card name (see 4.1.5)
14	H11		
15	H12	00	RFU
16	H13	00	
17	H14	00	
18	H15	00	
19	TCK	XX	Checksum (XOR of bytes 1 to 18)

4.1.3. Utiliser l'instruction GET DATA

Avec l'instruction **GET DATA** (documentée dans § 3.2.1), l'application hôte est capable de retrouve chaque information nécessaire pour identifier une PICC:

- Numéro de série (UID ou PUPI),
- Valeurs liées au protocole (ATQA et SAKA ou ATQB, …).

4.1.4. Protocole sans-contact

Le byte **standard** (**PIX.SS** dans la spécification PC/SC) est construite comme suit:

b7	b6	b5	b4	b3	b2	b1	b0	Value	Description
0	0	0	0	0	0	0	0	$_h$00	No information given
0	0	0	0	0	0	0	1	$_h$01	ISO 14443 A, level 1
0	0	0	0	0	0	1	0	$_h$02	ISO 14443 A, level 2
0	0	0	0	0	0	1	1	$_h$03	ISO 14443 A, level 3 or 4 (and Mifare) ISO 18092 @ 106 kbit/s “NFC-A”
0	0	0	0	0	1	0	1	$_h$05	ISO 14443 B, level 1
0	0	0	0	0	1	1	0	$_h$06	ISO 14443 B, level 2
0	0	0	0	0	1	1	1	$_h$07	ISO 14443 B, level 3 or 4
0	0	0	0	1	0	0	1	$_h$09	ICODE 1, EM4134
0	0	0	0	1	0	1	1	$_h$0B	ISO 15693
0	0	0	1	0	0	0	1	$_h$11	JIS:X6319-4 Felica cards ISO 18092 @ 212 or 424 kbit/s “NFC-F”

Note: PIX.SS est définit pour les cartes mémoires et basées sur les microprocesseurs, mais il est disponible dans l'ATR des cartes mémoire seulement. Dans l'autre cas, utiliser l'instruction GET DATA (avec les paramètres P1,P2=$_h$F1,00) pour connaître le protocole sous-jacent utilisé par la carte à puce.

4.1.5. Bytes nom des cartes sans-contact

Les bytes **nom** (**PIX.NN** dans la spécification PC/SC) sont précisées comme suit:

NN	Card name	From FW
	Values specified by PC/SC	
$_h$00 $_h$01	NXP Mifare Classic 1k	
$_h$00 $_h$02	NXP Mifare Classic 4k	
$_h$00 $_h$03	NXP Mifare UltraLight NFC Forum Type 2 Tag with a capacity <= 64 bytes	
$_h$00 $_h$06	ST Micro Electronics SR176	
$_h$00 $_h$07	ST Micro Electronics SRI4K, SRIX4K, SRIX512, SRI512, SRT512	1.70
$_h$00 $_h$0A	Atmel AT88SC0808CRF	
$_h$00 $_h$0B	Atmel AT88SC1616CRF	
$_h$00 $_h$0C	Atmel AT88SC3216CRF	
$_h$00 $_h$0D	Atmel AT88SC6416CRF	
$_h$00 $_h$12	Texas Instruments TAG IT	
$_h$00 $_h$13	ST Micro Electronics LRI512	
$_h$00 $_h$14	NXP ICODE SLI	
$_h$00 $_h$16	*Not available in this product (NXP ICODE1)*	
$_h$00 $_h$21	ST Micro Electronics LRI64	
$_h$00 $_h$24	ST Micro Electronics LR12	
$_h$00 $_h$25	ST Micro Electronics LRI128	
$_h$00 $_h$26	NXP Mifare Mini	
$_h$00 $_h$2F	Innovision/Broadcom Jewel	
$_h$00 $_h$30	Innovision/Broadcom Topaz NFC Forum Type 1 Tag	
$_h$00 $_h$34	Atmel AT88RF04C	
$_h$00 $_h$35	NXP ICODE SL2	
$_h$00 $_h$36	NXP Mifare Plus 2K SL1	1.81
$_h$00 $_h$37	NXP Mifare Plus 4K SL1	1.81
$_h$00 $_h$38	NXP Mifare Plus 2K SL2	1.81
$_h$00 $_h$39	NXP Mifare Plus 4K SL2	1.81
$_h$00 $_h$3A	NXP Mifare UltraLight C, NXP NTAG203 NFC Forum Type 2 Tag with a capacity > 64 bytes	
$_h$00 $_h$3A	Felica NFC Forum Type 3 Tag	

NN	Card name	
SpringCard proprietary extension[20]		
$_h$FF $_h$A0	Generic/unknown 14443-A card	
$_h$FF $_h$A1	ThinFilm	
$_h$FF $_h$B0	Generic/unknown 14443-B card	
$_h$FF $_h$B1	*Not available in this product (ASK CTS 256B)*	
$_h$FF $_h$B2	*Not available in this product (ASK CTS 512B)*	
$_h$FF $_h$B3	Pre-standard ST Micro Electronics SRI 4K	
$_h$FF $_h$B4	Pre-standard ST Micro Electronics SRI X512	
$_h$FF $_h$B5	Pre-standard ST Micro Electronics SRI 512	
$_h$FF $_h$B6	Pre-standard ST Micro Electronics SRT 512	
$_h$FF $_h$B7	Inside Contactless PICOTAG/PICOPASS	
$_h$FF $_h$B8	Generic Atmel AT88SC / AT88RF card	
$_h$FF $_h$C0	Calypso card using the Innovatron protocol	
$_h$FF $_h$D0	Generic ISO 15693 from unknown manufacturer	
$_h$FF $_h$D1	Generic ISO 15693 from EM Marin (or Legic)	
$_h$FF $_h$D2	Generic ISO 15693 from ST Micro Electronics, block number on 8 bits	
$_h$FF $_h$D3	Generic ISO 15693 from ST Micro Electronics, block number on 16 bits	
$_h$FF $_h$D5	Generic ISO 15693 from Infineon	
$_h$FF $_h$D6	EM MicroElectronic Marin EM4134 chip	1.81
$_h$FF $_h$FF	Virtual card (test only)	

Note: PIX.NN est précisé pour les cartes mémoires seulement. Même si l'instruction **GET DATA** permet de retrouver PIX.NN même pour les cartes basées sur les micro-processeurs (cartes à puce), la valeur renvoyée n'est pas précisée et ne doit pas être utilisée pour identifier la carte.

[20] The cards in this list are not referenced by PC/SC specification at the date of writing. In case they are added to the specification, the future firmware versions will have to use the new value. It is therefore advised **not to check those values** in the applications, as they are likely to be removed in the future. Set bit 6 of configuration register $_h$B3 (§ 7.4.3) to force PIX.NN = $_h$00 $_h$00 instead of using those proprietary values.

4.2. ISO 14443-4 PICC

4.2.1. Desfire première version (0.4)

Puisque cette PICC n'est pas compatible ISO 7816-4, les commandes Desfire doivent être enveloppées dans une instruction ENCAPSULATED, avec P1=$_h$00 (§ 3.3.7). Le **H663** traduit le C-APDU en commande Desfire native, retrouve la réponse Desfire native et la traduit en R-APDU valide.

4.2.2. Desfire EV0 (0.6) et EV1

Cette PICC est compatible ISO 7816-4. Les commandes natives sont enveloppées dans les APDU ISO 7816-4 avec une CLA spécifique à la cartec CLA = $_h$90. Merci de vous référer à la fiche technique de la carte pour plus de détails.

4.2.3. Cartes Calypso

Une carte Calypso est compatible ISO 7816-4. Vous travaillez peut-être avec une carte Calypso sans-contact comme si elle était insérée dans un coupleur de carte à puce à contact.

4.3. PICC à logique câblée basées sur ISO 14443-A

4.3.1. Mifare Classic

Les PICC couvertes par ce chapitre sont:

- Mifare 1k (NXP MF1ICS50, **PIX.NN = $_h$0001**),
- Mifare 4k (NXP MF1ICS70, **PIX.NN = $_h$0002**),
- Mifare Mini (NXP MF1ICS20, **PIX.NN = $_h$0026**),
- Mifare Plus (X or S) when used in level 1 (see § 4.3.2).

Merci de télécharger les fiches techniques de ces cartes sur www.nxp.com. Les informations utiles sont disponibles sur www.mifare.net.

Toutes ces PICC sont divisées en blocs de 16-byte. Les blocs sont groupés en secteurs. A la fin de chaque secteur un bloc spécifique (“annonce secteur”) est réservé pour les paramètres de sécurité (clés d'accès et conditions d'accès).

Faire fonctionner une PICC multi-standards comme une Mifare Classic

Certaines cartes à puce ou objets NFC compatibles avec ISO 14443-4 sont capables d'émuler les cartes Mifare Classic, mais à cause de leur compatibilité avec ISO 14443-4 (T=CL), le **H663** cachera leur **mode émulation** Mifare et les fera apparaître comme des cartes à puce de pointe.

Il y a 3 manières de forcer le **H663** à rester au niveau Mifare:

- Envoyer une commande T=CL DESELECT au PICC (instruction SLOT CONTROL avec P1,P2=$_h$20,00),
- Réinitialiser le champ RF et désactiver temporairement l'activation T=CL (instruction SLOT CONTROL avec P1,P2=$_h$10,03),
- Désactiver de manière permanente l'activation T=CL activation grâce au registre de configuration $_h$B3.

a. Instruction READ BINARY

Dans la READ BINARY commande APDU,

- P1 doit être $_h$00,
- P2 est l'adresse du premier bloc à lire (0 à 63 pour un Mifare 1k, 0à 255 pour un Mifare 4k),

Comme la taille de chaque bloc est 16, **Le** doit être un multiple de 16,

- Quand Le=$_h$00 est l'adresse est alignée sur un secteur limité, tous les blocs de données du secteur sont renvoyés (48 ou 240 bytes),

- Quand Le=$_h$00 et l'adresse n'est pas alignée, un bloc seul est renvoyé (16 bytes).

Noter que quand une annonce secteur (bloc sécurité) est lue, les clés ne sont pas lisibles (elles sont masquées par le PICC).

L'instruction **READ BINARY** ne peut pas traverser les limites; l'instruction GENERAL AUTHENTICATE doit être appelée pour chaque secteur immédiatement avant le READ BINARY.

Utiliser les instructions MIFARE CLASSIC READ (§ 3.3.5) est plus facile et pourra réduire le temps de transaction.

b. Instruction UPDATE BINARY

Dans UPDATE BINARY commande APDU,

- P1 doit être $_h$00,
- P2 est l'adresse du premier bloc à écrire (1 à 63 pour un Mifare 1k, 1 à 255 pour un Mifare 4k),

Comme la taille de chaque bloc est 16, **Lc** doit être un multiple de 16 (48 bytes pour les secteurs standards, 240 bytes pour les plus grands secteurs dans Mifare 4k).

L'instruction UPDATE BINARY ne peut pas traverser les limites du secteur; l'instruction GENERAL AUTHENTICATE doit être appelé pour chaque secteur immédiatement avant UPDATE BINARY.

Avertissement important

Écrire les annonces secteur (bloc sécurité) est possible si les conditions d'accès actuelles du secteur le permettent, mais les annonces secteur Mifare doivent suivre une règle de formatage spécifique (mélange des bits de conditions d'accès) pour être valide. Sinon le secteur devient inutilisable. Avant d'invoquer MIFARE CLASSIC WRITE, toujours vérifier que vous n'écrivez pas dans l'annonce secteur. Si vous devez vraiment le faire, vérifier que le nouveau contenu est formatté comme spécifié dans la fiche technique du PICC.

Utiliser les instructions MIFARE CLASSIC WRITE (§ 3.3.2) est plus facile et peut réduire le temps de transaction.

c. Instructions spécifiques pour les Mifare Classic

3 instructions spécifiques existent pour travailler avec les PICC Mifare Classic:

- MIFARE CLASSIC READ, voir § 3.3.1,
- MIFARE CLASSIC WRITE, voir § 3.3.2,
- MIFARE CLASSIC VALUE (qui implémentent INCREMENT, DECREMENT et RESTORE suivi par TRANSFER), voir § 3.3.3.

4.3.2. Mifare Plus X et Mifare Plus S

Merci de télécharger les fiches techniques des cartes sur www.nxp.com.

La **Mifare Plus** implémentent 4 niveaux de sécurité différents. L'attitude de la carte change drastiquement selon le niveau de sécurité sélectionné.

> ***SpringCard*** *a développé la librairie logicielle PCSC_MIFPLUS (disponible comme code source et DLL pré-compilé dans le SDK) pour aider à travailler avec les cartes* ***Mifare Plus*** *sans déscendre au niveau APDU et sans avoir besoin d'implémenter le schéma de sécurité vous-même.*
> *Pour la documentation de cette API, aller à*
> *http://www.springcard.com/support/apidoc/pcsc_mifplus/index.html*

a. Niveau 0

Au niveau 0, le PICC est compatible avec ISO 14443-4 (T=CL). Le **H663** construit un ATR carte à puce selon § 4.1.1. Les bytes historiques de l'ATS sont inclus dans l'ATR et aident à reconnaître la carte à ce niveau.

Comme le PICC n'est pas compatible avec ISO 7816-4, les commandes doivent être envoyées enveloppées dans une instruction ENCAPSULATED avec P1=$_h$00 (§ 3.3.7).

A la fin de ce processus de personnalisation, le champ RF doit être réinitialisé (pour que la PICC redémarre au niveau 1 ou plus). Envoyer l'instruction SLOT CONTROL avec P1,P2=$_h$10,02 pour ce faire (§ 3.3.5)[21].

b. Niveau 1

Au niveau 1, le PICC émule une Mifare Classic (§ 4.3.1). Le **H663** construit une carte mémoire ATR suivant § 4.1.1.

L'application doit utiliser les instructions MIFARE CLASSIC READ et MIFARE CLASSIC WRITE pour travailler avec la carte à ce niveau.

Le PICC supporte la nouvelle fonction authentification AES. Utiliser l'instruction ENCAPSULATE avec P1=$_h$01 (§ 3.3.7) pour implémenter cette fonction.

Pour augmenter le niveau de sécurité de la carte (allant du niveau 2 au niveau 3), une session ISO 14443-4 (T=CL) doit être commencée manuellement, même si la PICC annonce que ce n'est pas compatible T=CL. Envoyer l'instruction SLOT CONTROL avec P1,P2=$_h$20,01 pour ce faire (§ 3.3.5). Ensuite, procéder comme indiqué pour le niveau 0.

c. Niveau 2

Le niveau 2 n'est pas disponible sur les Mifare Plus S.

[21] As a consequence, the card with be reported as REMOVED, then a new CARD INSERT event will be triggered (but with a different ATR as the security level is different).

Travailler avec les **Mifare Plus X** à ce niveau est possible grâce aux appels d'instruction bas-niveau (SLOT CONTROL, ENCAPSULATE) mais il n'est pas implémenté dans le **H663** (et non supporté par notre librairie logicielle).

d. Niveau 3

Au niveau 3, le PICC est compatible ISO 14443-4 (T=CL). Le **H663** construit une carte à puce ATR suivant § 4.1.1. Ces bytes historiques de l'ATS sont inclus dans l'ATR et aident à reconnaître la carte à ce niveau.

Puisque la carte n'est pas compatible ISO 7816-4, les commandes doivent être envoyées enveloppées dans l'instruction ENCAPSULATED, avec P1=$_h$00 (§ 3.3.7).

4.3.3. NFC Forum Type 2 Tags – Mifare UltraLight et UltraLight C, NTAG203...

Les cartes couvertes dans ce chapitre sont:

- Mifare UL – NXP MF01CU1 (**PIX.NN = $_h$0003**),
- Mifare UL C – NXP MF01CU2 (**PIX.NN = $_h$003A**),
- Les PICC compatibles avec la spécification du NFC Forum sur les tags de type 2.

Merci de télécharger les fiches techniques des cartes sur www.nxp.com.

Merci de visiter www.nfcforum.org pour connaître les spécifications des tags de type 2.

Toutes ces cartes sont divisées en *pages* 4-bytes. Il n'est possible d'écrire qu'une page à la fois mais la lecture est généralement faite 4 pages par 4 pages (16bytes). Un tag NFC Forum de type 2 peut également être divisé en secteurs de 256 pages (1024bytes).

Il n'est pas possible de découvrir la capacité actuelle d'une PICC compatible au niveau protocolaire.
Si le PICC n'est pas déjà formaté selon la spécification du NFC Forum pour les tags de Type 2, la capacité est stockée parmis d'autres données dans la première page OTP (CC – bytes conteneur de capacité).
Dans d'autres cas, l'application pourra trouver le nombre de pages en envoyant l'instruction READ BINARY, incrémentant l'adresse jusqu'à ce qu'il échoue.
Faire attention car certains PICC n'échouent pas mais tronquent l'adresse; par exemple une PICC avec seulement 16 pages (0 à 15) peut renvoyer le contenu des pages 0, 1, 2 et 3 lorsque l'adresse 16 est lue. Comme les pages 0 et 1 sont stockées dans l'UID (le numéro de série) de la PICC, comparer les pages 16, 17 aux pages 0, 1 pour voir si la fin de l'espace mémoire a été atteinte.

a. *Instructions READ BINARY*

Dans le **READ BINARY** commande APDU,

- P1 est le numéro de secteur. Il doit être $_h$00 pour PICCs qui ont seulement un secteur,
- P2 est l'adresse de la première page. Merci de vous référer à la fiche technique de la puce pour connaître le nombre de pages qui peuvent être adressées.

Comme la taille des pages est de 4bytes, **Le** doit être un multiple de 4. Quand Le=$_h$00, 4 pages sont renvoyées (16 bytes).

Il est possible de lire les données complètes d'une Mifare UL en un seul appel en mettant Le à $_h$40 (64 bytes). Pour les Mifare UL C, le même résultat est atteint en mettant Le à $_h$90 (144 bytes).

b. Instruction UPDATE BINARY

Dans UPDATE BINARY commande APDU,

- P1 est le numéro du secteur. Il doit être $_h$00 pour les PICC qui n'ont qu'un seul secteur,
- P2 est l'adresse de la (seule) page pouvant être écrite. Merci de vous référer à la fiche technique de la puce pour connaître le nombre de pages qui peuvent être adressées.

Puisque la taille de la page est 4 bytes, **Lc** doit être 4, exactement.

Certaines pages peuvent contenir
- OTP (one-time-programming) bits,
- et/ou les bits lock qui sont désigné pour faire la lecture de la mémoire PICC seule.

Ne pas écrire sur ces pages sans une bonne compréhension des conséquences.

c. Mifare UltraLight C 3-DES authentification

La Mifare UltraLight C supporte une authentification à 3 passes Triple-DES.

Utilise l'instruction ENCAPSULATE avec P1=$_h$01 (§ 3.3.7) pour implémenter cette fonction.

***SpringCard** a développé la librairie logicielle PCSC_MIFULC (disponible comme code source et comme DLL pré-compilé dans le SDK) pour aider à travailler avec les cartes Mifare UltraLight C sans avoir besoin d'implémenter le schéma de sécurité vous-mêmes.*
Pour la documentation de cette API, aller sur
http://www.springcard.com/support/apidoc/pcsc_mifulc/index.html

4.3.4. NFC Forum Type 1 Tags – Puces Innovision/Broadcom

Firmware ≥ 1.75

Les PICC couvertes par ce chapitre sont:

- Innovision/Broadcom Topaz (**PIX.NN = $_h$002F**),
- Innovision/Broadcom Jewel (**PIX.NN = $_h$0030**),
- Toute PICC compatible avec la spécification NFC Forum des tags de type 1.

Merci de visiter www.nfcforum.org pour voir la spécification des tags de Type 1.

a. Structures de mémoires

Il y a deux groupes de PICC dans cette spécification:

- PICC avec une **structure de mémoire statique** qui fournit 120 bytes de données. Elles supportent seulement les fonctions RALL, READ, WRITE-E et WRITE-NE.
- PICC avec une **structure de mémoire dynamique** qui fournit plus de 120 bytes de données. Elles sont divisées en *blocs* de 8 bytes. Un segment est un groupe de 16 blocs (128 bytes de données). De nouvelles fonctions sont fournies pour adresser les *blocs* et *segments*: READ8, RSEG, WRITE-E8 et WRITE-NE8.

Ces PICCs ont 2 bytes d'information hardware appelés HR0 et HR1.

- HR0 = $_h$11 signifie une structure de mémoire statique,
- HR0 = $_h$1*y*, où $y \neq 1$, signifie une structure de mémoire dynamique,
- Les autres valeurs pour HR0 sont RFU, HR1 est ignoré.

Avant de lire/écrire les données PICC, l'application doit aller chercher HR0 pour savoir si le PICC a une structure de mémoire statique ou dynamique. Pour faire cela, l'application peut:

- Invoquer l'instruction READ BINARY, précisant s'il veut utiliser la fonction RALL de la PICC et attend 122 bytes de données (FF B0 00 00 7A). HR0 est le premier byte dans la réponse.
- Invoquer l'instruction GET DATA, précisant s'il veut connaître l'identification complète de la PICC (FF CA F0 00 00). HR0 est le premier byte dans la réponse.

b. Instruction READ BINARY

L_E		P1	P2	PICC function	Description
Both Static and Dynamic Structures					
$_h$00 $_h$78	0 120	$_h$00	$_h$00	RALL	The coupler returns the 120 bytes of data returned by the PICC in response to RALL. The HR0 and HR1 bytes are dropped.
$_h$7A	122	$_h$00	$_h$00	RALL	The coupler returns the complete frame returned by the PICC in response to RALL, i.e. HR0 and HR1 followed by 120 bytes of data.
$_h$01	1	$_h$00, $_h$00 to $_h$00, $_h$7F		READ	P2 specify the byte address within the card from 0 to 127. One byte is returned.
Dynamic Memory Structure only					
$_h$80	128	$_h$00, $_h$00 $_h$00, $_h$80 $_h$01, $_h$00 ...		RSEG	P1, P2 specify the byte address within the card. A complete segment (128 bytes of data) is returned. Therefore, P1, P2 must be aligned to a segment boundary *(≡ 0 mod 128)*.
$_h$08	8	$_h$00, $_h$00 $_h$00, $_h$08 $_h$00, $_h$10 ...		READ8	P1, P2 specify the byte address within the card. A complete block (8 bytes of data) is returned. Therefore, P1, P2 must be aligned to a block boundary *(≡ 0 mod 8)*.

Utiliser les fonctions RALL ou RSEG est bien plus rapide qu'utiliser READ/READ8 dans une boucle.

c. Instruction UPDATE BINARY

L_C		P1	P2	PICC function	Description
Both Static and Dynamic Structures					
$_h$01	1	$_h$00, $_h$00 to $_h$00, $_h$7F		WRITE-E	The coupler writes 1 byte of data into the Tag. P2 specify the byte address (from 0 to 127)
$_h$01	1	$_h$80, $_h$00 to $_h$80, $_h$7F		WRITE-NE	The coupler updates 1 byte of data to the Tag. The actual operation is a XOR between the current content of the card and the specified value. P2 specify the byte address (from 0 to 127)
Dynamic Memory Structure only					
$_h$01	1	$_h$00, $_h$00 $_h$00, $_h$08 $_h$00, $_h$10 ...		WRITE-E8	The coupler writes 8 byte of data into the Tag. P1, P2 specify the byte address within the card. Therefore, P1, P2 must be aligned to a block boundary *(≡ 0 mod 8)*.
$_h$01	1	$_h$80, $_h$00 $_h$80, $_h$08 $_h$80, $_h$10 ...		WRITE-NE8	The coupler updates 8 bytes of data to the Tag. The actual operation is a XOR between the current content of the card and the specified value. $P1_{0..6}$, P2 specify the byte address within the card. Therefore, $P1_{0..6}$, P2 must be aligned to a block boundary *(≡ 0 mod 8)*.

Certains blocs contiennent des bits OTP (one-time-programming), et/ou des bits lock qui sont destinés à faire la lecture de la mémoire PICC uniquement. N'écrivez pas dans ces bytes sans une bonne compréhension des conséquences.

4.4. PICC à logique câblée basées sur ISO 14443-B

4.4.1. ST Micro Electronics SR176

Ces PICC sont identifiés par **PIX.NN = $_h$0006**.

Elles sont divisées en *blocs* de 2-byte.

a. Instruction READ BINARY

Dans READ BINARY commande APDU,

- P1 doit être $_h$00,
- P2 est l'adresse du premier bloc a être lu (0 à 15),

Comme la taille de chaque bloc est 2, Le doit être un multiple de 2 (jusqu'à 32 bytes),

Quand Le=$_h$00, un bloc unique est renvoyé (2 bytes).

b. Instruction UPDATE BINARY

Dans le UPDATE BINARY commande APDU,

- P1 doit être $_h$00,
- P2 est l'adresse du bloc à écrire,

Comme la taille de chaque bloc est 2, Lc doit être 2, exactement.

Certains blocs jouent un rôle particulier dans la configuration de la PICC. N'écrivez pas dans ces blocs sans une bonne compréhension des conséquences.

4.4.2. ST Micro Electronics SRI4K, SRIX4K, SRI512, SRX512, SRT512

Ces PICCs sont identifiées par **PIX.NN = $_h$0007**.

Elles sont divisées en *blocs* de 4-byte.

a. Instruction READ BINARY

Dans READ BINARY commande APDU,

- P1 doit être $_h$00,
- P2 est l'adresse du premier bloc à être lu,

Comme la taille de chaque bloc est 2, Le doit être un multiple de 4,

Quand Le=$_h$00, un bloc unique est renvoyé (4 bytes).

b. Instruction UPDATE BINARY

Dans UPDATE BINARY commande APDU,

- P1 doit être $_h$00,
- P2 est l'adresse du bloc à écrire,

Comme la taille de chaque bloc est 4, Lc doit être 4, exactement.

Certains blocs jouent un rôle particulier dans la configuration des PICC. N'écrivez pas dans ces blocs sans une bonne compréhension des conséquences.

4.4.3. A l'intérieur du PicoPass sans-contact, mode ISO 14443-2

Cette partie s'applique aux puces nommées “PicoPass ou PicoTag” lorsque la compatibilité ISO 14443-3 n'est PAS activée sur cette carte (voir § 4.4.4 dans les autres cas).

Ces PICC existent en deux tailles (2K → 256 B, 16K → 2 kB), et en versions non-sécurisée (2K, 16K) ou sécurisée (2KS, 16KS). Elles sont divisées en blocs de 8 byte.

Elles sont actuellement identifiées par **PIX.NN = $_h$FFB7** et **PIX.SS = $_h$06** (ISO 14443-B niveau 2). Faites attention car cela pourrait changer dans les versions futures car PC/SC a enregistré de nouveaux PIX.NN pour ces PICCs.

Le **H663** peut lire/écrire dans les puces non-sécurisées uniquement (2K, 16K). L'attitude avec les puces sécurisées est indéfinie.

a. Instruction READ BINARY

Dans READ BINARY commande APDU,

- P1 doit être $_h$00,
- P2 est l'adresse du premier bloc a être lu (2K: 0 à 31; 16K: 0 à 255),

Comme la taille de chaque bloc est 8 , Le doit être un multiple de 8,

Quand Le=$_h$00, un bloc unique est renvoyé (8 bytes).

b. Instruction UPDATE BINARY

Dans UPDATE BINARY commande APDU,

- P1 doit être $_h$00,
- P2 est l'adresse du bloc à écrire (2K: 0 à 31; 16K: 0 à 255),

Comme la taille de chaque bloc est 8, Lc doit être 8, exactement.

Certains blocs jouent un rôle particulier dans la configuration des PICC. N'écrivez pas dans ces blocs sans une bonne compréhension des conséquences.

c. Sélectioner une Page

La fonciton intérieur d'une page spécifique n'est pas implémentée dans **H663**. Utiliser l'instruction ENCAPSULATE pour l'envoyer directement à la PICC.

4.4.4. A l'intérieur du PicoPass sans-contact, mode ISO 14443-3

Cette partie s'applique aux puces nommées “PicoPass ou PicoTag” lorsque la compatibilité ISO 14443-3 EST activée sur la carte (voir § 4.4.3 dans les autres cas).

Ces PICCs existent en deux tailles (2K → 256 B, 16K → 2 kB), et en versions non-sécurisée (2K, 16K) ou sécurisée (2KS, 16KS). Elles sont divisées en blocs de 8-byte.

Elles sont identifiées par **PIX.NN = $_h$FFB7** et **PIX.SS = $_h$07** (ISO 14443-B niveau 3 ou 4). Faites attention cela peut changer dans les versions futures car PC/SC a enregistré de nouveaux PIX.NN pour ces PICCs.

Le **H663** peut lire/écrire des puces non-sécurisées uniquement (2K, 16K). L'attitude avec les puces sécurisées est indéfinie.

a. Instruction READ BINARY

Dans READ BINARY commande APDU,

- P1 doit être $_h$00,
- P2 est l'adresse du premier bloc a être lu (2K: 0 à 31; 16K: 0 à 255),

Comme la taille de chaque bloc est 8, Le doit être un multiple de 8,

Quand Le=$_h$00, un bloc unique est renvoyé (8 bytes).

b. Instruction UPDATE BINARY

Dans UPDATE BINARY commande APDU,

- P1 doit être $_h$00,
- P2 est l'adresse du bloc à écrire (2K: 0 à 31; 16K: 0 à 255),

Comme la taille de chaque bloc est 8, Lc doit être 8, exactement.

Certains blocs jouent un rôle particulier dans la configuration des PICC. N'écrivez pas dans ces blocs sans une bonne compréhension des conséquences.

4.4.5. Atmel CryptoRF

Les PICC couvertes par ce chapitre sont:

- AT88SC0808CRF (**PIX.NN = $_h$000A**),
- AT88SC1616CRF (**PIX.NN = $_h$000B**),
- AT88SC3216CRF (**PIX.NN = $_h$000C**),
- AT88SC6416CRF (**PIX.NN = $_h$000D**),
- AT88SCRF04C (**PIX.NN = $_h$0034**).

Le **H663** implémente les fonctions lire et écrire en mode non-authentifié. Les fonctions avancées et la communication authentifiée doit être implémentée par l'application dans l'instruction ENCAPSULATE.

Le coupleur active toujours cette PICC avec CID=$_h$01. Utiliser ce CID pour construire la commande actuelle qui sera envoyée via l'instruction ENCAPSULATE.

a. Instruction READ BINARY

Dans READ BINARY commande APDU,

P1,P2 est la première adresse à être lue,

Le est la longueur qui sera lue (1 à 32 bytes).

Note: l'instruction READ BINARY cartographie les commandes bas-niveau de “la zone du système lecture”. La commande “zone du système lecture” n'est pas implémentée dans le **H663**, et doit donc être encapsulée.

b. Instruction UPDATE BINARY

Dans UPDATE BINARY commande APDU,

P1,P2 est la première adresse à écrire,

Lc est la longueur à écrire (1 à 32 bytes).

Note: l'instruction UPDATE BINARY cartographie les commandes de bas-niveau de la “zone d'écriture utilisateur”. La commande “zone d'écriture système” n'est pas implémentée dans le **H663**, et doit donc être encapsulée.

4.5. VICC ISO 15693

4.5.1. Commandes lire/écrire ISO 15693-3

La taille des blocs dépend de la puce. Les tailles connues sont

- 1 byte pour ST Micro Electronics LRI64 (**PIX.NN = $_h$0021**),
- 4 bytespour NXP ICODE-SLI (**PIX.NN = $_h$0014**) et les puces Texas Instrument TagIT (**PIX.NN = $_h$0012**) et autres puces ST Micro Electronics,
- 8 bytes pour les puces EM Marin (**PIX.NN = $_h$FFD1**).

Merci de lire la documentation de la VICC sur laquelle vous travaillez pour connaître la taille de ses blocs et le nombre de blocs existants.

Certaines VICC disposent de blocs spéciaux appelés OTP (one-time-programming) et WORM (write one, read many) qui ne peuvent pas être écrasés ou effacés après une première opération d'écriture. N'écrivez pas dans ces blocs sans une bonne compréhension des conséquences.

a. Instruction READ BINARY

Dans READ BINARY commande APDU,

- P1 doit être $_h$00,
- P2 est l'adresse du premier bloc à lire ; merci de lire la documentation de votre VICC pour connaître ses numéros de blocs,

Le doit être un multiple de la taille des blocs,

Si Le=$_h$00, un bloc unique est renvoyé (la longueur dépend de la VICC).

Note: ISO 15693 définit 2 fonctions pour lire les dates: READ SINGLE BLOCK et READ MULTIPLE BLOCKS. L'instruction du coupleur READ BINARY essaie les deux jusqu'à ce que l'un des deux réussisse.

b. Instruction UPDATE BINARY

Dans UPDATE BINARY commande APDU,

- P1 doit être $_h$00,
- P2 est l'adresse du bloc à écrire, merci de lire la documentation de votre VICC pour connaître son nombre de blocs,

Lc doit être de la taille du bloc, exactement.

Note: ISO 15693 définit 2 fonctions pour lire les dates: WRITE SINGLE BLOCK et WRITE MULTIPLE BLOCKS. L'instruction du coupleur UPDATE BINARY essaie les deux jusqu'à ce que l'un des deux réussisse.

4.5.2. Commandes Lire/écrire pour les puces ST Micro Electronics avec adresse 2-B bloc

Le M24LR16E de ST Micro Electronics (**PIX.NN = $_h$FFD3**) implémente une version étendue des commandes ISO 15693, où les adresses sont sur 2 bytes au lieu d'une.

Procéder comme avec les autres puces ISO 15693 avec cette différence: dans les instructions READ BINARY et UPDATE BINARY, P1 est le byte de haut ordre de l'adresse et peut être différent de zéro.

4.5.3. Complèter l'ensemble de commandes ISO 15693

Le standard ISO 15693 définit de nombreuses commandes avec ou sans pavillon 'option', et les fabricants de puce libres d'implémenter virtuellement n'importe quelle commande propriétaire ou personnalisée.

Commançant avec le firmware version 1.81, les commandes basiques, dans leur implémentation basique, sont disponibles via l'**instruction RFID MEMORY CONTROL** (§ 3.3.4), mais il demeure impossible d'implémenter toutes les commandes et toutes les variations dans un lecteur.

L'**instruction ENCAPSULATE** (INS = $_h$FE, voir § 3.3.7) pour ISO 15693 a donc été introduite, cette instruction permet d'envoyer une commande arbitraire à une puce 15693.

Comme le **H663** fait fonctionner une puce ISO 15693 en mode adressé (la VICC n'est jamais mise en mode *silencieux*), l'UID de la puce doit être fournit dans chaque frame de commande. L'insertion de l'UID est réalisée automatiquement par l'instruction ENCAPSULATE lorsque le paramètre P1 est réglé à $_h$05.

L'APDU doit être construit comme suit:

CLA	INS	P1	P2	Lc	Data In			Le
$_h$FF	$_h$FE	$_h$05	$_h$00	XX	Command flags	Command code	Command data (optional)	$_h$00

Note: Le peut être omis.

Valuers autorisées pour le byte 'commande pavillons'

Bit		Value	Description
7	RFU	0	
6	Option	0/1	Meaning is defined by the command description. Please refer to the ISO 15693:3 standard and/or to the datasheet of the VICC for details
5	Address	1	The UID of the VICC is included in the command frame
4	Select	0	Not using the VICC quiet state
3	Protocol extension	0/1	Must be 0 for standard commands Some VICC may implement vendor-specific commands that require to have this bit set to 1
2	Inventory	0	It is not allowed to invoke the INVENTORY command through an ENCAPSULATE APDU
1	Data rate	1	High data rate shall be used
0	Sub carrier	0	A single sub-carrier shall be used

En résumé, les valeurs typiques du byte 'commandes pavillon' sont:

- $_h$22 lorsque l'option pavillon n'est pas créée
- $_h$62 lorsque l'option pavillon est demandée par la PICC ou la commande

4.5.4. Implémentation des commandes basiques ISO 15693

Commencant avec le firmware version 1.81, les commandes ci-dessous sont disponibles via l'instruction RFID MEMORY CONTROL (§ 3.3.4)

a. Lire bloc unique

ISO 15693 code commande: $_h$20

L'APDU est

```
FF FE 05 00 03 22 20 <block number>
```

b. Écrire un bloc unique

ISO 15693 code commande: $_h$21

L'APDU est

```
FF FE 05 00 <3 + data length > 22 21 <block number> <...data...>
```

La longueur de la donnée doit correspondre à la taille du bloc. Merci de vous référer à la documentation de la VICC pour connaître la taille de son bloc.

c. Bloc Lock

ISO 15693 code commande: $_h$22

L'APDU est

```
FF FE 05 00 03 22 22 <block number>
```

Verrouiller un bloc fait qu'on ne peut plus que le lire. Cette opération ne peut pas être annulée. Ne réalisez pas cette opération sans une bonne compréhension des conséquences.

d. Écrire AFI

ISO 15693 code commande: $_h$27

L'APDU est

```
FF FE 05 00 03 22 27 <new AFI>
```

e. Lock AFI

ISO 15693 code commande: $_h$28

L'APDU est

```
FF FE 05 00 02 22 28
```

Verrouiller l'AFI ne peut pas être annulé. Ne réalisez pas cette opération sans une bonne compréhension des conséquences.

f. Écrire DSFID

ISO 15693 code commande: $_h$29

L'APDU est

```
FF FE 05 00 03 22 29 <new DSFID>
```

g. Lock DSFID

ISO 15693 code commande: $_h$2A

L'APDU est

```
FF FE 05 00 02 22 2A
```

Verrouiller le DSFID ne peut pas être annulé. Ne réalisez pas cette opération sans une bonne compréhension des conséquences.

h. Connaître les informations système

ISO 15693 code commande: $_h$2B

L'APDU est

```
FF FE 05 00 02 22 2B
```

Note: le **H663** envoie toujours la commande *connaître les informations système* au VICC, dans le cadre du processus de découverte; Invoquer l'instruction GET DATA (§ 3.2.1) pour retrouver la valeur déjà renvoyée par le VICC au **H663**.

4.6. Les autres PICC non-ISO

4.6.1. NFC Forum Type 3 Tags / Felica

Les PICC couverts par ce chapitre sont:

- Felica Lite, Felica Lite-S (**PIX.NN = $_h$003B**),
- Toutes PICC compatible avec la spécification du NFC Forum pour les tags de type 3.

Merci de visiter www.nfcforum.org pour connaître les spécifications des tag de type 3.

a. Instruction READ BINARY

Dans READ BINARY commande APDU,

- P1 doit être $_h$00,
- P2 est l'adresse du premier bloc à lire.

Comme la taille du bloc est 16 bytes, **Le** doit être un multiple de 16 ($_h$10). Quand Le=$_h$00, un bloc unique est renvoyé (16 bytes).

Il est possible de lire jusqu'à 8 blocs en même temps.

L'instruction READ BINARY est traduite en commande Felica “CHECK”, en utilisant le *SERVICE CODE pour la valeur READ BINARY* comme le paramètre “Service Code” pour la commande. La valeur par défaut pour ce paramètre est $_h$000B. Voir § 3.3.6 si vous avez besoin de changer de valeur.

b. Instruction UPDATE BINARY (byte unique)

Dans UPDATE BINARY commande APDU,

- P1 doit être $_h$00,
- P2 est l'adresse du bloc (unique) à écrire.

Comme la taille du bloc est 16 bytes, **Lc** doit être 16 ($_h$10), exactement.

L'instruction UPDATE BINARY est traduite en commande Felica “UPDATE”, en utilisant le *SERVICE CODE pour la valeur UPDATE BINARY* comme le paramètre “Service Code” pour la commande. La valeur par défaut pur ce paramètre est $_h$0009. Voir § 3.3.6 si vous avez besoin de changer la valeur.

4.7. Les autres VICC non-ISO

4.7.1. EM4134

Ces VICC utilise la modulation bit ISO 15693, sauf pour le format de frame spécifique au vendeur et l'ensemble des commandes. Elles sont reconnues par **PIX.NN = hFF D6**. Elles sont divisées en 16 mots, chaque mot étant de 32 bit (4 byte).

a. Instruction READ BINARY

Dans READ BINARY commande APDU,

- P1 doit être $_h$00,
- P2 est l'adresse du premier mot à lire (0 à 15).

Comme la taille du mot est de 4 bytes, **Le** doit être un multiple de 4 ($_h$04). Si Le=$_h$00, un seul mot est renvoyé (4 bytes).

Il est possible de lire le contenu complet de la carte en une fois (16 mots).

b. Instruction UPDATE BINARY

Dans UPDATE BINARY commande APDU,

- P1 doit être $_h$00,
- P2 est l'adresse du mot à écrire.

Comme la taille d'un mot est 4 bytes, **Lc** doit être 4 ($_h$04), exactement.

c. Verrouiller

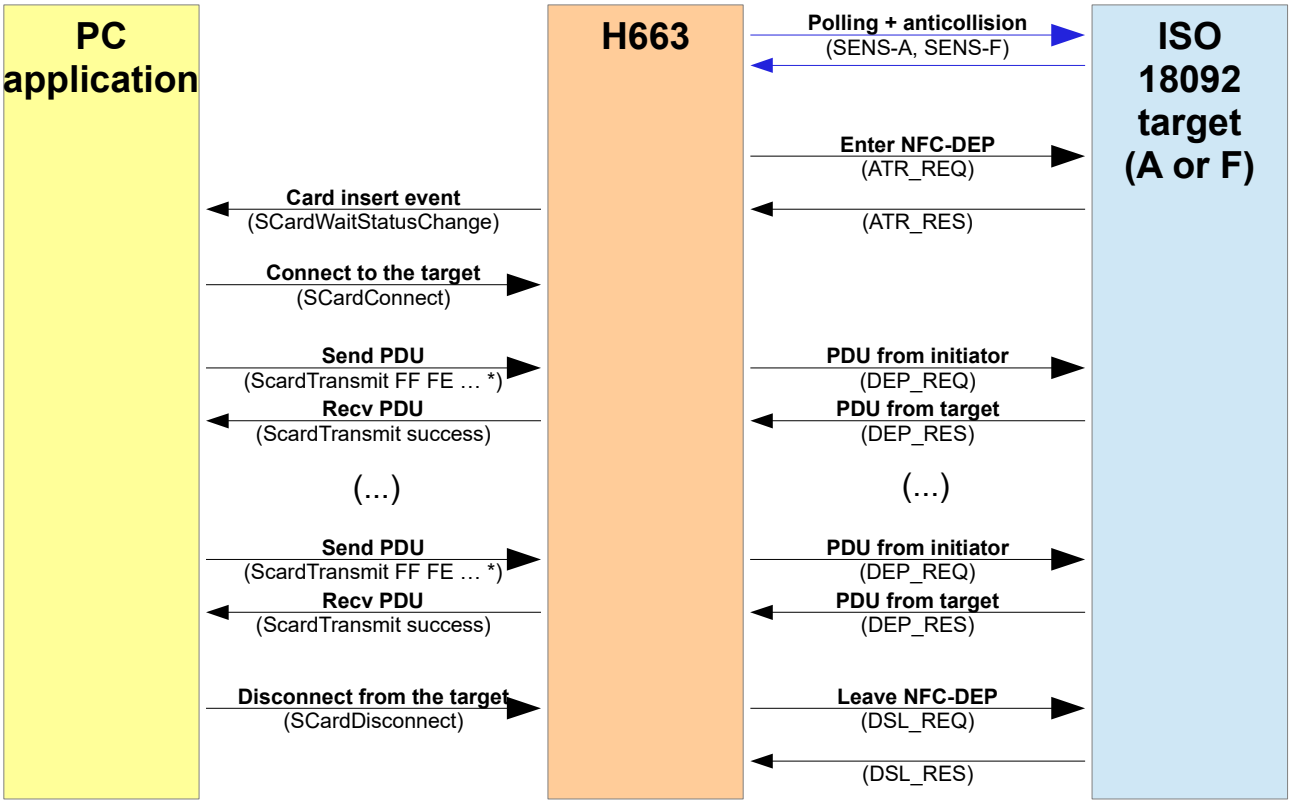
Verrouiller un mot est implémenté via l'**instruction RFID MEMORY CONTROL**, en utilisant le code de la fonction **Lock Block** (§ 3.3.4.c).

5. Utiliser H663 avec une cible NFCIP-1

5.1. Introduction

Le **H663** est un initiateur NFC. Il peut activer un cible NFC à distance (seul le schéma de communication passive est disponible).

Le **H663** implémente le protocole de transport ISO 18092 “NFCIP-1”, également appelé NFC-DEP par le NFC Forum.



* The PDU must be ENCAPSULATEd if it doesn't meet ISO 7816-4 constraints.

5.1.1. Fonctions réalisées par le coupleur

Le **H663** gère le protocole de transport NFC en interne:

- Transmission de *ATR_REQ* lorsqu'une cible NFC potentielle est détectée, manutention de *ATR_RES*,
- Échange initial de paramètres si besoin (*PSL_RES* / *PSL_RES*),
- Fragmentation de *DEP_REQ*, enchaînement de *DEP_RES*,

- Détection des erreurs de transmission et procédure de récupération,
- Détection de l'enlèvement des cibles.

5.1.2. Fonctions a implémenter sur le PC

Dans l'architecture NFC Forum, NFC-DEP (ISO 18092) est vu comme une couche de transmission bas-niveau (“MAC”) d'un protocole de plus haut niveau orienté connexion appelé LLCP.

Comme le **H663** implémente seulement le ISO 18092, les protocoles de haut niveau et les applications (par exemple, LLCP et SNEP en plus de LLCP) doivent être implémentées par une application PC. **SpringCard SDK pour PC/SC + NFC** fournit de nombreux échantillon pour le faire. Merci de télécharger le SDK de notre site web.

Comme le support pour LLCP doit être réclamé par l'initiateur NFC dans son *ATR_REQ*, le **H663** a des bytes G_I configurables, la valeur par défaut étant la valeur suivante, compatible avec LLCP:

```
46 66 6D 01 01 11 03 02 00 13 04 01 96
```

Pour changer les bytes G_I, typiquement pour désactiver le LLCP, se référer à § 5.3.1

5.2. Cartographie des fonctions NFC dans les fonctions PC/SC

5.2.1. ATR d'une cible ISO 18092

Le **H663** construit un pseudo-ATR en utilisant le format standard définit dans la spécification PC/SC:

Offset	Name	Value	Meaning (according to 7816-3)
0	TS	$_h$3B	Direct convention
1	T0	$_h$8…	Higher nibble 8 means: no TA1, no TB1, no TC1. TD1 to follow Lower nibble is the number of historical bytes (0 to 15)
2	TD1	$_h$80	Higher nibble 8 means: no TA2, no TB2, no TC2. TD2 to follow Lower nibble 0 means: protocol T=0
3	TD2	$_h$01	Higher nibble 8 means: no TA3, no TB3, no TC3, no TD3 Lower nibble 1 means: protocol T=1
4	H1	…	G_T bytes from ATR_RES
…	…		
3+k	Hk		
4+k	TCK	XX	Checksum (XOR of bytes 1 to 3+k)

Cette cible est compatible LLCP si ses G_T bytes commencent avec

```
46 66 6D
```

5.2.2. Utiliser SCardTransmit (ENCAPSULATE) pour échanger des PDU

ENCAPSULATE commande APDU = DEP_REQ

CLA	INS	P1	P2	Lc	Data In	Le
$_h$FF	$_h$FE	$_h$00	$_h$00	XX	Transport data bytes	$_h$00

Jusqu'à 255 bytes de données transport peuvent être transmises de cette manière.

Le **H663** ajoute le PFB (et le DID si nécessaire) et transmet le bloc valide. Si le tampon de reception de la cible est plus court que la taille du transport PDU, les blocs enchaînés sont automatiques. NAD n'est pas supporté.

Pendant la réception des blocs enchaînés, le **H663** les ré-assemble et les renvoie comme une seule réponse. Jusqu'à 256 bytes de données transport peuvent être reçues.

ENCAPSULATE réponse = DEP_RES

Data Out	SW1	SW2
Transport data bytes	See below	

ENCAPSULATE Statut

SW1	SW2	Meaning
$_h$90	$_h$00	Success
$_h$6F	XX	Error reported by the contactless interface. See chapter 6 for the list of possible values and their meaning.
$_h$62	$_h$82	Le is greater than actual response from target
$_h$6C	XX	Le is shorter than actual response from target

5.3. Options avancées

5.3.1. Changer les G_I bytes dans ATR_REQ

Les bytes générales a transmettre au **H663** *ATR_REQ* (G_I bytes) sont sotckées dans le **registre $_h$E1**.

Si le registre reste vide, la valeur par défaut est:

46 66 6D	LLCP magic number
01 01 11	LLCP version 1.1
03 02 00 13	Services = LLC Link Management + SNEP (NDEF exchange protocol)
04 01 96	Link time-out = 1.5 seconds

Utiliser la commande *PUSH REGISTER* (§ 6.3.7) pour fixer les nouveaux bytes généraux avant de mettre une nouvelle cible NFC en face de l'antenne du **H663**.

Utiliser la commande *WRITE REGISTER* (§ 6.3.6) si vous souhaitez que la nouvelle configuration soit permanente. Faites attention car la mémoire non-volatile à une endurance d'écriture limitée.

6. Contrôle direct du H663

6.1. Bases

Dans l'architecture PC/SC, la fonction ***SCardControl*** implémente le dialogue entre l'application et le coupleur, même s'il n'y a pas de carte dans le port.

L'accès au coupleur doit être obtenu en utilisant ***SCardConnect***, spécifiant SCARD_SHARE_DIRECT comme mode de partage coupleur.

> *Tous les drivers PC/SC ne permettent pas à l'application d'obtenir un accès direct au coupleur. Si vous utilisez le **driver SpringCard SDD480 PC/SC pour Windows**, il n'y a rien de spécifique à faire, mais pour les autres drivers une configuration spécifique doit être réalisée. Merci de vous référer au chapitre 9: Annexe B – activer SCardControl avec les différents Drivers.*

6.2. Détails d'implémentations

6.2.1. Échantillon de code

```
#include <winscard.h>

// dwControlCode for SpringCard SDD480 driver
#define IOCTL_SC_PCSC_ESCAPE       SCARD_CTL_CODE(2048)
// dwControlCode for Microsoft CCID drivers
#define IOCTL_MS_PCSC_ESCAPE       SCARD_CTL_CODE(3050)

// This function is a wrapper around SCardControl
// It creates its own PC/SC context for convenience, but you
// may remain into a previously open context

// Note: Use SCardListReaders to get reader_name

LONG reader_control(const char *reader_name,
                    const BYTE in_buffer[],
                    DWORD      in_length,
                    BYTE       out_buffer[],
                    DWORD      max_out_length,
                    DWORD      *got_out_length)
{
  SCARDCONTEXT hContext;
  SCARDHANDLE  hCard;

  LONG rc;
  DWORD dwProtocol;

  rc = SCardEstablishContext(SCARD_SCOPE_SYSTEM,
                             NULL,
                             NULL,
                             &hContext);
  if (rc != SCARD_S_SUCCESS)
    return rc;
```

```
  // get a direct connection to the reader
  // this must succeed even when there's no card

  rc = SCardConnect(hContext,
                    reader_name,
                    SCARD_SHARE_DIRECT,
                    0,
                    &hCard,
                    &dwProtocol);
  if (rc != SCARD_S_SUCCESS)
  {
    SCardReleaseContext(hContext);
    return rc;
  }

  // direct control through SCardControl
  // dwControlCode for SpringCard SDD480 driver

  rc = SCardControl(hCard,
                    IOCTL_SC_PCSC_ESCAPE,
                    in_buffer,
                    in_length,
                    out_buffer,
                    max_out_length,
                    got_out_length);

  if ((rc == ERROR_INVALID_FUNCTION)
   || (rc == ERROR_NOT_SUPPORTED)
   || (rc == RPC_X_BAD_STUB_DATA))
  {
    // direct control through SCardControl
    // dwControlCode for Microsoft CCID drivers

    rc = SCardControl(hCard,
                      IOCTL_MS_PCSC_ESCAPE,
                      in_buffer,
                      in_length,
                      out_buffer,
                      max_out_length,
                      got_out_length);
  }

  // close the connection
  // the dwDisposition parameter is coherent with the fact
  // that we didn't do anything with the card (or that there's
  // no card in the reader)

  SCardDisconnect(hCard, SCARD_LEAVE_CARD);
  SCardReleaseContext(hContext);

  return rc;
}
```

6.2.2. Lien vers le protocole d'héritage SpringProx

Envoyer une séquence de fuite via *SCardControl* (avec la valeur appropriée pour *dwControlCode*) est exactement la même chose que d'envoyer une “commande héritage” à un coupleur **SpringCard** fonctionnant en **mode héritage**.

La référence détaillée de toutes les commandes supportées par nos lecteurs est disponible dans les kits de développement **SpringCard CSB4**, **K531**, **K632** ou **K663**. Les paragraphes ci-dessous décrivent seulement un sous-ensemble de la liste des fonctions mais les fonctions listées sont les plus utiles dans un contexte PC/SC.

6.2.3. Format de réponses, codes de renvoi

Lorsque le dialogue avec **H663** a été réalisé avec succès, *SCardControl* renvoie SCARD_S_SUCCESS, et au moins un byte est renvoyé dans out_buffer (en position 0).

La valeur de ce byte est le code de statut du coupleur: $_h$00 lors d'un succès, une valeur différente de zéro en cas d'échec. La liste complète des codes d'erreur **H663** est donnée dans le chapitre 8: Annexe A – Codes erreur spécifiques.

Lorsqu'il y a des données disponibles, les données sont renvoyées en position 1 dans out_buffer.

6.2.4. Redirection à l'interprète APDU embarqué

SCardControl charge en commençant par $_h$FF (CLA byte de l'interprète APDU embarqué) géré comme s'ils étaient reçus par le stream *SCardTransmit*.

6.3. Liste des séquences de contrôle disponibles

6.3.1. Action sur les LED

a. Fixer les LED du coupleur manuellement

Pour un coupleur avec des LED rouges et vertes seulement, envoyer la séquence:

```
58 1E <red> <green>
```

Pour un coupleur avec des LED rouges, vertes et jaunes/bleues, envoyer la séquence:

```
58 1E <red> <green> <yellow/blue>
```

Choisir les valzurs pour rouge, verte et jaune/bleu dans ce tableau:

h00	LED is switched OFF
h01	LED is switched ON
h02	LED blinks slowly
h04	LED blinks quickly
h05	LED performs the “heart-beat” sequence

Une fois qu'une telle commande a été envoyée à **H663**, le firmware ne gère plus les LEDs automatiquement: les LED restent en permanence dans le dernier état spécifié par l'application.

Utilier la commande ci-dessous pour que le firmaware gère les LED automatiquement à nouveau.

b. Revenir au mode par défaut (les LED sont gérées par le coupleur du firmware)

Envoyer la séquence

```
58 1E
```

Pour revenir au mode par défaut.

6.3.2. Action sur le buzzer

a. Démarrer/arrêter le buzzer

Certains hardware disposent d'un beeper à tonalité simple. Pour démarrer le buzzer, envoyer la séquence:

```
58 1C <duration MSB> <duration LSB>
```

Où la durée spécifie la longueur de la tonalité, en millisecondes (max est 60000ms).

Mettre la durée à 0 si vous avez besoin de stopper le buzzer avant la durée commencée lors de l'appel précédent.

Une fois qu'une telle commande à été envoyée à **H663**, le firmware ne gère plus le buzzer automatiquement.

Utiliser la commande ci-dessous pour que le firmware gère automatiquement le buzzer de nouveau.

b. Revenir au mode par défaut (buzzer gérer par le firmware du coupleur)

Envoyer la séquence

```
58 1C
```

Aller au mode par défaut.

6.3.3. Obtenir l'information sur le coupleur et les ports

Les séquences ci-dessous sont utiles pour retrouver l'informations textuelle pour le nom du produit, le nom du port, etc… Les informations numériques (comme la version, le numéro de série) sont renvoyées en fils hexadécimaux.

Rappelez-vous que la valeur renvoyée (s'il y en a) est préfixée par le code statut ($_h$00 lors d'un succès).

a. Information sur la gamme de produits coupleur

Sequence	Will return…
58 20 01	Vendor name (“SpringCard”)
58 20 02	Product name
58 20 03	Product serial number (in ASCII)
58 20 04	USB vendor ID and product ID (in ASCII)
58 20 05	Product version (in ASCII)
58 20 80	Number of slots (raw value on 1 byte)
58 20 83	Product serial number (raw value on 4 bytes)
58 20 84	USB vendor ID and product ID (raw value on 4 bytes)
58 20 85	Product version (raw value on 3 bytes: major/minor/build)

b. Informations liées au port

Sequence	Will return…
58 21	Name of the current slot
58 21 00	Name of slot 0
58 21 01	Name of slot 1
…	
58 21 NN	Name of slot N

La nomination du port obéit à la règle suivante:

- Le port sans-contact est nommé “Sans-contact”,
- Lorsqu'un port de carte à puce sans-contact est présent, il est nommé “Contact”,
- Lorsque seul un port SIM/SAM est présent, il est appelé “SAM A” ou “SAM” selon la configuration fixée dans l'usine,
- Lorsque plus d'un port SIM/SAM est présent, ils sont nommés “SIM/SAM A”, “SIM/SAM B”, “SIM/SAM C” et “SIM/SAM D”.

*Le driver CCID **SpringCard** pour Windows (ref. SDD480) utilise ces noms pour construire la liste renvoyée à SCardListReaders. Les autres drivers sont suceptibles d'implémenter une convention de nomination différente.*

6.3.4. Arrêter/commencer un port

Lorsqu'un port est arrêter, le **H663**

- éteint le port de la carte à puce (s'il y en a un),
- désactive le port[22],
- envoie l'évènement “carte retirée” s'il y avait une carte dans le port.

Lorsque le port est redémarré, le **H663**

- active le port[23],
- essaie d'alimenté la carte à puce dans le port (s'il y en a une),
- si une carte a été trouvée, envoyer l'évènement “carte insérée”.

a. Arrêter un port

Sequence	Action
58 22	Stop current slot

b. Démarrer un port

Sequence	Action
58 23	Start current slot

[22] On contactless slot, the antenna RF field is switched OFF

[23] On contactless slot, the antenna RF field is switched ON

6.3.5. Séquences retirer/insérer de force

Utiliser ces séquences pour émuler l'insertion ou l'enlèvement de la carte. C'est utile pour le hardware lorsqu'il n'y a pas d'interrupteur de “présence de carte” disponible.

Faites attention à ce qu'aucune carte ne soit présente lorsque l'insertion carte est réalisée, le **H663** serait occupé un long moment (pour rien), avant d'éventuellement abandonner et renvoyer un statut “carte muette”.

N'UTILISEZ PAS cette option sauf si cela est explicitement conseillé par l'équipe support de SpringCard.

a. Insertion de la carte

Sequence	Action
58 40 01 01	Simulate a card insertion in the 1st card slot (ID-1 slot if existing)
58 40 01	Simulate a card insertion in the current slot

b. Enlèvement de la carte

Sequence	Action
58 40 01 00	Simulate a card removal from the 1st card slot (ID-1 slot if existing)
58 40 00	Simulate a card removal from the current slot

6.3.6. Lire/écrire les registres de configuration H663

Le **H663** dispose d'une mémoire non-volatile pour stocker les registres de configuration.

Voir le chapitre 7 pour la liste de ces registres, et leurs valeurs autorisées.

a. Lire les registres du coupleur

Pour lire la valeur du registre configuration à <index>, envoyer la séquence:

```
58 0E <index>
```

Rappelez-vous que la valeur renvoyée (s'il y en une) est préfixée par le code statut ($_h$00 si succès, $_h$16 si la valeur n'est pas définie dans la mémoire non-volatile).

b. Écrire les registres du coupleur

Pour définir la valeur du registre de configuration à <index>, envoyer la séquence:

```
58 0D <index> <…data…>
```

Envoyer un <data> empty (longueur-zéro) pour effacer la valeur courante. Dans ce cas, la valeur par défaut sera utilisée.

La mémoire non-volatile a une endurance écrire/effacer limitée.
Écrire une valeur différente dans le registre de configuration plus de 100 fois peut endommager votre produit de manière permanente.

La configuration est chargée lors de la réinitialisation. Pour appliquer une nouvelle configuration, vous devez réinitialiser H663 (ou combiner les cycles).

Vous pouvez également charger temporairement les paramètres de configuration comme expliqué dans le prochain paragraphe.

6.3.7. Pousser une nouvelle configuration temporaire

Pour renverser temporairement la valeur du registre de configuration à <index>, envoyer la séquence:

```
58 8D <index> <…data…>
```

Envoyer une <data> vide (longueur-zéro) pour recharger la valeur par défaut.

Cette valeur sera appliquée immédiatement, mais lors de la prochaine réinitialisation le **H663** rechargera ses registres de configuration de la mémoire non-volatile.

7. Registres de Configuration

Le **H663** dispose d'une mémoire non-volatile pour stocker sa configuration.

La mémoire est divisée en “registres”. Chaque registre est identifié par son adresse (une valeur 1-B) et est documenté dans ce chapitre.

Avertissement 1
Certains registres ne sont pas listés dans ce chapitre, mais ils ont pu être définis en usine, ou doivent utiliser la valeur par défaut pour la bonne opération. Ne pas écrire ou effacer les registres qui ne sont pas listés dans ce chapitre.

Avertissement 2
La mémoire non-volatile a une endurance écrire/effacer limitée.
Écrire une valeur différente dans le registre de configuration plus de 100 fois peut endommager votre produit de manière permanente.

7.1. Modifier la configuration du coupleur

7.1.1. Avec un logiciel

Les registres de configuration du coupleur sont disponibles grâce à l'appel de la fonction **SCardControl**. Référez-vous à § 6.3.6 pour plus de détails.

La configuration est chargée lors de la réinitialisation. Pour appliquer cette nouvelle configuration, le logiciel doit demander à l'utilisateur de réinitialiser ou débrancher/brancher le H663.

7.1.2. Utiliser le logicieil SpringCard MultiConf

SpringCard a développé un logiciel de configuration polyvalent[24] qui couvre la plupar des produits, dont le **H663** et tous les lecteurs de la **famille H663**.

Télecharger **SpringCard MultiConf** à http://www.springcard.com/en/download/find/file/sn14007

[24] Available for the Windows platform only.

7.2. Liste des registres de configuration disponiblies pour l'utilisateur final ou l'intégrateur

Address	Section	Name	See §
hB0	Contactless	Enabled protocols	7.5.1
hB2	PC/SC	CLA of the APDU interpreter	7.4.2
hB3	PC/SC	RF behaviour in PC/SC mode	7.4.3
hB4	Contactless	Parameters for polling	7.5.2
hC3	7816	Options for the smartcard slots	7.8.1
hC4	Contactless	Allowed baudrates in T=CL	7.5.4
hC5	Contactless	Options for T=CL	7.5.5
hC8	Contactless	Number of antennas + compatibility settings	7.5.6
hC9	Contactless	Options for polling	7.5.3
hCA	Core	Configuration of the LEDs	7.3.1
hCB	Core	Options for the LEDs and GPIOs	7.3.2
hCC	Core	Behaviour of the LEDs and buzzer	7.3.3
hCF	Felica	Service Codes for Felica read/write	7.6.1
hE1	NFC P2P	Global Bytes bytes in ATR_REQ	7.7.1

Ne rien écrire ou effacer dans les registres listés dans ce chapitre.

7.3. Configuration Centrale

7.3.1. Configuration des LEDs

Adresse: $_h$CA – Taille: 2 bytes

	Bit	Action if set	Note
msb	15 - 12	LED 1 $_h$0: colour is undefined $_h$1: colour is red $_h$2: colour is green $_h$3: colour is yellow $_h$4: colour is blue	
	11 - 8	LED 2 $_h$0: colour is undefined $_h$1: colour is red $_h$2: colour is green $_h$3: colour is yellow $_h$4: colour is blue	
	7 - 4	LED 3 $_h$0: colour is undefined $_h$1: colour is red $_h$2: colour is green $_h$3: colour is yellow $_h$4: colour is blue	
lsb	3 - 0	LED 4 $_h$0: colour is undefined $_h$1: colour is red $_h$2: colour is green $_h$3: colour is yellow $_h$4: colour is blue	

Valeur par défaut: $_h$0000

7.3.2. Options des LED et GPIO

Adresse: $_h$C9 – Taille: 1 byte1

	Bit	Action if set	Note
msb	7	Use PWM for buzzer	
	6	RFU	
	5	RFU	
	4	RFU	
	3	Invert logic for LED 4	
	2	Invert logic for LED 3	
	1	Invert logic for LED 2	
lsb	0	Invert logic for LED 1	

Valeur par défaut: $_h$00

7.3.3. Attitude des LED et buzzer

Si le coupleur a des LED, le coupleur montre son état (carte présente, carte absente, erreur) par ces LED. Vous pouvez désactiver cette option en fixant le bit 7 de ce registre à 1 (si l'application est toujours capable de contrôler les LED comme documenté en § 6.3.1.a et 3.4.1.a).

Si le coupleur a un buzzer, le buzzer sonne à chaque fois que le PICC est activé. Les 6 bytes d'ordre bas de ce registre définissent la durée de ce beep en intervalle de 10ms. Pour désactiver le bip automatique lors de l'arrivée de la carte, fixer cette valeur à 0 (si l'application est toujours capable de contrôler le buzzer comme documenté dans § 6.3.2 et 3.4.1.b).

Adresse: $_h$CC – Taille: 1 byte

	Bit	Values / Meaning
msb	7	1 : the H663 does signal its state on the LEDs 0 : the H663 doesn't signal its state on the LEDs
	6	RFU, must be 0
lsb	5	Duration of the automatic beep on card arrival, x 10ms (0 to 630ms) Set to $_h$00 to disable the automatic beep

Valeur par défaut: $_h$88 (80ms beep lors de l'arrivée de PICC + état des LEDs)

7.4. PC/SC configuration

7.4.1. Nomination des ports et mode startup

Adresse: $_h$B1 – Taille: 1 byte

	Bit	Action if set	Note
msb	7	Force a letter in the name of the SAM slots	Even if there's only one slot, it will be named “SAM A”
	6	Force a letter in the name of the ID-1 slots	Even if there's only one slot, it will be named “Contact A”
	5	RFU	
	4	Prefix the slot name using the product's serial number (in hex)	This is useful for computers with numerous products attached
	3	Start with SAM slot(s) OFF	All the SAM slot(s) will not run until resumed by a Control command
	2	Start with Contact slot OFF	The Contact slot will not run until resumed by a Control command
	1	Start with Contactless slot OFF	The Contactless slot will not run until resumed by a Control command
msb	0	No contactless slot	The Contactless slot will not be enumerated (and will never run)

Valeur par défaut: $_h$00

7.4.2. Byte CLA de l'interprète APDU

Ce registre définit le byte CLA (class) affecté à l'interprète APDU (voir § 3.1.1).

Pour désactiver l'interprète APDU, définir ce registre à $_h$00.

Adresse: $_h$B2 – Taille: 1 byte

Valeur par défaut: $_h$FF

7.4.3. Attitude du port sans-contact en mode PC/SC

Ce registre définit l'attitude du port sans-contact **H663** en mode PC/SC.

Adresse: $_h$B3 – Taille: 1 byte

	Bit	Action if set	Note
msb	7	Innovatron: return the “real” T=0 ATR (as supplied in REPGEN) instead of building a pseudo ATR	Setting this bit breaks the compatibility with MS CCID driver, because the card is connected in T=1 where its ATR claims it is T=0 only
	6	Use only standard values for PIX.NN in the ATR	Numerous contactless PICCs/VICCs have not been registered by their vendor in the PC/SC specification to get a standard PIX.NN. SpringCard has defined vendor-specific values for those cards (see 4.1.5). If this bit is set, these non-standard values will not be used, and PIX.NN will be fixed to $_h$0000 for all PICCs/VICCs that are not in the standard.
	5	Disable the pause in RF field after the PICC/VICC has been removed	When the PICC/VICC stops responding, the H663 pauses its RF field for 10 to 20ms. Setting this bit disable this behaviour.
	4	Disable the pause in RF field after the PICC/VICC during the polling	During the polling sequence, the H663 pauses its RF field for 10 to 20ms between the polling loops. Setting this bit disable this behaviour.
	3	No NFC-DEP activation over Felica (ISO 18092 @ 212 or 424 kbit/s)	
	2	No NFC-DEP activation over ISO 14443-A (ISO 18092 @ 106 kbit/s)	
	1	No T=CL (ISO-DEP) activation over ISO 14443-B	Send SLOT CONTROL P1,P2=$_h$20,01 to activate the PICC manually
lsb	0	No T=CL (ISO-DEP) activation over ISO 14443-A	Send SLOT CONTROL P1,P2=$_h$20,02 to activate the PICC manually

Valeur par défaut: $_h$00 (T=CL active sur 14443 A et B, NFC-DEP active sur 14443 A et Felica)

7.5. Configuration sans-contact

7.5.1. Protocoles activés

Ce registre définit la liste des protocoles que le **H663** va rechercher pendant la boucle d'attente. N'importe quelle PICC/VICC compatible avec l'un de ces protocoles actifs sera “vu”, et les autres seront ignorés.

Adresse: $_h$B0 – Taille: 2 bytes (en premier MSB)

	Bit	Active. protocol (if set)	Version
msb	15	*RFU*	
	14	*RFU*	
	13	*RFU*	
	12	JIS:X6319-4 (Felica) ISO 18092 @ 212 kbit/s and 424 kbit/s NFC Forum Type 3 Tags	
	11	ThinFilm	
	10	NFC Forum Type 1 Tags (Innovision/Broadcom chips)	
	9	*RFU*	
	8	EM 4134	≥ 1.81
	7	Innovatron (legacy Calypso cards – sometimes called 14443-B’)	
	6	*RFU*	
	5	ST Micro Electronics SRxxx	
	4	Inside Contactless PicoPass (also HID iClass)	
	3	*RFU*	
	2	ISO 15693	
	1	ISO 14443-B NFC Forum Type 4-B Tags	
lsb	0	ISO 14443-A ISO 18092 @ 106kbit/s NFC Forum Type 2 and Type 4-A Tags	

Valeur par défaut: $_h$F7FF (tous protocoles supportés sauf les ThinFilm sont activés)

7.5.2. Paramètres d'attente

Le registre définit les paramètes utilisés par le **H663** pour l'attente PICC/VICC.

Adresse: $_h$B4 – Taille: 5 bytes

Byte	Data	Default value	Remark
0	AFI for ISO 14443-B	$_h$00	Specify the *Application Family Identifier* to be used during ISO 14443-B polling. $_h$00 means that all PICCs shall answer.
1	AFI for ISO 15693	$_h$00	Specify the *Application Family Identifier* to be used during ISO 15693 polling. $_h$00 means that all VICCs shall answer.
2 - 3	SC for JIS:X6319-4 and ISO 18092 @ 212 and 424 kbit/s	$_h$FFFF	Specify the *System Code* to used during Felica polling (SENSF_REQ). The value is stored MSB first. $_h$FFFF means that all targets shall answer.
4	RC for JIS:X6319-4 and ISO 18092 @ 212 and 424 kbit/s	$_h$00	Specify the *Request Code* to used during Felica polling (SENSF_REQ). This value shall be $_h$00 to accept both NFC Type 3 Tags and NFC devices running in P2P mode (NFC-DEP), or $_h$01 to accept only NFC Type 3 Tags

7.5.3. Options pour l'attente

Utiliser ce registre pour configurer le support ATQB étendu pour les cartes ISO 14443-B, et pour désactiver JIS:X6319-4 / ISO 18092 @ 424 kbit/s.

Adresse: $_h$C9 – Taille: 1 byte

	Bit	Action if set	Note
msb	7	RFU	
	6	RFU	
	5	RFU	
	4	Activate extended ATQB	If this bit is set, the H663 will ask for an extended ATQB from ISO 14443-B. Not all cards do support this feature.
	3	Disable JIS:X6319-4 / ISO 18092 @ 424 kbit/s	If this bit is set, the H663 will communicate with Felica cards and NFC P2P targets up to 212 kbit/s only
	2	RFU	
	1	RFU	
lsb	0	RFU	

Valeur par défaut: $_h$00 (normal ATQB, permet 424kbit/s pour JIS:X6319-4)

7.5.4. Débits autorisés en T=CL (ISO 14443-4)

Utiliser ce registre pour laisser le **H663** négocier un débit plus important que 106 kbit/s avec ISO 14443-4 PICCs (DSI, DRI définit en PPS pour ISO 14443 A, en ATTRIB pour ISO 14443 B).

*Le **H663** est théorétiquement capable de communiquer avec les PICC à 848 kbit/s dans les deux directions, mais la vitesse maiximum dépend surtout des caractérisitiques du PICC, et de l'antenne actuelle du coupleur et de son environnement.*

Adresse: $_h$C4 – Taille: 2 bytes (en premier MSB)

	Bit	Meaning (if set)
		ISO 14443-A DS
msb	15	RFU, must be 0
	14	Allow ISO 14443 A PICC → H663 @ 848 kbit/s (DSI = 3 in PPS)
	13	Allow ISO 14443 A PICC → H663 @ 424 kbit/s (DSI = 2 in PPS)
	12	Allow ISO 14443 A PICC → H663 @ 212 kbit/s (DSI = 1 in PPS)
		ISO 14443-A DR
	11	RFU, must be 0
	10	Allow ISO 14443 A H663 → PICC @ 848 kbit/s (DRI = 3 in PPS)
	9	Allow ISO 14443 A H663 → PICC @ 424 kbit/s (DRI = 2 in PPS)
	8	Allow ISO 14443 A H663 → PICC @ 212 kbit/s (DRI = 1 in PPS)
		ISO 14443-B DS
	7	RFU, must be 0
	6	Allow ISO 14443 B PICC → H663 @ 848 kbit/s (DSI = 3 in ATTRIB)
	5	Allow ISO 14443 B PICC → H663 @ 424 kbit/s (DSI = 2 in ATTRIB)
	4	Allow ISO 14443 B PICC → H663 @ 212 kbit/s (DSI = 1 in ATTRIB)
		ISO 14443-B DR
	3	RFU, must be 0
	2	Allow ISO 14443 B H663 → PICC @ 848 kbit/s (DRI = 3 in ATTRIB)
	1	Allow ISO 14443 B H663 → PICC @ 424 kbit/s (DRI = 2 in ATTRIB)
lsb	0	Allow ISO 14443 B H663 → PICC @ 212 kbit/s (DRI = 1 in ATTRIB)

Valeur par défaut: $_h$3333 (jusqu'à 424 kbit/s).

Vous devez baisser les débits autorisés à 106kbps ($_h$0000) si votre antenne n'est pas capable de gérer des débits plus importants sans erreurs de communication.

7.5.5. Options pour T=CL (ISO 14443-4)

Le registre déifnit l'attitude du sous-système ISO 14443-4.

Adresse: $_h$C5 – Taille: 4 bytes

Byte	Data	Default value	Remark
0	Extra guard time	$_h$00	Guard time (specified in ms) to add before sending a frame to the PICC.
1	Retries on card mute	$_h$03	Number of retries before giving up when the PICC does not answer (communication timeout, and no other error detected)
2	Retries on comm. error	$_h$03	Number of retries before giving up when the PCC does not understand the PICC's response (CRC, parity, framing errors...)
3	RFU	$_h$00	*This byte must be $_h$00*

7.5.6. Nombre d'antennes + auto-stop

Adresse: $_h$C8 – Taille: 1 byte

	Bit	Action if set	Note
msb	7	RFU	
	6	RFU	
	5	RFU	
	4	Be compliant with RCTIF	This feature changes the polling sequence and the behaviour against Calypso Innovatron cards
	3	RFU	
	2	RFU	
	1	RFU	
lsb	0	Activate the secondary antenna	

Valeur par défaut: $_h$00 (seulement une antenne)

Merci de vous référer au document PNA2236 "H663 integration guide" pour plus d'informations sur la seconde antenne. Cette option est disponibles pour les H663 (non-balancés) seulement. Noter que les deux antennes ont exactement les mêmes caractéristiques RF.

7.6. Configuration Felica

7.6.1. Codes Service Codes pour lire/écrire sur Felica

Utiliser le registre pour définir comme le **H663** traite les cartes Felica et les Tags NFC de type 3.

Adresse: $_h$CF – Taille: 4 bytes

Byte	Data	Default value	Remark
0 - 1	Read Service Code	$_h$000B	Service Code used when the READ BINARY instruction is invoked (MSB first) The value $_h$000B is mandated by the specification of the NFC Forum Type 3 Tag
2 - 3	Update Service Code	$_h$0009	Service Code used when the UPDATE BINARY instruction is invoked (MSB first) The value $_h$0009 is mandated by the specification of the NFC Forum Type 3 Tag

Ces valeurs peuvent être temporairement écrasées dans le stream *SCardTransmit* en utilisant les instructions **SET FELICA RUNTIME PARAMETERS** (§ 3.3.6).

7.7. Configuration ISO 18092 / NFC-DEP

7.7.1. Global Bytes dans ATR_REQ

Adresse: $_h$E1 – Taille: 0 à 15 bytes

Ce registre définit les bytes G_I envoyées dans ATR_REQ.

Si ce registre reste vide, la valeur par défaut est:

46 66 6D	LLCP magic number
01 01 11	LLCP version 1.1
03 02 00 13	Services = LLC Link Management + SNEP (NDEF exchange protocol)
04 01 96	Link timeout = 1.5 seconds

7.8. Configuration ISO 7816

7.8.1. Options pour les ports cartes à puce

Ce registre définit les paramètres utilisés par **H663** pour les ports SIM/SAM et les cartes à puce.

Adresse: $_h$C3 – Taille: 5 bytes

Byte	Data	Default value	Remark
0	Configuration of the ID-1 slot	$_h$B3	- Contact if the coupler has a ID-1 slot - not used otherwise
1	Configuration of the SAM1 slot	$_h$B3	- “SAM A” if the coupler has 1 or 4 SAMs - not used otherwise
2	Configuration of the SAM2 slot	$_h$B3	- “SAM A” if the coupler has 3 SAMs - “SAM B” if the coupler has 4 SAMs
2	Configuration of the SAM3 slot	$_h$B3	- “SAM B” if the coupler has 3 SAMs - “SAM C” if the coupler has 4 SAMs
4	Configuration of the SAM4 slot	$_h$B3	- “SAM C” if the coupler has 3 SAMs - “SAM D” if the coupler has 3 SAMs

Chaque bits de bytes est définit comme suit:

	Bit	Action if set	Note
msb	7	Enable automatic PPS	
	6	Enable HSP	Use this only with Calypso SAMs
	5	Enable EMV power on	EMV mode is tried before standard mode
	4	Enable non-EMV power on	Non-EMV cards will be rejected
	3	RFU	
	2	Enable class C (1.8V)	The coupler tries the lower voltage class first, and then increments until one matches.
	1	Enable class B (3V)	
lsb	0	Enable class A (5V)	

$_h$B3 signifie

- PPS automatique
- HSP désactivé
- EMV essai d'alimentation avant l'allumage standard
- Class = AB (3V puis 5V)

8. Annexe A – Codes erreur spécifiques

Lorsque l'interprète APDU renvoie SW1 = $_h$6F, la valeur de SW2 cartographie l'un des codes erreur spécifiques **H663** listés ci-dessous.

SW2	Symbolic name[25]	Meaning
$_h$01	MI_NOTAGERR	No answer received (no card in the field, or card is mute)
$_h$02	MI_CRCERR	CRC error in card's answer
$_h$03	MI_EMPTY	No data available
$_h$04	MI_AUTHERR	Card authentication failed
$_h$05	MI_PARITYERR	Parity error in card's answer
$_h$06	MI_CODEERR	Invalid card response opcode
$_h$07	MI_CASCLEVEX	Bad anti-collision sequence
$_h$08	MI_SERNRERR	Card's serial number is invalid
$_h$09	MI_LOCKED	Card or block locked
$_h$0A	MI_NOTAUTHERR	Card operation denied, must be authenticated first
$_h$0B	MI_BITCOUNTERR	Wrong number of bits in card's answer
$_h$0C	MI_BYTECOUNTERR	Wrong number of bytes in card's answer
$_h$0D	MI_VALUEERR	Card counter error
$_h$0E	MI_TRANSERR	Card transaction error
$_h$0F	MI_WRITEERR	Card write error
$_h$10	MI_INCRERR	Card counter increment error
$_h$11	MI_DECRERR	Card counter decrement error
$_h$12	MI_READERR	Card read error
$_h$13	MI_OVFLERR	RC: FIFO overflow
$_h$15	MI_FRAMINGERR	Framing error in card's answer
$_h$16	MI_ACCESSERR	Card access error
$_h$17	MI_UNKNOWN_COMMAND	RC: unknown opcode
$_h$18	MI_COLLERR	A collision has occurred
$_h$19	MI_COMMAND_FAILED	RC: command execution failed
$_h$1A	MI_INTERFACEERR	RC: hardware failure
$_h$1B	MI_ACCESSTIMEOUT	RC: timeout
$_h$1C	MI_NOBITWISEANTICOLL	Anti-collision not supported by the card(s)
$_h$1D	MI_EXTERNAL_FIELD	An external RF field is already present, unable to activate the coupler's RF field

[25] As used in SpringProx API (defines in springprox.h)

h1F	MI_CODINGERR	Bad card status
h20	MI_CUSTERR	Card: vendor specific error
h21	MI_CMDSUPERR	Card: command not supported
h22	MI_CMDFMTERR	Card: format of command invalid
h23	MI_CMDOPTERR	Card: option of command invalid
h24	MI_OTHERERR	Card: other error
h3C	MI_WRONG_PARAMETER	Coupler: invalid parameter
h64	MI_UNKNOWN_FUNCTION	Coupler: invalid opcode
h70	MI_BUFFER_OVERFLOW	Coupler: internal buffer overflow
h7D	MI_WRONG_LENGTH	Coupler: invalid length

9. Annexe B – activer SCardControl avec les différents Drivers

Étant compatible avec la spécification CCID, le **H663** est supporté par (au moins) 4 drivers USB:

- SpringCard CCID driver pour Windows (ref. SDD480),
- Microsoft CCID kernel-mode driver (USBCCID) avec Windows 2000/XP/Vista,
- Microsoft CCID mode-utilisateur driver (WUDFUsbccidDriver) avec Windows 7,
- Le driver open-source CCID du package PCSC-Lite sur Linux, MacOS X, et autres systèmes d'exploitation UNIX.

9.1. Contrôle direct utilisant SpringCard SDD480

Le contrôle direct est toujours permit dans le **SpringCard SDD480 driver**.

Avec ce driver, dans l'appel de fonction SCardControl, le paramètre dwControlCode doit être fixé à **SCARD_CTL_CODE(2048)**.

> *SCARD_CTL_CODE est une macro définie dans le titre winscard.h du SDK Windows. Pour les langages non-C/C++, remplacer SCARD_CTL_CODE(2048) par une valeur constante „**00241FE4** (₀3219456).*

9.2. Contrôle direct en utilisant MS USBCCID

Avec le driver **MS USBCCID**, le contrôle direct du coupleur doit être activé sur une base par coupleur: chaque coupleur à son numéro de série USB, et le contrôle direct doit être activé sans équivoque pour ce numéro de série.

Cela est fait en écrivant une valeur dans le registre, en utilisant **regedit** ou un logiciel personnalisé. Voir par exemple l'outil de ligne de commande **ms_ccid_escape_enable**, disponible avec son code source dans **SpringCard PC/SC SDK**.

La clé cible dans le registre est

```
HKEY_LOCAL_MACHINE
          SYSTEM
               CurrentControlSet
                    Enum
                         USB
                              VID_1C34&PID_91B1
                                   yyyyyyyy
                                        Device Parameters
```

où *yyyyyyyy* est le numéro de série du coupleur.

Sous la clé d'enregistrement, créer l'entrée registre **EscapeCommandEnabled**, ou taper **DWORD**, et fixer la valeur **1**. Une fois que la valeur a été écrite, débrancher et brancher le coupleur (ou redémarrer l'ordinateur) pour que le driver redémarre, en prenant le nouveau paramètre en compte.

Avec ce driver, en appel de fonction SCardControl, le paramètre dwControlCode doit être fixé à **SCARD_CTL_CODE(3050)**.

> *SCARD_CTL_CODE est une macro définie dans le titre winscard.h du SDK Windows. Pour les languages non-C/C++, remplacer SCARD_CTL_CODE(3500) par une valeur constante* ***h004074F8*** *(d3225264).*

9.3. Contrôle direct utilisant MS WUDFUsbccidDriver

Avec **MS WUDFUsbccidDriver** (nouveau driver mode-utilisateur introduit dans Windows 7), le contrôle direct du coupleur doit être activé sur une base par coupleur: chaque coupleur à son numéro de série USB et le contrôle direct doit être activé sans équivoque pour le numéro de série.

Cela est fait en écrivant une valeur dans le registre, en utilisant **regedit** ou un logiciel personnalisé. Voir par exemple l'outil de ligne de commande **ms_ccid_escape_enable**, disponible avec son code source dans **SpringCard PC/SC SDK**.

La clé cible dans le registre est

```
HKEY_LOCAL_MACHINE
        SYSTEM
            CurrentControlSet
                Enum
                    USB
                        VID_1C34&PID_91B1
                            yyyyyyyy
                                Device Parameters
                                    WUDFUsbccidDriver
```

où *yyyyyyyy* est le numéro de série du coupleur.

Sous cette clé registre, créer l'entrée du registre **EscapeCommandEnabled**, ou taper **DWORD**, et fixer la valeur à **1**. Une fois que la valeur a été écrite, débrancher et brancher le coupleur (ou redémarrer l'ordinateur) le driver va redémarrer, prenant le nouveau paramètre en compte.

Avec ce driver, dans l'appel de la fonction ScardControl, le paramètre dwControlCode doit être fixé à **SCARD_CTL_CODE(3050)**.

> *SCARD_CTL_CODE est une macro définie dans le titre winscard.h du SDK Windows. Pour les langages non-C/C++, remplacer SCARD_CTL_CODE(3500) par une valeur constante* ***h004074F8*** *(d3225264).*

9.4. Contrôle direct en utilisant PCSC-Lite CCID

A écrire.

10. Licences de partie tierce

SpringCard H663 utilise un composant logiciel open-source d'une partie tierce.

10.1. FreeRTOS

FreeRTOS est un système d'exploitation en temps réel (ou RTOS) de Real Time Engineers Ltd.

Commençant par le firmware version 2.00, **SpringCard H663** fonctionne sur FreeRTOS v8.2.0.

FreeRTOS est distribué sous licence mofifiée GNU General Public License (GPL) qui permet un usage commercial et sur des produits à source fermée.

Pour plus d'informations, ou pour télécharger le code source de FreeRTOS, merci de visiter

www.freertos.org

Disclaimer

This document is provided for informational purposes only and shall not be construed as a commercial offer, a license, an advisory, fiduciary or professional relationship between SPRINGCARD and you. No information provided in this document shall be considered a substitute for your independent investigation.

The information provided in document may be related to products or services that are not available in your country.

This document is provided "as is" and without warranty of any kind to the extent allowed by the applicable law. While SPRINGCARD will use reasonable efforts to provide reliable information, we don't warrant that this document is free of inaccuracies, errors and/or omissions, or that its content is appropriate for your particular use or up to date. SPRINGCARD reserves the right to change the information at any time without notice.

SPRINGCARD doesn't warrant any results derived from the use of the products described in this document. SPRINGCARD will not be liable for any indirect, consequential or incidental damages, including but not limited to lost profits or revenues, business interruption, loss of data arising out of or in connection with the use, inability to use or reliance on any product (either hardware or software) described in this document.

These products are not designed for use in life support appliances, devices, or systems where malfunction of these product may result in personal injury. SPRINGCARD customers using or selling these products for use in such applications do so on their own risk and agree to fully indemnify SPRINGCARD for any damages resulting from such improper use or sale.

Copyright notice

Editor's information

SPRINGCARD SAS company with a capital of 227 000 €

RCS EVRY B 429 665 482

Parc Gutenberg, 2 voie La Cardon

91120 Palaiseau – FRANCE

Contact information

For more information and to locate our sales office or distributor in your country or area, please visit

www.springcard.com