PMDE050-BA
26/05/2009

K531 & K632 SDK

Getting started guide

Headquarters, Europa

SpringCard
13 voie la Cardon
Parc Gutenberg
91120 Palaiseau
FRANCE

Phone : +33 (0) 164 53 20 10
Fax : +33 (0) 164 53 20 18

Americas

SpringCard
964 Fifth Avenue
Suite 235
San Diego, CA 92101
USA

Phone : +1 (619) 544 1450
Fax : +1 (619) 573 6867

www.springcard.com

DOCUMENT INFORMATION

Category :	Developer's manual
Group :	K531 Family SDK
Reference :	PMDE050
Version :	BA
Status :	CHECKED

Keywords :
K531 - K632

Abstract :

pmde050-ba.doc
saved 26/05/09 - printed 26/05/09

REVISION HISTORY

Ver.	Date	Author	Valid. by Tech.	Valid. by Qual.	Approv. by	Remarks :
BA	26/05/09	LTX	JDA	ECL	JDA	SpringCard branding.

TABLE OF CONTENT

1. INTRODUCTION

Thanks for using SpringCard K531 & K632 SDK.

This "getting started guide" contains important information to start developing with SpringCard K531 & K632 module quickly, and efficiently.

Please take a few minutes to read this document carefully.

1.1. AUDIENCE

This reference manual is designed for use by system integrators or application developers. He assumes that the reader has expert knowledge of electronics and computer development, especially in C language.

1.2. ABOUT THE K531 & K632 SDK

1.2.1. Content

The K531 & K632 Developer's Kit "SDK" contains sample software and documentation to work with the K531 or the K632 OEM module.

1.2.2. Installation

The SDK is available only as a self-decompressing package for Windows (password protected)[1]. Download the SDK file, and uncompress it to a folder on your hard drive.

The folder where you've uncompressed the SDK will be referred as %SDK_ROOT%.

1.2.3. Exploring the tree

%SDK_ROOT%	
+ docs	General documentation
+ direct	***K531 & K632 development without the SpringProx API***
+ docs	Documentation for this part
+ samples	Sample projects (with source code)
+ runimage	
+ win32_i386	Binaries for Windows
+ linux_i386	Binaries for Linux
+ springprox_api	***K531 & K632 development using the SpringProx API***
+ docs	Documentation for this part
+ source	Source code of the API
+ samples	Sample projects using the API (with source code)
+ runimage	
+ win32_i386	Binaries for Windows
+ linux_i386	Binaries for Linux

[1] Other OS : please contact us if you really do need a ZIP version of the SDK.

```
+ usb_driver
  + win32_i386          Windows driver for the USB interface
  + linux_i386          Linux library to access the USB interface
```

1.2.4. Prerequisites

There is no general prerequisite to use the K531 & K632 SDK, but understanding the examples may be easier if you use the same tools as we did :

- Windows : sample projects come both with configurations for Visual C++ 6 (Visual Studio 98) and Visual C++ 8 (Visual C++ 2005 Express edition)
- Linux : sample projects and USB library come with a Makefile for GNU Make + GCC C compiler.

1.3. SUPPORT AND UPDATES

Interesting related materials (datasheet, application notes, sample software…) are available at SpringCard's web site : www.springcard.com .

Updated versions of this document and others will be posted on this web site as soon as they are made available.

For technical support enquiries, please refer to SpringCard support page, on the web at address www.springcard.com/support .

1.4. COPYRIGHT NOTICE FOR THE CALYPSO SAMPLES

This SDK contains sample code to access Calypso cards, either using the specific Innovatron radio protocol, or the standard ISO 14443-B protocol. Working with Calypso cards is subject to a specific license fee. Therefore, only readers having the Calypso compliance feature enabled (K531-C and alike) are suitable for to work with those cards.

2. First sight on K531 or K632 from a Windows PC

Before writing code to work with the K531 or the K632, it is interesting to have a first contact with the device.

In this chapter we focus on the basic actions that can be performed from a regular computer, even "manually" through a terminal connection.

2.1. Installing and configuring FTDI USB driver

If you intend to work only with serial products, you don't need to install the FTDI USB driver.

Do not plug the USB interface (nor any SpringCard USB device such as CSB4U) until the driver has been installed.

- Connect with administrator privileges on your computer.
- Go to `%SDK_ROOT%/usb_driver/win32_i386`.
- Launch `SDD100.exe`.
- Follow the installation process.

FTDI USB Driver for SpringCard Devices is not WHQL certified, and therefore not signed. Your computer may issue security warning, or even deny driver installation.

Accept driver installation or decrease security level to install our driver correctly.

Once driver successfully installed,

- Plug the USB interface onto a computer USB connector.
- Wait until device has been recognised and activated (you may have to locate the driver installation folder manually here).
- Go to Control Panel / "System" icon / "Hardware" tab.
- Click "Device manager".
- Browse the tree, and locate the new (virtual) communication port that has been assigned to the USB interface. Note the port number assigned here (must be something between COM2 and COM99 …).

If your device have been assigned a port number above 9 (COM10 to COM99), it is recommended to change this number, as some software fail to work with those numbers.

- Double click the (virtual) communication port.
- Go to "Port Settings" tab.
- Click "Advanced".
- In the "COM port number", select a port between COM2 and COM9.

2.2. CONNECTING WITH HYPER TERMINAL

Connect your device to the computer.

If working with K531-TTL or K632-TTL plus TTL-USB interface, don't forget to connect the antenna to the interface now.

- Launch HyperTerminal.
- Create a new connection to the communication port your device is connected to (USB : this is the virtual communication port as chosen above).
- Define the communication parameters as follow :
  - Baudrate = 38400 bps
  - 8 data bits, 1 stop bit, no parity
  - No flow control.
- Press the Enter key. Module replies by its prompt ">".
- Type "`info`" followed by Enter. Module replies by a few lines of information.

Congratulation, you're now connected to OEM reader !

2.3. INVOKING K531 & K632 COMMANDS USING ASCII PROTOCOL

The K531 and K632 modules are "slave" devices. That means that without command from the host, they won't do anything on their own. Commands are sent to K531 or K632 using one of the three available protocols : OSI3964, "fast" binary, and ASCII.

2.3.1. First try with HyperTerminal

Let's start with ASCII as this is the only one that can be easily "simulated" by user input in HyperTerminal.

- Remove any contactless card from the antenna.
- Type "`$4000`" followed by Enter. Module replies by "`+0100`".
- Now put a Mifare Standard 1k contactless card on the antenna.
- Type "`$4000`" again, followed by Enter. Module replies by "`+0007…`".

A few explanations :

40 is the op-code for K531 & K632 "select any" command. When you enter "$4000", you tell the module you want to use the ASCII protocol ("$" marker) to execute command #40, without parameters ("00" is the length of the parameter block, so in our case there's no data).

Module acknowledges our command ("+" marker), then tries to select a contactless card in the RF field.

If there's one, module replies with "00" (execution successful) followed by "07" to introduce 7 bytes of data. The 7 bytes are :

- Card's serial number (UID) on 4 bytes,
- Card's answer to query (ATQ) on 2 bytes,
- Card's answer to query (SAK) on 1 byte.

If there's no card in the field, module replies with "01" (no card, or no reply from card) followed by "00" as there's no data.

You may now terminate your HyperTerminal session, and close the program.

2.3.2. A real world demo

- Launch a command prompt (Start Menu → Run → "cmd.exe")
- Go to `%SDK_ROOT%/direct/runimage/win32_i386`.
- Enter the command `ref_direct.exe –asc –poll COM2` .
  (if your device is connected to another port, replace `COM2` by actual port number)

Put various cards in front of the antenna, and see that the module sees them all.

Terminate the program using Ctrl+C keys

2.3.3. A first sight on source code

Complete source code is provided under `%SDK_ROOT%/direct/samples`.

Note that one there's only "main" file common to every projects (`ref_direct.c`).

The demo we've just seen ("active polling") is implemented in `polling.c` .

Communication with the module using ASCII protocol is implemented in `ascii.c`.

OS or hardware dependant part is implemented in `hal_win32.c` (or `hal_linux.c` for Linux).

2.3.4. Porting the project(s) to another OS or hardware

Using the `hal_stub.c` file as skeleton, the project can be easily ported to another OS or even to a micro-controller without OS.

2.4. Invoking K531 & K632 commands using Binary protocol

"Fast" binary protocol is faster than ASCII, can be implemented more efficiently, but can't be simulated by user !

2.4.1. A real world demo

Let's try the same demo :

- Launch a command prompt (Start Menu → Run → "cmd.exe")
- Go to `%SDK_ROOT%/direct/runimage/win32_i386`.
- Enter the command `ref_direct.exe -bin -poll COM2` .
  (if your device is connected to another port, replace `COM2` by actual port number)

Put various cards in front of the antenna, and see that the module sees them all.

Terminate the program using Ctrl+C keys

2.4.2. A first sight on source code

Complete source code is provided under `%SDK_ROOT%/direct/samples`.

Note that one there's only "main" file common to every projects (`ref_direct.c`).

The demo we've just seen ("active polling") is implemented in `polling.c` .

Communication with the module using "fast" binary protocol is implemented in `binary.c` .

OS or hardware dependant part is implemented in `hal_win32.c` (or `hal_linux.c` for Linux).

2.4.3. Porting the project(s) to another OS or hardware

Using the `hal_stub.c` file as skeleton, the project can be easily ported to another OS or even to a micro-controller without OS.

2.5. Using SpringProx API to work with K531 or K632

2.5.1. A short introduction

As seen before, it isn't difficult to send command to the K531 or the K632 and receive its replies.

Anyway, advanced operation with contactless cards may involve more than one command with numerous parameters. Frames returned by the card itself must be properly interpreted before continuing the sequence.

SpringProx API is a software library that implements most of those complex operations, and exposes them as easy-to-use and well-documented high-level functions. SpringProx API eases developer's job by hiding most of the complexity of the smartcard itself, allowing focusing on the application core only.

On Windows, SpringProx API is provided as compiled dynamic-load library "`springprox.dll`".

2.5.2. A real world demo

Let's try the same demo :

- Launch a command prompt (Start Menu → Run → "cmd.exe")
- Go to `%SDK_ROOT%/springprox_api/runimage/win32_i386`.
- Enter the command `ref_showuid.exe COM2` .
  (if your device is connected to another port, replace `COM2` by actual port number)

Put various cards in front of the antenna, and see that the module sees them all.

Terminate the program using Ctrl+C keys

2.5.3. A first sight on source code

Complete source code is provided under `%SDK_ROOT%/springprox_api/samples`.

The demo we've just seen ("active polling") is implemented in `ref_polling.c` .

Note that only two functions are used (one to lookup for ISO/IEC 14443-A cards, the other for ISO/IEC 14443-B).

2.5.4. SpringProx API at the highest level

SpringProx API is a C library, available as a dynamic library under Windows. Some software development languages or environments can't use the DLL directly, either because they aren't able to link against a C-based library (Visual Basic 6 for example) or because they run in a virtual machine providing classical access to the binaries (.NET framework for example).

In both cases, it is still possible (and furthermore recommended) to use SpringProx API instead of trying to access the K531 or the K632 module directly. This can be done using a wrapper, either the SpringProx ActiveX, or the SpringProx class library for .NET .

For details regarding those topics, please refer to SpringCard **SpringProx SDK for PC**, which focuses on high-level Windows (or Linux) development using CSB4 or CSB5.

3. GOING FURTHER

3.1. LIST OF PRODUCTS COVERED BY THIS SDK

The K531 & K632 SDK covers a wide range of contactless couplers, having a common hardware basis –the K531 or the K632 OEM modules– and various antennas or host interface.

Here's a quick glance of the products that can be used together with this SDK :

Product name	Description	Host interface
K531	The K531 OEM module	TTL (5V)
K531-TTL	K531 module + 45x69mm PCB with antenna 8-pin connector as host interface	TTL (3V, 5V tolerant)
K531-232		RS-232
K531-485		RS-485
CSB4-S	K531 module + 85x140mm PCB with antenna	RS-232
CSB4-U		USB
K632	The K632 OEM module	TTL (5V)
K632-TTL	K632 module + 45x69mm PCB with antenna 8-pin connector as host interface	TTL (3V, 5V tolerant)
K632-232		RS-232

Note than some other products, even not listed here (either newer or in different families), still comply with this SDK. Please contact support-dev@springcard.com for any question regarding the use of this SDK with others products.

3.2. CHANGING OR UPDATING PRODUCT FIRMWARE

Note that all devices based on K531 & K632 module may be flashed with any of the K531 & K632 family firmware.

The firmware download utility, together with its detailed documentation for the firmware update operation, is available online at

www.springcard.com/download/flash

There are 4 different firmwares compliant with K531 & K632 operation as described in this SDK :

Firmware	Description	Remark
K531	Default firmware	
K531 485	For RS-485 devices only	USER pin drives the RS-485 transmit buffer
CSB4	For CSB-4U or CSB4-S	Same as default with only 2 differences: ■ RF field remains OFF after power up or reset ■ LEDs are driven by firmware to report device's activity
K632	For device based on K632 module	Same as default with compliance with ISO/IEC 15693

Each firmware comes in 2 releases :

- "Mk2" release is suitable for products based on K531-2R or K531-2R4 (CPU is Renesas R8C/25)
- "Mk1" (or blank) release is suitable for products based on K531 or K531-R (CPU is Renesas H8/3664)

Hardware "Mk1" is now deprecated, and all products now ship with K531-2R4 "Mk2". If working with a former hardware, be careful that "Mk1" firmwares are not upgraded anymore (last version will remain 1.40).

3.3. SPRINGPROX API OR LOWER LEVEL DEVELOPMENT ?

As seen in the previous chapter, there are two ways to work with the K531 or the K632 module :

- Direct access using one of the low level protocols,
- High-level access through functions provided by SpringProx API.

Choosing the best options may be difficult for beginners, so let's discuss it.

3.3.1. Availability of SpringProx API

OS	CPU	Binary available ?	Remark
Win32	i386	As a DLL	Source code also provided in this SDK
Windows CE PocketPC	ARMv4	As a DLL	Not provided in this SDK, please refer to **SpringProx SDK for PocketPC**
Linux	i386	As a static LIB	Source code also provided in this SDK

Apart from those 3 configurations, you'll have to compile the SpringProx API for your actual target[2].

Successful implementations of SpringProx API have already been done either by SpringCard or its customers on the following target :

- Nucleus running on ARM 7
- Linux running on Axis Etrax 100LX
- Renesas H8/300 CPU without OS
- …

Depending both on compiler and chosen options, the library size may vary between 32 and 128kB. RAM requirement is less than 4kB, or 10kB when built-in cryptographic functions (DES, 3-DES, MD5) are chosen.

3.3.2. Availability of direct access

Direct access is generally speaking available everywhere, as only one serial port (UART) is needed.

Numerous projects use K531 & K632 direct access from various CPU/MCU, most of them without OS. Here's a short list :

[2] SpringCard can also provide upon request a DLL for Windows CE running on i386 or on MIPS. Please contact us if you're in one of these cases.

- Renesas H8/300 CPU (10MHz)
- Renesas R8C CPU (10MHz)
- PIC 16F and 18F (8MHz)
- Neuron Chip 3150 (10MHz)
- Atmel 8051 and AVR (8MHz)
- …

In order to implement direct access through "fast" binary protocol, minimal requirements are :

- UART at 38400bps (or 115200bps for high speed)
- One buffer to implement UART's FIFO[3]. Same buffer can be used for input and output. Minimal size of buffer is
    - 60 bytes if working only with Mifare or other memory cards
    - 80 bytes if working with Desfire or T=CL cards with a frame size <= 64 bytes
    - 280 bytes if working with T=CL cards supporting a frame size up to 256 bytes.
- A timer to detect communication timeouts (10ms accuracy. If overall performance is not an issue, 100ms accuracy is largely enough).

3.3.3. Conditions where direct access is really easy

In most basic situation, the power of SpringProx API is not interesting, or even porting the API is not possible (small micro-controller). Here's a list of basic operations that can be easily performed using only direct access :

- Retrieving only standard serial numbers (ISO/IEC 14443-A UID or 14443-B PUPI)
- Mifare UltraLight read / write
- Mifare Standard read / write[4]
- Desfire select / read in plain mode (no ciphering, no MACing)
- T=CL (ISO/IEC 14443-4 + 7816-4) select / read in plain mode

3.3.4. Conditions where using the SpringProx API is definitively better

As explain earlier, SpringProx API "hides" the intrinsic complexity of certain contactless cards.

---

[3] Since there's no flow control, the module sends its frame one byte after the other, without guard time. The host CPU shall enqueue the frame in a buffer in an ISR or IRQ handler, and check only afterwards what has been received.

[4] Be careful that a "change key" operation is technically speaking no more than a write into sector's trailer, but the security block to be written MUST be properly formatted (any error in access condition bits may permanently lock the sector). There's a function SpringProx API to do this without risk.

For example, getting authenticated on a Desfire card is a three-step operation, with cryptographic computation at each step. SpringProx API exposes this sequence as a single function call with 2 parameters, where direct access involves that you (re)write the complete stuff.

Here's a list of a few operations that may take longer to (re)write :

- Mifare Standard "change key" function (see note 4)
- ISO/IEC 14443-B anti-collision
- Desfire authentication and secure communication
- Working with non-T=CL contactless cards (Calypso, …)
- Working with non-Mifare memory cards (Innovision Jewel, ASK CTS, …)

4. WHAT'S NEXT ?

This short guide has shown you how the K531 and the K632 work. You're now ready to write your own piece of software, using one of the two options (direct access or SpringProx API) depending on your host and your requirements.

In both cases, you'll find in the `docs` sub-directory all relevant information you'll need to develop your application. Also remember that almost ready-to-use examples are provided in this SDK, in C-language :

Typical application	Direct sample	SpringProx API sample
Polling, getting serial numbers	`ref_direct -poll`	`ref_showuid`
Mifare Ultralight operation	`ref_direct –mif_ul`	`ref_mif_ul`
Mifare Standard operation	`ref_direct –mif`	`ref_mifare`
Desfire full test		`ref_desfire`
T=CL small demo	`ref_direct –tcl`	`ref_tcl`
Reading ASK CTS cards		`ref_askcts`
Reading Innovision Jewel cards		`ref_jewel`
ISO15693 and ICODE1 operation		`ref_vicc`

DISCLAIMER

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

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

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

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

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

COPYRIGHT NOTICE

All information in this document is either public information or is the intellectual property of PRO ACTIVE and/or its suppliers or partners.

You are free to view and print this document for your own use only. Those rights granted to you constitute a license and not a transfer of title : you may not remove this copyright notice nor the proprietary notices contained in this documents, and you are not allowed to publish or reproduce this document, either on the web or by any mean, without written permission of PRO ACTIVE.

EDITOR'S INFORMATION

PRO ACTIVE SAS company with a capital of 227 000 €
RCS EVRY B 429 665 482
Parc Gutenberg, 13 voie La Cardon
91120 Palaiseau – France