



**Advanced Card Systems Ltd.**  
Card & Reader Technologies

# ACR1251T USB Token NFC Reader II



Reference Manual V1.02



## Revision History

| Release Date | Revision Description   | Version Number |
|--------------|--|----------------|
| 2018-06-04   | <ul style="list-style-type: none"><li>Initial Release</li></ul>  | 1.00           |
| 2020-01-10   | <ul style="list-style-type: none"><li>Updated MIFARE Trademarks</li><li>Updated Section 2.0 Features</li><li>Updated Section 5.1 PCSC API</li><li>Updated Section 5.3.6 Set Automatic PICC Polling</li><li>Updated Section 5.3.7 Read Automatic PICC Polling</li><li>Updated Section 5.3.10 Set Auto PPS</li><li>Updated Section 5.3.11 Read Auto PPS</li><li>Re-organized Section 5.2</li></ul> | 1.01           |
| 2020-08-28   | <ul style="list-style-type: none"><li>Updated Section 5.3.4 Set Led Status Indicator Behavior</li><li>Updated Section 5.3.8 Set PICC Operating Parameter</li><li>Updated Section 5.3.9 Read PICC Operating Parameter</li></ul>   | 1.02           |



## Table of Contents

|             |   |          |
|-------------|---|----------|
| <b>1.0.</b> | <b>Introduction .....</b>                                       | <b>5</b> |
| <b>2.0.</b> | <b>Features .....</b>   | <b>6</b> |
| <b>3.0.</b> | <b>Acronyms and Abbreviations .....</b>                         | <b>7</b> |
| <b>4.0.</b> | <b>Architecture .....</b>                                       | <b>8</b> |
| <b>5.0.</b> | <b>Host Programming (PC-linked) API .....</b>                   | <b>9</b> |
| 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 Flow .....   | 17       |
| 5.1.8.      | Escape Command Flow .....                                       | 18       |
| 5.2.        | Contactless Smart Card Protocol .....                           | 19       |
| 5.2.1.      | ATR Generation .....  | 19       |
| 5.2.2.      | Pseudo APDU for Contactless Interface .....                     | 22       |
| 5.2.3.      | APDU commands for PCSC 2.0 Part 3 (version 2.02 or above) ..... | 23       |
| 5.2.4.      | PICC Commands for MIFARE Classic (1K/4K) memory cards .....     | 37       |
| 5.2.5.      | Accessing PCSC-compliant tags (ISO 14443-4) .....               | 46       |
| 5.2.6.      | Accessing FeliCa tags .....                                     | 48       |
| 5.3.        | Peripherals Control .....                                       | 49       |
| 5.3.1.      | Get Firmware Version .....                                      | 49       |
| 5.3.2.      | LED Control .....   | 50       |
| 5.3.3.      | LED Status .....  | 51       |
| 5.3.4.      | Set LED Status Indicator Behavior .....                         | 52       |
| 5.3.5.      | Read LED Status Indicator Behavior .....                        | 53       |
| 5.3.6.      | Set Automatic PICC Polling .....                                | 54       |
| 5.3.7.      | Read Automatic PICC Polling .....                               | 56       |
| 5.3.8.      | Set PICC Operating Parameter .....                              | 57       |
| 5.3.9.      | Read PICC Operating Parameter .....                             | 58       |
| 5.3.10.     | Set Auto PPS .....  | 59       |
| 5.3.11.     | Read Auto PPS .....   | 60       |
| 5.4.        | ACR122T Compatible Commands .....                               | 61       |
| 5.4.1.      | Bi-color LED Control .....                                      | 61       |
| 5.4.2.      | Get Firmware Version .....                                      | 63       |
| 5.4.3.      | Get PICC Operating Parameter .....                              | 64       |
| 5.4.4.      | Set PICC Operating Parameter .....                              | 65       |

## List of Figures

|                   |   |           |
|-------------------|---|-----------|
| <b>Figure 1 :</b> | <b>ACR1251T Architecture .....</b>        | <b>8</b>  |
| <b>Figure 2 :</b> | <b>ACR1251T APDU Flow .....</b>           | <b>17</b> |
| <b>Figure 3 :</b> | <b>ACR1251T Escape Command Flow .....</b> | <b>18</b> |



## List of Tables

|   |    |
|---|----|
| <b>Table 1</b> : Acronyms and Abbreviations .....   | 7  |
| <b>Table 2</b> : MIFARE Classic 1K Memory Map ..... | 39 |
| <b>Table 3</b> : MIFARE Classic 4K Memory Map ..... | 39 |
| <b>Table 4</b> : MIFARE Ultralight Memory Map ..... | 40 |



## 1.0. Introduction

The ACR1251T is the token version of the ACR1251U PC-linked NFC smart card reader developed based on the 13.56 MHz contactless technology. Following ACR122T, the token version of the ACR122U, the world's first CCID-compliant contactless reader, the ACR1251T offers more and advanced features. It is designed to support not only ISO 14443 Type A and B cards, but also MIFARE®, FeliCa and all four types of NFC tags and devices.

The ACR1251T acts as the intermediary device between the computer and the card. The reader, which specifically communicates with the contactless tag or the device peripheral (LED), will carry out commands issued from the computer. It has a PICC reader interface that follows the PC/SC specifications. This reference manual will discuss in detail how the PC/SC APDU commands were implemented for the contactless interface and device peripheral of the ACR1251T.



## 2.0. Features

- USB Full Speed Interface
- CCID-compliant
- Smart Card Reader:
  - Contactless Interface:
    - Read/Write speed of up to 424 Kbps
    - Built-in antenna for contactless tag access, with card reading distance of up to 30 mm (depending on tag type)
    - Supports ISO 14443 Part 4 Type A and B cards, MIFARE Classic®, FeliCa, and all four types of NFC (ISO/IEC 18092 tags)
    - Built-in anti-collision feature (only one tag is accessed at any time)
    - NFC Support:
      - Card Reader/Writer mode
- Built-in Peripherals:
  - User-controllable Bi-color LED
- Application Programming Interface:
  - Supports PC/SC
  - Supports CT-API (through wrapper on top of PC/SC)
- USB Firmware Upgradeability
- Supports Android™ 3.1 and later<sup>1</sup>
- Compliant with the following standards:
  - EN 60950/IEC 60950
  - ISO 14443
  - ISO 18092
  - PC/SC
  - CCID
  - CE
  - FCC
  - RoHS
  - REACH
  - VCCI (Japan)
  - MIC (Japan)
  - Microsoft® WHQL

---

<sup>1</sup> Uses an ACS-defined Android Library



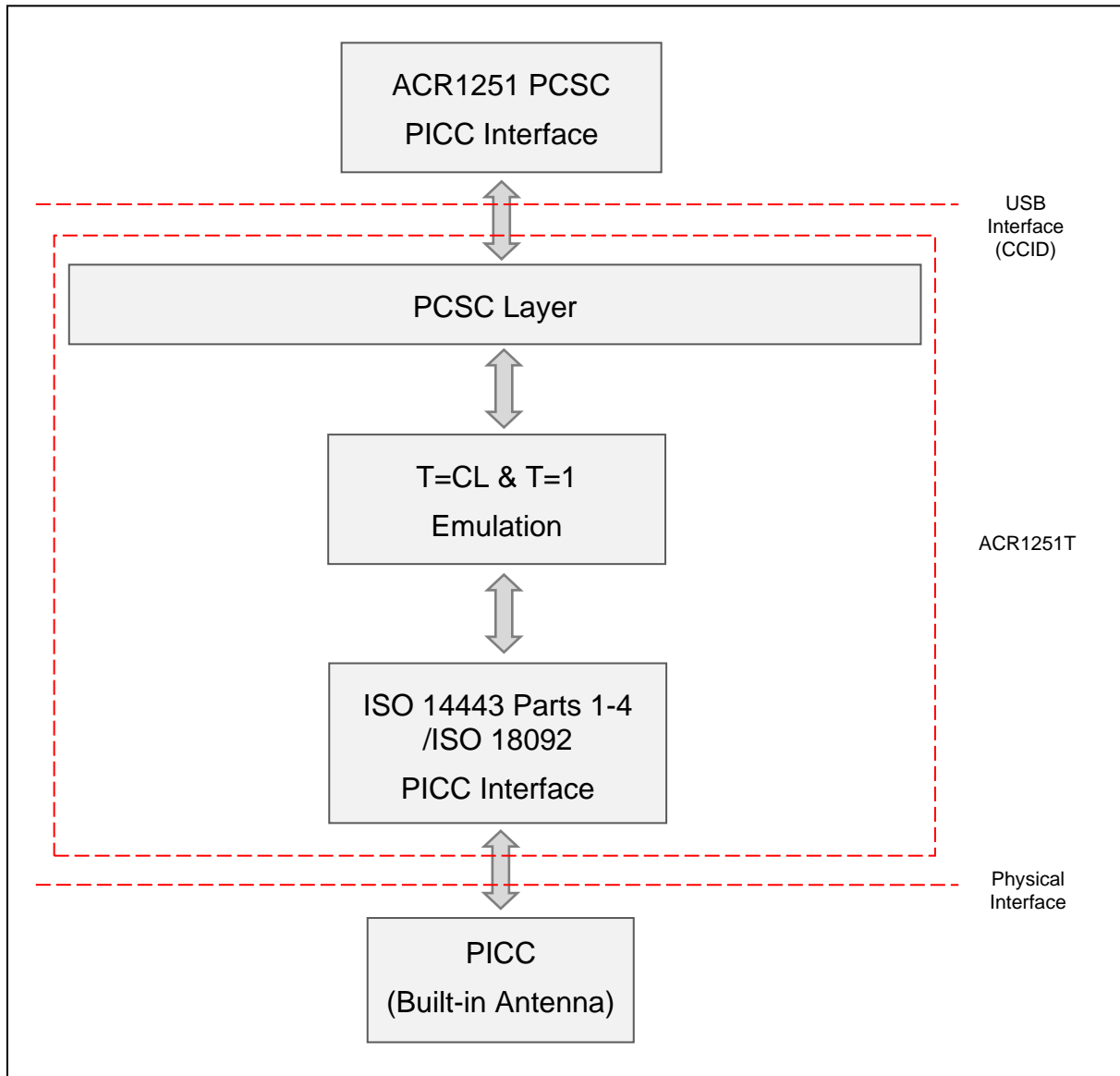
### 3.0. Acronyms and Abbreviations

| Acronym/Abbreviation | Description  |
|----------------------|--|
| ATR                  | Attribute Request and Attribute Response                           |
| DEP                  | Data Exchange Protocol Request and Data Exchange Protocol Response |
| DSL                  | Deselect Request and Deselect Response                             |
| PSL                  | Parameter Selection Request and Parameter Selection Response       |
| RLS                  | Release Request and Release Response                               |
| WUP                  | Wakeup Request and Wakeup Response                                 |
| DID                  | Device ID  |
| BS                   | Sending bit duration   |
| BR                   | Receiving bit duration   |
| PP                   | Protocol Parameters  |
| Gi                   | Optional information field for Initiator                           |
| PFB                  | Control information for transaction                                |
| FSL                  | maximum value for the Frame Length                                 |
| LLCP                 | Logical Link Control Protocol                                      |

**Table 1:** Acronyms and Abbreviations

## 4.0. Architecture

For communication architecture, the protocol used between the ACR1251T reader and the computer is CCID. All communications between PICC are PCSC-compliant.



**Figure 1:** ACR1251T Architecture





## 5.0. Host Programming (PC-linked) API

### 5.1. PCSC API

This section will describe some of the PCSC API commands for application programming usage. For more details, please refer to Microsoft MSDN Library or PCSC workgroup.

#### 5.1.1. SCardEstablishContext

The **SCardEstablishContext** function establishes the resource manager context within which database operations are performed.

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

This function should be performed first before any other PCSC operation.

Example:

```
#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

The **SCardListReaders** function provides the list of readers within a set of named reader groups, eliminating duplicates.

The caller supplies a list of reader groups and receives the list of readers within the named groups. Unrecognized group names are ignored. This function only returns readers within the named groups that are currently attached to the system and available for use.

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

Example:

```
#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

The **SCardConnect** function establishes a connection (using a specific resource manager context) between the calling application and a smart card contained by a specific reader. If no card exists in the specified reader, an error is returned.

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

Example:

```
#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

The **SCardControl** function gives you direct control of the reader. You can call it any time after a successful call to **SCardConnect** and before a successful call to **SCardDisconnect**. The effect on the state of the reader depends on the control code.

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

**Note:** Commands from **Peripherals Control** use this API for sending.

Example:

```
#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

The **SCardTransmit** function sends a service request to the smart card and expects to receive data back from the card.

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

**Note:** APDU Commands (i.e. the command sent to connected card, **PICC Commands for MIFARE Classic (1K/4K) memory cards**, and **Pseudo APDU for Contactless Interface**) use this API for sending.

Example:

```
#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
BYTE              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

The **SCardDisconnect** function terminates a connection previously opened between the calling application and a smart card in the target reader.

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

This function ends the PCSC Operation.

Example:

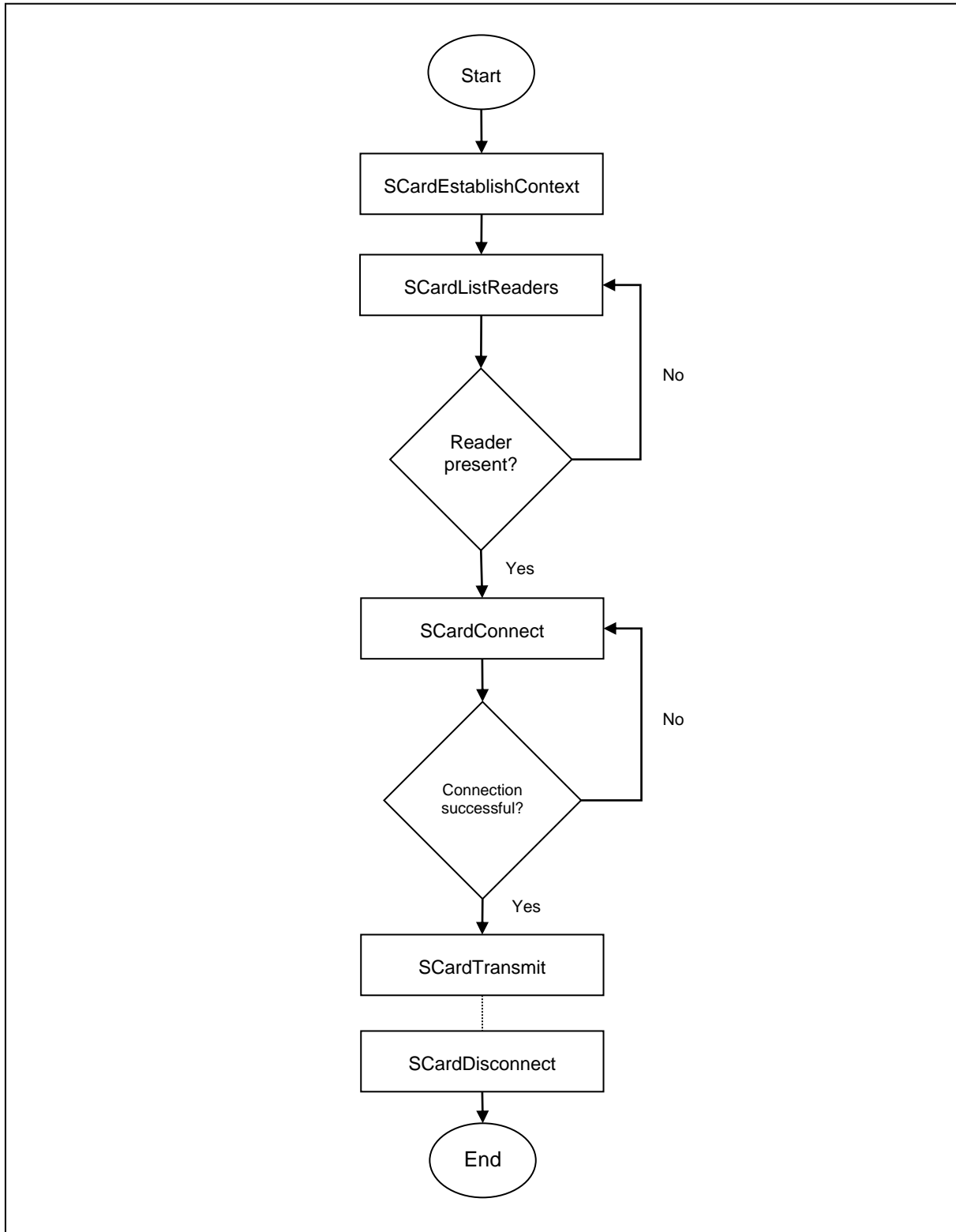
```
#define SCARD_SCOPE_USER 0

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
    }
}
}
```

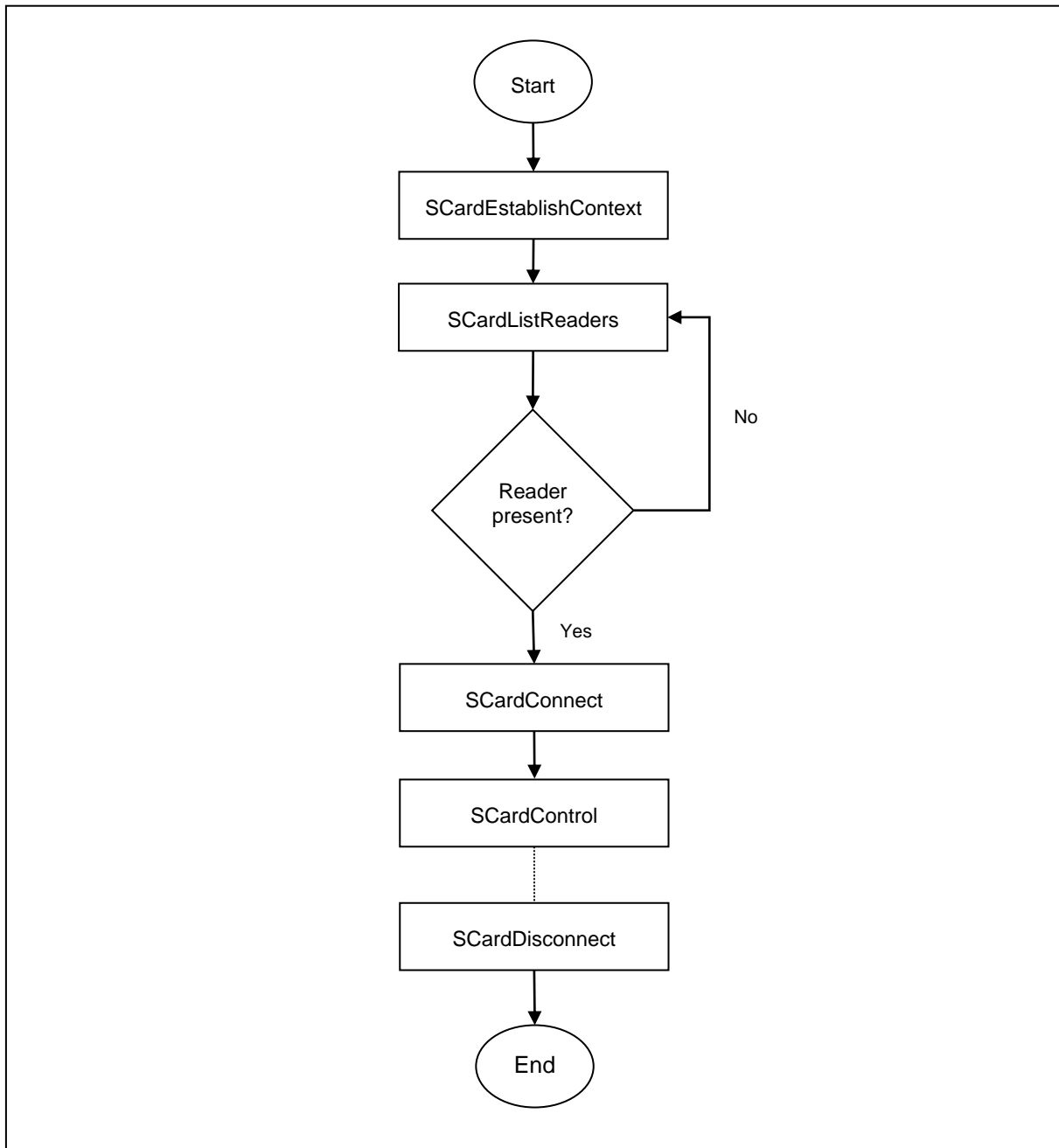


### 5.1.7. APDU Flow



**Figure 2: ACR1251T APDU Flow**

### 5.1.8. Escape Command Flow



**Figure 3:** ACR1251T Escape Command Flow



## 5.2. Contactless Smart Card Protocol

### 5.2.1. ATR Generation

If the reader detects a PICC, an ATR will be sent to the PCSC driver to identify the PICC.

#### 5.2.1.1. ATR Format for ISO 14443 Part 3 PICCs

| Byte                   | Value        | Designation    | Description   |
|------------------------|--------------|----------------|---|
| 0                      | 3Bh          | Initial Header |   |
| 1                      | 8Nh          | T0             | Higher nibble 8 means: no TA1, TB1, TC1 only TD1 is following.<br>Lower nibble N is the number of historical bytes (HistByte 0 to HistByte N-1) |
| 2                      | 80h          | TD1            | Higher nibble 8 means: no TA2, TB2, TC2 only TD2 is following.<br>Lower nibble 0 means T = 0  |
| 3                      | 01h          | TD2            | Higher nibble 0 means no TA3, TB3, TC3, TD3 following.<br>Lower nibble 1 means T = 1  |
| 4<br><br>To<br><br>3+N | 80h          | T1             | Category indicator byte, 80 means A status indicator may be present in an optional COMPACT-TLV data object.                                     |
|                        | 4Fh          | Tk             | Application identifier Presence Indicator.  |
|                        | 0Ch          |                | Length  |
|                        | RID          |                | Registered Application Provider Identifier (RID) # A0 00 00 03 06   |
|                        | SS           |                | Byte for standard.  |
|                        | C0 .. C1h    |                | Bytes for card name.  |
|                        | 00 00 00 00h |                | RFU   |
| 4+N                    | UU           | TCK            | Exclusive-oring of all the bytes T0 to Tk   |

#### Example:

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

Where:

- Length (YY)** = 0Ch
- RID** = A0 00 00 03 06h (PC/SC Workgroup)
- Standard (SS)** = 03h (ISO 14443A, Part 3)
- Card Name (C0 .. C1)** = [00 01h] (MIFARE Classic 1K)
  
- Standard (SS)** = 03h: ISO 14443A, Part 3  
= 11h: FeliCa



| <b>Card Name (C0 .. C1)</b> |                             |                            |
|-----------------------------|-----------------------------|----------------------------|
|                             | 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 and 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]: undefined tags   |
|                             | 00 37: MIFARE Plus® SL1 4K  |                            |



**5.2.1.2. ATR Format for ISO 14443 Part 4 PICCs**

| Byte                       | Value                        | Designation  | Description   |         |         |       |                            |                              |  |
|----------------------------|------------------------------|--|---|---------|---------|-------|----------------------------|------------------------------|--|
| 0                          | 3Bh                          | Initial Header   |   |         |         |       |                            |                              |  |
| 1                          | 8N                           | T0   | Higher nibble 8 means: no TA1, TB1, TC1 only TD1 is following.<br>Lower nibble N is the number of historical bytes (HistByte 0 to HistByte N-1)   |         |         |       |                            |                              |  |
| 2                          | 80h                          | TD1  | Higher nibble 8 means: no TA2, TB2, TC2 only TD2 is following.<br>Lower nibble 0 means T = 0  |         |         |       |                            |                              |  |
| 3                          | 01h                          | TD2  | Higher nibble 0 means no TA3, TB3, TC3, TD3 following.<br>Lower nibble 1 means T = 1  |         |         |       |                            |                              |  |
| 4<br>to<br>3 + N           | XX                           | T1   | Historical Bytes:<br><br>ISO 14443-A:<br>The historical bytes from ATS response. Refer to the ISO 14443-4 specification.<br><br>ISO 14443-B:<br><table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Byte1-4</th> <th>Byte5-7</th> <th>Byte8</th> </tr> </thead> <tbody> <tr> <td>Application Data from ATQB</td> <td>Protocol Info Byte from ATQB</td> <td>Higher nibble=MBLI from ATTRIB command<br/>Lower nibble (RFU)=0</td> </tr> </tbody> </table> | Byte1-4 | Byte5-7 | Byte8 | Application Data from ATQB | Protocol Info Byte from ATQB | Higher nibble=MBLI from ATTRIB command<br>Lower nibble (RFU)=0 |
|                            | Byte1-4                      | Byte5-7  |   | Byte8   |         |       |                            |                              |  |
| Application Data from ATQB | Protocol Info Byte from ATQB | Higher nibble=MBLI from ATTRIB command<br>Lower nibble (RFU)=0 |   |         |         |       |                            |                              |  |
| XX<br>XX<br>XX             | Tk                           |  |   |         |         |       |                            |                              |  |
| 4+N                        | UU                           | TCK  | Exclusive-oring of all the bytes T0 to Tk   |         |         |       |                            |                              |  |

**Example 1:** ATR for MIFARE® DESFire® = {3B 81 80 01 80 80h} // 6 bytes of ATR

**Note:** Use the APDU “FF CA 01 00 00h” to distinguish the ISO 14443A-4 and ISO 14443B-4 PICCs, and retrieve the full ATS if available. ISO 14443A-3 or ISO 14443B-3/4 PICCs do have ATS returned.

APDU Command = FF CA 01 00 00h  
 APDU Response = 06 75 77 81 02 80 90 00h  
 ATS = {06 75 77 81 02 80h}

**Example 2:** ATR for EZ-link = {3B 88 80 01 1C 2D 94 11 F7 71 85 00 BEh}  
 Application Data of ATQB = 1C 2D 94 11h  
 Protocol Information of ATQB = F7 71 85h  
 MBLI of ATTRIB = 00h



## 5.2.2. Pseudo APDU for Contactless Interface

### 5.2.2.1. Get Data

This command returns the serial number or ATS of the connected PICC.

Get UID APDU Format (5 Bytes)

| Command  | Class | INS | P1         | P2  | Le                  |
|----------|-------|-----|------------|-----|---------------------|
| Get Data | FFh   | CAh | 00h<br>01h | 00h | 00h<br>(Max Length) |

If P1 = 00h, Get UID Response Format (UID + 2 Bytes)

| Response | Data Out     |     |     |              |     |     |
|----------|--------------|-----|-----|--------------|-----|-----|
| Result   | UID<br>(LSB) | ... | ... | UID<br>(MSB) | SW1 | SW2 |

If P1 = 01h, Get ATS of a ISO 14443 A card (ATS + 2 Bytes)

| Response | Data Out |  |  |     |     |
|----------|----------|--|--|-----|-----|
| Result   | ATS      |  |  | SW1 | SW2 |

Response Codes

| Results | SW1 | SW2 | Meaning  |
|---------|-----|-----|--|
| Success | 90h | 00h | The operation was completed successfully.  |
| Warning | 62h | 82h | End of UID/ATS reached before Le bytes (Le is greater than UID Length).                                    |
| Error   | 6Ch | XXh | Wrong length (wrong number Le: 'XX' encodes the exact number) if Le is less than the available UID length. |
| Error   | 63h | 00h | The operation failed.  |
| Error   | 6Ah | 81h | Function not supported   |

#### Examples:

To get the serial number of the “connected PICC”:

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

To get the ATS of the “connected ISO 14443 A PICC”:

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



### 5.2.3. APDU commands for PCSC 2.0 Part 3 (version 2.02 or above)

PCSC 2.0 Part 3 commands are used to transparently pass data from an application to a contactless tag, return the received data transparently to the application and protocol, and switch the protocol simultaneously.

#### 5.2.3.1. Command and Response APDU Format

Command Format

| CLA | INS | P1  | P2       | Lc      | Data In       |
|-----|-----|-----|----------|---------|---------------|
| FFh | C2h | 00h | Function | DataLen | Data[DataLen] |

Where:

**Functions**                    1 byte.  
                                       00h = Manage Session  
                                       01h = Transparent Exchange  
                                       02h = Switch Protocol  
                                       Other = RFU

Response Format

| Data Out                   | SW1 | SW2 |
|----------------------------|-----|-----|
| Data Field BER-TLV encoded |     |     |

Every command returns SW1 and SW2 together with the response data field (if available). The SW1 SW2 is based on ISO 7816. SW1 SW2 from the C0 data object below should also be used.

C0 data element Format

| Tag | Length (1 byte) | SW2          |
|-----|-----------------|--------------|
| C0h | 03h             | Error Status |



### Error Status Description

| Error Status | Description  |
|--------------|--|
| XX SW1 SW2   | XX = number of the bad data object in the APDU<br>00 = general error of APDU<br>01 = error in the 1 <sup>st</sup> data object<br>02 = error in the 2 <sup>nd</sup> data object |
| 00 90 00h    | No error occurred  |
| XX 62 82h    | Data object XX warning, requested information not available  |
| XX 63 00h    | No information   |
| XX 63 01h    | Execution stopped due to failure in other data object  |
| XX 6A 81h    | Data object XX not supported   |
| XX 67 00h    | Data object XX with unexpected length  |
| XX 6A 80h    | Data object XX with unexpected vale  |
| XX 64 00h    | Data Object XX execution error (no response from IFD)  |
| XX 64 01h    | Data Object XX execution error (no response from ICC)  |
| XX 6F 00h    | Data object XX failed, no precise diagnosis  |

The first value byte indicates the number of the erroneous data object XX, while the last two bytes indicate the explanation of the error. SW1 SW2 values based on ISO 7816 are allowed.

If there are more than one data objects in the C-APDU field and one data object failed, IFD can process the following data objects if they do not depend on the failed data objects.





### 5.2.3.2. Manage Session Command

This command is used to manage the transparent session. This includes starting and ending a transparent session. Through this command, you can also manage the operation environment and the capabilities of the IFD within the transparent session.

Manage Session Command

| Command        | Class | INS | P1  | P2  | Lc      | Data In              |
|----------------|-------|-----|-----|-----|---------|----------------------|
| Manage Session | FFh   | C2h | 00h | 00h | DataLen | DataObject (N bytes) |

Where:

#### Data Object (1 byte)

| Tag    | Data Object               |
|--------|---------------------------|
| 80h    | Version Data Object       |
| 81h    | Start Transparent Session |
| 82h    | End Transparent Session   |
| 83h    | Turn Off RF Field         |
| 84h    | Turn On RF Field          |
| 5F 46h | Timer                     |
| FF 6Dh | Get Parameter             |
| FF 6Eh | Set Parameter             |

Manage Session Response Data Object

| Tag    | Data Object               |
|--------|---------------------------|
| C0h    | Generic Error status      |
| 80h    | Version data object       |
| FF 6Dh | IFD parameter data object |

#### 5.2.3.2.1. Start Session Data Object

This command is used to start a transparent session. Once the session has started, auto-polling will be disabled until the session is ended.

Start Session Data Object

| Tag | Length (1 byte) | Value |
|-----|-----------------|-------|
| 81h | 00h             | -     |



### 5.2.3.2.2. End Session Data Object

This command ends the transparent session. The auto-polling will be reset to the state before the session has started.

End Session Data Object

| Tag | Length (1 byte) | Value |
|-----|-----------------|-------|
| 82h | 00h             | -     |

### 5.2.3.2.3. Version Data Object

This command returns the version number of the IFD handler.

Version Data Object

| Tag | Length (1 byte) | Value |       |       |
|-----|-----------------|-------|-------|-------|
| 80h | 03h             | Major | Minor | Build |

### 5.2.3.2.4. Turn Off the RF Data Object

This command turns off the antenna field.

Turn off RF Field Data Object

| Tag | Length (1 byte) | Value |
|-----|-----------------|-------|
| 83h | 00h             | -     |

### 5.2.3.2.5. Turn On the RF Data Object

This command turns on the antenna field.

Turn on the RF Field Data Object

| Tag | Length (1 byte) | Value |
|-----|-----------------|-------|
| 84h | 00h             | -     |

### 5.2.3.2.6. Timer Data Object

This command creates a 32-bit timer data object in unit of 1  $\mu$ s.

**Example:** If there is timer data object with 5000  $\mu$ s between RF Turn Off Data Object and RF Turn On Data Object, the reader will turn off the RF field for about 5000 $\mu$ s before it is turned on.

Timer Data Object

| Tag    | Length (1 byte) | Value           |
|--------|-----------------|-----------------|
| 5F 46h | 04h             | Timer (4 bytes) |



### 5.2.3.2.7. Get Parameter Data Object

This command gets the different parameters from the IFD.

Get Parameter Data Object

| Tag    | Length (1 byte) | Value       |     |       |
|--------|-----------------|-------------|-----|-------|
|        |                 | Tag         | Len | Value |
| FF 6Dh | Var             | TLV_Objects |     |       |

TLV\_Objects

| Parameters Requested                          | Tag | Length |
|---|-----|--------|
| Frame size for IFD integer (FSDI)             | 01h | 00h    |
| Frame size for ICC integer (FSCI)             | 02h | 00h    |
| Frame waiting time integer (FWTI)             | 03h | 00h    |
| Max. Communication Speed supported by the IFD | 04h | 00h    |
| Communication Speed of the ICC                | 05h | 00h    |
| Modulation Index                              | 06h | 00h    |
| PCB for ISO/IEC14443                          | 07h | 00h    |
| CID for ISO/IEC14443                          | 08h | 00h    |
| NAD for ISO/IEC14443                          | 09h | 00h    |
| Param 1 – 4 for for ISO/IEC14443 type B       | 0Ah | 00h    |



### 5.2.3.2.8. Set Parameter Data Object

This command sets different parameters from the IFD.

Set Parameter Data Object

| Tag    | Length (1 byte) | Value       |     |       |
|--------|-----------------|-------------|-----|-------|
|        |                 | Tag         | Len | Value |
| FF 6Eh | Var             | TLV_Objects |     |       |

TLV\_Objects

| Parameters Requested                          | Tag | Length |
|---|-----|--------|
| Frame size for IFD integer (FSDI)             | 01h | 01h    |
| Frame size for ICC integer (FSCI)             | 02h | 01h    |
| Frame waiting time integer (FWTI)             | 03h | 01h    |
| Max. Communication Speed supported by the IFD | 04h | 01h    |
| Communication Speed of the ICC                | 05h | 01h    |
| Modulation Index                              | 06h | 01h    |
| PCB for ISO/IEC14443                          | 07h | 01h    |
| CID for ISO/IEC14443                          | 08h | 01h    |
| NAD for ISO/IEC14443                          | 09h | 01h    |
| Param 1 – 4 for for ISO/IEC14443 type B       | 0Ah | 04h    |



### 5.2.3.3. Transparent Exchange Command

This command transmits and receives any bit or bytes from ICC.

Transparent Exchange Command

| Command  | Class | INS | P1  | P2  | Lc      | Data In              |
|----------|-------|-----|-----|-----|---------|----------------------|
| TranspEx | FFh   | C2h | 00h | 01h | DataLen | DataObject (N bytes) |

Where:

#### Data Object (1 byte)

| Tag    | Data Object                       |
|--------|-----------------------------------|
| 90h    | Transmission and Reception Flag   |
| 91h    | Transmission Bit Framing          |
| 92h    | Reception Bit Framing             |
| 93h    | Transmit                          |
| 94h    | Receive                           |
| 95h    | Transceive – Transmit and Receive |
| FF 6Dh | Get Parameter                     |
| FF 6Eh | Set Parameter                     |

Transparent Exchange Response Data Object

| Tag    | Data Object  |
|--------|--|
| C0h    | Generic Error status                                   |
| 92h    | Number of valid bits in the last byte of received data |
| 96h    | Response Status  |
| 97h    | ICC response   |
| FF 6Dh | IFD parameter data object                              |



### 5.2.3.3.1. Transmission and Reception Flag Data Object

This command defines the framing and RF parameters for the following transmission.

Transmission and Reception Flag Data Object

| Tag | Length (1 byte) | Value |  |
|-----|-----------------|-------|--|
|     |                 | bit   | Description  |
| 90h | 02h             | 0     | 0 – append CRC in the transmit data<br>1 – do not append CRC in the transmit data  |
|     |                 | 1     | 0 – discard CRC from the received data<br>1 – do not discard CRC from the received data (i.e. no CRC checking)   |
|     |                 | 2     | 0 – insert parity in the transmit data<br>1 – do not insert parity   |
|     |                 | 3     | 0 – expect parity in received date<br>1 – do not expect parity (i.e. no parity checking)   |
|     |                 | 4     | 0 – append protocol prologue in the transmit data or discard from the response<br>1 – do not append or discard protocol prologue if any (e.g. PCB, CID, NAD) |
|     |                 | 5-15  | RFU  |

### 5.2.3.3.2. Transmission Bit Framing Data Object

This command defines the number of valid bits of the last byte of data to transmit or transceive.

Transmission bit Framing Data Object

| Tag | Length (1 byte) | Value |  |
|-----|-----------------|-------|--|
|     |                 | bit   | Description  |
| 91h | 01h             | 0-2   | Number of valid bits of the last byte (0 means all bits are valid) |
|     |                 | 3-7   | RFU  |

Transmission bit framing data object shall be together with “transmit” or “transceive” data object only. If this data object does not exist, it means all bits are valid.

### 5.2.3.3.3. Reception bit Framing Data Object

For the command APDU, this data object defines the number of expected valid bits of the last byte of data received.

For the response APDU, this data object mentions the number of valid bits in the last byte of received data.

Reception bit Framing Data Object

| Tag | Length (1 byte) | Value |  |
|-----|-----------------|-------|--|
|     |                 | bit   | Description  |
| 92h | 01h             | 0-2   | Number of valid bits of the last byte (0 means all bits are valid) |
|     |                 | 3-7   | RFU  |

If this data object does not exist, it means all bits are valid.

### 5.2.3.3.4. Transmit Data Object

This command transmits the data from IFD to the ICC. No response is expected from the ICC after transmit is complete.

Transmit Data Object

| Tag | Length (1 byte) | Value          |
|-----|-----------------|----------------|
| 93h | DataLen         | Data (N bytes) |

### 5.2.3.3.5. Receive Data Object

This command forces the reader into receiving mode within the time, given in the following timer object.

Receive Data Object

| Tag | Length (1 byte) | Value |
|-----|-----------------|-------|
| 94h | 00h             | -     |

### 5.2.3.3.6. Transceive Data Object

This command transmits and receives data from the ICC. After transmission is complete, the reader will wait until the time given in the timer data object.

If no timer data object was defined in the data field, the reader will wait for the duration given in the Set Parameter FWTI Data Object. If no FWTI is set, the reader will wait for about 302  $\mu$ s.

Transceive Data Object

| Tag | Length (1 byte) | Value          |
|-----|-----------------|----------------|
| 95h | DataLen         | Data (N Bytes) |



### 5.2.3.3.7. Response Status Data Object

Inside the response, this command is used to notify the received data status.

Response Status Data Object

| Tag | Length (1 byte) | Value  |  |   |
|-----|-----------------|--------|--|---|
|     |                 | Byte 0 |  | Byte 1  |
|     |                 | Bit    | Description  |   |
| 96h | 02h             | 0      | 0 – CRC is OK or no checked<br>1 – CRC check fail  | If a collision is detected, these bytes will tell the collision position. Otherwise, "00h" will be shown. |
|     |                 | 1      | 0 – no collision<br>1 – collision detected         |   |
|     |                 | 2      | 0 – no parity error<br>1 – parity error detected   |   |
|     |                 | 3      | 0 – no framing error<br>1 – framing error detected |   |
|     |                 | 4 - 7  | RFU  |   |

### 5.2.3.3.8. Response Data Object

Inside the response, this command is used to notify the received data status.

Response Data Object

| Tag | Length (1 byte) | Value              |
|-----|-----------------|--------------------|
| 97h | DataLen         | ReplyData (N Byte) |



### 5.2.3.4. Switch Protocol Command

This command specifies the protocol and different layers of the standard within the transparent session.

Switch Protocol Command

| Command    | Class | INS | P1  | P2  | Lc      | Data In              |
|------------|-------|-----|-----|-----|---------|----------------------|
| SwProtocol | FFh   | C2h | 00h | 02h | DataLen | DataObject (N bytes) |

Where:

#### Data Object (1 byte)

| Tag    | Data Object                 |
|--------|-----------------------------|
| 8Fh    | Switch Protocol Data Object |
| FF 6Dh | Get Parameter               |
| FF 6Eh | Set Parameter               |

Switch Protocol Response Data Object

| Tag    | Data Object               |
|--------|---------------------------|
| C0h    | Generic Error status      |
| FF 6Dh | IFD parameter data object |

#### 5.2.3.4.1. Switch Protocol Data Object

This command specifies the protocol and different layers of the standard.

Switch Protocol Data Object

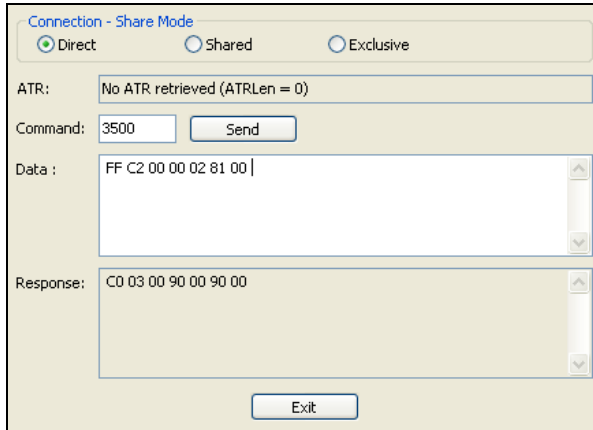
| Tag | Length (1 byte) | Value   |  |
|-----|-----------------|---|--|
|     |                 | Byte 0  | Byte 1   |
| 8Fh | 02h             | 00h – ISO/IEC14443 Type A<br>01h – ISO/IEC14443 Type B<br>03h – FeliCa<br>Other – RFU | 00h – If no layer separation<br>02h – Switch to Layer 2<br>03h – Switch or activate to layer 3<br>04h – Activate to layer 4<br>Other - RFU |

### 5.2.3.5. PCSC 2.0 Part 3 Example

1. Start Transparent Session.

Command: **FF C2 00 00 02 81 00**

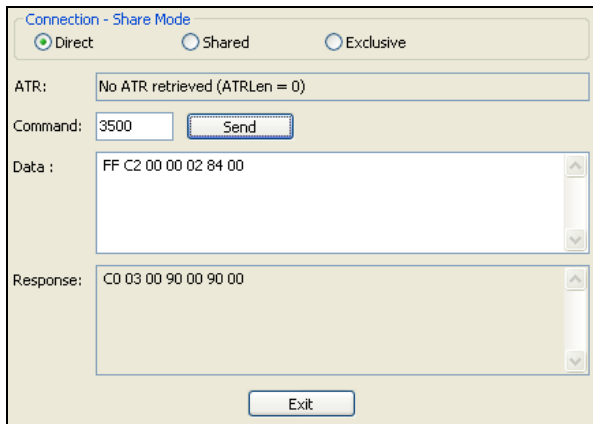
Response: **C0 03 00 90 00 90 00**



2. Turn the Antenna Field on.

Command: **FF C2 00 00 02 84 00**

Response: **C0 03 00 90 00 90 00**



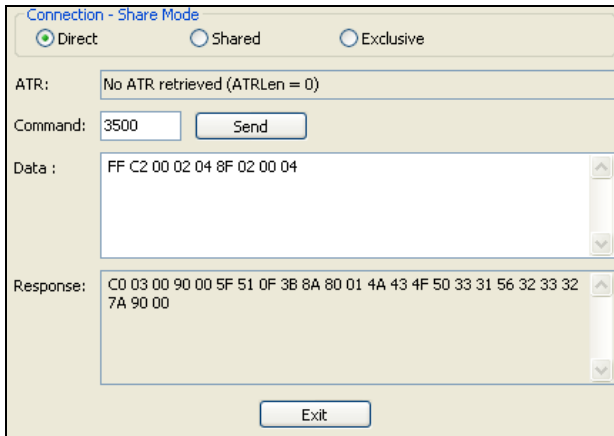


3. ISO 14443-4A Active.

Command: **FF C2 00 02 04 8F 02 00 04**

Response: **C0 03 01 64 01 90 00** (if no card present)

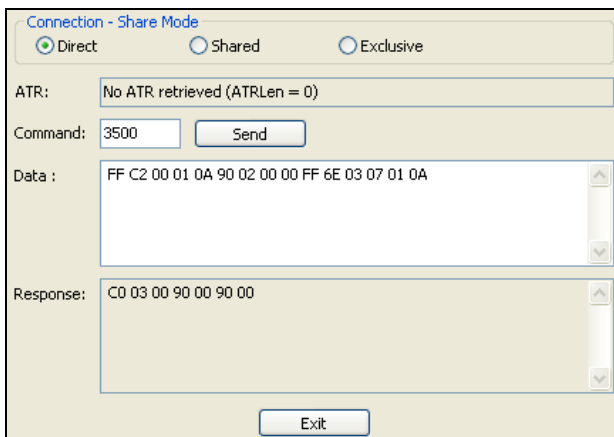
**C0 03 00 90 00 5F 51 [ATR] 90 00**



4. Set the PCB to 0Ah and enable the CRC, parity and protocol prologue in the transmit data.

Command: **FF C2 00 01 0A 90 02 00 00 FF 6E 03 07 01 0A**

Response: **C0 03 00 90 00 90 00**





5. Send the APDU “80B2000008” to card and get response.

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

Response: **C0 03 00 90 00 92 01 00 96 02 00 00 97 0C [Card Response] 90 00**

The screenshot shows a software window titled "Connection - Share Mode" with three radio buttons: "Direct" (selected), "Shared", and "Exclusive". Below the radio buttons, there are three text fields: "ATR:" containing "No ATR retrieved (ATRLen = 0)", "Command:" containing "3500", and "Data:" containing the hex command "FF C2 00 01 0E 5F 46 04 40 42 0F 00 95 05 80 B2 00 00 08". A "Send" button is located to the right of the Command field. Below the Data field, there is a "Response:" field containing the hex response "C0 03 00 90 00 92 01 00 96 02 00 00 97 0C 0B 00 01 02 03 04 05 06 07 08 90 00 90 00". An "Exit" button is at the bottom center.

6. End Transparent Session.

Command: **FF C2 00 00 02 82 00**

Response: **C0 03 00 90 00 90 00**

The screenshot shows the same software window as above. The "Command:" field now contains "3500" and the "Data:" field contains the hex command "FF C2 00 00 02 82 00". The "Response:" field contains the hex response "C0 03 00 90 00 90 00". The "Exit" button remains at the bottom center.



## 5.2.4. PICC Commands for MIFARE Classic (1K/4K) memory cards

### 5.2.4.1. Load Authentication Keys

This command loads the authentication keys to the reader. The authentication keys are used to authenticate the particular sector of the MIFARE Classic (1K/4K) memory card.

Load Authentication Keys APDU Format (11 bytes)

| Command                  | Class | INS | P1            | P2         | Lc  | Data In       |
|--------------------------|-------|-----|---------------|------------|-----|---------------|
| Load Authentication Keys | FFh   | 82h | Key Structure | Key Number | 06h | Key (6 bytes) |

Where:

**Key Structure** 1 byte.

00h = Key is loaded into the reader volatile memory.

Other = Reserved.

**Key Number** 1 byte.

00h ~ 01h = Volatile memory for storing a temporary key. The key will disappear once the reader is disconnected from the computer. Two volatile keys are provided. The volatile key can be used as a session key for different sessions. *Default Value = {FF FF FF FF FF FFh}*

**Key** 6 bytes.

The key value loaded into the reader. Example: FF FF FF FF FF FFh

Load Authentication Keys Response Format (2 Bytes)

| Response | Data Out |     |
|----------|----------|-----|
| Result   | SW1      | SW2 |

Load Authentication Keys Response Codes

| Results | SW1 | SW2 | Meaning                                   |
|---------|-----|-----|---|
| Success | 90h | 00h | The operation was completed successfully. |
| Error   | 63h | 00h | The operation failed.                     |

#### Example:

// Load a key {FF FF FF FF FF FFh} into the volatile memory location 00h.

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



### 5.2.4.2. Authentication for MIFARE Classic (1K/4K)

This command uses the keys stored in the reader to do authentication with the MIFARE Classic (1K/4K) card (PICC). Two types of authentication keys are used: TYPE\_A and TYPE\_B.

Load Authentication Keys APDU Format (6 bytes) [Obsolete]

| Command        | Class | INS | P1  | P2           | P3       | Data In    |
|----------------|-------|-----|-----|--------------|----------|------------|
| Authentication | FFh   | 88h | 00h | Block Number | Key Type | Key Number |

Load Authentication Keys APDU Format (10 bytes)

| Command        | Class | INS | P1  | P2  | Lc  | Data In                 |
|----------------|-------|-----|-----|-----|-----|-------------------------|
| Authentication | FFh   | 86h | 00h | 00h | 05h | Authenticate Data Bytes |

Authenticate Data Bytes (5 bytes)

| Byte1       | Byte 2 | Byte 3       | Byte 4   | Byte 5     |
|-------------|--------|--------------|----------|------------|
| Version 01h | 00h    | Block Number | Key Type | Key Number |

Where:

**Block Number** 1 byte. The memory block to be authenticated.

For MIFARE Classic 1K card, it has totally 16 sectors and each sector consists of four consecutive blocks (e.g., Sector 00h consists of blocks {00h, 01h, 02h and 03h}; sector 01h consists of blocks {04h, 05h, 06h and 07h}; the last sector 0Fh consists of blocks {3Ch, 3Dh, 3Eh and 3Fh}. Once the authentication is done successfully, there is no need to do the authentication again provided that the blocks to be accessed are belonging to the same sector. Please refer to the MIFARE Classic (1K/4K) specification for more details.

**Note:** Once the block is authenticated successfully, all the blocks belonging to the same sector are accessible.

**Key Type** 1 byte.

60h = Key is used as a TYPE A key for authentication.

61h = Key is used as a TYPE B key for authentication.

**Key Number** 1 byte.

00 ~ 01h = Volatile memory for storing keys. The keys will disappear when the reader is disconnected from the PC. Two volatile keys are provided. The volatile key can be used as a session key for different sessions.

Load Authentication Keys Response Format (2 bytes)

| Response | Data Out |     |
|----------|----------|-----|
| Result   | SW1      | SW2 |



Load Authentication Keys Response Codes

| Results | SW1 | SW2 | Meaning                                   |
|---------|-----|-----|---|
| Success | 90  | 00h | The operation was completed successfully. |
| Error   | 63  | 00h | The operation failed.                     |

| Sectors<br>(Total 16 sectors. Each sector consists of 4 consecutive blocks) | Data Blocks<br>(3 blocks, 16 bytes per block) | Trailer Block<br>(1 block, 16 bytes) |            |
|---|---|--------------------------------------|------------|
| Sector 0  | 00h – 02h                                     | 03h                                  | } 1K bytes |
| Sector 1  | 04h – 06h                                     | 07h                                  |            |
| ..  | ..  | ..                                   |            |
| ..  | ..  | ..                                   |            |
| Sector 14   | 38h – 0Ah                                     | 3Bh                                  |            |
| Sector 15   | 3Ch – 3Eh                                     | 3Fh                                  |            |

**Table 2:** MIFARE Classic 1K Memory Map

| Sectors<br>(Total 32 sectors. Each sector consists of 4 consecutive blocks) | Data Blocks<br>(3 blocks, 16 bytes per block) | Trailer Block<br>(1 block, 16 bytes) |        |
|---|---|--------------------------------------|--------|
| Sector 0  | 00h ~ 02h                                     | 03h                                  | } 2 KB |
| Sector 1  | 04h ~ 06h                                     | 07h                                  |        |
| ..  |   |                                      |        |
| ..  |   |                                      |        |
| Sector 30   | 78h ~ 7Ah                                     | 7Bh                                  |        |
| Sector 31   | 7Ch ~ 7Eh                                     | 7Fh                                  |        |

| Sectors<br>(Total 8 sectors. Each sector consists of 16 consecutive blocks) | Data Blocks<br>(15 blocks, 16 bytes per block) | Trailer Block<br>(1 block, 16 bytes) |        |
|---|--|--------------------------------------|--------|
| Sector 32   | 80h ~ 8Eh                                      | 8Fh                                  | } 2 KB |
| Sector 33   | 90h ~ 9Eh                                      | 9Fh                                  |        |
| ..  |  |                                      |        |
| ..  |  |                                      |        |
| Sector 38   | E0h ~ EEh                                      | EFh                                  |        |
| Sector 39   | F0h ~ FEh                                      | FFh                                  |        |

**Table 3:** MIFARE Classic 4K Memory Map



| Byte Number     | 0      | 1        | 2      | 3      | Page |
|-----------------|--------|----------|--------|--------|------|
| Serial Number   | SN0    | SN1      | SN2    | BCC0   | 0    |
| Serial Number   | SN3    | SN4      | SN5    | SN6    | 1    |
| Internal/Lock   | BCC1   | Internal | Lock0  | Lock1  | 2    |
| OTP             | OPT0   | OPT1     | OTP2   | OTP3   | 3    |
| Data read/write | Data0  | Data1    | Data2  | Data3  | 4    |
| Data read/write | Data4  | Data5    | Data6  | Data7  | 5    |
| Data read/write | Data8  | Data9    | Data10 | Data11 | 6    |
| Data read/write | Data12 | Data13   | Data14 | Data15 | 7    |
| Data read/write | Data16 | Data17   | Data18 | Data19 | 8    |
| Data read/write | Data20 | Data21   | Data22 | Data23 | 9    |
| Data read/write | Data24 | Data25   | Data26 | Data27 | 10   |
| Data read/write | Data28 | Data29   | Data30 | Data31 | 11   |
| Data read/write | Data32 | Data33   | Data34 | Data35 | 12   |
| Data read/write | Data36 | Data37   | Data38 | Data39 | 13   |
| Data read/write | Data40 | Data41   | Data42 | Data43 | 14   |
| Data read/write | Data44 | Data45   | Data46 | Data47 | 15   |

512 bits  
or  
64 bytes

**Table 4:** MIFARE Ultralight Memory Map

**Examples:**

//Authenticate the Block 04h with a {TYPE A, key number 00h}. For PC/SC V2.01, Obsolete.

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

//Authenticate the Block 04h with a {TYPE A, key number 00h}. For PC/SC V2.07

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

**Note:** MIFARE Ultralight does not need to do any authentication. The memory is free to access.



### 5.2.4.3. Read Binary Blocks

This command retrieves multiple data blocks from the PICC. The data block/trailer block must be authenticated first before executing this command.

Read Binary APDU Format (5 bytes)

| Command            | Class | INS | P1  | P2           | Le                      |
|--------------------|-------|-----|-----|--------------|-------------------------|
| Read Binary Blocks | FFh   | B0h | 00h | Block Number | Number of Bytes to Read |

Where:

- Block Number** 1 byte. The starting block.
- Number of Bytes to Read** 1 byte.  
Multiple of 16 bytes for MIFARE Classic (1K/4K) or Multiple of 4 bytes for MIFARE Ultralight  
Maximum of 16 bytes for MIFARE Ultralight.  
Maximum of 48 bytes for MIFARE Classic 1K. (Multiple Blocks Mode; 3 consecutive blocks)  
Maximum of 240 bytes for MIFARE Classic 4K. (Multiple Blocks Mode; 15 consecutive blocks)

**Example 1:** 10h (16 bytes). The starting block only. (Single Block Mode)

**Example 2:** 40h (64 bytes). From the starting block to starting block+3. (Multiple Blocks Mode)

**Note:** For security reasons, the Multiple Block Mode is used for accessing Data Blocks only. The Trailer Block is not supposed to be accessed in Multiple Blocks Mode. Please use Single Block Mode to access the Trailer Block.

Read Binary Block Response Format (Multiply of 4/16 + 2 bytes)

| Response | Data Out                      |     |     |
|----------|-------------------------------|-----|-----|
| Result   | Data (Multiple of 4/16 Bytes) | SW1 | SW2 |

Read Binary Block Response Codes

| Results | SW1 | SW2 | Meaning                                   |
|---------|-----|-----|---|
| Success | 90h | 00h | The operation was completed successfully. |
| Error   | 63h | 00h | The operation failed.                     |

Examples:

// Read 16 bytes from the binary block 04h (MIFARE Classic 1K/ 4K)

APDU = FF B0 00 04 10h

// Read 240 bytes starting from the binary block 80h (MIFARE Classic 4K)

// Block 80h to Block 8Eh (15 blocks)

APDU = FF B0 00 80 F0h



### 5.2.4.4. Update Binary Blocks

This command writes multiple data blocks into the PICC. The data block/trailer block must be authenticated first before executing this command.

Update Binary APDU Format (Multiple of 16 + 5 bytes)

| Command              | Class | INS | P1  | P2           | Lc                        | Data In                           |
|----------------------|-------|-----|-----|--------------|---------------------------|-----------------------------------|
| Update Binary Blocks | FFh   | D6h | 00h | Block Number | Number of bytes to update | Block Data (Multiple of 16 Bytes) |

Where:

- Block Number** 1 byte. The starting block to be updated.
- Number of bytes to update** 1 byte.
- Multiply of 16 bytes for MIFARE Classic (1K/4K) or 4 bytes for MIFARE Ultralight.
  - Maximum 48 bytes for MIFARE Classic 1K. (Multiple Blocks Mode; 3 consecutive blocks)
  - Maximum 240 bytes for MIFARE Classic 4K. (Multiple Blocks Mode; 15 consecutive blocks)
- Block Data** Multiple of 16 + 2 Bytes, or 6 bytes. The data to be written into the binary block/blocks.

**Example 1:** 10h (16 bytes). The starting block only. (Single Block Mode)

**Example 2:** 30h (48 bytes). From the starting block to starting block +2. (Multiple Blocks Mode)

**Note:** For safety reasons, the Multiple Block Mode is used for accessing data blocks only. The Trailer Block is not supposed to be accessed in Multiple Blocks Mode. Please use Single Block Mode to access the Trailer Block.

Update Binary Block Response Codes (2 bytes)

| Results | SW1 SW2 | Meaning                                   |
|---------|---------|---|
| Success | 90 00h  | The operation was completed successfully. |
| Error   | 63 00h  | The operation failed.                     |

**Examples:**

```
// Update the binary block 04h of MIFARE Classic (1K/4K) with Data {00 01 .. 0Fh}
APDU = {FF D6 00 04 10 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0Fh}

// Update the binary block 04h of MIFARE Ultralight with Data {00 01 02 03h}
APDU = {FF D6 00 04 04 00 01 02 03h}
```

### 5.2.4.5. Value Block Operation (INC, DEC, STORE)

This command is used for manipulating value-based transactions (e.g., increment a value of the value block).

Value Block Operation APDU Format (10 bytes)

| Command               | Class | INS | P1  | P2           | Lc  | Data In |                                       |
|-----------------------|-------|-----|-----|--------------|-----|---------|---------------------------------------|
| Value Block Operation | FFh   | D7h | 00h | Block Number | 05h | VB_OP   | VB_Value<br>(4 Bytes)<br>{MSB .. LSB} |

Where:

- Block Number** 1 byte. The value block to be manipulated.
- VB\_OP** 1 byte.  
 00h = Store the VB\_Value into the block. The block will then be converted to a value block.  
 01h = Increment the value of the value block by the VB\_Value. This command is only valid for value block.  
 02h = Decrement the value of the value block by the VB\_Value. This command is only valid for value block.
- VB\_Value** 4 bytes. The value used for value manipulation. The value is a signed long integer (4 bytes).

**Example 1:** Decimal -4 = {FFh, FFh, FFh, FCh}

| VB_Value |     |     |     |
|----------|-----|-----|-----|
| MSB      |     |     | LSB |
| FFh      | FFh | FFh | FCh |

**Example 2:** Decimal 1 = {00h, 00h, 00h, 01h}

| VB_Value |     |     |     |
|----------|-----|-----|-----|
| MSB      |     |     | LSB |
| 00h      | 00h | 00h | 01h |

Value Block Operation Response Format (2 bytes)

| Response | Data Out |     |
|----------|----------|-----|
| Result   | SW1      | SW2 |

Value Block Operation Response Codes

| Results | SW1 SW2 | Meaning                                   |
|---------|---------|---|
| Success | 90 00h  | The operation was completed successfully. |
| Error   | 63 00h  | The operation failed.                     |



### 5.2.4.6. Read Value Block

This command retrieves the value from a value block. This command is valid only for value blocks.

Read Value Block APDU Format (5 bytes)

| Command          | Class | INS | P1  | P2           | Le  |
|------------------|-------|-----|-----|--------------|-----|
| Read Value Block | FFh   | B1h | 00h | Block Number | 04h |

Where:

**Block Number** 1 byte. The value block to be accessed.

Read Value Block Response Format (4 + 2 bytes)

| Response | Data Out              |     |     |
|----------|-----------------------|-----|-----|
| Result   | Value<br>{MSB .. LSB} | SW1 | SW2 |

Where:

**Value** 4 bytes. The value returned from the card. The value is a signed long integer (4 bytes).

**Example 1:** Decimal -4 = {FFh, FFh, FFh, FCh}

| Value |     |     |     |
|-------|-----|-----|-----|
| MSB   |     |     | LSB |
| FFh   | FFh | FFh | FCh |

**Example 2:** Decimal 1 = {00h, 00h, 00h, 01h}

| Value |     |     |     |
|-------|-----|-----|-----|
| MSB   |     |     | LSB |
| 00h   | 00h | 00h | 01h |

Read Value Block Response Codes

| Results | SW1 | SW2 | Meaning                                   |
|---------|-----|-----|---|
| Success | 90h | 00h | The operation was completed successfully. |
| Error   | 63h | 00h | The operation failed.                     |



### 5.2.4.7. Copy Value Block

This command copies a value from a value block to another value block.

Copy Value Block APDU Format (7 bytes)

| Command          | Class | INS | P1  | P2                  | Lc  | Data In |                     |
|------------------|-------|-----|-----|---------------------|-----|---------|---------------------|
| Copy Value Block | FFh   | D7h | 00h | Source Block Number | 02h | 03h     | Target Block Number |

Where:

**Source Block Number** 1 byte. The value of the source value block will be copied to the target value block.

**Target Block Number** 1 byte. The value block to be restored. The source and target value blocks must be in the same sector.

Copy Value Block Response Format (2 bytes)

| Response | Data Out |     |
|----------|----------|-----|
| Result   | SW1      | SW2 |

Copy Value Block Response Codes

| Results | SW1 SW2 | Meaning                                   |
|---------|---------|---|
| Success | 90 00h  | The operation was completed successfully. |
| Error   | 63 00h  | The operation failed.                     |

#### Examples:

// Store a value "1" into block 05h

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

// Read the value block 05h

APDU = {FF B1 00 05 04h}

// Copy the value from value block 05h to value block 06h

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

// Increment the value block 05h by "5"

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



### 5.2.5. Accessing PCSC-compliant tags (ISO 14443-4)

All ISO 14443-4 compliant cards (PICCs) understand the ISO 7816-4 APDUs. The ACR1251T reader just has to communicate with the ISO 14443-4 compliant cards through exchanging ISO 7816-4 APDUs and responses. ACR1251T will handle the ISO 14443 Parts 1-4 Protocols internally.

MIFARE Classic (1K/4K), MIFARE Mini and MIFARE Ultralight tags are supported through the T=CL emulation. Just simply treat the MIFARE tags as standard ISO 14443-4 tags. For more information, please refer to **Section 5.2.2**.

ISO 7816-4 APDU Format

| Command                       | Class | INS | P1 | P2 | Lc                       | Data In | Le  |
|-------------------------------|-------|-----|----|----|--------------------------|---------|---|
| ISO 7816<br>Part 4<br>Command |       |     |    |    | Length of the<br>Data In |         | Expected<br>length of the<br>Response<br>Data |

ISO 7816-4 Response Format (Data + 2 bytes)

| Response | Data Out      |     |     |
|----------|---------------|-----|-----|
| Result   | Response Data | SW1 | SW2 |

Common ISO 7816-4 Response Codes

| Results | SW1    | Meaning                                   |
|---------|--------|---|
| Success | 90 00h | The operation was completed successfully. |
| Error   | 63 00h | The operation failed.                     |

Typical sequence may be:

1. Present the tag and connect the PICC Interface.
2. Read/Update the memory of the tag.

To do this:

1. Connect the tag.

The ATR of the tag is 3B 88 80 01 00 00 00 00 33 81 81 00 3Ah.

In which,

The Application Data of ATQB = 00 00 00 00, protocol information of ATQB = 33 81 81. It is an ISO 14443-4 Type B tag.

2. Send an APDU, Get Challenge.

<< 00 84 00 00 08h

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

**Note:** For ISO 14443-4 Type A tags, the ATS can be obtained by using the APDU “FF CA 01 00 00h.”



**Example:**

```
// Read 8 bytes from an ISO 14443-4 Type B PICC (ST19XR08E)
```

```
APDU = {80 B2 80 00 08h}
```

```
Class = 80h
```

```
INS = B2h
```

```
P1 = 80h
```

```
P2 = 00h
```

```
Lc = None
```

```
Data In = None
```

```
Le = 08h
```

```
Answer: 00 01 02 03 04 05 06 07h [$9000h]
```



### 5.2.6. Accessing FeliCa tags

For FeliCa access, the command is different from the one used in PCSC-compliant tags and MIFARE. The command follows the FeliCa specification with an added header.

FeliCa Command Format

| Command        | Class | INS | P1  | P2  | Lc                    | Data In                                 |
|----------------|-------|-----|-----|-----|-----------------------|---|
| FeliCa Command | FFh   | 00h | 00h | 00h | Length of the Data In | FeliCa Command (start with Length Byte) |

FeliCa Response Format (Data + 2 bytes)

| Response | Data Out      |
|----------|---------------|
| Result   | Response Data |

#### Read Memory Block Example:

1. Connect the FeliCa.

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

In which, **11 00 3Bh** = FeliCa

2. Read FeliCa IDM.

CMD = FF CA 00 00 00h

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

e.g., FeliCa IDM = 01 01 06 01 CB 09 57 03h

3. FeliCa command access.

Example: "Read" Memory Block.

CMD = FF 00 00 00 10 10 06 **01 01 06 01 CB 09 57 03** 01 09 01 01 80 00h

where:

Felica Command = 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. Peripherals Control

The reader's peripherals control commands are implemented by using **SCardControl** with Control Code **SCARD\_CTL\_CODE(3500)**.

#### 5.3.1. Get Firmware Version

This command is used to get the reader's firmware message.

Get Firmware Version Format (5 bytes)

| Command              | Class | INS | P1  | P2  | Lc  |
|----------------------|-------|-----|-----|-----|-----|
| Get Firmware Version | E0h   | 00h | 00h | 18h | 00h |

Get Firmware Version Response Format (5 bytes + Firmware Message Length)

| Response | Class | INS | P1  | P2  | Le                         | Data Out         |
|----------|-------|-----|-----|-----|----------------------------|------------------|
| Result   | E1h   | 00h | 00h | 00h | Number of bytes to receive | Firmware Version |

#### Example:

Response = E1 00 00 00 0F 41 43 52 31 32 35 31 55 5F 56 44 30 21 2E 30

Firmware Version (HEX) = 41 43 52 31 32 35 31 55 5F 56 36 32 31 2E 30

Firmware Version (ASCII) = "ACR1251T\_V621.0"



### 5.3.2. LED Control

This command is used to control the LED's output.

LED Control Format (6 bytes)

| Command     | Class | INS | P1  | P2  | Lc  | Data In    |
|-------------|-------|-----|-----|-----|-----|------------|
| LED Control | E0h   | 00h | 00h | 29h | 01h | LED Status |

LED Control Response Format (6 bytes)

| Response | Class | INS | P1  | P2  | Le  | Data Out   |
|----------|-------|-----|-----|-----|-----|------------|
| Result   | E1h   | 00h | 00h | 00h | 01h | LED Status |

LED Status (1 byte)

| LED Status | Description | Description     |
|------------|-------------|-----------------|
| Bit 0      | RED LED     | 1 = ON; 0 = OFF |
| Bit 1      | GREEN LED   | 1 = ON; 0 = OFF |
| Bit 2 - 7  | RFU         | RFU             |



### 5.3.3. LED Status

This command is used to check the existing LED's status.

LED Status Format (5 bytes)

| Command    | Class | INS | P1  | P2  | Lc  |
|------------|-------|-----|-----|-----|-----|
| LED Status | E0h   | 00h | 00h | 29h | 00h |

LED Status Response Format (6 bytes)

| Response | Class | INS | P1  | P2  | Le  | Data Out   |
|----------|-------|-----|-----|-----|-----|------------|
| Result   | E1h   | 00h | 00h | 00h | 01h | LED Status |

LED Status (1 byte)

| LED Status | Description | Description     |
|------------|-------------|-----------------|
| Bit 0      | RED LED     | 1 = ON; 0 = OFF |
| Bit 1      | GREEN LED   | 1 = ON; 0 = OFF |
| Bit 2 - 7  | RFU         | RFU             |



### 5.3.4. Set LED Status Indicator Behavior

This command sets the behaviors of LEDs as status indicators.

**Note:** The setting will be saved into non-volatile memory.

Set LED Status Indicator Behavior Format (6 bytes)

| Command                           | Class | INS | P1  | P2  | Lc  | Data In  |
|-----------------------------------|-------|-----|-----|-----|-----|----------|
| Set LED Status Indicator Behavior | E0h   | 00h | 00h | 21h | 01h | Behavior |

Behavior (1 byte)

| Behavior  | MODE                        | Description  |
|-----------|-----------------------------|--|
| Bit 0     | Card Operation Blinking LED | To blink the LED whenever the PICC card is being accessed                      |
| Bit 1     | PICC Polling Status LED     | To show the PICC polling status.<br>1 = Enable; 0 =Disable                     |
| Bit 2     | PICC Activation Status LED  | To show the activation status of the PICC interface.<br>1 = Enable; 0 =Disable |
| Bit 3 – 5 | RFU                         | RFU  |
| Bit 6     | Color Select (GREEN)        | GREEN LED for status change<br>1 = Enable; 0 = Disable                         |
| Bit 7     | Color Select (RED)          | RED LED for status change<br>1 = Enable; 0 = Disable                           |

**Note:** Default value of behavior = 47h.

Set LED Status Indicator Behavior Response Format (6 bytes)

| Response | Class | INS | P1  | P2  | Le  | Data Out          |
|----------|-------|-----|-----|-----|-----|-------------------|
| Result   | E1h   | 00h | 00h | 00h | 01h | Default Behaviors |



### 5.3.5. Read LED Status Indicator Behavior

This command reads the current default behavior of LEDs.

Read LED Status Indicator Behavior Format (5 bytes)

| Command                            | Class | INS | P1  | P2  | Lc  |
|------------------------------------|-------|-----|-----|-----|-----|
| Read LED Status Indicator Behavior | E0h   | 00h | 00h | 21h | 00h |

Read LED Status Indicator Behavior Response Format (6 bytes)

| Response | Class | INS | P1  | P2  | Le  | Data Out  |
|----------|-------|-----|-----|-----|-----|-----------|
| Result   | E1h   | 00h | 00h | 00h | 01h | Behaviors |

Behavior (1 byte)

| Behavior  | MODE                        | Description  |
|-----------|-----------------------------|--|
| Bit 0     | Card Operation Blinking LED | To blink the LED whenever the PICC card is being accessed                      |
| Bit 1     | PICC Polling Status LED     | To show the PICC polling status.<br>1 = Enable; 0 =Disable                     |
| Bit 2     | PICC Activation Status LED  | To show the activation status of the PICC interface.<br>1 = Enable; 0 =Disable |
| Bit 3 - 5 | RFU                         | RFU  |
| Bit 6     | Color Select (GREEN)        | GREEN LED for status change<br>1 = Enable; 0 = Disable                         |
| Bit 7     | Color Select (RED)          | RED LED for status change<br>1 = Enable; 0 = Disable                           |

**Note:** Default value of Behavior = 47h.



### 5.3.6. Set Automatic PICC Polling

This command is used to set the reader's polling mode.

Whenever the reader is connected to a PC, the PICC polling function will start the PICC scanning to determine if a PICC is placed on/removed from the built-in antenna.

You can send a command to disable the PICC polling function. The command is sent through the PCSC Escape command interface. To meet the energy saving requirement, special modes are provided for turning off the antenna field whenever the PICC is inactive, or no PICC is found. The reader will consume less current in power saving mode.

**Note:** The setting will be saved into non-volatile memory.

Set Automatic PICC Polling Format (6 bytes)

| Command                    | Class | INS | P1  | P2  | Lc  | Data In         |
|----------------------------|-------|-----|-----|-----|-----|-----------------|
| Set Automatic PICC Polling | E0h   | 00h | 00h | 23h | 01h | Polling Setting |

Set Automatic PICC Polling Response Format (6 bytes)

| Response | Class | INS | P1  | P2  | Le  | Data Out        |
|----------|-------|-----|-----|-----|-----|-----------------|
| Result   | E1h   | 00h | 00h | 00h | 01h | Polling Setting |

Polling Setting (1 byte)

| Polling Setting | Mode  | Description   |
|-----------------|---|---|
| Bit 0           | Auto PICC Polling                               | 1 = Enable; 0 =Disable  |
| Bit 1           | Turn off Antenna Field if no PICC is found.     | 1 = Enable; 0 =Disable  |
| Bit 2           | Turn off Antenna Field if the PICC is inactive. | 1 = Enable; 0 =Disable  |
| Bit 3           | RFU   | RFU   |
| Bit 5 .. 4      | PICC Poll Interval for PICC                     | <Bit 5 – Bit 4><br><0 – 0> = 250 ms<br><0 – 1> = 500 ms<br><1 – 0> = 1000 ms<br><1 – 1> = 2500 ms |
| Bit 6           | RFU   | RFU   |
| Bit 7           | Enforce ISO 14443-A Part 4                      | 1= Enable; 0= Disable.  |

**Note:** Default value of Polling Setting = 8Bh.



**Reminders:**

1. It is recommended to enable the option “Turn Off Antenna Field if the PICC is inactive”, so that the “Inactive PICC” will not be exposed to the field all the time to prevent the PICC from “warming up”.
2. The longer the PICC Poll Interval, the more efficient of energy saving. However, the response time of PICC Polling will become longer. The Idle Current Consumption in Power Saving Mode is about 60 mA, while the Idle Current Consumption in Non-Power Saving mode is about 130mA. **Note:** Idle Current Consumption = PICC is not activated.
3. The reader will activate the ISO 14443A-4 mode of the “ISO 14443A-4 compliant PICC” automatically. Type B PICC will not be affected by this option.
4. The JCOP30 card comes with two modes: ISO 14443A-3 (MIFARE Classic 1K) and ISO 14443A-4 modes. The application has to decide which mode should be selected once the PICC is activated.



### 5.3.7. Read Automatic PICC Polling

This command is used to check the current PICC polling setting.

Read Automatic PICC Polling Format (5 bytes)

| Command                     | Class | INS | P1  | P2  | Lc  |
|-----------------------------|-------|-----|-----|-----|-----|
| Read Automatic PICC Polling | E0h   | 00h | 00h | 23h | 00h |

Read Automatic PICC Polling Response Format (6 bytes)

| Response | Class | INS | P1  | P2  | Le  | Data Out        |
|----------|-------|-----|-----|-----|-----|-----------------|
| Result   | E1h   | 00h | 00h | 00h | 01h | Polling Setting |

Polling Setting (1 byte)

| Polling Setting | Mode  | Description   |
|-----------------|---|---|
| Bit 0           | Auto PICC Polling                               | 1 = Enable; 0 =Disable  |
| Bit 1           | Turn off Antenna Field if no PICC is found.     | 1 = Enable; 0 =Disable  |
| Bit 2           | Turn off Antenna Field if the PICC is inactive. | 1 = Enable; 0 =Disable  |
| Bit 3           | RFU   | RFU   |
| Bit 5 .. 4      | PICC Poll Interval for PICC                     | <Bit 5 – Bit 4><br><0 – 0> = 250 ms<br><0 – 1> = 500 ms<br><1 – 0> = 1000 ms<br><1 – 1> = 2500 ms |
| Bit 6           | RFU   | RFU   |
| Bit 7           | Enforce ISO 14443-A Part 4                      | 1= Enable; 0= Disable.  |

**Note:** Default value of Polling Setting = 8Bh.





### 5.3.8. Set PICC Operating Parameter

This command is used to set the PICC operating parameter.

**Note:** The setting will be saved into non-volatile memory.

Set PICC Operating Parameter Format (6 bytes)

| Command                      | Class | INS | P1  | P2  | Lc  | Data In             |
|------------------------------|-------|-----|-----|-----|-----|---------------------|
| Set PICC Operating Parameter | E0h   | 00h | 00h | 20h | 01h | Operating Parameter |

Set PICC Operating Parameter Response Format (6 bytes)

| Response | Class | INS | P1  | P2  | Le  | Data Out            |
|----------|-------|-----|-----|-----|-----|---------------------|
| Result   | E1    | 00h | 00h | 00h | 01h | Operating Parameter |

Operating Parameter (1 byte)

| Operating Parameter | Parameter        | Description                                       | Option                 |
|---------------------|------------------|---|------------------------|
| Bit 0               | ISO 14443 Type A | The Tag Types to be detected during PICC Polling. | 1 = Detect<br>0 = Skip |
| Bit 1               | ISO 14443 Type B |   | 1 = Detect<br>0 = Skip |
| Bit 2               | FeliCa 212 Kbps  |   | 1 = Detect<br>0 = Skip |
| Bit 3               | FeliCa 424 Kbps  |   | 1 = Detect<br>0 = Skip |
| Bit 4               | Topaz            |   | 1 = Detect<br>0 = Skip |
| Bit 5 - 7           | RFU              | RFU   | RFU                    |

**Note:** Default value of Operating Parameter = 1Fh.



### 5.3.9. Read PICC Operating Parameter

This command is used to check the current PICC operating parameter.

Read PICC Operating Parameter Format (5 bytes)

| Command                       | Class | INS | P1  | P2  | Lc  |
|-------------------------------|-------|-----|-----|-----|-----|
| Read PICC Operating Parameter | E0h   | 00h | 00h | 20h | 00h |

Read PICC Operating Parameter Response Format (6 bytes)

| Response | Class | INS | P1  | P2  | Le  | Data Out            |
|----------|-------|-----|-----|-----|-----|---------------------|
| Result   | E1h   | 00h | 00h | 00h | 01h | Operating Parameter |

Operating Parameter (1 byte)

| Operating Parameter | Parameter        | Description                                       | Option                 |
|---------------------|------------------|---|------------------------|
| Bit 0               | ISO 14443 Type A | The Tag Types to be detected during PICC polling. | 1 = Detect<br>0 = Skip |
| Bit 1               | ISO 14443 Type B |   | 1 = Detect<br>0 = Skip |
| Bit 2               | FeliCa 212 Kbps  |   | 1 = Detect<br>0 = Skip |
| Bit 3               | FeliCa 424 Kbps  |   | 1 = Detect<br>0 = Skip |
| Bit 4               | Topaz            |   | 1 = Detect<br>0 = Skip |
| Bit 5 - 7           | RFU              | RFU   | RFU                    |

**Note:** Default value of Operating Parameter = 1Fh.



### 5.3.10. Set Auto PPS

Whenever a PICC is recognized, the reader will try to change the communication speed between the PCD and PICC as defined by the maximum connection speed. If the card does not support the proposed connection speed, the reader will try to connect the card with a slower speed setting.

Set Auto PPS Format (7 bytes)

| Command      | Class | INS | P1  | P2  | Lc  | Data In      |              |
|--------------|-------|-----|-----|-----|-----|--------------|--------------|
| Set Auto PPS | E0h   | 00h | 00h | 24h | 02h | Max Tx Speed | Max Rx Speed |

Set Auto PPS Response Format (9 bytes)

| Response | Class | INS | P1  | P2  | Le  | Data Out     |                  |              |                  |
|----------|-------|-----|-----|-----|-----|--------------|------------------|--------------|------------------|
| Result   | E1h   | 00h | 00h | 00h | 04h | Max Tx Speed | Current Tx Speed | Max Rx Speed | Current Rx Speed |

Where:

**Max Tx Speed** Maximum Tx Speed (1 Byte)

**Current Tx Speed** Current Tx Speed (1 Byte)

**Max Rx Speed** Maximum Rx Speed (1 Byte)

**Current Rx Speed** Current Rx Speed (1 Byte)

00h = 106 Kbps; default setting, equal to No Auto PPS

01h = 212 Kbps

02h = 424 Kbps

#### Notes:

1. Normally, the application should know the maximum connection speed of the PICCs being used. The environment also affects the maximum achievable speed. The reader just uses the proposed communication speed to talk with the PICC. The PICC will become inaccessible if the PICC or environment does not meet the requirement of the proposed communication speed.
2. The reader supports different speeds for sending and receiving.



### 5.3.11. Read Auto PPS

This command checks the current auto PPS setting.

Read Auto PPS Format (5 bytes)

| Command       | Class | INS | P1  | P2  | Lc  |
|---------------|-------|-----|-----|-----|-----|
| Read Auto PPS | E0h   | 00h | 00h | 24h | 00h |

Read Auto PPS Response Format (9 bytes)

| Response | Class | INS | P1  | P2  | Le  | Data Out     |                  |              |                  |
|----------|-------|-----|-----|-----|-----|--------------|------------------|--------------|------------------|
| Result   | E1h   | 00h | 00h | 00h | 04h | Max Tx Speed | Current Tx Speed | Max Rx Speed | Current Rx Speed |

Where:

**Max Tx Speed** Maximum Tx Speed (1 Byte)

**Current Tx Speed** Current Tx Speed (1 Byte)

**Max Rx Speed** Maximum Rx Speed (1 Byte)

**Current Rx Speed** Current Rx Speed (1 Byte)

00h = 106 Kbps; default setting, equal to No Auto PPS

01h = 212 Kbps

02h = 424 Kbps



## 5.4. ACR122T Compatible Commands

### 5.4.1. Bi-color LED Control

This command is used to control the states of the bi-color LED.

Bi-color LED Control Command Format (9 bytes)

| Command              | Class | INS | P1  | P2                | Lc  | Data In (4 bytes)         |
|----------------------|-------|-----|-----|-------------------|-----|---------------------------|
| Bi-color LED Control | FFh   | 00h | 40h | LED State Control | 04h | Blinking Duration Control |

#### P2 LED State Control

Bi-color LED Control Format (1 byte)

| CMD   | Item                             | Description                           |
|-------|----------------------------------|---------------------------------------|
| Bit 0 | Final Red LED State              | 1 = On; 0 = Off                       |
| Bit 1 | Final Green LED State            | 1 = On; 0 = Off                       |
| Bit 2 | Red LED State Mask               | 1 = Update the State<br>0 = No change |
| Bit 3 | Green LED State Mask             | 1 = Update the State<br>0 = No change |
| Bit 4 | Initial Red LED Blinking State   | 1 = On; 0 = Off                       |
| Bit 5 | Initial Green LED Blinking State | 1 = On; 0 = Off                       |
| Bit 6 | Red LED Blinking Mask            | 1 = Blink<br>0 = Not Blink            |
| Bit 7 | Green LED Blinking Mask          | 1 = Blink<br>0 = Not Blink            |

#### Data In Blinking Duration Control

Bi-color LED Blinking Duration Control Format (4 bytes)

| Byte 0   | Byte 1  | Byte 2                | Byte 3 |
|--|---|-----------------------|--------|
| T1 Duration<br>Initial Blinking State<br>(unit = 100 ms) | T2 Duration<br>Toggle Blinking State<br>(unit = 100 ms) | Number of repetitions | 00h    |

#### Data Out SW1 SW2. Status Code returned by the reader.

Status Code

| Results | SW1 | SW2               | Meaning                                   |
|---------|-----|-------------------|---|
| Success | 90h | Current LED State | The operation was completed successfully. |
| Error   | 63  | 00h               | The operation failed.                     |



Current LED State (1 byte)

| Status     | Item              | Description     |
|------------|-------------------|-----------------|
| Bit 0      | Current Red LED   | 1 = On; 0 = Off |
| Bit 1      | Current Green LED | 1 = On; 0 = Off |
| Bits 2 – 7 | Reserved          |                 |

**Reminders:**

1. *The LED State operation will be performed after the LED Blinking operation is completed.*
2. *The LED will not change if the corresponding LED Mask is not enabled.*
3. *The LED will not blink if the corresponding LED Blinking Mask is not enabled. Also, the number of repetitions must be greater than zero.*
4. *T1 and T2 duration parameters are used for controlling the duty cycle of LED blinking and Buzzer Turn-On duration. For example, if T1=1 and T2=1, the duty cycle = 50%.*

**Note:** *Duty Cycle =  $T1 / (T1 + T2)$ .*



### 5.4.2. Get Firmware Version

This command is used to retrieve the firmware version of the reader.

Get Firmware Version Command Format (5 bytes)

| Command      | Class | INS | P1  | P2  | Le  |
|--------------|-------|-----|-----|-----|-----|
| Get Firmware | FFh   | 00h | 48h | 00h | 00h |

Get Firmware Version Response Format (X bytes)

| Response | Data Out         |
|----------|------------------|
| Result   | Firmware Version |

#### Example:

Response = 41 43 52 31 32 35 31 54 5F 56 44 30 30 2E 30h = ACR1251T\_VD00.0 (ASCII)



### 5.4.3. Get PICC Operating Parameter

This command is used to get the PICC operating parameter of the reader.

Get the PICC Operating Parameter Command Format (5 bytes)

| Command                      | Class | INS | P1  | P2  | Le  |
|------------------------------|-------|-----|-----|-----|-----|
| Get PICC Operating Parameter | FFh   | 00h | 50h | 00h | 00h |

Get the PICC Operating Parameter Response Format (2 byte)

| Response | Data Out |                          |
|----------|----------|--------------------------|
| Result   | 90h      | PICC Operating Parameter |

PICC Operating Parameter

| Bit | Parameter   | Description   | Option                    |
|-----|---|---|---------------------------|
| 7   | Auto PICC Polling   | To enable the PICC polling.   | 1 = Enable<br>0 = Disable |
| 6   | Auto ATS Generation   | To issue ATS request whenever an ISO 14443-4 Type A tag is activated. | 1 = Enable<br>0 = Disable |
| 5   | Polling Interval  | To set the time interval between successive PICC polling.             | 1 = 250 ms<br>0 = 500 ms  |
| 4   | FeliCa 424 Kbps   | The Tag Types to be detected during PICC polling.                     | 1 = Detect<br>0 = Skip    |
| 3   | FeliCa 212 Kbps   |   | 1 = Detect<br>0 = Skip    |
| 2   | Topaz   |   | 1 = Detect<br>0 = Skip    |
| 1   | ISO 14443 Type B  |   | 1 = Detect<br>0 = Skip    |
| 0   | ISO 14443 Type A<br><i>Note: To detect the MIFARE tags, the Auto ATS Generation must be disabled first.</i> |   | 1 = Detect<br>0 = Skip    |





#### 5.4.4. Set PICC Operating Parameter

This command is used to set the PICC operating parameter of the reader.

Set PICC Operating Parameter Command Format (5 bytes)

| Command                      | Class | INS | P1  | P2                       | Le  |
|------------------------------|-------|-----|-----|--------------------------|-----|
| Set PICC Operating Parameter | FFh   | 00h | 51h | PICC Operating Parameter | 00h |

Set PICC Operating Parameter Response Format (2 byte)

| Response | Data Out                     |
|----------|------------------------------|
| Result   | 90h PICC Operating Parameter |

PICC Operating Parameter

| Bit | Parameter   | Description   | Option                    |
|-----|---|---|---------------------------|
| 7   | Auto PICC Polling   | To enable the PICC polling.   | 1 = Enable<br>0 = Disable |
| 6   | Auto ATS Generation   | To issue ATS request whenever an ISO 14443-4 Type A tag is activated. | 1 = Enable<br>0 = Disable |
| 5   | Polling Interval  | To set the time interval between successive PICC polling.             | 1 = 250 ms<br>0 = 500 ms  |
| 4   | FeliCa 424 Kbps   | The Tag Types to be detected during PICC polling.                     | 1 = Detect<br>0 = Skip    |
| 3   | FeliCa 212 Kbps   |   | 1 = Detect<br>0 = Skip    |
| 2   | Topaz   |   | 1 = Detect<br>0 = Skip    |
| 1   | ISO 14443 Type B  |   | 1 = Detect<br>0 = Skip    |
| 0   | ISO 14443 Type A<br><i>Note: To detect the MIFARE tags, the Auto ATS Generation must be disabled first.</i> |   | 1 = Detect<br>0 = Skip    |

Android is a trademark of Google, LLC.  
Microsoft is a registered trademark of the Microsoft Corporation in the United States and/or other countries.  
MIFARE, MIFARE Classic, MIFARE DESFire and MIFARE Ultralight are trademarks of NXP B.V.