15370 Barranca Parkway
Irvine. CA 92618

# 55x3 PC/SC Driver

## USER GUIDE

**Driver Version** 2.1.1.3

**Document Number** 4505-USM-01, Rev A.1

Mai 2011

## Copyright / Trademarks



## Version History

| Date | Author | Description | Document Version |
|---|---|---|---|
| 05/17/11 | HBauer | EMVCo support | A.1 |
| 10/28/10 | L. Dixon | Reformat | A.0 |
| 09/20/07 | C Hubmann | Initial Version | 1.0 |

## Contact

www.HIDGlobal.com

| | |
|---|---|
| **North America** | 949 732 2000 |
| **Toll Free** | 800 237 7769 |
| **Europe, Middle East, Africa** | + 49 6123 791 0 |
| **Asia Pacific** | 852 3160 9800 |
| **Latin America** | + 52 477 779 1492 |
| **Email** | support@hidglobal.com |

## Contents

**1 Features ..... 3**

**2 Extended Features ..... 7**

**3 Virtual COM-Port Interface (VCOM) ..... 14**

**4 Contactless EMVCo Support ..... 15**

**5 Appendix – Code Example ..... 18**

# 1 Features

This driver is fully compliant to the PC/SC 2.01.4 specification for contactless Smart cards / Storage cards.

See http://www.pcscworkgroup.com > Specifications > Download Specifications for the latest PC/SC Specification.

## 1.1 Summary

- ISO14443(A/B) cards
- Multiple Antennas
- Multiple Tags per Antenna
- ATR driver
- Get Data command
- Load Keys command
- Authenticate command (PC/SC 2.01.3)
- General Authenticate command
- Read Binary command
- Update Binary command
- Increment command
- Decrement command

## 1.2 ISO14443(A/B) cards

This driver supports contactless cards, compliant with ISO14443 of type A (part 3 and part 4) and type B, both up to a communication rate of 848 kBit/s on the air interface.

## 1.3 Multiple Antennas

This driver supports readers with multiple antenna configurations. Furthermore, completely customize the commands that trigger the antenna toggling.

## 1.4 Multiple Tags per Antenna

Using recent firmware, the driver handles multiple tags on the same antenna at the same time. It also allows the simultaneous reading of multiple tags.

## 1.5 ATR

This driver constructs ATRs for contactless cards as defined in PC/SC 2.01.4 specification, July 2007, Part 3 - Requirements for PC-Connected Interface Devices, Section 3.1.3.2.3. and further adjusts them for compliance to the next revision of the PC/SC Workgroup specification.

## 1.6 Get Data command

See to PC/SC 2.01.4 specification, July 2007, Part 3 - Requirements for PC- Connected Interface Devices, Section 3.2.2.1.3.

## 1.7 PC/SC 2.01.4 compliant Load Keys command

See to PC/SC 2.01.4 specification, July 2007, Part 3 - Requirements for PC- Connected Interface Devices, Section 3.2.2.1.4.

## 1.8 PC/SC 2.01.3 compliant Authenticate command

See to PC/SC 2.01.3 specification, January 2006, Part 3 - Requirements for PC- Connected Interface Devices, Section 3.2.2.1.5.

**Note:** This command only supports MIFARE® cards at the moment

## 1.9 PC/SC 2.01.4 compliant General Authenticate command

See to PC/SC 2.01.4 specification, July 2007, Part 3 - Requirements for PC- Connected Interface Devices, Section 3.2.2.1.5.

**Note:** This command only supports MIFARE® cards at the moment

## 1.10 PC/SC 2.01.4 compliant Read Binary command

See to PC/SC 2.01.4 specification, July 2007, Part 3 - Requirements for PC- Connected Interface Devices, Section 3.2.2.1.7.

**Note:** This command only supports MIFARE® cards at the moment

## 1.11 PC/SC 2.01.4 compliant Update Binary command

See to PC/SC 2.01.4 specification, July 2007, Part 3 - Requirements for PC- Connected Interface Devices, Section 3.2.2.1.8.

**Note:** This command only supports MIFARE® cards at the moment

## 1.12 Increment command

This command increments the value of a block on a storage card.

**Command Syntax**

| Command | Data |
|---|---|
| CLA | 'FF' |
| INS | 'D4' |
| P1 | MSB of block address |
| P2 | LSB of block address |
| LC | 4 |
| Data Field | byte value array indicating block increment |
| Le | Empty |

**Response Syntax**

| | Data | | Comment |
|---|---|---|---|
| **Data Field** | SW1 | SW2 | Status word as described below |
| **Empty** | '90' | '00' | Success |
| | '65' | '81' | Memory failure (unsuccessful increment) |
| | '69' | '82' | Security status not satisfied |
| | '6A' | '82' | Invalid block address |
| | '6C' | 'XX' | Wrong length (wrong number Lc; 'XX' is the exact |

**Note:** This command only supports MIFARE® cards at the moment

## 1.13 Decrement command

This command decrements the value of a block on a storage card.

**Command Syntax**

| Command | Data |
|---|---|
| CLA | 'FF' |
| INS | 'D8' |
| P1 | MSB of block address |
| P2 | LSB of block address |
| LC | 4 |
| Data Field | byte value array indicating block decrement |
| Le | Empty |

**Response Syntax**

| | Data | | Comment |
|---|---|---|---|
| **Data Field** | SW1 | SW2 | status word as described below |
| **Empty** | '90' | '00' | Success |
| | '65' | '81' | memory failure (unsuccessful decrement) |
| | '69' | '82' | security status not satisfied |
| | '6A' | '82' | invalid block address |
| | '6C' | 'XX' | Wrong length (wrong number Lc; 'XX' is the exact |

**Note:** This command only supports MIFARE® cards at the moment

# 2 Extended Features

Additional features are implemented into the driver as I/O-Controls. The purpose of these features is to extend the functionality / interoperability and to ease development using a combination of 55x3 readers and this driver.

Features listed are accessible through the included Driver Applet. See individual commands for detailed usage information.

For an example of how to invoke I/O-Controls, see to Appendix – Code Example, page 18.

## 2.1 Summary

- Query Driver Version
- Query Card
- Query CID
- Get UID
- Query Reader Version
- Query Running
- Set Airspeed
- Query Airspeed
- Query AsyncBaudRates
- Set Timeout
- Query Timeout
- Request Read Settings
- Set ISO14443-4 Support
- Query ISO14443-4 Support
- Set Antenna

## 2.2 Query Driver Version

This I/O-Control returns the driver's version string in ASCII-Coding.

| IOCTL-Code | Input Buffer | Output Buffer size |
|---|---|---|
| 2065 | - | >= 8 bytes recommended |

**Response**

| Data | Comment |
|---|---|
| Char* | Version String |

## 2.3 Query Card

This I/O-Control returns whether a card is available in a requested slot or not.

| IOCTL-Code | Input Buffer | Output Buffer size |
|---|---|---|
| 2050 | 1 byte | 1 byte |

**Input values**

| Data | Comment |
|---|---|
| 0x00 - 0xXX | Desired card slot |

**Response values**

| Data | Comment |
|---|---|
| 0x00 | No card is present |
| 0x01 | A card is present |

## 2.4 Query CID

This I/O-Control returns whether the card in a requested slot supports/uses CID or not.

| IOCTL-Code | Input Buffer | Output Buffer size |
|---|---|---|
| 2062 | 1 byte | 1 byte |

**Input values**

| Data | Comment |
|---|---|
| 0x00 - 0xXX | Desired card slot |

**Response values**

| Data | Comment |
|---|---|
| 0x00 | Present card supports CID |
| 0x01 | Present card does not support CID |

## 2.5 Get UID

This I/O-Control returns the UID preceded by the tag type of the card in a requested slot.

| IOCTL-Code | Input Buffer | Output Buffer size |
|---|---|---|
| 2304 | 1 byte | 1 byte + UID size |

**Input values**

| Data | Comment |
|---|---|
| 0x00 - 0xXX | Desired card slot |

**Response**

Returns the present card's UID preceded by the card's type (1 Byte). The byte-mapping is described in your reader manual within the New Serial Mode section.

## 2.6 Query Reader Version

This I/O-Control returns the reader's version string in ASCII-Coding.

| IOCTL-Code | Input Buffer | Output Buffer size |
|---|---|---|
| 2053 | - | >= 16 bytes recommended |

**Response**

| Data | Comment |
|---|---|
| Char* | Version String |

## 2.7 Query Polling

This I/O-Control returns whether card polling is enabled or not.

| IOCTL-Code | Input Buffer | Output Buffer size |
|---|---|---|
| 2064 | - | 1 byte |

**Response values**

| Data | Comment |
|---|---|
| 0x00 | Drivers polling routines have been stopped |
| 0x01 | Driver is operating normal and polls card as usual |

## 2.8 Set Airspeed

This I/O-Control sets the reader's current maximum RFID communication baud rate.

| IOCTL-Code | Input Buffer | Output Buffer |
|---|---|---|
| 2070 | sizeof(unsigned long) | - |

**Input values**

| Data | Comment |
|---|---|
| 0x106 or 0x00 | Set maximum baud rate to 106 kBit/s |
| 0x212 or 0x02 | Set maximum baud rate to 212 kBit/s |
| 0x424 or 0x04 | Set maximum baud rate to 424 kBit/s |
| 0x848 or 0x08 | Set maximum baud rate to 848 kBit/s |

## 2.9 Query Airspeed

This I/O-Control returns the reader's current maximum RFID communication baud rate.

| IOCTL-Code | Input Buffer | Output Buffer size |
|---|---|---|
| 2071 | - | sizeof(unsigned long) |

**Response values**

| Data | Comment |
|---|---|
| 0x106 | Maximum baud rate is 106 kBit/s |
| 0x212 | Maximum baud rate is 212 kBit/s |
| 0x424 | Maximum baud rate is 424 kBit/s |
| 0x848 | Maximum baud rate is 848 kBit/s |

## 2.10 Query AsyncBaudRates

| IOCTL-Code | Input Buffer | Output Buffer size |
|---|---|---|
| 2073 | - | 1 byte |

**Response values**

| Data | Comment |
|---|---|
| 0x00 | Reader supports asynchronous baud rates |
| 0x01 | Reader does not support asynchronous baud rates |

## 2.11 Set Timeout

This I/O-Control sets the reader's current contactless timeout value.

| IOCTL-Code | Input Buffer | Output Buffer |
|---|---|---|
| 2074 | sizeof(unsigned long) | - |

**Input values**

| Data | Comment |
|---|---|
| 0x00 | Automatic values are applied for each card |
| 1 - 19000 | Manual timeout in milliseconds |

## 2.12 Query Timeout

This I/O-Control returns the reader's current contactless timeout value in milliseconds.

| IOCTL-Code | Input Buffer | Output Buffer size |
|---|---|---|
| 2075 | - | sizeof(unsigned long) |

**Response value**

| Data | Comment |
|---|---|
| 0x00 | Automatic values are applied for each card |
| 1 - 19000 | Current timeout in milliseconds |

## 2.13 Request Read Settings

This I/O-Control advises the driver to reload and apply settings from registry.

| IOCTL-Code | Input Buffer | Output Buffer |
|---|---|---|
| 2076 | - | - |

## 2.14 Set ISO14443-4 Support

| IOCTL-Code | Input Buffer | Output Buffer |
|---|---|---|
| 2077 | 1 byte | - |

**Note:** This I/O-Control has a different behavior than the previous 2.0.X driver series!

**Input values**

| Data | Comment |
|---|---|
| 0x00 | Disable ISO14443-4 support |
| 0x01 | Enable ISO14443-4 support |

## 2.15 Query ISO14443-4 Support

| IOCTL-Code | Input Buffer | Output Buffer |
|---|---|---|
| 2078 | - | 1 byte |

**Note:** This I/O-Control has a different behavior than the previous 2.0.X driver series!

**Response values**

| Data | Comment |
|---|---|
| 0x00 | ISO14443-4 support is disabled |
| 0x01 | ISO14443-4 support is enabled |

## 2.16 Set Antenna

Fixates an antenna and/or advises reader to start/stop toggling.

| IOCTL-Code | Input Buffer | Output Buffer |
|---|---|---|
| 2080 | 1 byte | - |

**Input values**

| Data | Comment |
|---|---|
| 0x00 – 0xXX | Stop toggling and fixate to antenna with the given index. |
| null OR >= number of | Enable antenna togging again |

# 3 Virtual COM-Port Interface (VCOM)

55x3 PC/SC Driver also registers a virtual COM-Port interface that behaves like a native COM-Port – although with limited functionality.

The Name of this COM-Port consists of the usual COM prefix (e.g.: COM1, COM3)

with a driver specific postfix (e.g.: OK55x3_0, OK55x3_1).

So the resulting VCOM name is: COM_OK55x3_X, where X stands for the device- number.

You can connect to this COM-Port using the standard Win32 function calls like

CreateFile and CloseHandle or using a recent ReaderDLL.

Be aware that it is not necessary to detect the reader manually / to switch baud rates, the driver already did this prior to any VCOM connection.

You should have a stable binary connection at the highest baud rate by default.

**Note:** While a connection to the VCOM port is present, the PC/SC card polling is switched off because it would interfere with user communication.

# 4 Contactless EMVCo Support

The following functionality is only available for Multi ISO Reader Boards with firmware versions above 1.2.5. Additional features are implemented into the driver as I/O-Controls. The purpose of these features is to extend the functionality / interoperability and to ease development using a combination of 55x3 readers and this driver.

For an example of how to invoke I/O-Controls, see to Appendix – Code Example, page 18.

## 4.1 Summary

- Set RFID Operation Mode
- Query RFID Operation Mode – TLV encoded
- Query Airspeed

## 4.2 Set RFID Operation Mode

| IOCTL-Code | Input Buffer | Output Buffer size |
|---|---|---|
| 3107 | 1 byte | - |

**Input values**

| Data | Comment |
|---|---|
| 0x00 | Set RFID Operation Mode ISO |
| 0x11 | Set RFID Operation Paypass |

**Response values**

SCardControl function call fails with error code 0x1F when reader firmware doesn't support Contactless EmvCo functionality.

## 4.3 Query RFID Operation Mode

| IOCTL-Code | Input Buffer | Output Buffer size |
|---|---|---|
| 3200 | - | >=9 byte |

**Response**

| Data | Comment |
|---|---|
| Byte* | TLV encoded byte stream |

The response of this request is a TLV encoded byte stream:

**TLV** decoding structure:

**T**ype (1byte) **L**ength (1 byte) **V**alue(s)

**Defined TLV Tags:**

```
TLV_TAG_VERSION_PAYPASS_LIBRARY          0xB3 (hex)
TLV_TAG_READER_OPERATION_MODE            0xB6 (hex)
```

**Defined return values for 'Operation Mode Tag':**

```
OP_MODE_RFID_ISO                         0x10 (hex)
```

## 4.4 Query PayPass Library Version

| IOCTL-Code | Input Buffer | Output Buffer size |
|---|---|---|
| 3200 | - | >=9 byte |

**Response**

| Data | Comment |
|---|---|
| Byte* | TLV encoded byte stream |

The response of this request is a TLV encoded byte stream:

**TLV** decoding structure:

**T**ype (1byte) **L**ength (1 byte) **V**alue(s)

**Defined TLV Tags:**

```
TLV_TAG_VERSION_PAYPASS_LIBRARY          0xB3 (hex)
```

**Defined return values for PayPass Library Version:**

| Byte: | Definition: | Range: |
|---|---|---|
| 1: | PAYPASS_LIB_MAJOR_VERSION | 0 - 255 (dec) |
| 2: | PAYPASS_LIB_MINOR_VERSION | 0 - 255 (dec) |
| 3: | PAYPASS_LIB_BUILD_NUMBER | 0 - 255 (dec) |
| 4: | PAYPASS_LIB_PATCH_NUMBER | 0 – 255 (dec) |

# 5 Appendix – Code Example

The following tested code establishes a connection to windows smart card service, enumerates all readers, stores them into an array of defined structures, connects to the first reader found, gets/sets some data and disconnects from both, reader and smart card service.

**Include:** **winscard.h**

**Link:** **winscard.lib**

```
// Defines the driver-
structure struct
PCSC_Instance
{
    // Name of driver instance
    UCHAR Name[128];
    // The current connection-mode
    DWORD Mode;
    // Handle to the instance
    SCARDHANDLE Handle;
};
// Used to filter applicable smartcard devices
#define DRIVER_STRING    55x3

// Handle to the SmartCard service
SCARDCONTEXT PCSC_Context;

// List of found driver instances
PCSC_Instance     *PCSC_Instances;

// Number of driver instances in list
ULONG             PCSC_NumInstances;

// Temporary variable
DWORD dw_temp;

// Establish connection to the smartcard-service
LONG result = SCardEstablishContext(SCARD_SCOPE_USER, NULL, NULL,
                                    &PCSC_Context);

// Check for error
```

```
if (result != SCARD_S_SUCCESS) {
    return;
}

// Buffer that holds all smartcard reader names
UCHAR buffer[2048];
ULONG buffer_length = sizeof(buffer);

// Pointer to a single reader within the buffer
UCHAR* pReader;

// Enumerate all driver instances
result = SCardListReadersA(PCSC_Context, NULL, (LPSTR)buffer,
                           &buffer_length
                                 );
// Check for error
if (result != SCARD_S_SUCCESS) {
    SCardReleaseContext(PCSC_Context
    ); return;
}

// First of all check how much readers are connected
PCSC_NumInstances = 0;
pReader = buffer;
while (pReader && *pReader)
{
    // Check each smartcard reader's name
    if (_strnicmp((CHAR*)pReader,
                  DRIVER_STRING,
                  min(strlen((CHAR*)pReader),
                  strlen(DRIVER_STRING))) ==
                  0)
    {
        // Increment instance count if we found a reader
        PCSC_NumInstances++;
    }
    // Advance to next reader-string
    pReader +=
    strlen((CHAR*)pReader)+1;
}

// Exit if no readers have been
found if (!PCSC_NumInstances) {
```

```
    SCardReleaseContext(PCSC_Context);
    return;
}
// Now that we know how much readers we have, initialize the readers-array
PCSC_Instances = new PCSC_Instance[PCSC_NumInstances];

// Fill the detected readers into reader-
array pReader = buffer;
PCSC_NumInstances = 0;
while (pReader && *pReader)
{
    if (_strnicmp((CHAR*)pReader,
                  DRIVER_STRING,
                  min(strlen((CHAR*)pReader),
                  strlen(DRIVER_STRING))) ==
                  0)
    {
        // Store each reader's name
        memcpy(PCSC_Instances[PCSC_NumInstances].Name,
        pReader,
               strlen((CHAR*)pReader));

        // Terminate each name correctly
        PCSC_Instances[PCSC_NumInstances].Name[strlen(
                                              (CHAR*)pReader)] = '\0';
        // Increment instance count
        PCSC_NumInstances++;

    }

    pReader += strlen((CHAR*)pReader)+1;

}
// Connect to first driver instance
result = SCardConnectA(PCSC_Context,
                       (LPCSTR)PCSC_Instances[0].Name,
                       SCARD_SHARE_DIRECT, NULL,
                       &PCSC_Instances[0].Handle, &dw_temp);

// Exit if we can't connect to
reader if (result !=
SCARD_S_SUCCESS) {
    delete PCSC_Instances;
    SCardReleaseContext(PCSC_Context
```

```
    ); return;
}
// Get Airspeed of first driver instance
DWORD b_length;
result = SCardControl(PCSC_Instances[0].Handle, SCARD_CTL_CODE(2071),
NULL,
                      0, buffer, sizeof(buffer), &b_length);

if (result == SCARD_S_SUCCESS)
{
    switch(*((ULONG*)buffer))
    {
        case 0x106:
            printf(Airspeed = 106 kBit/s\n);
            break;
        case 0x212:
            printf(Airspeed = 212 kBit/s\n);
            break;
        case 0x424:
            printf(Airspeed = 424 kBit/s\n);
            break;
        case 0x848:
            printf(Airspeed = 848 kBit/s\n);
            break;
        default:
            printf(Airspeed Unknown\n);
            break;
    }
}
else {
    printf(Error retrieving airspeed!\n);
}

// Set card timeout of first driver instance
ULONG tout = 5000;
memcpy(buffer, &tout, sizeof(ULONG));
result = SCardControl(PCSC_Instances[0].Handle, SCARD_CTL_CODE(2074),
                      buffer, sizeof(ULONG), NULL, NULL, &b_length);
```

```
if (result == SCARD_S_SUCCESS) {
    printf(timeout set!\n);
}
else {
    printf(timeout setting failed!\n);
}
// Close connection to reader
SCardDisconnect(PCSC_Instances[0].Handle, SCARD_LEAVE_CARD);

delete PCSC_Instances;

// Close connection to smartcard-service
SCardReleaseContext(PCSC_Context);
```