# MUSCLE PC/SC IFD Driver API

David Corcoran & Ludovic Rousseau
corcoran@musclecard.com, ludovic.rousseau@free.fr

June 11, 2004

**Abstract**

This toolkit and documentation is provided on an as is basis. The authors shall not be held responsible for any mishaps caused by the use of this software.
For more information please visit http://www.musclecard.com/.

Document history:

| | | |
|---|---|---|
| 3.0.1 | August 9, 2004 | latest PDF only version |
| 3.1.0 | June 11, 2004 | reformat using LaTeX and correct bugs |

# Contents

# 1 Introduction/Overview

This document describes the API calls required to make a PC/SC driver for a device to be supported under the MUSCLE PC/SC resource manager. By implementing these calls correctly in a driver or shared object form, reader manufacturers can fit their hardware into an already existing infrastructure under several operating systems and hardware platforms. This IFD Handler interface is not restricted to smart cards and readers and could also be used for other types of smart card like devices. I would really like to hear from you. If you have any feedback either on this documentation or on the MUSCLE project please feel free to email me at: `corcoran@musclecard.com`.

# 2 Definitions

## 2.1 Defined types

The following is a list of commonly used type definitions in the following API. These definitions and more can be found in the `ifdhandler.h` file.

| PC/SC type | C type |
|---|---|
| DWORD | unsigned long |
| LPSTR | char * |
| PDWORD | unsigned long * |
| PUCHAR | unsigned char * |
| RESPONSECODE | long |
| VOID | void |

## 2.2 Error codes

The following is a list of returned values:

| IFD_SUCCESS |
|---|
| IFD_COMMUNICATION_ERROR |
| IFD_ERROR_CONFISCATE |
| IFD_ERROR_EJECT |
| IFD_ERROR_NOT_SUPPORTED |
| IFD_ERROR_POWER_ACTION |
| IFD_ERROR_PTS_FAILURE |
| IFD_ERROR_SET_FAILURE |
| IFD_ERROR_SWALLOW |
| IFD_ERROR_TAG |
| IFD_ERROR_VALUE_READ_ONLY |
| IFD_ICC_NOT_PRESENT |
| IFD_ICC_PRESENT |

```
IFD_NOT_SUPPORTED
IFD_PROTOCOL_NOT_SUPPORTED
IFD_RESPONSE_TIMEOUT
```

# 3 Readers configuration

## 3.1 USB readers

USB readers use the bundle approach so that the reader can be loaded and unloaded upon automatic detection of the device. The bundle approach is simple: the actual library is just embedded in a directory so additional information can be gathered about the device.

A bundle looks like the following:

```
GenericReader.bundle/
  Contents/
    Info.plist  - XML file describing the reader
    MacOS/      - Driver directory for OS X
    Solaris/    - Driver directory for Solaris
    Linux/      - Driver directory for Linux
    HPUX/       - Driver directory for HPUX
```

The `Info.plist` file describes the driver and gives the loader all the necessary information. The following must be contained in the `Info.plist` file:

- `ifdVendorID`

  The vendor ID of the USB device.

  Example:

  ```
  <key>ifdVendorID</key>
  <string>0x04E6</string>
  ```

  You may have an OEM of this reader in which an additional `<string>` can be used like in the below example:

  ```
  <key>ifdVendorID</key>
  <array>
    <string>0x04E6</string>
    <string>0x0973</string>
  </array>
  ```

If multiples exist all the other parameters must have a second value also. You may chose not to support this feature but it is useful when reader vendors OEM products so you only distribute one driver.

The CCID driver from Ludovic Rousseau[1] uses this feature since the same driver supports many different readers.

- `ifdProductID`

  The product id of the USB device.

  ```
  <key>ifdProductID</key>
  <string>0x3437</string>
  ```

- `ifdFriendlyName`

  Example:

  ```
  <key>ifdFriendlyName</key>
  <string>SCM Microsystems USB Reader</string>
  ```

- `CFBundleExecutable`

  The executable name which exists in the particular platform's directory.

  Example:

  ```
  <key>CFBundleExecutable</key>
  <string>libccid.so.0.4.2</string>
  ```

## 3.2 Serial readers

Serial drivers must be configured to operate on a particular port and respond to a particular name. The `reader.conf` file is used for this purpose.

It has the following syntax:

```
# Configuration file for pcsc-lite
# David Corcoran <corcoran@musclecard.com>

FRIENDLYNAME  Generic Reader
DEVICENAME    /dev/ttyS0
LIBPATH       /usr/lib/pcsc/drivers/libgen_ifd.so
CHANNELID     1
```

- The pound sign `#` denotes a comment.

[1] `http://pcsclite.alioth.debian.org/ccid.html`

- The `FRIENDLYNAME` field is an arbitrary text used to identify the reader. This text is displayed by commands like `pcsc_scan`[2] that prints the names of all the connected and detected readers.

- The `DEVICENAME` field was not used for old drivers (using the IFD handler version 2.0 or previous). It is now (IFD handler version 3.0) used to identify the physical port on which the reader is connected. This is the device name of this port. It is dependent of the OS kernel. For example the first serial port device is called `/dev/ttyS0` under Linux and `/dev/cuaa0` under FreeBSD.

- The `LIBPATH` field is the filename of the driver code. The driver is a dynamically loaded piece of code (generally a `drivername.so*` file).

- The `CHANNELID` is no more used for recent drivers (IFD handler 3.0) and has been superseded by `DEVICENAME`. If you have an old driver this field is used to indicate the port to use. You should read your driver documentation to know what information is needed here. It should be the serial port number for a serial reader.

  `CHANNELID` was the numeric version of the port in which the reader will be located. This may be done by a symbolic link where `/dev/pcsc/1` is the first device which may be a symbolic link to `/dev/ttyS0` or whichever location your reader resides.

# 4 API Routines

The routines specified hereafter will allow you to write an IFD handler for the PC/SC Lite resource manager. Please use the complement developer's kit complete with headers and Makefile at: `http://www.musclecard.com/drivers.html`.

This gives a common API for communication to most readers in a homogeneous fashion. This document assumes that the driver developer is experienced with standards such as ISO-7816-(1, 2, 3, 4), EMV and MCT specifications. For listings of these specifications please access the above web site.

## 4.1 IFDHCreateChannel

**Synopsis:**

```
#include <PCSC/ifdhandler.h>

RESPONSECODE IFDHCreateChannel(DWORD Lun,
    DWORD Channel);
```

[2] `http://ludovic.rousseau.free.fr/softwares/pcsc-tools/`

**Parameters:**

| | | |
|---|---|---|
| `Lun` | IN | Logical Unit Number |
| `Channel` | IN | Channel ID |

**Description:**

This function is required to open a communications channel to the port listed by `Channel`. For example, the first serial reader on COM1 would link to `/dev/pcsc/1` which would be a symbolic link to `/dev/ttyS0` on some machines This is used to help with inter-machine independence.

On machines with no `/dev` directory the driver writer may choose to map their `Channel` to whatever they feel is appropriate.

Once the channel is opened the reader must be in a state in which it is possible to query `IFDHICCPresence()` for card status.

- `Lun` - Logical Unit Number

  Use this for multiple card slots or multiple readers. `0xXXXXYYYY` - `XXXX` multiple readers, `YYYY` multiple slots. The resource manager will set these automatically. By default the resource manager loads a new instance of the driver so if your reader does not have more than one smart card slot then ignore the Lun in all the functions.

  PC/SC supports the loading of multiple readers through one instance of the driver in which `XXXX` is important. `XXXX` identifies the unique reader in which the driver communicates to. The driver should set up an array of structures that associate this `XXXX` with the underlying details of the particular reader.

- `Channel` - Channel ID

  This is denoted by the following:

  ```
  0x000001  /dev/pcsc/1
  0x000002  /dev/pcsc/2
  0x000003  /dev/pcsc/3
  0x000004  /dev/pcsc/4
  ```

  USB readers can ignore the `Channel` parameter and query the USB bus for the particular reader by manufacturer and product id.

**Returns:**

| | |
|---|---|
| `IFD_SUCCESS` | Successful |
| `IFD_COMMUNICATION_ERROR` | Error has occurred |

## 4.2 IFDHCreateChannelByName

**Synopsis:**

```
#include <PCSC/ifdhandler.h>

RESPONSECODE IFDHCreateChannelByName(DWORD Lun,
    LPSTR deviceName);
```

**Parameters:**

| | | |
|---|---|---|
| `Lun` | IN | Logical Unit Number |
| `DeviceName` | IN | String device path |

**Description:**

This function is required to open a communications channel to the port listed by `DeviceName`.

Once the channel is opened the reader must be in a state in which it is possible to query `IFDHICCPresence()` for card status.

- `Lun` - Logical Unit Number

  Use this for multiple card slots or multiple readers. `0xXXXXYYYY` - `XXXX` multiple readers, `YYYY` multiple slots. The resource manager will set these automatically. By default the resource manager loads a new instance of the driver so if your reader does not have more than one smart card slot then ignore the Lun in all the functions.

  PC/SC supports the loading of multiple readers through one instance of the driver in which `XXXX` is important. `XXXX` identifies the unique reader in which the driver communicates to. The driver should set up an array of structures that associate this `XXXX` with the underlying details of the particular reader.

- `DeviceName` - filename to use by the driver.

  For drivers configured by `/etc/reader.conf` this is the value of the field `DEVICENAME`.

  For USB drivers under platforms using `libusb`[3] for USB abstraction (Any Unix except MacOSX) the `DeviceName` field uses the string generated by:

  ```
  printf("usb:%04x/%04x:libusb:%s:%s",
      idVendor, idProduct,
      bus->dirname, dev->filename)
  ```

  So it is something like: `usb:08e6/3437:libusb:001:042` under Linux.

  It is the responsability of the driver to correctly identify the reader. This scheme was put in place to be able to distinguish two identical readers connected at the same time.

[3] `http://libusb.sourceforge.net/`

**Returns:**

| | |
|---|---|
| `IFD_SUCCESS` | Successful |
| `IFD_COMMUNICATION_ERROR` | Error has occurred |

## 4.3 IFDHCloseChannel

**Synopsis:**

```
#include <PCSC/ifdhandler.h>

RESPONSECODE IFDHCloseChannel(DWORD Lun);
```

**Parameters:**

| | | |
|---|---|---|
| `Lun` | IN | Logical Unit Number |

**Description:**

This function should close the reader communication channel for the particular reader. Prior to closing the communication channel the reader should make sure the card is powered down and the terminal is also powered down.

**Returns:**

| | |
|---|---|
| `IFD_SUCCESS` | Successful |
| `IFD_COMMUNICATION_ERROR` | Error has occurred |

## 4.4 IFDHGetCapabilities

**Synopsis:**

```
#include <PCSC/ifdhandler.h>

RESPONSECODE IFDHGetCapabilities(DWORD Lun,
    DWORD Tag,
    PDWORD Length,
    PUCHAR Value);
```

**Parameters:**

| | | |
|---|---|---|
| `Lun` | IN | Logical Unit Number |
| `Tag` | IN | Tag of the desired data value |
| `Length` | INOUT | Length of the desired data value |
| `Value` | OUT | Value of the desired data |

**Description:**

This function should get the slot/card capabilities for a particular slot/card specified by `Lun`. Again, if you have only 1 card slot and don't mind loading a new driver for each reader then ignore `Lun`.

- `Tag` - the tag for the information requested
  - `TAG_IFD_ATR`
    Return the ATR and it's size (implementation is mandatory).
  - `SCARD_ATTR_ATR_STRING`
    Same as `TAG_IFD_ATR` but this one is not mandatory. It is defined in Microsoft PC/SC `SCardGetAttrib()`.
  - `TAG_IFD_SIMULTANEOUS_ACCESS`
    Return the number of sessions the driver can handle.
    This is used for multiple readers sharing the same driver.
  - `TAG_IFD_SLOTS_NUMBER`
    Return the number of slots in this reader.
  - `IOCTL_SMARTCARD_VENDOR_VERIFY_PIN`
    Return a non null value if the driver/reader is able to perform a `SCardControl(hCard, IOCTL_SMARTCARD_VENDOR_VERIFY_PIN, ...)`. See pcsc-lite documentation [1].
- `Length` - the length of the returned data
- `Value` - the value of the data

This function is also called when the application uses the PC/SC `SCardGetAttrib()` function. The list of supported tags is not limited. The ones above are used by the PC/SC lite resource manager.

**Returns:**

| | |
|---|---|
| `IFD_SUCCESS` | Successful |
| `IFD_ERROR_TAG` | Invalid tag given |

## 4.5 IFDHSetCapabilities

**Synopsis:**

```
#include <PCSC/ifdhandler.h>

RESPONSECODE IFDHSetCapabilities(DWORD Lun,
    DWORD Tag,
    DWORD Length,
    PUCHAR Value);
```

**Parameters:**

| | | |
|---|---|---|
| `Lun` | IN | Logical Unit Number |
| `Tag` | IN | Tag of the desired data value |
| `Length` | INOUT | Length of the desired data value |
| `Value` | OUT | Value of the desired data |

**Description:**

This function should set the slot/card capabilities for a particular slot/card specified by Lun. Again, if you have only 1 card slot and don't mind loading a new driver for each reader then ignore Lun.

- `Tag` - the tag for the information needing set
- `Length` - the length of the data
- `Value` - the value of the data

This function is also called when the application uses the PC/SC `SCardGetAttrib()` function. The list of supported tags is not limited.

**Returns:**

| | |
|---|---|
| `IFD_SUCCESS` | Success |
| `IFD_ERROR_TAG` | Invalid tag given |
| `IFD_ERROR_SET_FAILURE` | Could not set value |
| `IFD_ERROR_VALUE_READ_ONLY` | Trying to set read only value |

## 4.6 IFDHSetProtocolParameters

**Synopsis:**

```
#include <PCSC/ifdhandler.h>

RESPONSECODE IFDHSetProtocolParameters(DWORD Lun,
    DWORD Protocol,
    UCHAR Flags,
    UCHAR PTS1,
    UCHAR PTS2,
    UCHAR PTS3);
```

**Parameters:**

| | | |
|---|---|---|
| `Lun` | IN | Logical Unit Number |
| `Protocol` | IN | Desired protocol |
| `Flags` | IN | OR'd Flags (See below) |
| `PTS1` | IN | 1st PTS Value |
| `PTS2` | IN | 2nd PTS Value |
| `PTS3` | IN | 3rd PTS Value |

**Description:**

This function should set the Protocol Type Selection (PTS) of a particular card/slot using the three PTS parameters sent

- `Protocol` - 0...14

  T=0 to T=14 protocol

- `Flags` - Logical OR of possible values to determine which PTS values to negotiate
  - `IFD_NEGOTIATE_PTS1`
  - `IFD_NEGOTIATE_PTS2`
  - `IFD_NEGOTIATE_PTS3`
- `PTS1`, `PTS2`, `PTS3` - PTS Values

  See ISO 7816/EMV documentation.

**Returns:**

| | |
|---|---|
| `IFD_SUCCESS` | Success |
| `IFD_ERROR_PTS_FAILURE` | Could not set PTS value |
| `IFD_COMMUNICATION_ERROR` | Error has occurred |
| `IFD_PROTOCOL_NOT_SUPPORTED` | Protocol is not supported |

## 4.7 IFDHPowerICC

**Synopsis:**

```
#include <PCSC/ifdhandler.h>

RESPONSECODE IFDHPowerICC(DWORD Lun,
    DWORD Action,
    PUCHAR Atr,
    PDWORD AtrLength);
```

**Parameters:**

| | | |
|---|---|---|
| `Lun` | IN | Logical Unit Number |
| `Action` | IN | Action to be taken |
| `Atr` | OUT | Answer to Reset (ATR) value of the inserted card |
| `AtrLength` | INOUT | Length of the ATR |

**Description:**

This function controls the power and reset signals of the smart card reader at the particular reader/slot specified by Lun.

- `Action` - Action to be taken on the card
  - `IFD_POWER_UP`
    Power and reset the card if not done so (store the ATR and return it and it's length)
  - `IFD_POWER_DOWN`
    Power down the card then power up if not done already (`Atr` and `AtrLength` should be zeroed)
  - `IFD_RESET`
    Perform a quick reset on the card. If the card is not powered power up the card. (Store and return `Atr` and `Length`)
- `Atr` - Answer to Reset of the card

  The driver is responsible for caching this value in case `IFDHGetCapabilities()` is called requesting the ATR and it's length. The ATR length should not exceed `MAX_ATR_SIZE`.
- `AtrLength` - Length of the Atr

  This should not exceed `MAX_ATR_SIZE`.

**Notes:**

Memory cards without an ATR should return `IFD_SUCCESS` on reset but the `Atr` should be zeroed and the length should be zero Reset errors should return zero for the `AtrLength` and return `IFD_ERROR_POWER_ACTION`.

**Returns:**

| | |
|---|---|
| `IFD_SUCCESS` | Success |
| `IFD_ERROR_POWER_ACTION` | Error powering/resetting card |
| `IFD_COMMUNICATION_ERROR` | An error has occurred |
| `IFD_NOT_SUPPORTED` | Action not supported |

## 4.8 IFDHTransmitToICC

**Synopsis:**

```
#include <PCSC/ifdhandler.h>

RESPONSECODE IFDHTransmitToICC(DWORD Lun,
    SCARD_IO_HEADER SendPci,
    PUCHAR TxBuffer,
    DWORD TxLength,
    PUCHAR RxBuffer,
    PDWORD RxLength,
    PSCARD_IO_HEADER RecvPci);
```

**Parameters:**

| | | |
|---|---|---|
| `Lun` | IN | Logical Unit Number |
| `SendPci` | IN | Protocol structure |
| `TxBuffer` | IN | APDU to be sent |
| `TxLength` | IN | Length of sent APDU |
| `RxBuffer` | OUT | APDU response |
| `RxLength` | INOUT | Length of APDU response |
| `RecvPci` | INOUT | Receive protocol structure |

**Description:**

This function performs an APDU exchange with the card/slot specified by `Lun`. The driver is responsible for performing any protocol specific exchanges such as T=0, 1, etc. differences. Calling this function will abstract all protocol differences.

- `SendPci` - contains two structure members

  - `Protocol` - 0, 1, ... 14
    T=0 ... T=14
  - `Length` - Not used.

- `TxBuffer` - Transmit APDU

  example: `"\x00\xA4\x00\x00\x02\x3F\x00"`

- `TxLength` - Length of this buffer

- `RxBuffer` - Receive APDU

  example: `"\x61\x14"`

- `RxLength` - Length of the received APDU

  This function will be passed the size of the buffer of `RxBuffer` and this function is responsible for setting this to the length of the received APDU response. This should be ZERO on all errors. The resource manager will take responsibility of zeroing out any temporary APDU buffers for security reasons.

- `RecvPci` - contains two structure members

  - `Protocol` - 0, 1, ... 14
    T=0 ... T=14
  - `Length` - Not used.

**Notes:**

The driver is responsible for knowing what type of card it has. If the current slot/card contains a memory card then this command should ignore the `Protocol` and use the MCT style commands for support for these style cards and transmit them appropriately. If your reader does not support memory cards or you don't want to implement this functionality, then ignore this.

`RxLength` should be set to zero on error.

The driver is *not* responsible for doing an automatic Get Response command for received buffers containing `61 XX`.

**Returns:**

| | |
|---|---|
| `IFD_SUCCESS` | Success |
| `IFD_COMMUNICATION_ERROR` | An error has occurred |
| `IFD_RESPONSE_TIMEOUT` | The response timed out |
| `IFD_ICC_NOT_PRESENT` | ICC is not present |
| `IFD_PROTOCOL_NOT_SUPPORTED` | Protocol is not supported |

## 4.9 IFDHControl

**Synopsis:**

```
#include <PCSC/ifdhandler.h>

RESPONSECODE IFDHControl(DWORD Lun,
    DWORD dwControlCode,
    PUCHAR TxBuffer,
    DWORD TxLength,
    PUCHAR RxBuffer,
    DWORD RxLength,
    PDWORD pdwBytesReturned);
```

**Parameters:**

| | | |
|---|---|---|
| `Lun` | IN | Logical Unit Number |
| `dwControlCode` | IN | Control code for the operation |
| `TxBuffer` | IN | Bytes to be sent |
| `TxLength` | IN | Length of sent bytes |
| `RxBuffer` | OUT | Response |
| `RxLength` | IN | Length of response buffer |
| `pdwBytesReturned` | OUT | Length of response |

**Description:**

This function performs a data exchange with the reader (not the card) specified by `Lun`. It is responsible for abstracting functionality such as PIN pads, biometrics, LCD panels, etc. You should follow the MCT and CTBCS specifications for a list of accepted commands to implement. This function is fully voluntary and does not have to be implemented unless you want extended functionality.

- `dwControlCode` - Control code for the operation

  This value identifies the specific operation to be performed. This value is driver specific.

- `TxBuffer` - Transmit data
- `TxLength` - Length of this buffer
- `RxBuffer` - Receive data
- `RxLength` - Length of the response buffer
- `pdwBytesReturned` - Length of response

This function will be passed the length of the buffer `RxBuffer` in `RxLength` and it must set the length of the received data in `pdwBytesReturned`.

**Notes:**

`*pdwBytesReturned` should be set to zero on error.

**Returns:**

| | |
|---|---|
| `IFD_SUCCESS` | Success |
| `IFD_COMMUNICATION_ERROR` | An error has occurred |
| `IFD_RESPONSE_TIMEOUT` | The response timed out |

## 4.10 IFDHICCPresence

**Synopsis:**

```
#include <PCSC/ifdhandler.h>

RESPONSECODE IFDHICCPresence(DWORD Lun);
```

**Parameters:**

| | | |
|---|---|---|
| `Lun` | IN | Logical Unit Number |

**Description:**

This function returns the status of the card inserted in the reader/slot specified by `Lun`. In cases where the device supports asynchronous card insertion/removal detection, it is advised that the driver manages this through a thread so the driver does not have to send and receive a command each time this function is called.

**Returns:**

| | |
|---|---|
| `IFD_ICC_PRESENT` | ICC is present |
| `IFD_ICC_NOT_PRESENT` | ICC is not present |
| `IFD_COMMUNICATION_ERROR` | An error has occurred |

# 5 API provided by pcsc-lite

pcsc-lite also provides some API to ease the development of the driver.

## 5.1 LTPBundleFindValueWithKey

**Synopsis:**

```
#include <PCSC/parser.h>

int LTPBundleFindValueWithKey(char *fileName,
    char *tokenKey,
    char *tokenValue,
    int tokenIndice);
```

**Parameters:**

| | | |
|---|---|---|
| `fileName` | IN | configuration file name |
| `tokenKey` | IN | searched token string |
| `tokenValue` | OUT | found token value |
| `tokenIndice` | IN | indice of the desired token in an array |

**Description:**

This function is used to parse the `Info.plist` configuration file. The `tokenIndice` field is used to get the n-th element of an array.

**Returns:**

| | |
|---|---|
| `-1` | Error |
| `0` | No error |
| `1` | Configuration file not found |

## 5.2 debug_msg

**Synopsis:**

```
#include <PCSC/debuglog.h>

void debug_msg(const char *fmt,
    ...);
```

**Parameters:**

| | | |
|---|---|---|
| `fmt` | IN | format string as in `printf()` |
| `...` | IN | optionnal parameters as in `printf()` |

**Description:**

This function is used by the driver to send debug information. The advantage of using the same debug function as pcsc-lite is that you also benefit from the debug redirection provided by pcsc-lite. For example you can use `pcscd -foreground` to start `pcscd` and keep it in the forground with the debug sent to stderr. You will then get `pcscd` and the driver' debug messages in the same place.

**Example:**

```
#define DebugLogB(fmt, data) debug_msg("%s:%d:%s " fmt, \
    __FILE__, __LINE__, __FUNCTION__, data)

DebugLogB("received bytes: %d", r);
```

## 5.3 debug_xxd

**Synopsis:**

```
#include <PCSC/debuglog.h>

void debug_xxd(const char *msg,
    const unsigned char *buffer,
    const int size);
```

**Parameters:**

| | | |
|---|---|---|
| `msg` | IN | text string |
| `buffer` | IN | buffer you want to dump in hex |
| `size` | IN | size of the buffer |

**Description:**

Same idea as `debug_msg()` put print the hex dump of a buffer.

**Example:**

```
debug_xxd("received frame: ", buff, buff_size);
```

# 6 API changes

The IFD handler API changed over the time.

If the driver provides a `IFDHCreateChannelByName()` function is supposed to use API v3.0. Otherwise it is used with API v2.0.

## 6.1 API version 2.0

- `DEVICENAME` in `reader.conf` is not used.
- `IFDHControl()` API was:

```
RESPONSECODE IFDHControl(DWORD Lun,
    PUCHAR TxBuffer,
    DWORD TxLength,
    PUCHAR RxBuffer,
    PDWORD RxLength);
```

## 6.2 API version 3.0

- introduction of `IFDHCreateChannelByName()`.

  For serial drivers, `CHANNELID` is no more used and `DEVICENAME` is used instead.

  For USB drivers the device name if `usb:%04x/%04x:libusb:%s:%s`. See 4.2.

- `IFDHControl()` API changed

  See 4.9.

# References

[1] David Corcoran and Ludovic Rousseau. MUSCLE PC/SC Lite API, Toolkit API Reference Documentation, 2004. `http://pcsclite.alioth.debian.org/pcsc-lite/`.