Advanced Card Systems Ltd.
Card & Reader Technologies

ACR1251T

トークンタイプ NFC リーダー II (USB インターフェイス)

リファレンスマニュアル V1.02

事前に告知なく変更される場合があります

info@acs.com.hk
www.acs.com.hk

改定履歴

リリス日付	改訂説明	バージョン
2018-06-04	● 初回発布	1.00
2020-01-10	● MIFARE の商標および帰属の説明を追加。 ● セクション 2.0 更新：特性 ● セクション 5.1 更新： PCSC API ● セクション 5.3.6 更新： 自動的な PICC のポーリングを設置する（Set Automatic PICC Polling） ● セクション 5.3.7 更新： 自動的な PICC のポーリングを読取る（Read Automatic PICC Polling） ● セクション 5.3.10 更新： 自動的な PPS を設定する（Set Auto PPS） ● セクション 5.3.11 更新： 自動的な PPS を読み取る（Read Auto PPS） ● セクション 5.2 の非接触型スマートカード関連コマンドの再グループ化	1.01
2020-08-28	● セクション 5.3.4 更新：LED ステータスインジケータの動作を設定する（Set LED Status Indicator Behavior） ● セクション 5.3.8 更新：PICC 操作のパラメーターを設定する（Set PICC Operating Parameter） ● セクション 5.3.9 更新：PICC 操作のパラメーターを読取る（Read PICC Operating Parameter）	1.02

カタログ

1.0. 紹介 5

2.0. 特性 6

3.0. 略語 7

4.0. アーキテクチャ 8

5.0. ホストプログラミング（PC リンク）API 9

5.1. PCSC API 9
5.1.1. SCardEstablishContext 9
5.1.2. SCardListReaders 10
5.1.3. SCardConnect 11
5.1.4. SCardControl 12
5.1.5. SCardTransmit 14
5.1.6. SCardDisconnect 16
5.1.7. APDU の流れ 17
5.1.8. ダイレクトコマンド（Escape Command）の流れ 18
5.2. 非接触スマートカード プロトコル 19
5.2.1. ATR の生成 19
5.2.2. 非接触インターフェースの疑似 APDU コマンド 23
5.2.3. PCSC 2.0 パート 3 の APDU コマンド（2.02 もしくはもっと新しいバージョン） 24
5.2.4. MIFARE Classic 1K/4K メモリカードの PICC コマンド（T=CL エミュレーション） 40
5.2.5. PC/SC 規格に準拠しているタグにアクセスする（ISO 14443-4） 51
5.2.6. FeliCa タグのアクセス 53
5.3. 周辺デバイス制御 54
5.3.1. ファームウェアのバージョンを取得する（Get Firmware Version） 54
5.3.2. LED 制御（LED Control） 55
5.3.3. LED 状態（LED Status） 56
5.3.4. LED ステータスインジケータの動作を設定する（Set LED Status Indicator Behavior） 57
5.3.5. LED ステータスインジケータの動作を読み取り（Read LED Status Indicator Behavior） 58
5.3.6. 自動的な PICC のポーリングを設置する（Set Automatic PICC Polling） 59
5.3.7. 自動的な PICC のポーリングを読取る（Read Automatic PICC Polling） 61
5.3.8. PICC 操作のパラメータを設定する（Set PICC Operating Parameter） 62
5.3.9. PICC 操作のパラメータを読取る（Read PICC Operating Parameter） 63
5.3.10. 自動的な PPS を設定する（Set Auto PPS） 64
5.3.11. 自動的な PPS を読み取る（Read Auto PPS） 65
5.4. ACR122U 互換性のあるコマンド 66
5.4.1. 二色 LED 制御（Bi-color LED Control） 66
5.4.2. ファームウェアのバージョンを取得する（Get Firmware Version） 68
5.4.3. PICC 操作のパラメータを入手する（Read the PICC Operating Parameter） 69
5.4.4. PICC 操作のパラメータを設定する（Set PICC Operating Parameter） 70

図示カタログ

図示 1：ACR1251T のアーキテクチャ 8
図示 2:ACR1251T の APDU の流れ 17
図示 3:ACR1251T 直接コマンド（Escape Command）の流れ 18

チャートカタログ

表 1 ：略語 7
表 2 ：MIFARE Classic 1K カードのメモリマップ 42
表 3 ：MIFARE Classic 4K カードのメモリマップ 43
表 4 ：MIFARE Ultralight カードのメモリマップ 44

1.0. 紹介

ACR1251T は, 13.56MHz の非接触技術に基づいて開発された ACR1251U PC リンク式 NFC スマートカードリーダーのトークンタイプ USB リーダーです。世界初の CCID 準拠の非接触型リーダーである ACR122U のトークンバージョンの ACR122T に続き, ACR1251T はより高度な機能を提供します。 これは, ISO 14443 タイプ A および B カードだけでなく, MIFARE®, FeliCa, および 4 種類の NFC タグとデバイスをサポートするように設計されています。

ACR1251T は、コンピューターとカードの間の中間デバイスとして、コンピューターからのコマンドを実行し、非接触タグまたはデバイスコンポーネント（LED）と通信します。 その PICC リーダーインターフェイスは、PC / SC 仕様に準拠しています。 このリファレンスマニュアルでは、PC / SC APDU コマンドを実行して非接触インターフェースをサポートし、ACR1251T の周辺機器を制御する方法について詳しく説明します。PC/ SC の APDU コマンドを実行することによって、どのように非接触インターフェースとは ACR1251T の周辺機器をサポートする。この API ドキュメントはこれについて詳しく説明している。

2.0. 特性

- USB フルスピード･インターフェース
- CCID 準拠
- スマートカードリーダー：
  - 非接触インターフェース：
    - 最大 424 kbps の書き込み速度
    - 内蔵アンテナを使って, ACR1251T の 通信距離は最大 30mm（タグのタイプに応じて）
    - ISO 14443 の 4 パート A および B カード, MIFARE Classic®, FeliCa, 4 タイプ の NFC タグ（ISO/IEC 18092）も サポート
    - 衝突防止機能保有（一枚のタグのみアクセス）
    - NFC サポート
      - カードリーダ/ライタモード
- 内蔵されている周辺機器：
  - ユーザーコントロールできる二色 LED パイロットランプ
- アプリケーション プログラミング インターフェース
  - PC/SC サポート
  - (PC / SC の上のラッパー経由で), CT- API をサポート
- USB ファームウェアのアップグレード機能
- Android™ 3.1 と以降のバージョンサポート[1]
- 以下の規格に準拠：
  - EN 60950/IEC 60950
  - ISO 14443
  - ISO 18092
  - PC/SC
  - CCID
  - CE
  - FCC
  - RoHS 2
  - REACH
  - VCCI（日本）
  - MIC（日本）
  - Microsoft® WHQL

[1] *ACS の Android ライブラリを使用*

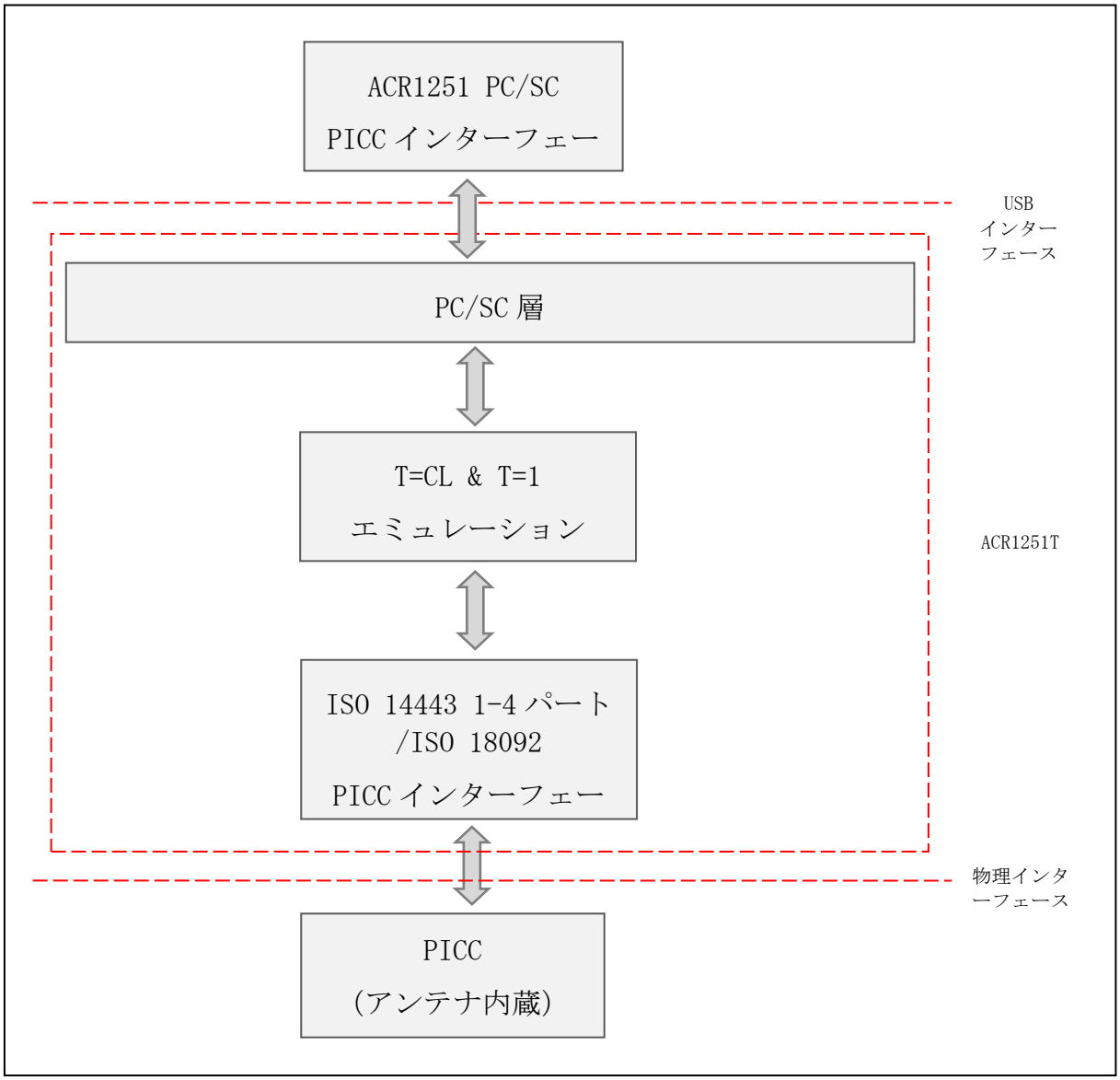
3.0. 略語

略語	説明
ATR	属性請求と属性応答
DEP	データ交換プロトコルの請求と応答
DSL	請求と応答のキャンセル
PSL	パラメ―タ―オプションの請求と応答
RLS	請求リリースと応答リリース
WUP	ウェイクアップ請求とウェイクアップ応答
DID	設備 ID
BS	ビット期間を送信します
BR	ビット期間を受信します
PP	プロトコルパラメ―タ―
Gi	イニシエータのオプションの情報フィールド
PFB	取引のための制御情報
FSL	フレーム長さの最大値
LLCP	論理リンク制御プロトコル

表1 ：略語

4.0. アーキテクチャ

ACR1251T と PC のデータ通信は CCID プロトコルを採用していますが、PICC 間の通信は PC/SC 規格に準拠しています。



図示 1：ACR1251T のアーキテクチャ

5.0. ホストプログラミング（PC リンク）API

5.1. PCSC API

このセッションでは、いくつのアプリケーションプログラミングに使用する PC/SC API コマンドを説明します。これらの API の詳しい情報について、Microsoft MSDN ライブラリまたは PC/SC ワークグループを参照してください。

5.1.1. SCardEstablishContext

SCardEstablishContext 関数はデータベース操作を実行するリソースマネージャのコンテキストを確立するのためです。

ほかの PCSC 実行する前に、この関数を実行するはずです。.

参照のウェブサイト：

http://msdn.microsoft.com/en-us/library/windows/desktop/aa379479%28v=vs.85%29.aspx

例：

```
#define SCARD_SCOPE_USER 0

SCARDCONTEXT hContext;
int retCode;
void main ()
{
   // To establish the resource manager context and assign it to "hContext"
   retCode = SCardEstablishContext(SCARD_SCOPE_USER,
                  NULL,
                  NULL,
                  &hContext);
   if (retCode != SCARD_S_SUCCESS)
   {
      // Establishing resource manager context failed
   }
   else
   {
      // Establishing resource manager context successful
      // Further PCSC operation can be performed
   }

}
```

5.1.2. SCardListReaders

SCardListReaders 関数は、重複をなくして、一つのセットの名前付きリーダーグループリストを提供します。

呼び出し側はリーダーグループのリストを供給します。関数は指定しているセット中の名前付きリーダーのリストを返します。認識できないグループの名前は無視されます。この関数は現在システムに接続されて利用できるグループ中のリーダーだけに返されます。

参照のウェブサイト：

http://msdn.microsoft.com/en-us/library/windows/desktop/aa379793%28v=vs.85%29.aspx

例：

```
#define SCARD_SCOPE_USER 0

SCARDCONTEXT hContext; // Resource manager context
int retCode;
char readerName [256]; // List reader name

void main ()
{
   // To establish the resource manager context and assign to
"hContext"
   retCode = SCardEstablishContext(SCARD_SCOPE_USER,
                  NULL,
                  NULL,
                  &hContext);
   if (retCode != SCARD_S_SUCCESS)
   {
      // Establishing resource manager context failed
   }
   else
   {
      // Establishing resource manager context successful
      // List the available reader which can be used in the system
      retCode = SCardListReaders (hContext,
                  NULL,
                  readerName,
                  &size);
      if (retCode != SCARD_S_SUCCESS)
      {
         // Listing reader fail
      }
      if (readerName == NULL)
      {
         // No reader available
      }
      else
      {
         // Reader listed
      }
   }
}
```

5.1.3. SCardConnect

SCardConnect 関数は（特別のリソースマネージャのコンテキスを利用して）アプリケーションと特定のリーダーを含めているスマートカードの間に接続を確立します。特定のリーダー中はカードがない場合、エラーメッセージが返されます。

参照のウェブサイト：

http://msdn.microsoft.com/en-us/library/windows/desktop/aa379473%28v=vs.85%29.aspx

例：

```
#define SCARD_SCOPE_USER 0

SCARDCONTEXT      hContext;          // Resource manager context
SCARDHANDLE    hCard;             // Card context handle
unsigned long     dwActProtocol; // Establish active protocol
int               retCode;
char              readerName [256]; // List reader name
char              rName [256];      // Reader name for connection

void main ()
{
   …
      if (readerName == NULL)
      {
         // No reader available
      }
      else
      {
         // Reader listed
      rName = "ACS ACR1251 CL Reader PICC 0";    // Depends on what
                                                 reader be used
                                                 // Should connect to
                                                 PICC interface
         retCode = SCardConnect(hContext,
               rName,
               SCARD_SHARE_SHARED,
               SCARD_PROTOCOL_T0,
               &hCard,
               &dwActProtocol);
         if (retCode != SCARD_S_SUCCESS)
         {
            // Connection failed (May be because of incorrect reader
         name, or no card was detected)
         }
         else
         {
            // Connection successful
         }
      }
   }
```

5.1.4. SCardControl

SCardControl 関数はユーザーにカードリーダーをダイレクトに制御する機能を提供しています。**SCardConnect** 関数が成功に呼び出されして、**SCardDisconnect** 関数を呼び出す前に、ユーザーはこの関数を自由に呼び出すことができます。リーダーの状態に対する影響は、制御コードに依存しています。

参照のウェブサイト：

http://msdn.microsoft.com/en-us/library/windows/desktop/aa379474%28v=vs.85%29.aspx

注釈：*5.3* セクションのコマンドはこの *API* で送信します。

例：

```
#define SCARD_SCOPE_USER   0

#define EscapeCommand 0x310000 + 3500*4
   SCARDCONTEXT         hContext;          // Resource manager context
   SCARDHANDLE       hCard;             // Card context handle
   unsigned long        dwActProtocol; // Established active protocol
   int                  retCode;
   char                 readerName [256]; // Lists reader name
   char                 rName [256];      // Reader name for connection
   BYTE                 SendBuff[262],    // APDU command buffer
                        RecvBuff[262]; // APDU response buffer
   BYTE              FWVersion [20],   // For storing firmware version
                                           message
   BYTE                 ResponseData[50]; // For storing card response
   DWORD             SendLen,       // APDU command length
                        RecvLen;       // APDU response length
void main ()
   {
      …
      rName = "ACS ACR1251 CL Reader PICC 0";      // Depends on what
                                                   reader will be used
                                                   // Should connect to
                                                   PICC interface
         retCode = SCardConnect(hContext,
            rName,
            SCARD_SHARE_DIRECT,
            SCARD_PROTOCOL_T0| SCARD_PROTOCOL_T1,
            &hCard,
            &dwActProtocol);
         if (retCode != SCARD_S_SUCCESS)
         {
            // Connection failed (may be because of incorrect reader
         name, or no card was detected)
         }
         else
         {
            // Connection successful
            RecvLen = 262;
            // Get firmware version
            SendBuff[0] = 0xE0;
            SendBuff[1] = 0x00;
            SendBuff[2] = 0x00;
            SendBuff[3] = 0x18;
            SendBuff[4] = 0x00;
```

```
SendLen = 5;
retCode = SCardControl ( hCard,
         EscapeCommand,
         SendBuff,
         SendLen,
         RecvBuff,
         RecvLen,
         &RecvLen);
if (retCode != SCARD_S_SUCCESS)
{
   // APDU sending failed
   return;
}
else
{
   // APDU sending successful
   // The RecvBuff stores the firmware version message.
   for (int i=0;i< RecvLen-5;i++)
   {
      FWVersion[i] = RecvBuff [5+i];
   }
}
// Connection successful
RecvLen = 262;

// Turn Green LED on, turn Red LED off
SendBuff[0] = 0xE0;
SendBuff[1] = 0x00;
SendBuff[2] = 0x00;
SendBuff[3] = 0x29;
SendBuff[4] = 0x01;
SendBuff[5] = 0x02;  // Green LED On, Red LED off
SendLen = 6;
retCode = SCardControl ( hCard,
         EscapeCommand,
         SendBuff,
         SendLen,
         RecvBuff,
         RecvLen,
         &RecvLen);
if (retCode != SCARD_S_SUCCESS)
{
   // APDU sending failed
   return;
}
else
{
   // APDU sending success
}
```

5.1.5. SCardTransmit

SCardTransmit 関数はサービスリクエストをスマートカードに送信するために、またはスマートカードから返されるデータを受信するために使われます。

参照のウェブサイト：

http://msdn.microsoft.com/en-us/library/windows/desktop/aa379804%28v=vs.85%29.aspx

注： *APDU* コマンド（即ち：接続を確立されたカードに送信するコマンド、***5.2.4*** セクション - *PICC* コマンドそして ***5.2.2*** セクション - 非接触インターフェースの疑似 *APDU* コマンド）はこの *API* で送信されます。

例：

```
#define SCARD_SCOPE_USER      0

SCARDCONTEXT      hContext;          // Resource manager context
SCARDHANDLE    hCard;             // Card context handle
unsigned long     dwActProtocol; // Established active protocol
int               retCode;
char              readerName [256]; // List reader name
char              rName [256];      // Reader name for connect
BYTE              SendBuff[262], // APDU command buffer
                  RecvBuff[262]; // APDU response buffer
BYTE           CardID [8],          // For storing the FeliCa IDM/
                                    MIFARE UID
BYTE              ResponseData[50]; // For storing card response
DWORD          SendLen,       // APDU command length
                  RecvLen;       // APDU response length
SCARD_IO_REQUEST  ioRequest;

void main ()
{
   …
   rName = "ACS ACR1251 CL Reader PICC 0";    // Depends on what reader
                                              should be used
                                              // Should connect to PICC
                                              interface
      retCode = SCardConnect(hContext,
                     rName,
                     SCARD_SHARE_SHARED,
                     SCARD_PROTOCOL_T0,
                     &hCard,
                     &dwActProtocol);
      if (retCode != SCARD_S_SUCCESS)
      {
         // Connection failed (May be because of incorrect reader
         name, or no card was detected)
      }
      else
      {
         // Connection successful
         ioRequest.dwProtocol = dwActProtocol;
         ioRequest.cbPciLength = sizeof(SCARD_IO_REQUEST);
         RecvLen = 262;
```

```
        // Get MIFARE UID/ FeliCa IDM
        SendBuff[0] = 0xFF;
        SendBuff[1] = 0xCA;
        SendBuff[2] = 0x00;
        SendBuff[3] = 0x00;
        SendBuff[4] = 0x00;
        SendLen = 5;
        retCode = SCardTransmit( hCard,
                                 &ioRequest,
                                 SendBuff,
                                 SendLen,
                                 NULL,
                                 RecvBuff,
                                 &RecvLen);

        if (retCode != SCARD_S_SUCCESS)
        {
           // APDU sending failed
           return;
        }
        else
        {
           // APDU sending successful
           // The RecvBuff stores the IDM for FeliCa / the UID for
           MIFARE.
           // Copy the content for further FeliCa access
           for (int i=0;i< RecvLen-2;i++)
           {
              CardID [i] = RecvBuff[i];
           }
        }
```

5.1.6. SCardDisconnect

SCardDisconnect 関数は前に確立されたアプリケーションとターゲットリーダー間の接続を終了するためです。。

参照のウェブサイト：

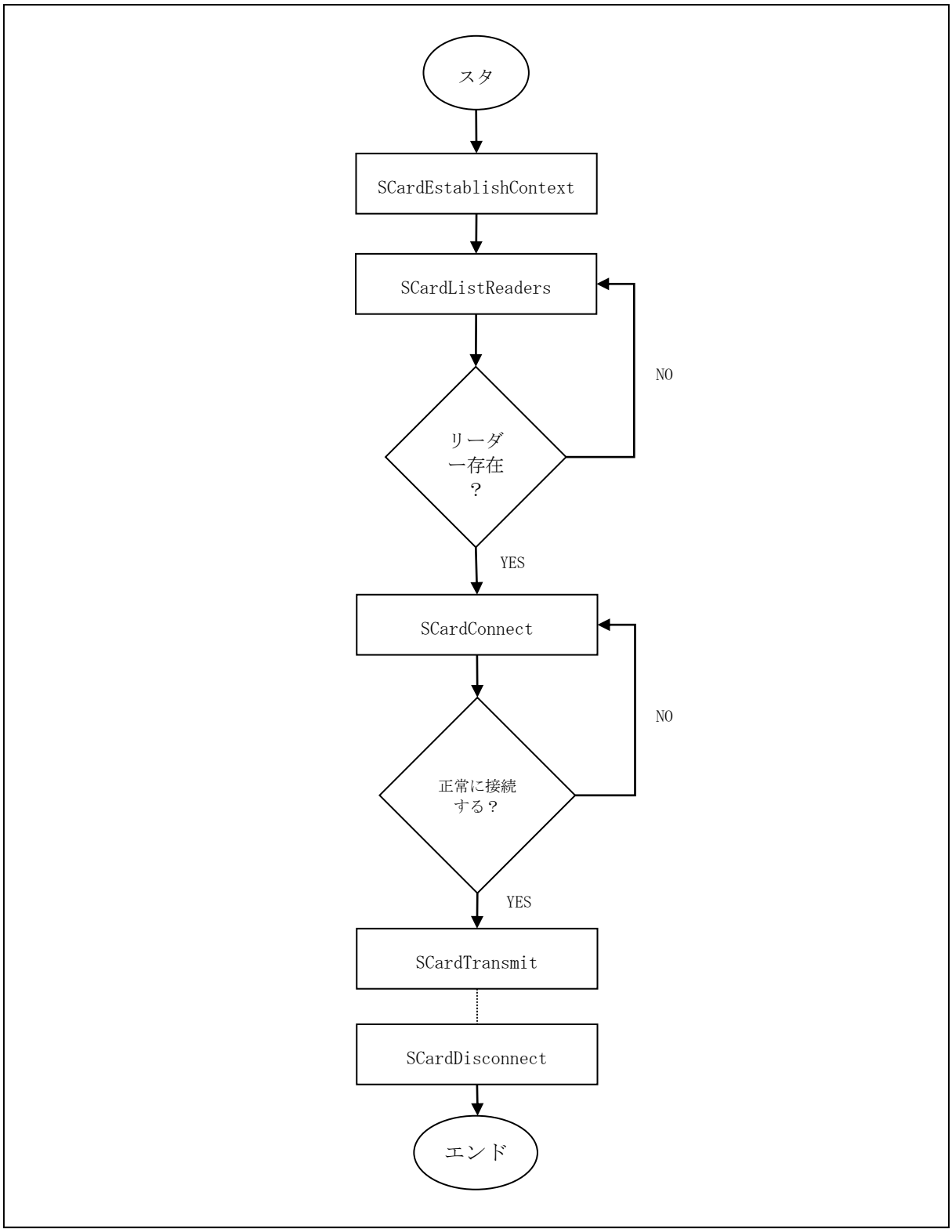
http://msdn.microsoft.com/en-us/library/windows/desktop/aa379475%28v=vs.85%29.aspx

この関数 PCSC 操作を終止します。.

例：

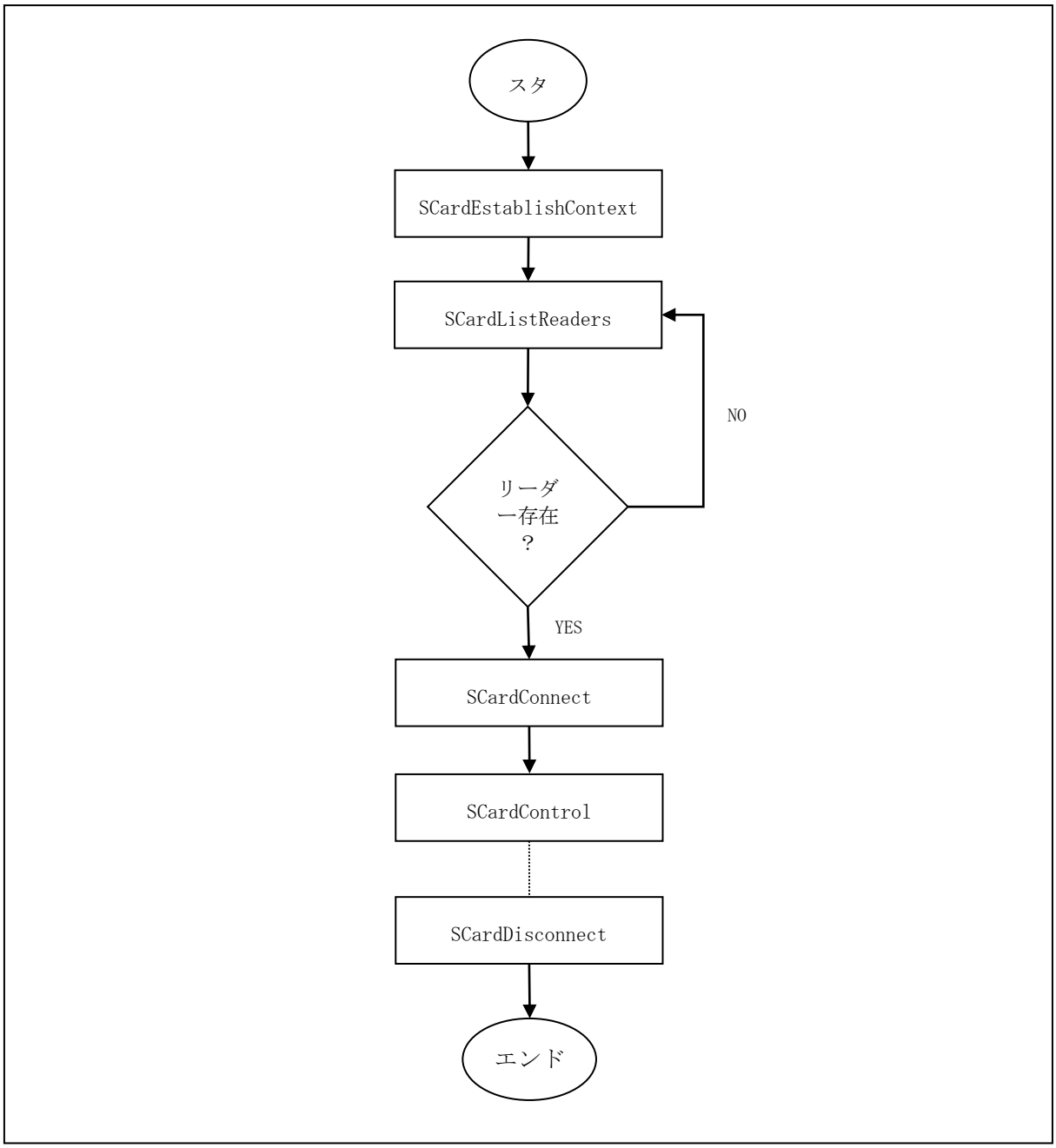
```
#define SCARD_SCOPE_USER 0

SCARDCONTEXT      hContext;          // Resource manager context
SCARDHANDLE    hCard;             // Card context handle
unsigned long     dwActProtocol; // Established active protocol
int               retCode;

void main ()
{
   …
         // Connection successful
      …
      retCode = SCardDisconnect(hCard,SCARD_RESET_CARD);
      if (retCode != SCARD_S_SUCCESS)
      {
         // Disconnection failed
      }
      else
      {
         // Disconnection successful
      }
   }
}
```

5.1.7. APDU の流れ



図示 2:ACR1251T の APDU の流れ

5.1.8. ダイレクトコマンド（Escape Command）の流れ



図示 3:ACR1251T　直接コマンド（Escape Command）の流れ

5.2. 非接触スマートカード プロトコル

5.2.1. ATR の生成

リーダーが PICC を検出すると、PICC を識別するために、ATR が PC/SC ドライバに送されます。

5.2.1.1. ATR フォーマット（ISO 14443-3 PICC に適用）

バイト	Value	標記	説明
0	3Bh	最初のヘッダー	-
1	8Nh	T0	高いニブル 8 の意味は：TA1、TB1 と TC1 がなくて、TD1 だけが続いている。 下位ニブル N は歴史的なバイトの数です（HistByte 0 - HistByte N-1)
2	80h	TD1	高いニブル 8 の意味は：TA2、TB2 と TC2 がなくて、TD2 だけが続いている。 下位ニブル 0 の意味は T=0
3	01h	TD2	高いニブル 0 の意味は：TA3、TB3、TC3 および TD3 が全部続いていない。 下位ニブル 1 の意味は T=1
4 から 3+N	80h	T1	カテゴリインジケータバイトは、80 のステータスインジケータが任意の COMPACT-TLV データオブジェクトに存在するかもしれない意味です。
	4Fh	Tk	アプリケーション識別子にはインジケータが存在している
	0Ch		Length
	RID		登録されたアプリケーションプロバイダ識別子（RID）# A0 00 00 03 06
	SS		基準のバイト
	C0 ..C1h		カードネームバイト
	00 00 00 00h	RFU	RFU # 00 00 00 00
4+N	UU	TCK	T0 から Tk までのすべてのバイトの排他的論理和

例：

MIFARE Classic 1K カード ATR = {3B 8F 80 01 80 4F 0C A0 00 00 03 06 03 00 01 00 00 00 00 6Ah}

その中：

長さ (YY) = 0Ch

RID = A0 00 00 03 06h (PC/SC ワークグループ)

基準 （SS） = 03h （ISO 14443A、3 パート）

カードネーム （C0 ..C1） ={00 01h} (MIFARE Classic 1K)

基準 （SS） = 03h：ISO 14443A、3 パート

= 11h：FeliCa

カードネーム(C0 ..C1) 00 01: MIFARE Classic 1K 00 38: MIFARE Plus® SL2 2K

00 02: MIFARE Classic 4K 00 39: MIFARE Plus® SL2 4K

00 03: MIFARE Ultralight® 00 30: Topaz と Jewel

00 26: MIFARE Mini® 00 3B: FeliCa

00 3A: MIFARE Ultralight® C FF 28: JCOP 30

00 36: MIFARE Plus® SL1 2K FF [SAK]: 定義していないタグ

00 37: MIFARE Plus® SL1 4K

5.2.1.2. ATR フォーマット（ISO 14443-4 PICC に適用）

<table>
<tr><th>バイト</th><th>Value</th><th>標記</th><th>説明</th></tr>
<tr><td>0</td><td>3Bh</td><td>最初のヘッダー</td><td>-</td></tr>
<tr><td>1</td><td>8N</td><td>T0</td><td>高いニブル 8 の意味は：TA1、TB1 と TC1 がなくて、TD1 だけが続いている。
下位ニブル N は歴史的なバイトの数です（HistByte 0 - HistByte N-1)</td></tr>
<tr><td>2</td><td>80h</td><td>TD1</td><td>高いニブル 8 の意味は：TA2、TB2 と TC2 がなくて、TD2 だけが続いている。
下位ニブル 0 の意味は T=0</td></tr>
<tr><td>3</td><td>01h</td><td>TD2</td><td>高いニブル 0 の意味は：TA3、TB3、TC3 および TD3 が全部続いていない。
下位ニブル 1 の意味は T=1</td></tr>
<tr><td rowspan="2">4
から
3 + N</td><td>XX</td><td>T1</td><td rowspan="2">歴史的なバイト

ISO 14443-A：
ATS 応答の歴史的なバイト。ISO 14443-4 基準を参照してください。

ISO 14443-B：
<table>
<tr><th>Byte1-4</th><th>Byte5-7</th><th>Byte8</th></tr>
<tr><td>ATQB のアプリケーションデータ</td><td>ATQB からのプロトコル情報バイト</td><td>高いニブル =ATTRIB コマンドの MBLI；
下位ニブル（RFU）=0</td></tr>
</table></td></tr>
<tr><td>XX
XX
XX</td><td>Tk</td></tr>
<tr><td>4+N</td><td>UU</td><td>TCK</td><td>T0 から Tk までのすべてのバイトの排他的論理和</td></tr>
</table>

例 1：MIFARE® DESFire® 的 ATR = {3B 81 80 01 80 80h} // 6 bytes of ATR

注釈：APDU“FF CA 01 00 00h”を使用して、ISO 14443A-4 の PICC に準拠しているまたは ISO 14443B-4 の PICC に準拠していることを区別します。可能な場合、完全な ATS を取得します。*ISO 14443A-3* または *ISO 14443B-3/4* の *PICC* に準拠する場合、*ATS* が返される。

APDU コマンド = FF CA 01 00 00h

APDU 応答 = 06 75 77 81 02 80 90 00h

ATS = {06 75 77 81 02 80h}

例 2：EZ-link の ATR = {3B 88 80 01 1C 2D 94 11 F7 71 85 00 BEh}

ATQB の応答データ = 1C 2D 94 11h

ATQB からのプロトコル情報 = F7 71 85h

ATTRIB の MBLI =00h

5.2.2. 非接触インターフェースの疑似 APDU コマンド

5.2.2.1. データを取得する（Get Data）

GET DATA コマンドは“接続された PICC”のシリアルナンバーもしくは ATS を取得します。

GET UID の APDU フォーマット（5 バイト）

コマンド	CLA	INS	P1	P2	Le
Get Data	FFh	CAh	00h 01h	00h	00h （最大長さ）

応答	データ出力					
結果	UID (LSB)	…	…	UID (MSB)	SW1	SW2

例え **P1 = 01h**、ISO14443 A タイプのカードの ATS を入手する（ATS + 2 バイト）

応答	データ出力		
結果	ATS	SW1	SW2

応答コード

結果	SW1	SW2	意味
成功	90h	00h	操作が成功に完了しました。
警告	62h	82h	UID/ATS の終わりが Le バイトの前に達しました（Le は UID の長さより大きいです）
エラー	6Ch	XXh	間違った長さ（間違ったナンバーLe：‘XX’は精確な数字を表す）、Le は、利用可能な UID の長さ未満である場合
エラー	63h	00h	操作が失敗しました。
エラー	6Ah	81h	この機能をサポートできません。

例：

“接続された PICC”のシリアルナンバーを取得します

```
UINT8 GET_UID[5] = {FF, CA, 00, 00, 00};
```

“接続された ISO 14443-A PICC”の ATS を取得します

```
UINT8 GET_ATS[5] = {FF, CA, 01, 00, 00};
```

5.2.3. PCSC 2.0 パート 3 の APDU コマンド（2.02 もしくはもっと新しいバージョン）

PCSC 2.0 パート 3 のコマンドが透過的にアプリケーションからデータを非接触タグへ渡し、アプリケーションとプロトコルに透過的に受信したデータを返し、同時にプロトコルを切り替えるために使用されています。

5.2.3.1. コマンドと応答の APDU フォーマット

コマンドのフォーマット

CLA	INS	P1	P2	Lc	データイン
FFh	C2h	00h	機能	DataLen	Data[DataLen]

その中：

機能　1 バイト。

00h = セッション管理

01h = 透明交換

02h = プロトコルの切り替え

他 = RFU

応答フォーマット

データ出力	SW1	SW2
符号化されました BER-TLV データフィールド		

すべてのコマンドは、レスポンスデータフィールド（利用可能な場合）と一緒に SW1 と SW2 を返します。SW1 と SW2 は ISO7816 に基づいて、以下の C0 データオブジェクトの SW1 SW2 も使用する必要があります。

C0 データ要素のフォーマット

タグ	長さ（1 バイト）	SW2
C0h	03h	エラーステータス

エラーステータスの説明

エラーステータス	説明
XX SW1 SW2	XX = APDU 内の不正なデータオブジェクトの数量 00 = APDU の一般的なエラー 01 = 一番目のデータオブジェクト内のエラー 02 = 二番目のデータオブジェクト内のエラー
00 90 00h	エラーは発生していません
XX 62 82h	データオブジェクトの XX 警告、要求された情報は存在していません
XX 63 00h	情報なし
XX 63 01h	実行は他のデータオブジェクトの障害のため停止しました
XX 6A 81h	データオブジェクトはサポートされていません XX
XX 67 00h	予期しない長さのデータオブジェクト XX
XX 6A 80h	予期しない値のデータオブジェクト XX
XX 64 00h	データオブジェクト XX の実行エラー（IFD からの応答がありません）
XX 64 01h	データオブジェクト XX の実行エラー（ICC からの応答がありません）
XX 6F 00h	データオブジェクト XX は正確な診断なしで失敗しました

最後の 2 バイトは、エラーの説明を示しながら、一番目のバイトの数値は、誤ったデータオブジェクトの XX の数を示します。ISO7816 に基づいて、SW1 SW2 の値が許可されています。

C-APDU データフィールドには複数のデータオブジェクトがあって、1 つのデータオブジェクトが失敗した場合、他のデータオブジェクトが失敗したデータオブジェクトに依存しない場合、IFD は次のデータオブジェクトを処理することができます。

5.2.3.2. セッションを管理するコマンド（Manage Session Command）

このコマンドは、透明なセッションを管理するために使用されます。起動と透明セッションの終了が含まれています。このコマンドを使用して、ユーザーは動作環境や透明セッション内の IFD の機能を管理することができます。

セッションを管理するコマンド

コマンド	CLA	INS	P1	P2	Lc	データイン
Manage Session	FFh	C2h	00h	00h	データ長さ	データオブジェクト（N バイト）

その中：

データオブジェクト（1 バイト)

タグ	データオブジェクト
80h	バージョンのデータオブジェクト
81h	透明セッションを開始する
82h	透明セッションを終了する
83h	RF フィールドをオフにする
84h	RF フィールドをオンにする
5F 46h	タイマー
FF 6Dh	パラメ―タ―を取得する
FF 6Eh	パラメ―タ―を設定する

セッション管理の応答データオブジェクト

タグ	データオブジェクト
C0h	一般的なエラーステータス
80h	バージョンのデータオブジェクト
FF 6Dh	IFD パラメ―タ―データオブジェクト

5.2.3.2.1. セッションデータオブジェクトを開始する（Start Session Data Object）

このコマンドは、透過的なセッションを開始するために使用されています。セッションが開始されると、セッションが終了されるまで、自動ポーリングが無効になります。

セッションデータオブジェクトを開始する

タグ	長さ（1 バイト）	数値
81h	00h	-

5.2.3.2.2. セッションデータオブジェクトを終了する（End Session Data Object）

このコマンドは、透過的なセッションを終了するために使用されています。セッションが開始される前に自動ポーリング状態にリセットされます。

セッションデータオブジェクトを終了する

タグ	長さ（1 バイト）	数値
82h	00h	-

5.2.3.2.3. バージョンのデータオブジェクト

このコマンドは、IFD Handler のバージョン番号を返すために使用されます。

バージョンのデータオブジェクト

タグ	長さ（1 バイト）	数値		
80h	03h	メジャーのバージョン	マイナーのバージョン	内部のバージョン

5.2.3.2.4. RF データオブジェクトをオフにする（Turn Off the RF Data Object）

このコマンドはアンテナフィールドをオフにする時に使われます。

RF データオブジェクトをオフにする

タグ	長さ（1 バイト）	数値
83h	00h	-

5.2.3.2.5. RF データオブジェクトをオンにする（Turn On the RF Data Object）

このコマンドはアンテナフィールドをオンにする時に使われます。

RF データオブジェクトをオンにする

タグ	長さ（1 バイト）	数値
84h	00h	-

5.2.3.2.6. タイマーデータオブジェクト（Timer Data Object）

このコマンドは、1 µs の単位で 32 ビットのタイマーデータオブジェクトを作成するために使用されます。

例：RF をオフにするデータオブジェクトと RF をオンにするデータオブジェクト間には 5000µs のタイマデータオブジェクトがある場合、RF がオンになっている前に、りーだーは 5000µs 程度の RF フィールドをオフにします。

タイマーデータオブジェクト

タグ	長さ（1 バイト）	数値
5F 46h	04h	タイマー（4 バイト）

5.2.3.2.7. パラメータデータオブジェクトを取得する（Get Parameter Data Object）

このコマンドは、IFD から異なるパラメータを取得するために使用されます。

パラメータデータオブジェクトを取得する

タグ	長さ（1 バイト）	数値		
		タグ	長さ	数値
FF 6Dh	バール	TLV_Objects		

TLV_Objects

要求のパラメータ	タグ	長さ
IFD フレームサイズの整数（FSDI）	01h	00h
ICC フレームサイズの整数（FSCI）	02h	00h
フレーム待ち時間の整数（FWTI）	03h	00h
IFD でサポートされている最大な通信速度	04h	00h
ICC の通信速度	05h	00h

要求のパラメータ	タグ	長さ
指数変調	06h	00h
ISO/IEC14443 基準の PCB	07h	00h
ISO/IEC14443 基準の CID	08h	00h
ISO/IEC14443 基準の NAD	09h	00h
ISO/IEC14443 B タイプのパラメ—タ—1 - 4	0Ah	00h

5.2.3.2.8. パラメータデータオブジェクトを設定する（Set Parameter Data Object）

このコマンドは、IFD とは異なるパラメータを設定するために使用されます。

パラメータデータオブジェクトを設定する

タグ	長さ（1 バイト）	数値		
		タグ	長さ	数値
FF 6Eh	バール	TLV_Objects		

TLV_Objects

要求のパラメータ	タグ	長さ
IFD フレームサイズの整数（FSDI）	01h	01h
ICC フレームサイズの整数（FSCI）	02h	01h
フレーム待ち時間の整数（FWTI）	03h	01h
IFD でサポートされている最大な通信速度	04h	01h
ICC の通信速度	05h	01h
指数変調	06h	01h
ISO/IEC14443 基準の PCB	07h	01h
ISO/IEC14443 基準の CID	08h	01h
ISO/IEC14443 基準の NAD	09h	01h
ISO/IEC14443 B タイプのパラメ—タ—1 - 4	0Ah	04h

5.2.3.3. 透明交換のコマンド（Transparent Exchange Command）

このコマンドは、送信および ICC から任意のビットまたはバイトを受信するために使用されます。

透明交換のコマンド

コマンド	CLA	INS	P1	P2	Lc	データイン
TranspEx	FFh	C2h	00h	01h	DataLen	データオブジェクト（N バイト）

その中：

データオブジェクト（1 バイト)

タグ	データオブジェクト
90h	送受信フラグ
91h	伝送ビットフレーミング
92h	受信ビットフレーミング
93h	送信
94h	受信
95h	送受信ー送信と受信
FF 6Dh	パラメ—タ—を取得する
FF 6Eh	パラメ—タ—を設定する

透明交換の応答データオブジェクト

タグ	データオブジェクト
C0h	一般的なエラーステータス
92h	受信したデータの最後のバイトでの有効なビットの数量
96h	応答メッセージの状態コード
97h	ICC 応答
FF 6Dh	IFD パラメ—タ—データオブジェクト

5.2.3.3.1. 送受信のフラグデータオブジェクト（Transmission and Reception Flag Data Object）

このコマンドは、次の送信のためのフレーミングおよび RF パラメータを定義するために使用されます。

送受信のフラグデータオブジェクト

タグ	長さ（1 バイト）	数値	
		ビット	説明
90h	02h	0	0 – 送信したデータに CRC を追加します 1 – 送信したデータに CRC を追加しません
		1	0 – 受信したデータから CRC を破棄します 1 – 受信したデータから CRC を破棄しません（CRC チェックなし）
		2	0 – 送信したデータにパリティを挿入します 1 – 送信したデータにパリティを挿入しません
		3	0 – 受信したデータのパリティを期待します 1 – パリティを期待していません（パリティチェックなし）
		4	0 – 送信したデータのプロトコルプロローグを追加したり、応答から捨てます 1 - 追加またはプロトコルのプロローグを破棄することはありません（ある場合）（例えば、PCB、CID、NAD）
		5-15	RFU

5.2.3.3.2. ビットフレーミングデータオブジェクトを送信する（Transmission Bit Framing Data Object）

このコマンドは、送受信されていないデータの最後のバイトの有効ビット数を定義するために使用されます。

ビットフレーミングデータオブジェクトを送信する

タグ	長さ（1 バイト）	数値	
		ビット	説明
91h	01h	0-2	最後のバイトの有効ビット数（0 はすべてのビットが有効であることを意味します）
		3-7	RFU

伝送ビットフレーミングデータオブジェクトは、「送信」または「送受信」のみのデータオブジェクトと一緒でなければなりません。このデータオブジェクトが存在しない場合、それはすべてのビットが有効であることを意味します。

Z

5.2.3.3.3. ビットフレーミングデータオブジェクトを受信する（Reception Bit Framing Data Object）

コマンド APDU の場合、このデータオブジェクトは、受信されたデータの最後のバイトの予期な有効ビット数を定義します。

コマンド APDU の場合、このデータオブジェクトは、受信されたデータの最後のバイトの予期な有効ビット数を通知します。

ビットフレーミングデータオブジェクトを受信する

タグ	長さ（1 バイト）	数値	
		ビット	説明
92h	01h	0-2	最後のバイトの有効ビット数（0 はすべてのビットが有効であることを意味します）
		3-7	RFU

このデータオブジェクトが存在しない場合、それはすべてのビットが有効であることを意味します。

5.2.3.3.4. データオブジェクトを送信する（Transmit Data Object）

このコマンドは、IFD から ICC にデータを送信するために使用されます。送信が完了した後、ICC からの応答が予想されていません。

データオブジェクトを送信する

タグ	長さ（1 バイト）	数値
93h	DataLen	データ（N バイト）

5.2.3.3.5. データオブジェクトを受信する（Receive Data Object）

このコマンドは、次のタイマーオブジェクトに与えられた時間内に受信モードに入るために、リーダーを強制する時に使用されます。

データオブジェクトを受信する

タグ	長さ（1 バイト）	数値
94h	00h	-

5.2.3.3.6. データオブジェクトを送受信する（Transceive Data Object）

このコマンドは、ICC からのデータを送受信するために使用されます。送信が完了すると、リーダーは、タイマーデータオブジェクトに指定された時間まで待機します。

何のタイマデータオブジェクトは、データフィールドで定義されていない場合、リーダーは Set Parameter FWTI データオブジェクトに指定された期間を待っています。FWTI が設定されていない場合、リーダーは、約 302 µs を待ちます。

データオブジェクトを送受信する

タグ	長さ（1 バイト）	数値
95h	DataLen	データ（N バイト）

5.2.3.3.7. ステータスデータオブジェクトを応答する（Response Status Data Object）

応答内では、このコマンドが受信されたデータの状態を通知するために使用されます。

ステータスデータオブジェクトを応答する

タグ	長さ（1 バイト）	数値		
		バイト 0		バイト 1
		ビット	説明	
96h	02h	0	0 - CRC が OK、若しくはチェックしていません 1 - CRC チェックが失敗しました	衝突が検出された場合、これらのバイトは、衝突位置を教えてくれます。じゃないと“00h”を表示します。
		1	0 – 衝突なし 1 – 衝突が検出されました	
		2	0 – パリティエラーなし 1 – パリティエラーが検出されました	
		3	0 – フレームエラーなし 1 – フレームエラーが検出されました	
		4 - 7	RFU	

5.2.3.3.8. データフォーマットを応答する

応答内では、このコマンドが受信されたデータの状態を通知するために使用されます。

データフォーマットを応答する

タグ	長さ（1 バイト）	数値
97h	DataLen	応答データ（N バイト）

5.2.3.4. プロトコルを切り替えるコマンド（Switch Protocol Command）

このコマンドは、プロトコルと透明セッション内の標準の異なる層を指定するために使用されます。

プロトコルを切り替えるコマンド

コマンド	CLA	INS	P1	P2	Lc	データイン
SwProtocol	FFh	C2h	00h	02h	DataLen	データオブジェクト（N バイト）

その中：

データオブジェクト（1 バイト)

タグ	データオブジェクト
8Fh	プロトコルデータオブジェクトを切り替える
FF 6Dh	パラメ—ター—を取得する
FF 6Eh	パラメ—ター—を設定する

プロトコルの応答データオブジェクトを切り替える

タグ	データオブジェクト
C0h	一般的なエラーステータス
FF 6Dh	IFD パラメ—ター—データオブジェクト

5.2.3.4.1. プロトコルデータオブジェクトを切り替える（Switch Protocol Data Object）

このコマンドは、プロトコルおよび規格の異なる層を指定するために使用されます。

プロトコルデータオブジェクトを切り替える

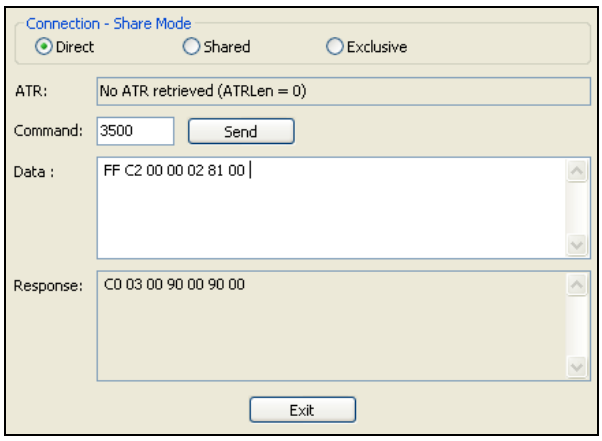
タグ	長さ（1 バイト）	数値	
		バイト 0	バイト 1
8Fh	02h	00h – ISO/IEC14443 Type A 01h – ISO/IEC14443 Type B 03h – FeliCa 他 – RFU	00h – 層分離がない場合 02h – ２層に切り替える 03h – ３層に切り替えるまたは活性化する 04h – ４層に活性化する 他 - RFU

5.2.3.4.2. PCSC 2.0 パート 3 の例

1. 透明セッションを開始する

   コマンド：**FF C2 00 00 02 81 00**

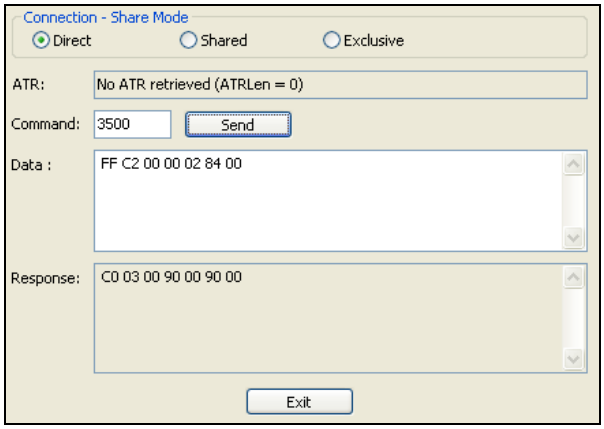
   応答：**C0 03 00 90 00 90 00**



2. アンテナフィールドをオンにする

   コマンド：**FF C2 00 00 02 84 00**

   応答：**C0 03 00 90 00 90 00**

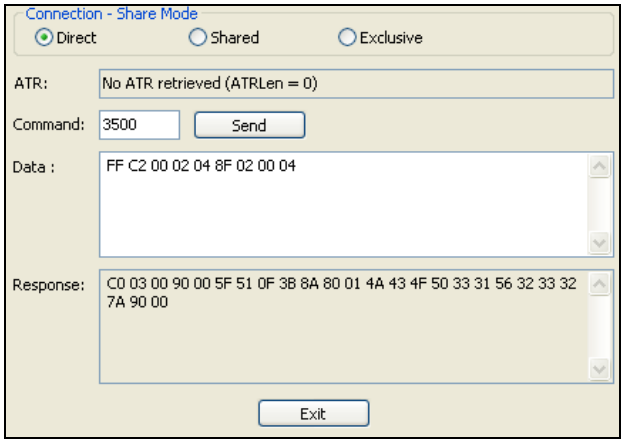


3. ISO 14443-4A 有効。

コマンド：**FF C2 00 02 04 8F 02 00 04**

応答： **C0 03 01 64 01 90 00** (カードがない場合)

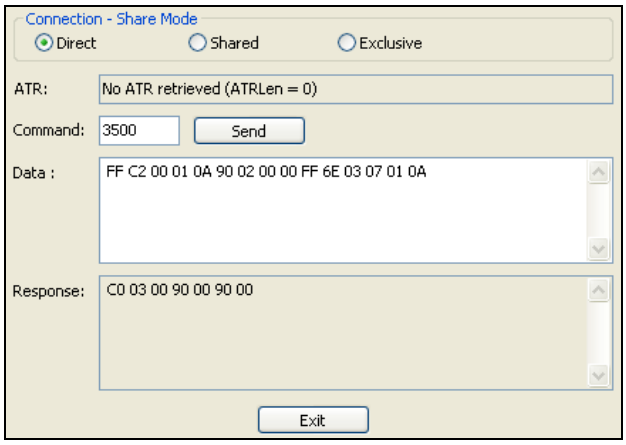
C0 3 00 90 00 5F 51 [ATR] 90 00



4. 0AH に PCB を設定し、送信データで CRC、パリティ、プロトコルプロローグを有効にします。

コマンド：**FF C2 00 01 0A 90 02 00 00 FF 6E 03 07 01 0A**

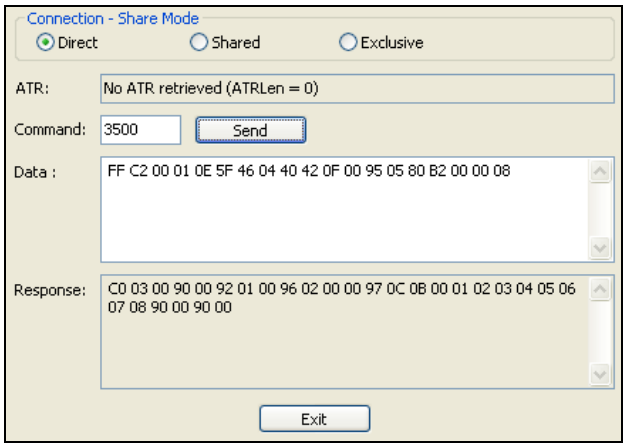
応答：**C0 03 00 90 00 90 00**



5. カードに APDU「80B2000008」を送信し、応答を取得します。

コマンド：**FF C2 00 01 0E 5F 46 04 40 42 0F 00 95 05 80 B2 00 00 08**

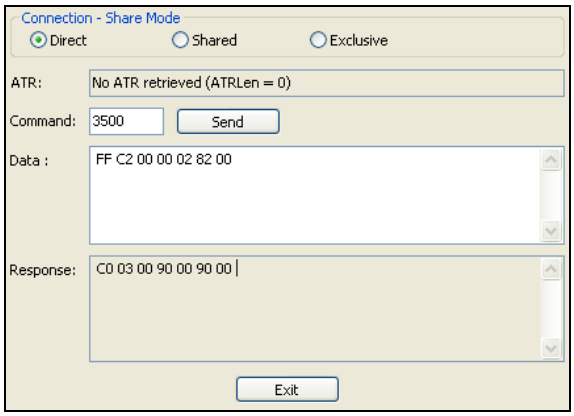
応答：**C0 03 00 90 00 92 01 00 96 02 00 00 97 0C [カードの応答] 90 00**



6. 透明セッションを終了する。

コマンド：**FF C2 00 00 02 82 00**

応答：**C0 03 00 90 00 90 00**



5.2.4. MIFARE Classic 1K/4K メモリカードの PICC コマンド（T=CL エミュレーション）

5.2.4.1. 認証キーのダンロード（Load Authentication Keys）

このコマンドはリーダーにキーをロードする時に使われる。このキーは MIFARE Classic メモリカードの特定なセクターを認証するために使用されます。リーダーは二種の認証キーのアドレスが提供られている：失いやすいキーのアドレスと失いやすくないキーのアドレス。

Load Authentication Keys APDU フォーマット（11 バイト）

コマンド	CLA	INS	P1	P2	Lc	データイン
Load Authentication Keys	FFh	82h	キー構造	キーナンバー	06h	キー（6 バイト）

その中：

キー構造 1 バイト。

00h = キーが失いやすいキーのメモリにロードされる。

その他＝予約済み

キーの番号 1 バイト。

00h – 01Fh = キーを保存するための失いやすくないキーのメモリ。キーは永遠にリーダーに保存されます。リーダーが PC から切断された場合でも、リーダーのメモリ内に保持されます。その失いやすいキーのメモリには最大３２個キーをストアーできる。

注釈：デフォルト値は FF FF FF FF FF FFh です。

キー 6 バイト。

リーダーにロードされるきーの値。例： FF FF FF FF FF FFh。

Load Authentication Keys 応答フォーマット（2 バイト）

応答	データ出力	
結果	SW1	SW2

Load Authentication Keys 応答コード

結果	SW1	SW2	意味
成功	90h	00h	操作が成功に完了しました。
エラー	63h	00h	操作が失敗しました。

例：

// 失いやすいキーのメモリに 00h キーをロードする {FF FF FF FF FF FFh}。

APDU = {FF 82 00 00 06 FF FF FF FF FF FFh}

5.2.4.2. MIFARE Classic (1K/4K)カードに対しての認証（Authentication for MIFARE Classic (1K/4K）

このコマンドはリーダーにストアーられているキーで MIFARE Classic® カード（PICC）を認証する時に使われます。二種の認証キーを使われる：TYPE_A と TYPE_B。

Load Authentication Keys APDU フォーマット（6 バイト）「廃止された」

コマンド	CLA	INS	P1	P2	P3	データイン
Authentication	FFh	88h	00h	ブロック番号	キーのタイプ	キーナンバー

Load Authentication Keys APDU フォーマット（10 バイト）

コマンド	CLA	INS	P1	P2	Lc	データイン
Authentication	FFh	86h	00h	00h	05h	データバイト認証

データバイト認証（5 バイト）

バイト 1	バイト 2	バイト 3	バイト 4	バイト 5
バージョン番号 01h	00h	ブロック番号	キーのタイプ	キーナンバー

その中：

ブロック番号 1 バイト。認証されていないメモリブロック。

一枚の MIFARE Classic 1K カードが 16 個と分けて、各セクターには 4 個の連続的なブロックが含めています。例：セクター00h が含めているブロック{00h、01h、02h および 03h}；セクター01h が含めているブロック{04h、05h、06h および 07h}；ラストセクター0Fh が含めているブロック{3Ch、3Dh、3Eh および 3Fh}。当ブロックが成功に認証されると、同じセクターの全てのブロックをアクセスできる。詳しい情報は MIFARE Classic 1K/4 k 基準を参照してください。

***注釈：**ブロックが正常に認証されると、同セクターに所属する全てのブロックがアクセス可能である。

キーのタイプ 1 バイト。

60h = TYPE A キーとして、認証用に使われる。

61h = TYPE B キーとして、認証用に使われます。

キーの番号 1 バイト。

00h – 01h = キーを保存するための失いやすくないキーのメモリ。リーダーが PC から切断された時、キーが消えます。二つの失いやすいキーを提供しています。 これらは異なるセッションにセッション鍵として使用できる。

Load Authentication Keys 応答フォーマット（2 バイト）

応答	データ出力	
結果	SW1	SW2

Load Authentication Keys 応答コード

結果	SW1	SW2	意味
成功	90	00h	操作が成功に完了しました。
エラー	63	00h	操作が失敗しました。

例：

// {TYPE A，キーナンバーの 00h}によって、ブロック 04h を認証します。PC/SC V2.01，廃止されます

APDU = {FF 88 00 04 60 00h};

// {TYPE A，キーナンバーの 00h}によって、ブロック 04h を認証します。PC/SC V2.07

APDU = {FF 86 00 00 05 01 00 04 60 00h}

セクター (16 個のセクター，各セクターには 4 個の連続的なブロックが含めている)	データブロック (3 つのブロック，各には 16 バイト)	トレーラーブロック (1 つのブロック，16 バイト)
セクター0	00h – 02h	03h
セクター1	04h – 06h	07h
..	..	..
..	..	..
セクター14	38h – 0Ah	3Bh
セクター15	3Ch – 3Eh	3Fh

1 KB

表2 ：MIFARE Classic 1K カードのメモリマップ

セクター (32 個のセクター，各セクターには 4 個の連続的なブロックが含めている)	データブロック (3 つのブロック，各には 16 バイト)	トレーラーブロック (1 つのブロック，16 バイト)
セクター0	00h ~ 02h	03h
セクター1	04h ~ 06h	07h
..		
..		
セクター30	78h ~ 7Ah	7Bh
セクター31	7Ch ~ 7Eh	7Fh

2 KB

セクター (8 個のセクター，各セクターには 16 個の連続的なブロックが含めている)	データブロック (15 つのブロック，各には 16 バイト)	トレーラーブロック (1 つのブロック，16 バイト)
セクター32	80h ~ 8Eh	8Fh
セクター33	90h ~ 9Eh	9Fh
..		
..		
セクター38	E0h ~ EEh	EFh
セクター39	F0h ~ FEh	FFh

2 KB

表3 ：MIFARE Classic 4K カードのメモリマップ

例：

// {TYPE A、キーナンバーの 00h}によって、ブロック 04h を認証します。

// PC/SC V2.01，廃止される

APDU = {FF 88 00 04 60 00h};

// {TYPE A、キーナンバーの 00h}によって、ブロック 04h を認証します。

// PC/SC V2.07

APDU = FF 86 00 00 05 01 00 04 60 00h

バイトナンバー	0	1	2	3	ページ
シリアルナンバー	SN0	SN1	SN2	BCC0	0
シリアルナンバー	SN3	SN4	SN5	SN6	1
内部/ロック	BCC1	内部	Lock0	Lock1	2
OTP	OPT0	OPT1	OTP2	OTP3	3
データリーダー/ライター	Data0	Data1	Data2	Data3	4
データリーダー/ライター	Data4	Data5	Data6	Data7	5
データリーダー/ライター	Data8	Data9	Data10	Data11	6
データリーダー/ライター	Data12	Data13	Data14	Data15	7
データリーダー/ライター	Data16	Data17	Data18	Data19	8
データリーダー/ライター	Data20	Data21	Data22	Data23	9
データリーダー/ライター	Data24	Data25	Data26	Data27	10
データリーダー/ライター	Data28	Data29	Data30	Data31	11
データリーダー/ライター	Data32	Data33	Data34	Data35	12
データリーダー/ライター	Data36	Data37	Data38	Data39	13
データリーダー/ライター	Data40	Data41	Data42	Data43	14
データリーダー/ライター	Data44	Data45	Data46	Data47	15

512 ビット
または
64 バイト

表4 ：MIFARE Ultralight カードのメモリマップ

注釈：MIFARE Ultralight のメモリは自由にアクセスできる。認証はいりません。

5.2.4.3. バイナリブロックの読み取る（Read Binary Blocks）

複数のデータブロックを PICC カードから取り出すことに使われます。このコマンドを実行する前に、データブロック/トレーラーブロックを認証しなければなりません

Read Binary の APDU フォーマット（5 バイト）

コマンド	CLA	INS	P1	P2	Le
Read Binary Blocks	FFh	B0h	00h	ブロック番号	更新していないバイト

その中：

ブロックの番号 1 バイト。開始ブロック

更新されていない 1 バイト。

更新していない MIFARE Classic 1K/4K のバイトは 16 の倍数です；更新していない MIFARE Ultralight のバイトは 4 の倍数です。

更新していない MIFARE Ultralight のバイトは最大に１６です。

更新していない MIFARE Classic 1K カードのバイトは最大に 48 です。（複数のブロックモード；3 個の連続のブロック）

更新していない MIFARE Classic 4K カードのバイトは最大に 240 です。（複数のブロックモード；１５個の連続のブロック）

例 1：10h（16 バイト）。開始ブロックだけ（単一のブロックモード）

例 2：40h（64 バイト）。開始ブロックから開始ブロックまで+3（複数のブロックモード）

注釈：安全のために、複数のブロックモードはデータブロックだけにアクセスすることに使用されます。トレーラーブロックは複数のブロックモードでアクセスされません。単一のブロックモードを使用してください。

Read Binary Block の応答フォーマット（4/16 の倍数 + 2 バイト）

応答	データ出力		
結果	データ（4/16 バイトの倍数）	SW1	SW2

Read Binary Block 応答コード

結果	SW1	SW2	意味
成功	90h	00h	操作が成功に完了しました。
エラー	63h	00h	操作が失敗しました。

例：

// バイナリブロックの 04h から 16 バイトを読み出す（MIFARE Classic 1K/4K）

APDU = FF B0 00 04 10h

// バイナリブロック 80h から 240 バイトを読み出す（MIFARE Classic 4K）

// ブロック 80 からブロック 8Eh まで（15 個ブロック）

APDU = FF B0 00 80 F0h

5.2.4.4. バイナリブロックの更新（Update Binary Blocks）

Update Binary Blocks コマンドは複数のデータブロックを PICC カードに書き入れるのに使われる。このコマンドを実行する前に、データブロック/トレーラーブロックを認証しなければなりません

Update Binary の APDU フォーマット（16 の倍数 ＋5 バイト）

コマンド	CLA	INS	P1	P2	Lc	データイン
Update Binary Blocks	FFh	D6h	00h	ブロック番号	更新していないバイト	データブロック（16 バイトの倍数）

その中：

ブロックの番号 1 バイト。更新していない開始ブロック

更新していないバイト： 1 バイト。

更新していない MIFARE Classic 1K/4K のバイトは 16 の倍数です；更新していない MIFARE Ultralight カードのバイトは 4 の倍数です。

更新していない MIFARE Classic 1K カードのバイトは最大に 48 です。（複数のブロックモード；3 個の連続のブロック）

更新していない MIFARE Classic 4K カードのバイトは最大に 240 です。（複数のブロックモード； 1 5 個の連続のブロック）

ブロックデータ 16 の倍数 ＋2 バイト、または 6 バイトバイナリブロックに書き入れていないデータ。

例 1：10h（16 バイト）。開始ブロックだけ（単一のブロックモード）

例 2：30h（48 バイト）。開始ブロックから開始ブロックまで+2（複数のブロックモード）

注釈：安全のために、複数のブロックモードはデータブロックだけにアクセスすることに使用される。トレーラーブロックは複数のブロックモードでアクセスされません。単一のブロックモードを使用してください。

Update Binary Block の応答コード（2 バイト）

結果	SW1 SW2	意味
成功	90 00h	操作が成功に完了しました。
エラー	63 00h	操作が失敗しました。

例：

// MIFARE Classic 1K/4K カード中のバイナリブロック 04h のデータを{00 01 ..0Fh}に更新します

APDU = {FF D6 00 04 10 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0Fh}

//MIFARE Ultralight 中のバイナリブロック 04 h を{00 01 02 03}に更新する

APDU = {FF D6 00 04 04 00 01 02 03h}

5.2.4.5. 数値ブロックの操作（Value Block Operation） （INC, DEC, STORE）

このコマンドは数値を基づいてのトランザクションを実行する時に使われます（例：数値ブロックの数値を増える）。

Value Block Operation の APDU フォーマット（10 バイト）

コマンド	CLA	INS	P1	P2	Lc	データイン	
Value Block Operation	FFh	D7h	00h	ブロック番号	05h	VB_OP	VB_Value（4 バイト）{MSB ..LSB}

その中：

ブロックの番号 1 バイト。操作されていない数値のブロック

VB_OP 1 バイト。

00h = VB_Value をブロックにストアーして、このブロックは数値ブロックになります。

01h = VB_Value によって、数値ブロックの数値をインクリメントするこのコマンドは数値ブロックしか実行されません。

02h = VB_Value によって、数値ブロックの数値をデクリメントする。このコマンドは数値ブロックしか実行されません。

VB_Value 4 バイト。数値の操作に使用される符号付き長い整数です（4 バイト）。

例 1：Decimal –4 = {FFh, FFh, FFh, FCh}

VB_Value			
MSB			LSB
FFh	FFh	FFh	FCh

例 2：Decimal 1 = {00h, 00h, 00h, 01h}

VB_Value			
MSB			LSB
00h	00h	00h	01h

Value Block Operation の応答フォーマット（2 バイト）

応答	データ出力	
結果	SW1	SW2

Value Block Operation 応答コード

結果	SW1 SW2	意味
成功	90 00h	操作が成功に完了しました。
エラー	63 00h	操作が失敗しました。

5.2.4.6. 数値ブロックを読み取る（Read Value Block）

Read Value Block コマンドは数値ブロックの数値を入手するために使われます。数値ブロックしか実行されません。

Read Value Block フォーマット（5 バイト）

コマンド	CLA	INS	P1	P2	Le
Read Value Block	FFh	B1h	00h	ブロック番号	04h

その中：

ブロック番号 1 バイト。読み書かれていない数値ブロック。

Read Value Block の応答フォーマット（4 + 2 バイト）

応答	データ出力		
結果	Value {MSB ..LSB}	SW1	SW2

その中：

値 4 バイト。カードから返された数値で、符号付き長い整数です（4 バイト）

例 1：Decimal –4 = {FFh, FFh, FFh, FCh}

Value			
MSB			LSB
FFh	FFh	FFh	FCh

例 2：Decimal 1 = {00h, 00h, 00h, 01h}

Value			
MSB			LSB
00h	00h	00h	01h

Read Value Block コマンドの応答コード

結果	SW1	SW2	意味
成功	90h	00h	操作が成功に完了しました。
エラー	63h	00h	操作が失敗しました。

5.2.4.7. 数値ブロックをコピーする（Copy Value Block）

このコマンドは一つの数値ブロック中の数値を別の数値ブロックにコピーする時に使われます。

Copy Value Block の APDU フォーマット（7 バイト）

コマンド	CLA	INS	P1	P2	Lc	データイン	
Copy Value Block	FFh	D7h	00h	**ソース** ブロック番号	02h	03h	ターゲットブロック番号

その中：

元ブロックの番号 1 バイト。ソース値のブロックの値が目標値ブロックにコピーされる。

ターゲットブロック番号 1 バイト。復元する値ブロック。ソースとターゲット値のブロックは、必ず同じセクター内にある。

Copy Value Block の応答フォーマット（2 バイト）

応答	データ出力	
結果	SW1	SW2

Copy Value Block の応答コード

結果	SW1 SW2	意味
成功	90 00h	操作が成功に完了しました。
エラー	63 00h	操作が失敗しました。

例：

//数値"1"を数値ブロック 05h にストアーします。

APDU = {FF D7 00 05 05 00 00 00 00 01h}

// 数値ブロック 05h を読み取ります。

APDU = {FF B1 00 05 04h}

数値ブロック 05h 中の数値を数値ブロック 06h にコピーします。

APDU = {FF D7 00 05 02 03 06h}

//ブロック 05h の値を 5 にインクリメントする。

APDU = {FF D7 00 05 05 01 00 00 00 05h}

5.2.5. PC/SC 規格に準拠しているタグにアクセスする（ISO 14443-4）

基本的に、すべての ISO14443-4 に準拠したカード（PICC カード）は、ISO7816-4 の APDU を理解できます。ACR1251T カードリーダーは ISO 7816-4 の APDU および応答を交換することによって、ISO14443-4 基準に準拠しているカードと通信します。ACR1251T は内部で ISO14443 の 1 – 4 パートのプロトコルを処理します。

MIFARE Classic 1K/4K、MIFARE Mini および MIFARE Ultralight タグは T=CL エミュレーションを介してサポートされます。MIFARE タグを標準な ISO 14443-4 タグとして取り扱えばいいです。 **詳しい情報が 5.2.2 セクションを参照してください。**

ISO 7816-4 仕様の APDU フォーマット

コマンド	CLA	INS	P1	P2	Lc	データイン	Le
ISO 7816 4 パートのコマンド	-	-	-	-	データの長さ	-	応答データの予想の長さ

ISO 7816-4 仕様の応答データフォーマット（データ+2 バイト）

応答	データ出力		
結果	応答データ	SW1	SW2

一般的な ISO 7816-4 コマンドの応答コード

結果	SW1	意味
成功	90 00h	操作が成功に完了しました。
エラー	63 00h	操作が失敗しました。

典型的なシーケンスは：

1. タグを提出して、PICC インターファースと接続します。
2. タグ中の情報を読み取り/更新する。

これを実行します：

1. タグと接続する。

   タグの ATR は 3B 88 80 01 00 00 00 00 33 81 81 00 3Ah です。

   その中、

   ATQB アプリケーションのデータ= 00 00 00 00、ATQB プロトコル 情報= 33 81 81。これは ISO 14443-4 Type B タグです。

2. APDU を送信して、乱数を入手する。

   00 84 00 00 08

   >> 1A F7 F3 1B CD 2B A9 58h [90 00h]

   注：*対于 ISO 14443-4 Type A のタグに対して、APDU"FF CA 01 00 00h"によって ATS を入手する。*

例：

// ISO 14443-4 Type B PICC（ST19XR08E）から 8 バイトを読み取ります。

APDU = {80 B2 80 00 08h}

CLA = 80h

INS = B2h

P1 = 80h

P2 = 00h

Lc = なし

データ＝なし

Le = 08h

応答：00 01 02 03 04 05 06 07h [$9000h]

5.2.6. FeliCa タグのアクセス

FeliCa タグをアクセスするコマンドは、PCSC タグおよび MIFARE をアクセスするコマンドと違っています。このコマンドは FeliCa 基準に準拠して、ヘッダが追加されています。

FeliCa コマンドのフォーマット

コマンド	CLA	INS	P1	P2	Lc	データイン
FeliCa コマンド	FFh	00h	00h	00h	データの長さ	FeliCa コマンド（長さのバイトから始まる）

FeliCa の応答データフォーマット（データ+2 バイト）

応答	データ出力
結果	応答データ

例のメモリブロックデータの読み取り

1. FeliCa を接続する。

   ATR = 3B 8F 80 01 80 4F 0C A0 00 00 03 06 11 00 3B 00 00 00 00 42h

   その中：11 00 3Bh = FeliCa

2. FeliCa IDM の読み取り。

   コマンド = FF CA 00 00 00h

   RES = [IDM (8bytes)] 90 00h

   例：FeliCa IDM = 01 01 06 01 CB 09 57 03h

3. FeliCa コマンドをアクセスします。

   メモリブロックデータの読み取りを例として：

   コマンド = FF 00 00 00 10 10 06 01 01 06 01 CB 09 57 03 01 09 01 01 80 00h

   その中：

   Felica コマンド = 10 06 01 01 06 01 CB 09 57 03 01 09 01 01 80 00h

   IDM = 01 01 06 01 CB 09 57 03h

   RES = Memory Block Data

5.3. 周辺デバイス制御

リーダーの周辺機器制御コマンドは、制御コードの **SCARD_CTL_CODE（3500）**で **SCardControl** を使用して実装されています。

5.3.1. ファームウェアのバージョンを取得する（Get Firmware Version）

このコマンドはファームウェアのバージョンを入手する時に使われます。

Get Firmware Version コマンドフォーマット（５バイト）

コマンド	CLA	INS	P1	P2	Lc
Get Firmware Version	E0h	00h	00h	18h	00h

Get Firmware Version 応答フォーマット（5 バイト ＋ ファームウェアメッセージの長さ）

応答	CLA	INS	P1	P2	Le	データ出力
結果	E1h	00h	00h	00h	受信していないバイトの数量	ファームウェアのバージョン番号

例：

応答 = E1 00 00 00 0F 41 43 52 31 32 35 31 55 5F 56 36 32 31 2E 30

ファームウェアのバージョン番号 (HEX) = 41 43 52 31 32 35 31 55 5F 56 36 32 31 2E 30

ファームウェアのバージョン番号 (ASCII) = “ACR1251T_V621.0”

5.3.2. LED 制御（LED Control）

このコマンドは LED の出力を制御するために使用されます。

LED Control コマンドフォーマット（6 バイト）

コマンド	CLA	INS	P1	P2	Lc	データイン
LED Control	E0h	00h	00h	29h	01h	LED 状態

LED Control 応答フォーマット（6 バイト）

応答	CLA	INS	P1	P2	Le	データ出力
結果	E1h	00h	00h	00h	01h	LED 状態

LED 状態（1 バイト）

LED 状態	説明	説明
Bit 0	レッド	1 = ON；0 = OFF
Bit 1	グリーン	1 = ON；0 = OFF
Bit 2 - 7	RFU	RFU

5.3.3. LED 状態（LED Status）

このコマンドは LED の状態を検査するために使用されます。

LED Control コマンドフォーマット（5 バイト）

コマンド	CLA	INS	P1	P2	Lc
LED Status	E0h	00h	00h	29h	00h

LED Status 応答フォーマット（6 バイト）

応答	CLA	INS	P1	P2	Le	データ出力
結果	E1h	00h	00h	00h	01h	LED 状態

LED 状態（1 バイト）

LED 状態	説明	説明
Bit 0	レッド	1 = ON；0 = OFF
Bit 1	グリーン	1 = ON；0 = OFF
Bit 2 - 7	RFU	RFU

5.3.4. LED ステータスインジケータの動作を設定する（Set LED Status Indicator Behavior）

このコマンドは LED ステータスインジケータの動作を設定する時に使われます。

注：この設定は、失いにくいメモリに保存されます。

Set LED Status Indicator Behaviors コマンドフォーマット（6 バイト）

コマンド	CLA	INS	P1	P2	Lc	データイン
Set LED Status Indicator Behavior	E0h	00h	00h	21h	01h	操作

操作（1 バイト）

操作	モード	説明
Bit 0	カード動作、LED 点滅	LED が PIC カードが読み書きされる時に点滅する。
Bit 1	PICC ポーリング状態 LED	PICC ポーリング状態を表示 1 = 有効；0 = 無効
Bit 2	PICC 有効化される状態 LED	PICC 有効化される状態を表示 1 = 有効；0 = 無効
Bit 3 - 5	RFU	RFU
Bit 6	色を選ぶ（緑）	緑の LED が状態が変更されたを表示 1 = 有効；0 = 無効
Bit 7	色を選ぶ（赤）	赤の LED が状態が変更されたを表示 1 = 有効；0 = 無効

注：操作のデフォルト値 = *47h*

Set LED Status Indicator Behavior 応答フォーマット（6 バイト）

応答	CLA	INS	P1	P2	Le	データ出力
結果	E1h	00h	00h	00h	01h	デフォルト操作

5.3.5. LED ステータスインジケータの動作を読み取り（Read LED Status Indicator Behavior）

このコマンドは、LED の現在のデーフォルト操作を読み取ることに使用されます。

Read LED Status Indicator Behavior コマンドフォーマット（5 バイト）

コマンド	CLA	INS	P1	P2	Lc
Read LED and Buzzer Status Indicator Behaviors	E0h	00h	00h	21h	00h

Read LED Status Indicator Behavior 応答フォーマット（6 バイト）

応答	CLA	INS	P1	P2	Le	データ出力
結果	E1h	00h	00h	00h	01h	操作

操作（1 バイト）

操作	モード	説明
Bit 0	カード動作、LED 点滅	LED が PIC カードが読み書きされる時に点滅する。
Bit 1	PICC ポーリング状態 LED	PICC ポーリング状態を表示 1 = 有効；0 = 無効
Bit 2	PICC 有効化される状態 LED	PICC 有効化される状態を表示 1 = 有効；0 = 無効
Bit 3 - 5	RFU	RFU
Bit 6	色を選ぶ（緑）	緑の LED が状態が変更されたを表示 1 = 有効；0 = 無効
Bit 7	色を選ぶ（赤）	赤の LED が状態が変更されたを表示 1 = 有効；0 = 無効

注：操作のデフォルト値 = *47h*

5.3.6. 自動的な PICC のポーリングを設置する（Set Automatic PICC Polling）

このコマンドはカードリーダーのポーリングモードを設置する時に使われます。

リーダーが PC に接続されるたびに、PICC ポーリング機能が自動的に PICC のスキャンを開始して、内蔵アンテナに置かれる/から削除される PICC があるかどうか確認します。

コマンドを送信して、PICC のポーリングを無効にできます。このコマンドは PCSC Escape コマンドのインターフェースで送信されます。エネルギを節約するために、PICC が活動していない、または PICC が見つからない時、いつでもアンテナフィールドをオフにするための特別なモードが設けられている。省電力モードで、リーダーはもっと少ない電流を消費します。

注釈：この設置は失いやすいキーのメモリに保存されます。

Set Automatic PICC Polling コマンドフォーマット（6 バイト）

コマンド	CLA	INS	P1	P2	Lc	データイン
Set Automatic PICC Polling	E0h	00h	00h	23h	01h	ポーリング設定

Set Automatic PICC Polling 応答フォーマット（6 バイト）

応答	CLA	INS	P1	P2	Le	データ出力
結果	E1h	00h	00h	00h	01h	ポーリング設定

ポーリング設定（1 バイト）

ポーリング設定	モード	説明
Bit 0	自動的に PICC ポーリング	1 = 有効；0 = 無効
Bit 1	PICC が見つかっていない場合は、アンテナフィールドをオフにします。	1 = 有効；0 = 無効
Bit 2	PICC が活動していない場合、アンテナフィールドをオフにします。	1 = 有効；0 = 無効
Bit 3	RFU	RFU
Bit 5 ..4	PICC の PICC ポーリング間隔	<Bit 5 – Bit 4> <0 – 0> = 250 ms <0 – 1> = 500 ms <1 – 0> = 1000 ms <1 – 1> = 2500 ms
Bit 6	RFU	RFU
Bit 7	ISO 14443-A 4 パートを強制に実行します。	1 = 有効；0 = 無効

注：ポーリング設置のデフォルト値 = *8Bh*

提示：

1. 「*PICC* が活動していない場合、アンテナフィールドをオフにする」、そのオプションを有効にすることをお勧めします。そうしたら、活動していない *PICC* はずっとアンテナフィールドに公開されなくて、*PICC* の「ウォーミングアップ」を防ぎます。
2. *PICC* ポーリング間隔の長さに関わって、省エネルギがより効率になります。しかし、*PICC* ポーリングの応答時間が長くなります。省エネルギ状態で *Idle* 消費電流は *60 mA* です；非省エネルギ状態で *Idle* 消費電流は *130 mA* です。注釈：dle 消費電流=PICC が活性化されていない。
3. リーダーは自動的に*"ISO 14443A-4 PICC"*の *ISO 14443A-4* モードを有効にします。*B* タイプの *PICC* はこのオプションによって影響を受けることはありません。
4. *JCOP30* カードには二つのモードを持っている：*ISO 14443A-3*（*MIFARE 1K*）と *ISO 14443A-4* モード。*PICC* を有効にすると、アプリケーションは一つのモードを選択しなければなりません。

5.3.7. 自動的な PICC のポーリングを読取る（Read Automatic PICC Polling）

このコマンドは現在の PICC のポーリングの状態の設置を検査するために使用されます。

Read Automatic PICC Polling コマンドフォーマット（5 バイト）

コマンド	CLA	INS	P1	P2	Lc
Read Automatic PICC Polling	E0h	00h	00h	23h	00h

Read Automatic PICC Polling 応答フォーマット（6 バイト）

応答	CLA	INS	P1	P2	Le	データ出力
結果	E1h	00h	00h	00h	01h	ポーリング設定

ポーリング設定（1 バイト）

ポーリング設定	モード	説明
Bit 0	自動的に PICC ポーリング	1 = 有効；0 = 無効
Bit 1	PICC が見つかっていない場合は、アンテナフィールドをオフにします。	1 = 有効；0 = 無効
Bit 2	PICC が活動していない場合、アンテナフィールドをオフにします。	1 = 有効；0 = 無効
Bit 3	RFU	RFU
Bit 5 ..4	PICC の PICC ポーリング間隔	<Bit 5 – Bit 4> <0 – 0> = 250 ms <0 – 1> = 500 ms <1 – 0> = 1000 ms <1 – 1> = 2500 ms
Bit 6	RFU	RFU
Bit 7	ISO 14443-A 4 パートを強制に実行します。	1 = 有効；0 = 無効

注：ポーリング設置のデフォルト値 = *8Bh*

5.3.8. PICC 操作のパラメ—ター—を設定する（Set PICC Operating Parameter）

このコマンドは PICC 操作のパラメ—ター—を設定するために使われます。

注釈：この設置は失いやすいキーのメモリに保存されます。

Set PICC Operating Parameter コマンドフォーマット（6 バイト）

コマンド	CLA	INS	P1	P2	Lc	データイン
Set PICC Operating Parameter	E0h	00h	00h	20h	01h	操作パラメ—ター—

Set PICC Operating Parameter 応答フォーマット（6 バイト）

応答	CLA	INS	P1	P2	Le	データ出力
結果	E1	00h	00h	00h	01h	操作パラメ—ター—

操作パラメ—ター—（1 バイト）

操作パラメ—ター—	パラメ—ター—	説明	オプション
Bit 0	ISO 14443 A タイプ	PICC のポーリング中に検出されるタグのタイプ	1 = 検出 0 = スキップ
Bit 1	ISO 14443 B タイプ		1 = 検出 0 = スキップ
Bit 2	FeliCa 212 Kbps		1 = 検出 0 = スキップ
Bit 3	FeliCa 424 Kbps		1 = 検出 0 = スキップ
Bit 4	Topaz		1 = 検出 0 = スキップ
Bit 5 - 7	RFU	RFU	RFU

注：操作のデフォルト値 = *1Fh*

5.3.9. PICC 操作のパラメーターを読取る（Read PICC Operating Parameter）

このコマンドは PICC 操作のパラメーターを検査するために使用されます。

Read PICC Operating Parameter コマンドフォーマット（5 バイト）

コマンド	CLA	INS	P1	P2	Lc
Read PICC Operating Parameter	E0h	00h	00h	20h	00h

Read PICC Operating Parameter 応答フォーマット（6 バイト）

応答	CLA	INS	P1	P2	Le	データ出力
結果	E1h	00h	00h	00h	01h	操作パラメーター

操作パラメーター（1 バイト）

操作パラメーター	パラメーター	説明	オプション
Bit 0	ISO 14443 A タイプ	PICC のポーリング中に検出されるタグのタイプ	1 = 検出 0 = スキップ
Bit 1	ISO 14443 B タイプ		1 = 検出 0 = スキップ
Bit 2	FeliCa 212 Kbps		1 = 検出 0 = スキップ
Bit 3	FeliCa 424 Kbps		1 = 検出 0 = スキップ
Bit 4	Topaz		1 = 検出 0 = スキップ
Bit 5 - 7	RFU	RFU	RFU

注：操作のデフォルト値 = *1Fh*

5.3.10. 自動的な PPS を設定する（Set Auto PPS）

PICC が認識されるたびに、リーダーは最大接続速度によって定義された PCD および PICC との間の通信速度を変更しようとします。カードが提案された接続速度をサポートしていない場合、リーダーはより遅い速度でとカードと接続しようとします。

Set Auto PPS コマンドフォーマット（7 バイト）

コマンド	CLA	INS	P1	P2	Lc	データイン	
Set Auto PPS	E0h	00h	00h	24h	02h	Max Tx Speed	Max Rx Speed

Set Auto PPS 応答フォーマット（9 バイト）

応答	CLA	INS	P1	P2	Le	データ出力			
結果	E1h	00h	00h	00h	04h	Max Tx Speed	Current Tx Speed	Max Rx Speed	Current Rx Speed

その中：

Max Tx Speed 最大 Tx 速度（1 バイト）

Current Tx Speed 現在の Tx 速度（1 バイト）

Max Rx Speed 最大 Rx 速度（1 バイト）

Current Rx Speed 現在の Rx 速度（1 バイト）

00h = 106 Kbps；デフォルト設定、自動 PPS が設定されていないと同じです。

01h = 212 Kbps

02h = 424 Kbps

注釈：

1. 通常、アプリケーションが使用中の PICC の最大接続速度を知っている必要があります。環境にも達成可能な最大速度に影響します。リーダーは提案されている通信速度を使用して、PICC と話をします。PICC や環境が提案されている通信速度の要件を満たしていない場合、PICC はアクセスできなくなります。
2. リーダーは、送信側と受信側との間の異なる速度をサポートしています。

5.3.11. 自動的な PPS を読み取る（Read Auto PPS）

このコマンドは現在の自動的な PPS の設置を検査するために使用されます。

Read Auto PPS コマンドフォーマット（5 バイト）

コマンド	CLA	INS	P1	P2	Lc
自動的な PPS を読み取る（Read Auto PPS）	E0h	00h	00h	24h	00h

Read Auto PPS 応答フォーマット（9 バイト）

応答	CLA	INS	P1	P2	Le	データ出力			
結果	E1h	00h	00h	00h	04h	Max Tx Speed	Current Tx Speed	Max Rx Speed	Current Rx Speed

その中：

Max Tx Speed 最大 Tx 速度（1 バイト）

Current Tx Speed 現在の Tx 速度（1 バイト）

Max Rx Speed 最大 Rx 速度（1 バイト）

Current Rx Speed 現在の Rx 速度（1 バイト）

00h = 106 Kbps；デフォルト設定、自動 PPS が設定されていないと同じです。

01h = 212 Kbps

02h = 424 Kbps

5.4. ACR122U 互換性のあるコマンド

5.4.1. 二色 LED 制御（Bi-color LED Control）

このコマンドは、二色の LED インジケータのステータスを制御するために使用されます。

Bi-color LEDs Contro コマンドフォーマット（9 バイト）

コマンド	CLA	INS	P1	P2	Lc	データフィールド（4 バイト）
Bi-color LED Control	FFh	00h	40h	LED 状態制御	04h	点滅周期の制御

P2 LED 状態制御

二色の LED とブザー制御のフォーマット（1 バイト）

コマンド	アイテム	説明
Bit 0	赤の LED の最後の状態	1 = ON；0 = OFF
Bit 1	緑の LED の最後の状態	1 = ON；0 = OFF
Bit 2	赤の LED のマスク	1 = 状態を更新する 0 = 変化なし
Bit 3	緑の LED のマスク	1 = 状態を更新する 0 = 変化なし
Bit 4	初期の赤の LED の点滅状態	1 = ON；0 = OFF
Bit 5	初期の緑の LED の点滅状態	1 = ON；0 = OFF
Bit 6	赤い LED の点滅マスク	1 = 点滅 0 = 点滅なし
Bit 7	緑の LED の点滅マスク	1 = 点滅 0 = 点滅なし

データ　　点滅周期の制御

Bi-color LED Blinking Duration Control コマンドフォーマット（4 バイト）

バイト 0	バイト 1	バイト 2	バイト 3
T1 周期 初期の LED の点滅状態 （単位 = 100 ms）	T2 周期 点滅状態を切り替える （単位 = 100 ms）	繰り返し回数	00h

データ出力　　SW1 SW2。リーダーから返された状態コード

状態コード

結果	SW1	SW2	意味
成功	90h	現在の LED 状態	操作が成功に完了しました。
エラー	63	00h	操作が失敗しました。

LED の現在の状態（1 バイト）

状態	アイテム	説明
Bit 0	現在の赤い LED	1 = ON；0 = OFF
Bit 1	現在の緑の LED	1 = ON；0 = OFF
Bits 2 – 7	保留	

提示：

1. ***LED* 状態**の操作は ***LED* 点滅**操作の後に実行されます。
2. ***LED* 状態のマスク**が有効になっていない場合、***LED* 状態**は変更しません。
3. ***LED* 状態のマスク**が有効になっている場合、***LED*** は点滅しません。。また、繰り返し回数は *0* より大きくなければなりません。
4. *T1* および *T2* 周期のパラメータは、*LED* の点滅周期とブザーターンオン周期を制御するために使用される。例えば：もし *T1=1*，*T2=1*，デューティサイクル= *50%*。

   注：デューティサイクル= *T1/*（*T1 + T2*）。

5.4.2. ファームウェアのバージョンを取得する（Get Firmware Version）

このコマンドはリーダーのファームウェアのバージョンを取得する時に使われます。

Get Firmware Version のコマンドフォーマット（５バイト）

コマンド	CLA	INS	P1	P2	Le
Get Firmware	FFh	00h	48h	00h	00h

Get Firmware Version の応答フォーマット（X バイト）

応答	データ出力
結果	ファームウェアのバージョン番号

例：

応答 = 41 43 52 31 32 35 31 54 5F 56 44 30 30 2E 30h = ACR1251T_VD00.0 (ASCII)

5.4.3. PICC 操作のパラメーターを入手する（Read the PICC Operating Parameter）

このコマンドはリーダーの PICC 操作のパラメーターを入手する時に使われます。

Get the PICC Operating Parameter フォーマット（5 バイト）

コマンド	CLA	INS	P1	P2	Le
Get PICC Operating Parameter	FFh	00h	50h	00h	00h

Get the PICC Operating Parameter 応答フォーマット（2 バイト）

応答	データ出力	
結果	90h	PICC 操作パラメーター

PICC 操作パラメーター

ビット	パラメーター	説明	オプション
7	自動的に PICC ポーリング	PICC ポーリングを有効する	1 = 有効にする 0 = 無効にする
6	自動に ATS 生成する	ISO 14443-4 A タイプのタグを活性化するたびに ATS 請求を送信します。	1 = 有効にする 0 = 無効にする
5	ポーリング間隔	連続した PICC のポーリング間の時間間隔を設定します。	1 = 250 ms 0 = 500 ms
4	FeliCa 424 Kbps	PICC のポーリング中に検出されるタグのタイプ	1 = 検出 0 = スキップ
3	FeliCa 212 Kbps		1 = 検出 0 = スキップ
2	Topaz		1 = 検出 0 = スキップ
1	ISO 14443 B タイプ		1 = 検出 0 = スキップ
0	ISO 14443 A タイプ **注：** *MIFARE* タグを検査するために、*ATS* の自動生成を無効しなければなりません。		1 = 検出 0 = スキップ

5.4.4. PICC 操作のパラメーターを設定する（Set PICC Operating Parameter）

このコマンドはリーダーの PICC 操作のパラメーターを設定する時に使われます。

Set PICC Operating Parameter コマンドフォーマット（5 バイト）

コマンド	CLA	INS	P1	P2	Le
Set PICC Operating Parameter	FFh	00h	51h	PICC 操作パラメーター	00h

Set PICC Operating Parameter 応答フォーマット（2 バイト）

応答	データ出力	
結果	90h	PICC 操作パラメーター

PICC 操作パラメーター

ビット	パラメーター	説明	オプション
7	自動的に PICC ポーリング	PICC ポーリングを有効する	1 = 有効にする 0 = 無効にする
6	自動に ATS 生成する	ISO 14443-4 A タイプのタグを活性化するたびに ATS 請求を送信します。	1 = 有効にする 0 = 無効にする
5	ポーリング間隔	連続した PICC のポーリング間の時間間隔を設定します。	1 = 250 ms 0 = 500 ms
4	FeliCa 424 Kbps	PICC のポーリング中に検出されるタグのタイプ	1 = 検出 0 = スキップ
3	FeliCa 212 Kbps		1 = 検出 0 = スキップ
2	Topaz		1 = 検出 0 = スキップ
1	ISO 14443 B タイプ		1 = 検出 0 = スキップ
0	ISO 14443 A タイプ **注：** *MIFARE タグを検査するために、ATS の自動生成を無効しなければなりません。*		1 = 検出 0 = スキップ

Android は Google Inc.の商標です。
Microsoft は Atmel また子会社がアメリカとまたはほかの国の登録商標です。
MIFARE、MIFARE Classic、MIFARE DESFire および MIFARE Ultralight は NXP B.V.の登録商標で、ライセンスに従って使用されます。