Application Note

Using the M519 in PC/SC Coupler mode over a Serial interface

Headquarters, Europa

SpringCard SAS
2, voie la Cardon
Parc Gutenberg
91120 Palaiseau
FRANCE

Phone: +33 (0)1 64 53 20 10

Americas

SpringCard Inc.
185 Alewife Brook Parkway,
ste 210
Cambridge, MA 02138
USA

Email: sales@springcard.com

www.springcard.com

Document Identification

Category	Application notes
Group/Family	SpringCore / M519
Reference & Version	PNA23174-AA
Date	10/10/2023
Diffusion	Public
Keywords	M519, PC/SC, CCID, serial, SDK

Revision History

Version	Date	Author	QC	Description
AA	16/08/2023	JDA	CFE	Initial release

Table of Contents

1 Introduction

1.1 Overview

This document describes the principle of using the M519 and its derivatives as a PC/SC Coupler over a Serial communication link.

The M519 is an advanced dual-interface OEM module, supporting contactless operation (NFC/RFID HF, ISO/IEC 14443 and 15693) through an external antenna, and, through an optional interface board, contact (ISO/IEC 7816) operation.

The M519 supports various operating modes: Smart Reader, RFID Scanner (keyboard wedge), PC/SC Coupler, and more. The PC/SC Coupler mode is both the most versatile and the most open (interoperable) of all modes, but it is generally limited to USB devices (and more specifically to the CCID protocol specified by the USB Workgroup).

Fortunately for developers/integrators who aim to build a smart card-aware application for a serial-only host in a seamless way, the M519 implements an equivalent “CCID over Serial” protocol; this makes it possible even for the tiniest host to run a “PC/SC-Like” software library. This document explains how to do so; it is associated with the `springcard-ccid-serial` SDK (software development kit) that can be downloaded freely from GitHub:

https://github.com/springcard/springcard-ccid-serial

The document will guide you through the process of setting up and operating the M519 in PC/SC Coupler mode over a Serial interface, porting the SDK to your target system or micro-controller, and start developing your own smart card-aware solution.

1.2 Target platforms — a foreword

It is difficult to present the integration of a product such as the M519 within an embedded target in the general case, as each embedded target has its own specific

features (bare-metal microcontroller, cooperative micro-kernel, pre-emptive micro-kernel, use of DMA and/or DPC, etc.).

We have chosen to focus our SDK and examples on just two targets

- The PC world (Linux or Windows), because the M519 development board (M519-SRK) provides easy access to the M519 serial interface via a FTDI USB-serial bridge,
- The Raspberry Pico (RP2040 processor), because it is an inexpensive development board that is very easy to supply and start up. Unlike other boards, all its I/Os are 3V, and its Cortex-M0+ core is an industry standard.

The following chapters will give a brief overview of how to set up projects for the PC and for the Raspberry Pico.

To make sure the source code provided in the SDK could be “easily” ported to any target, even a small MCU, the following compromises have been made:

- The code is purely imperative, procedural, and uses a single execution context (it does not involve multi-tasking or multi-threading[1]);
- No complex queues are involved, all the memory used is statically allocated.

Of course, implementers remain totally free to introduce more advanced programming patterns when creating their own solution.

1 Linux and Windows examples use a thread only to simulate the UART interrupt (ISR) that would exist on a MCU.

1.3 Related Documents

1.3.1 Documents available as PDF

Reference	Title / Description
PFT22217	M519 Data Sheet and Integration Guide
PMD23175	M519-SRK Getting Started Guide
PMI23209	M519-SUV Getting Started Guide
PMD15305	Zero-Driver CCID low level Implementation
PNA23207	Using the M519 in PC/SC Coupler mode over a USB interface

1.3.2 Online Material

Documentation of the SpringCore firmware.

https://docs.springcard.com/books/SpringCore/Welcome

SpringCard Tech Zone, the blog of the R&D Team

https://tech.springcard.com/

1.4 Related Literature

The present document makes use of concepts and vocabulary that are taken from existing standards and specifications. Please refer to the original documents for a complete understanding.

1.4.1 ISO Standards

1.4.1.1 ISO/IEC 7816

ISO/IEC 7816 is an international standard that defines the characteristics and requirements for integrated circuit cards (ICCs) or smart cards, as well as their interfaces with devices. This standard encompasses physical, electrical, communication, and

application-specific aspects of smart cards, providing a comprehensive framework for their design and use. It defines two protocols: T=0 (character) and T=1 (block).

According to this standard, the M519 is an ICC coupling device (CD) e.g. a smart card coupler.

The M519 provides the protocol-level (TPDU) interface; communication between the host application and the smart card through the M519 takes place at APDU level. This principle is fully described in ISO/IEC 7816-4.

Visit iso.org online store if you want to buy this standard.

1.4.1.2 ISO/IEC 14443 and 15693

ISO/IEC 14443 defines the standards for proximity cards used for identification and contactless communication. ISO/IEC 15693, specifies the requirements for vicinity cards, which have a longer communication range when used with a larger antenna. According to these standard the M519 is both a proximity coupling device (PCD) and a vicinity coupling device (VCD).

Both standards detail parameters like physical characteristics, radio frequency power, and signal interface for contactless ICC.

High-end contactless smart cards are operated at APDU-level over a block protocol ("T=CL") exactly like their contact counterparts, where low-end RFID ICs use a proprietary command set (this is for instance the case for the NXP Mifare family) or the low-level command set defined in ISO/IEC 15693.

Visit iso.org online store if you want to buy these standards.

1.4.2 PC/SC Standard

PC/SC (Personal Computer/Smart Card) is an open standard that ensures seamless integration and communication between smart cards and personal computers. It is used in applications like digital security, network security, and identity authentication.

The documents are publicly available on https://pcscworkgroup.com/

The M519 is compliant with the current version of PC/SC (v2).

Most high-end operating systems feature a PC/SC stack, generally associated to a set of USB drivers. But since this document focuses on Serial communication, we consider that the target is a microcontroller or PLC, that lacks PC/SC support. Therefore, a lightweight equivalent of the PC/SC stack has to be implemented on your side. The `springcard-ccid-serial` SDK, and its "PC/SC-Like" library, will help you doing so.

1.4.3 CCID Standard

CCID (Chip Card Interface Device) is a USB protocol that allows a smart card to be connected to a computer via a card reader. It facilitates communication between the computer and the smart card.

The documents is publicly available on https://www.usb.org/document-library/smart-card-ccid-version-11

Because CCID is a USB-only specification, when used over its Serial interface, the M519 uses a SpringCard-specific protocol. Yet, this protocol is so closely derived from CCID that reading this standard is helpful to be able to implement the "driver" for the M519 into your own target.

The `springcard-ccid-serial` SDK, and its "CCID over Serial" driver, will help you doing so. The reference document is [PMD15305].

1.5 Glossary

Reference	Title / Description
APDU	Application-Protocol Datagram Unit. The name comes from the OSI model to describe end-to-end command/response exchanges, between two applications.
ATR	Answer To Reset. This is the identification message that a smart card sends when starting up.
CCID	Chip Card Interface Device, the USB specification for smart card couplers
HAL	Hardware Abstraction Layer
PCD	Proximity Coupling Device; i.e. a contactless coupler compliant with ISO/IEC 14443
PC/SC	Personal Computer/Smart Card
PICC	Proximity Integrated Circuit Card; i.e. a contactless smart card compliant with ISO/IEC 14443
OSI	Open Systems Interconnection: a conceptual framework used to understand and standardize the functions of telecommunication and computing systems
RS-232	A standard serial communication standard, with 2 independent lines (RX/TX) for full-duplex communications. Electrical levels are - 0: +3 to +15V - 1: -15 to -3V
RS-485	A standard serial communication standard, with 1 differential pair allowing half-duplex communications only. Electrical levels are: - 0: $V_A < V_B$ - 1: $V_A > V_B$
RS-TTL	Not a standard, but a shortcut to describe serial communication with 2 independent lines (RX/TX) for full-duplex communications and electrical levels that are TTL or TTL/CMOS: - 0: < 0.8V - 1: > 2V
TPDU	Transport-Protocol Datagram Unit. The name comes from the OSI model to describe a packet or frame sent by a device to another.
USB	Universal Serial Bus

VCD	Vicinity Coupling Device; i.e. a contactless coupler compliant with ISO/IEC 15693
VICC	Vicinity Integrated Circuit Card; i.e. an RFID chip compliant with ISO/IEC 15693

2 Hardware setup

2.1 Quick-start for the M519-SRK

The M519-SRK “Starter Kit” is a mother board for the M519 and various peripherals (antenna, smart card interfaces). It is designed to let the implementer/developer evaluate the M519 in various configurations easily.

Warning: setting a wrong hardware configuration or applying a wrong power level is likely to damage the M519 or the host. Please refer to [PMD23175] for details.

2.1.1 M519-SRK + PC, using the FTDI USB to Serial bridge

- Disconnect any existing connection (USB or serial), unplug power jack,
- Set jumper JP4 “RF PWR SETUP” to position “EXT”
- Set jumper JP1 to position “JP4”,
- Set jumper JP5 “V_RF_IN” to position “JP1”,
- Plug a DC 12V power supply into jack “12V PWR”,
- Connect the host using an USB cable to the “USB UART” mini-B connector.

NB: in this configuration, you must provide external power to the SRK through its power jack. The schematics does not derivate any power to the M519 from the USB link arriving at the FTDI chip.

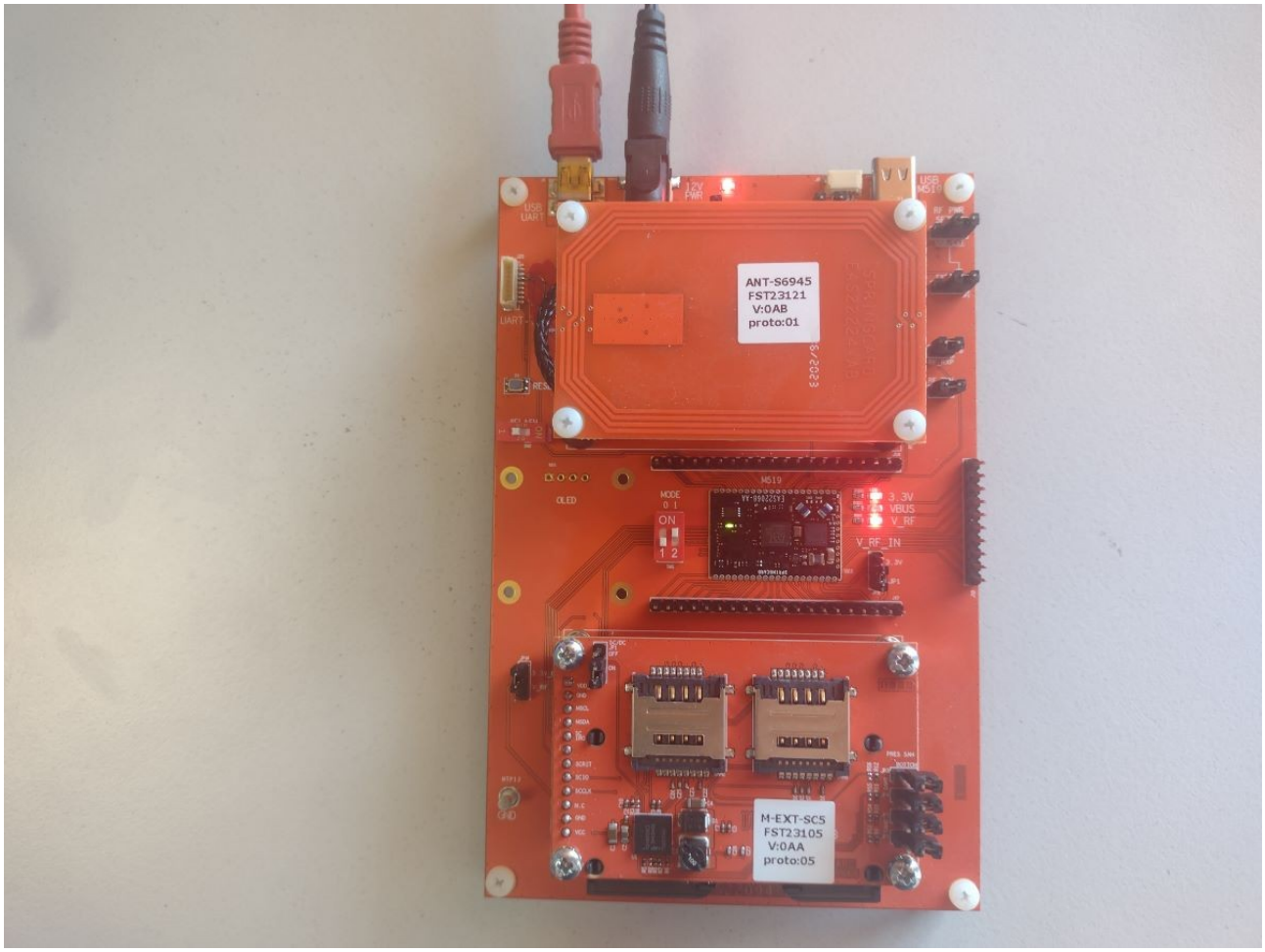

Illustration 1: M519-SRK connected to the PC though its FTDI USB to Serial bridge

2.1.2 M519-SRK + PC, using the JST 8-pin connector

- Disconnect any existing connection (USB or serial), unplug power jack,
- Set jumper **JP4** “RF PWR SETUP” to position “EXT”
- Set jumper **JP1** to position “JP4”
- Set jumper **JP5** “V_RF_IN” to position “JP1”,
- Connect the host UART (3V level) and the power supply (5V DC) to the JST PH 8-pin connector “UART”.

NB: in this configuration, you may provide external power to the SRK through its power jack if the power source at the JST connector is too weak.

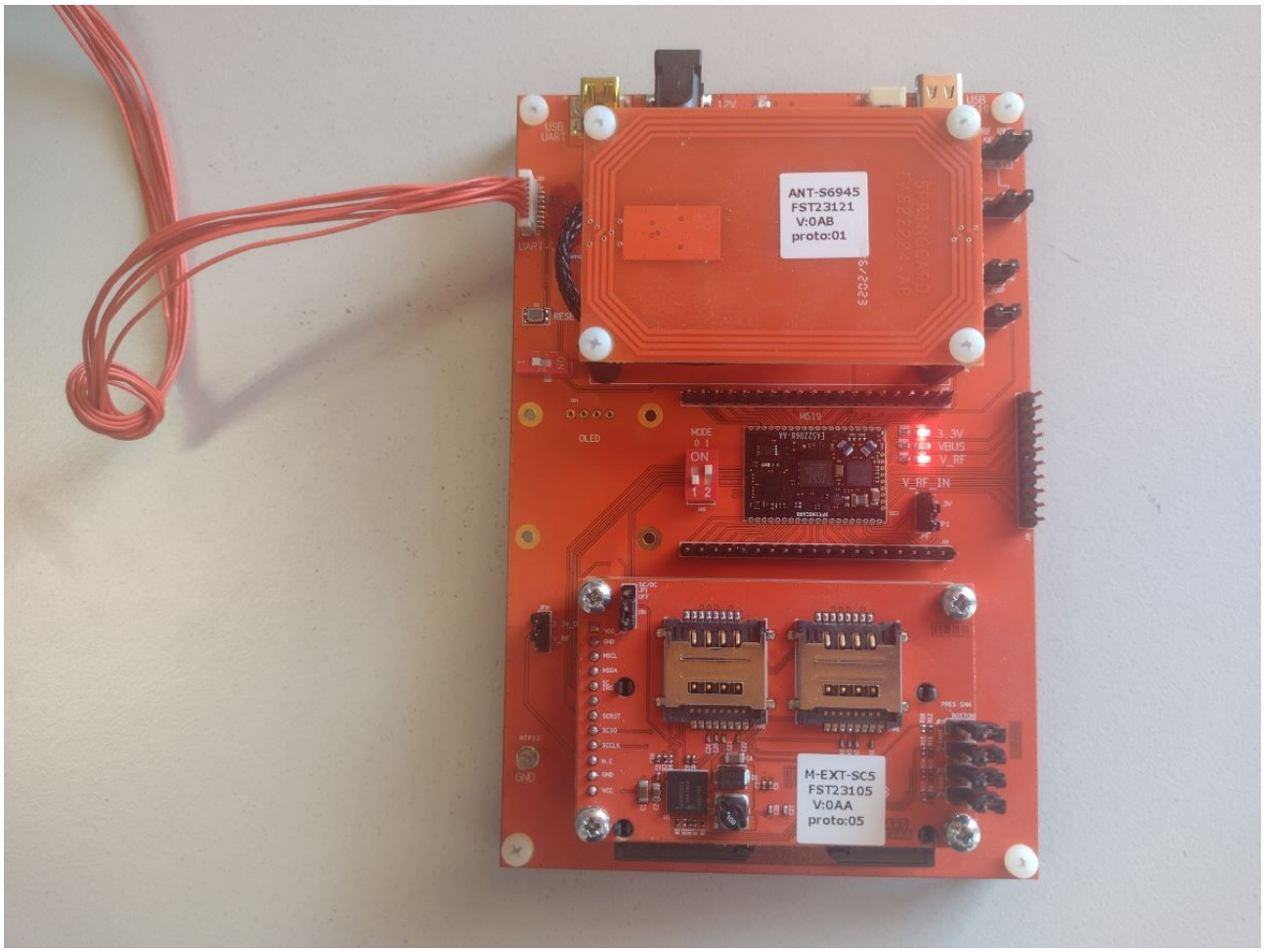

Illustration 2: M519-SRK connected and powered through its JST 8 connector

2.1.3 M519-SRK + Raspberry Pico

- Disconnect any existing connection (USB or serial), unplug power jack,
- Open jumper JP5 “V_RF_IN”,
- Use a breakout board and jumper wires to connect a Raspberry Pico, according to the following pinout:

Signal	Pin on the Raspberry Pico	Pin on the M519
Ground	38 (GND)	37 (GND)
3.3V power supply	36 (3V3 OUT)	39 (VIN_RF) 40 (VIN_3V3)
Serial, host to M519	1 (UART0 TX)	21 (M519 RX)
Serial, M519 to host	2 (UART0 RX)	21 (M519 TX)

It could be a good idea to add a few jumper wires to connect a logic analyser, as done in the picture below.

- We will be using the USB link of the Raspberry Pico to run the sample. The M519 is powered by the 3V3 output generated by the Raspberry Pico from the 5V delivered by the host through the USB link.

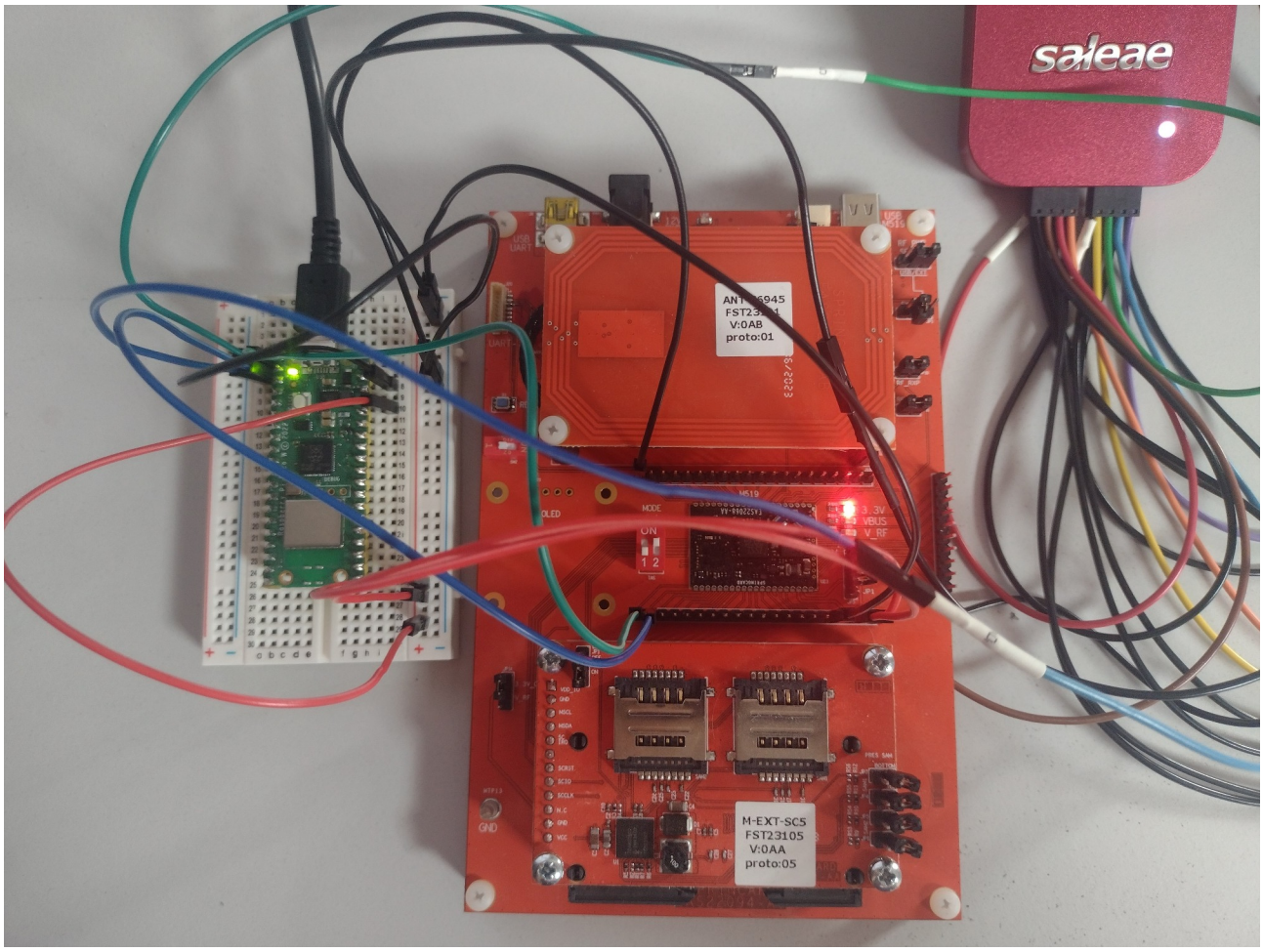

Illustration 3: Raspberry Pico W, M519-SRK and Logic analyser (Saleae)

2.2 Quick-start for the M519-SUV

The M519-SUV “Serial/ USB Versatile antenna” is a 69x45 antenna that holds the M519. It is designed to be integrated in virtually any equipment and either an USB or a Serial interface. The electrical level of the Serial interface (“RS-TTL”, RS-232 or RS-485) depends on the variant.

Warning: setting a wrong hardware configuration or applying a wrong power level is likely to damage the M519 or the host. Please refer to [PMI23209] for details.

2.2.1 M519-SUV + PC

TBD

2.2.2 M519-SUV + Raspberry Pico

TBD

2.3 Other platforms

This paragraph exposes only the requirements regarding the serial interface itself, the power supply, and the optional handshaking and configuration signals. Of course, an antenna and its matching circuits are necessary for contactless operation, and contact interfaces are necessary to use contact smart cards.

Please refer to [PFT22217] for details.

2.3.1 Serial interface

Using the M519 in PC/SC Coupler over a serial link is possible using only the 2 communication lines (RX, TX) of the serial interface.

On the M519 module itself, the serial interface operates as 0 / 3.3.V. Depending on the hardware behind, your system may communicate with the M519 by the mean of an RS-232, RS-422 or “RS-TTL” (0 / 5V) interface. RS-485 may also be used, but with particular precaution since this medium is half-duplex only *(more on that in § 5.4)*.

When using the M519-SUV, always check the actual configuration of the antenna, since this single hardware provides the 3 interface levels. Do not connect the USB interface together with the serial interfaces since USB takes priority.

When using the M519-SRK, you may connect to the M519 using either

- the USB to Serial bridge “USB UART” (USB mini-B connector, FTDI chipset),
- the JST PH 8 connector “UART” that supports either 0 / 3.3V or 0 / 5V levels,

- the header that is directly connected to the module's pin (0 / 3.3V only in this case).

Connect only one interface at once. Do not connect the main USB interface (USB type C connector, label "USB M519") together with the serial interfaces since USB takes priority.

Always report to the detailed documentation of the actual hardware involved to avoid damaging the M519 or the host system.

2.3.2 Power supply

The serial interface is available only when the M519 is powered at 3.3V (5V power is associated with USB).

Unless you are using a complete product that derivates a 3.3V rail from over power source, respect the following setup:

- Connect both VIN_3V3 (pin 40) and VIN_RF (pin 39) to a 3.3V power supply that can deliver at least 500mA,
- Connect both GND signals (pins 37 and 20) to the ground of the power supply.
- Leave VBUS (pin 38) unconnected.

2.3.3 Optional handshaking signals

Synchronization between the host system and the M519 can be simplified using a couple of I/O lines (0 / 3.3V).

- /WAKEUP (pin 16)

This signal is asserted (LOW level) by the M519 when it is not processing a command from the host (and therefore is HIGH while the M519 is processing a command, or starting-up).

A HIGH to LOW transition of the /WAKEUP signal let the host know that the module has finished processing the last command sent by the host. The host shall never send a command unless /WAKEUP is low.

When the M519 starts, the host should wait for the HIGH to LOW transition of the /WAKEUP signal before sending the first command. If the /WAKEUP signal is not made

available to the host, the host shall wait at least 500ms before probing the M519 with a first command.

When the M519 resets for any reason while no command is running, the /WAKEUP signal makes a LOW to HIGH transition. This make it possible for the host to detect immediately that the M519 has restarted. If the /WAKEUP signal is not made available to the host, only a failure or timeout on the next command will tell that something went wrong.

- /RESET (pin 19)

This signal may be driven by the host to reset the M519.

Start-up of the M519 may take up to 500ms (more if a firmware upgrade takes place). Therefore, the host shall way at least 500ms before probing the module (unless the module asserts the /WAKEUP signal earlier).

- /SUSPEND (pin 17) **[NOT IMPLEMENTED IN CURRENT FIRMWARE REVISION]**

This signal may be driven by the host to tell the M519 to enter a low power mode.

In PC/SC Coupler operation, when /SUSPEND is asserted by the host, the M519 switches its contactless (RF) interface off, and powers down all the contact cards (if some).

Entering suspend mode takes a few seconds, and resuming from suspend mode also takes a few seconds. Therefore, the host shall not toggle the /SUSPEND pin for less that 10 seconds.

2.3.4 Optional configuration signals

The M519 features two configuration pins, MODE0 (pin 31) and MODE1 (pin 32).

When MODE0 and MODE1 are HIGH level or left unconnected, the M519 uses the configuration defined in its non-volatile memory. Namely, its operating mode is the one that has been selected by register $_H$02C0. Value shall be $_H$02 for PC/SC operation.

https://docs.springcard.com/books/SpringCore/Non_volatile_memory/Configuration/Main_configuration/Profile

It is possible to override the configuration stored in the non-volatile memory by asserting either MODE0 or MODE1 at boot time (power up or action on /RESET). Tying MODE1 to a LOW level while MODE0 remains HIGH level enforce PC/SC operation, whatever the value in register $_{H}$02C0.

The subject of the configuration of the M519 is covered in details in the next chapter.

3 Configuring the M519 for PC/SC Coupler operation

M519 OEM module:

The M519 is delivered with a default configuration that enables only the SpringCard Direct protocol, not the CCID protocol that is required for PC/SC operation.

There are some exceptions to the fact that the M519 is delivered with only the SpringCard direct protocol enabled. For instance, sample products may have been pre-configured by SpringCard FAE / Support team for the ease of a particular demonstration. Also custom order codes may be placed for large batches of pre-configured modules.

Anyway, generally speaking, the integrator shall always plan for and execute a configuration step for the M519 module while assembling the final system.

There are two methods to configure the M519: either by the setting MODE0 and MODE1 signals, or by setting register $_H$02C0 to value $_H$02 in the non-volatile memory.

Products derived from the M519:

Products like the M519-SUV are generally delivered with a specific factory configuration. Consult SpringCard Sales team to select the appropriate order code to get M519-SUV pre-configured for PC/SC Coupler operation. Pay attention that writing a new configuration, as explain below, will overwrite the factory configuration.

3.1 Configuring the M519 using MODE0 and MODE1

To enforce PC/SC Coupler operation,

- Leave MODE0 unconnected, or connect it to the 3.3V level,
- Connect MODE1 to the ground,
- Perform a full reset of the module, either by cycling the power supply or asserting then releasing the /RESET signal.

Of course this method is not available on the M519-SUV since MODE0 and MODE1 are not made available on any of the connectors.

When using the M519-SRK, simply move the MODE1 switch to the ON position, while the MODE0 remains in the OFF position, and press the RESET button.

3.2 Configuring the M519's non-volatile memory

The simplest procedure uses a terminal software and human interaction with the M519. It should work whatever the current configuration of the M519 is, since the shell ("human console") is available in all modes.

- Connect to the M519 over a serial interface. Refer to next paragraph "Testing the connection" for the procedure and parameters,
- Enter command `cfgC0` to read the current value of register $_{H}$02C0,
- Enter command `cfgC0=02` to activate the CCID protocol and PC/SC operation,
- Enter command `reset` to activate the new configuration.

If the M519 is in its default configuration (SpringCard Direct protocol), the procedure may be completely automated using SpringCoreConfig command line tool, like this:

```
SpringCoreConfig --serial=COM5 --write 02C0=02
```

(Replace COM5 with the actual communication port on your production/test system).

The drawback is that this procedure with fail if the device has already been configured for PC/SC mode (or any other but Direct).

3.3 Testing the connection

3.3.1 Linux

- Download and install a terminal software for serial communication.

For the snapshots, we are using *Minicom* (on Ubuntu/Debian, use `sudo apt-get install minicom` to install it).

- Open a the serial port the M519 is connected to. Configuration is:
  - Baudrate: 38400bps,
  - Format: 8 data bits, 1 stop bit, no parity, no flow control.

The serial port shown in the examples is /dev/ttyS5. Adjust to our actual setup.

- Reset the M519,
- Wait at least 250ms to let the bootloader execute,
- Send <*CR*><*LF*>,
- The M519 echoes the new line sequence and replies

  `SpringCard M519 v1.26 PC/SC Serial`<*CR*><*LF*>
  `>` <*CR*><*LF*>

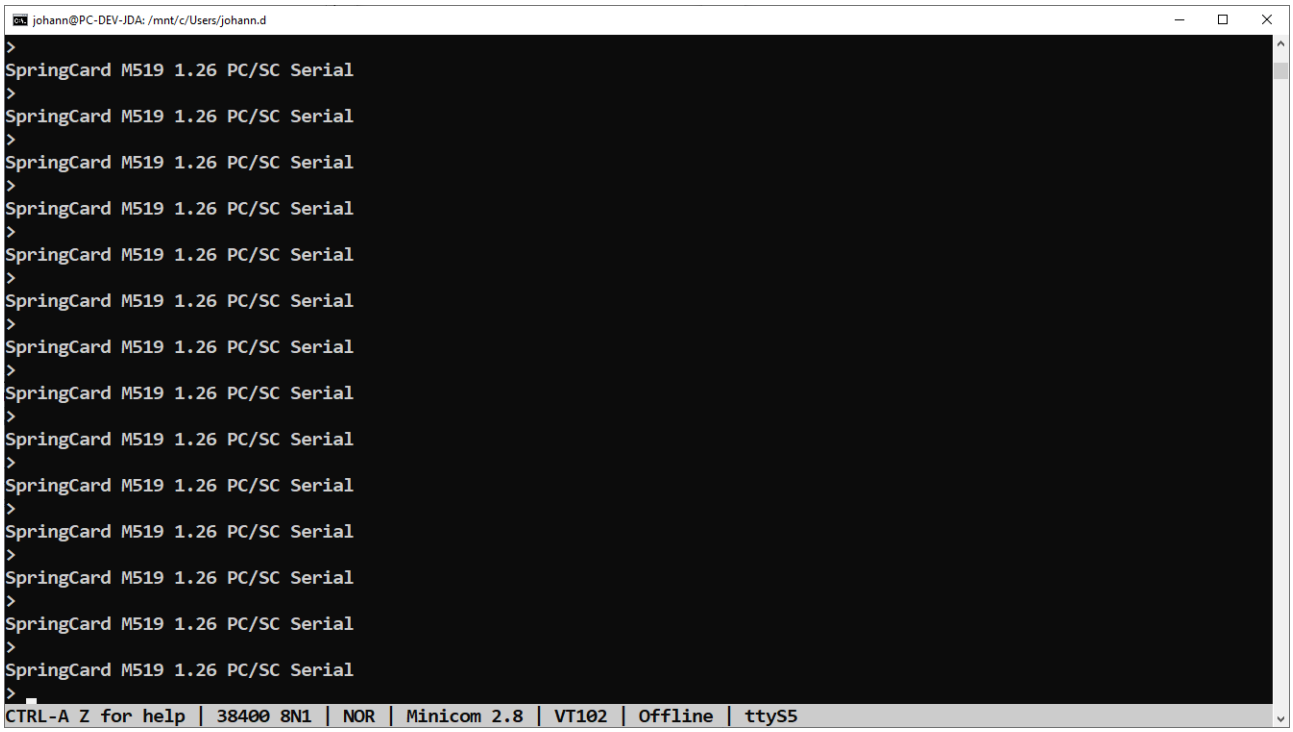

Illustration 4: Retrieving the prompt of the M519 with Minicom

Congratulation! This sequence shows that the M519 is correctly configured and connected to your Linux computer. Don't forget to disconnect the terminal software from the serial port before trying to connect to the M519 from another software.

3.3.2 Windows

- Download and install a terminal software for serial communication.

For the snapshots, we are using *Hterm*, a free software available at

https://www.der-hammer.info/pages/terminal.html

- Open a the serial port the M519 is connected to. Configuration is:
  - Baudrate: 38400bps,
  - Format: 8 data bits, 1 stop bit, no parity, no flow control.

The serial port shown in the examples is COM5. Adjust to our actual setup.

- Adjust the configuration of the terminal software so that
  - Terminal sends *<CR><LF>* when you hit the *<RETURN>* key,
  - Terminal use *<CR><LF>* as new line separator.
- Reset the M519,
- Wait at least 250ms to let the bootloader execute,
- Send *<CR><LF>*,
- The M519 echoes the new line sequence and replies

```
SpringCard M519 v1.26 PC/SC Serial<CR><LF>
> <CR><LF>
```

"v1.26" is the version number of the firmware. Newer firmwares will show another version number.

“PC/SC” is the operating mode. If your M519 answers with another operating mode, go back to previous paragraph “Configuring the M519 for PC/SC Coupler operation”.

“Serial” is the primary host interface. If your M519 answers “USB”, not “Serial”, go back to step “Connecting the M519 to your target”.

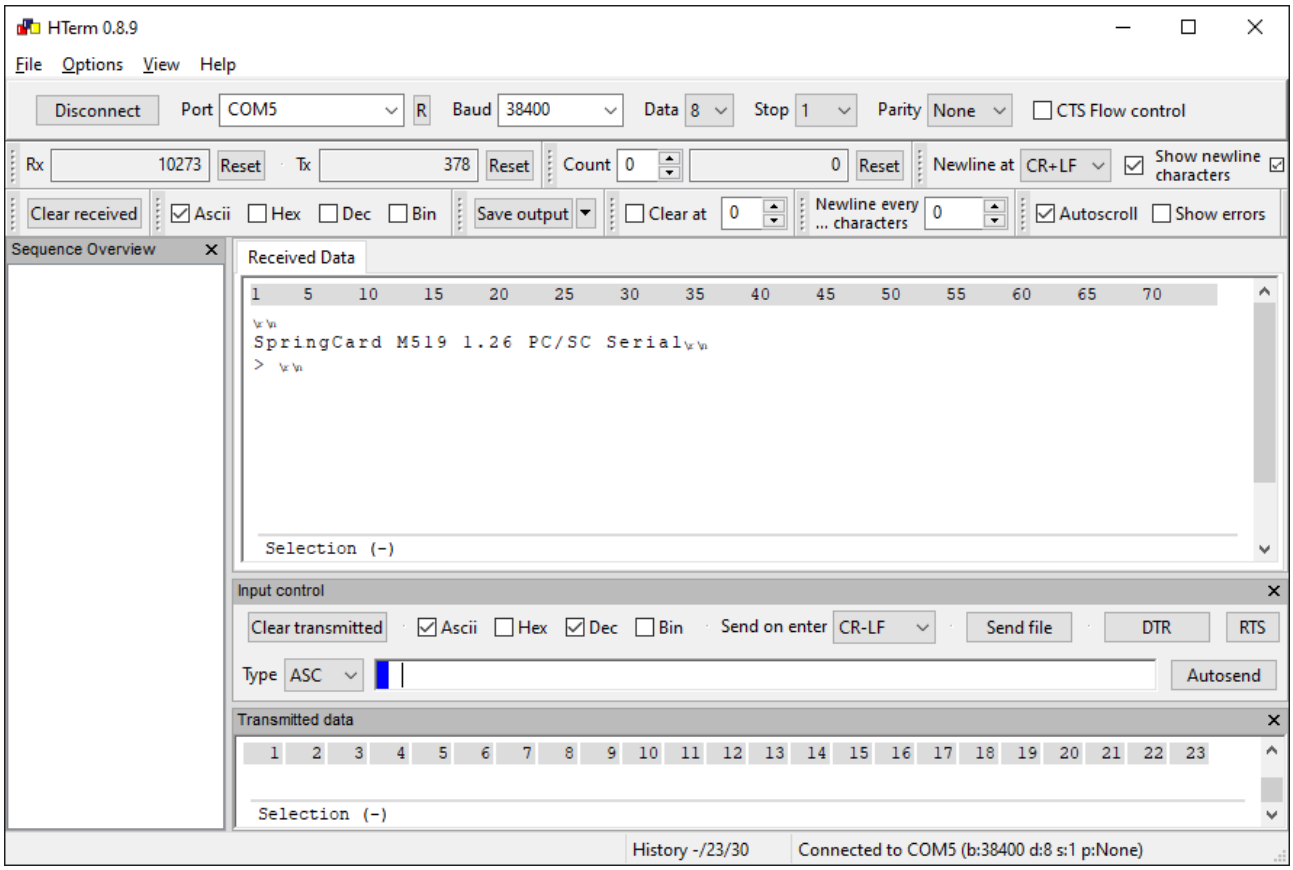

Illustration 5: retrieving the prompt of the M519 with HTerm

Congratulation! This sequence shows that the M519 is correctly configured and connected to your Windows computer. Don’t forget to disconnect the terminal software from the serial port before trying to connect to the M519 from another software.

3.3.3 Raspberry Pico

There is no easy way to test the communication with the Raspberry Pico. You may immediately proceed with the sample (§ 4.3.4).

If the sample does not work immediately, a logic analyser is a valuable tool to understand the problem and come to a solution.

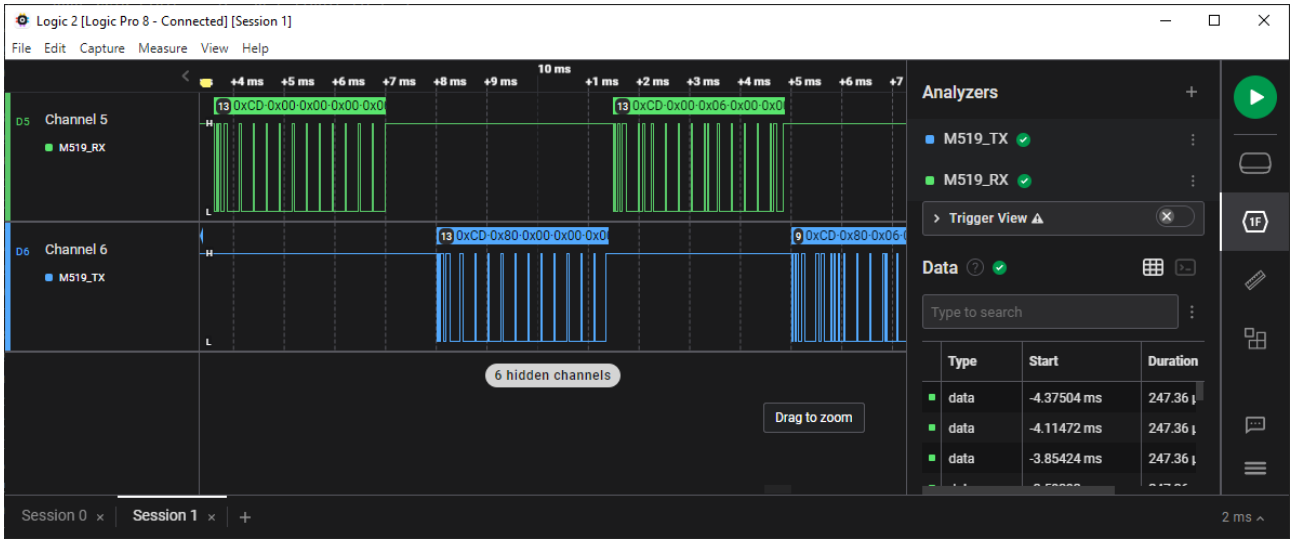

Illustration 6: Capture of the Saleae Logic Pro software showing the initial handshaking (ping then GET DESCRIPTOR) between the Raspberry Pico and the M519

4 Getting started with the SDK

4.1 Downloading the PC/SC-Like for Serial SDK from GitHub

The SDK that accompanies this application note is named `springcard-ccid-serial` and is hosted on GitHub. Direct link is:

https://github.com/springcard/springcard-ccid-serial

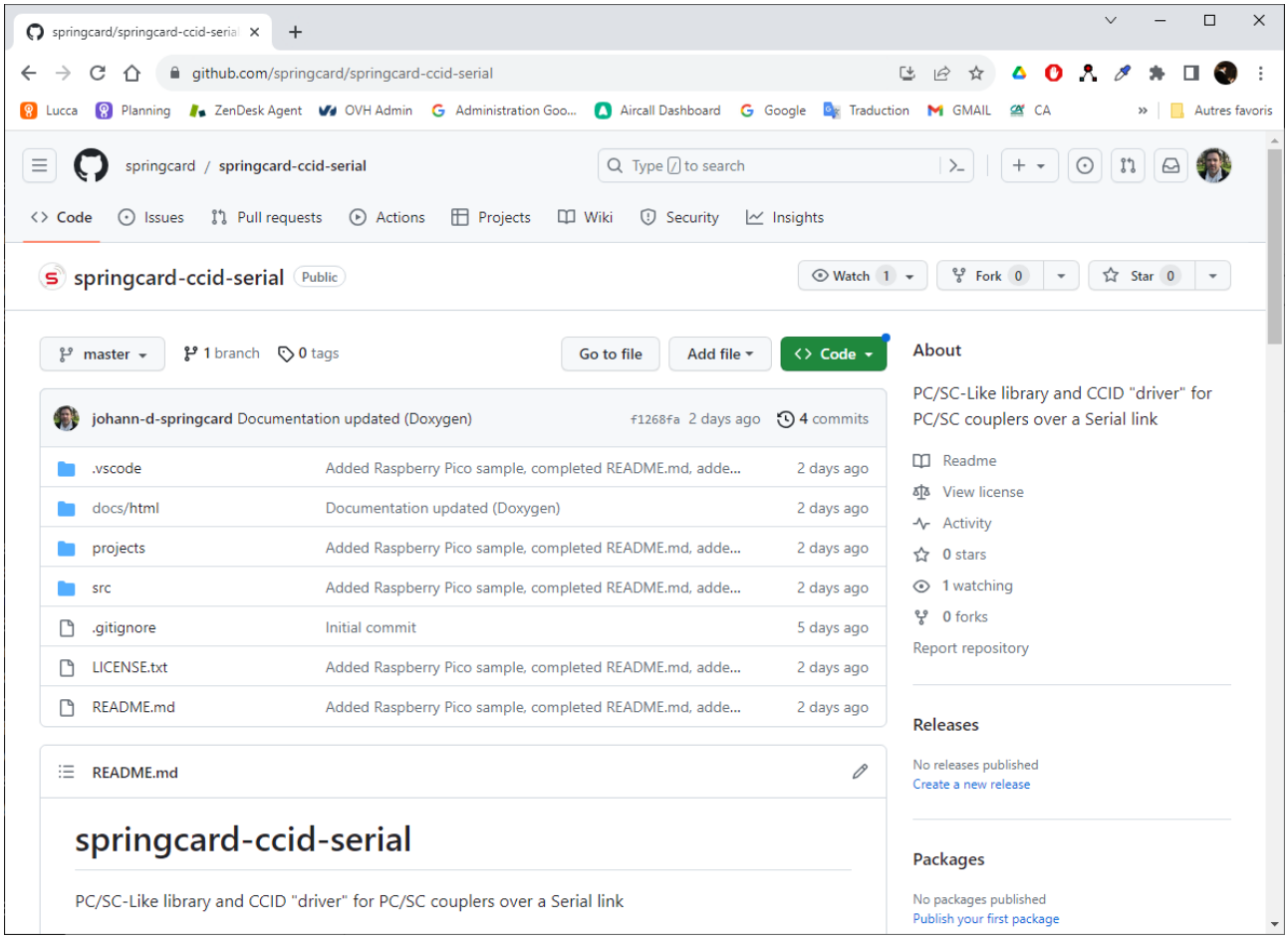

Illustration 7: springcard-ccid-serial project on GitHub

If you have Git installed, you may clone it directly

```
git clone https://github.com/springcard/springcard-ccid-serial.git
```

Then

```
cd springcard-ccid-serial
```

to enter the directory.

4.2 Exploring the SDK files and projects

The organisation of the SDK tree is pretty straightforward:

Directory	Content
/ docs	The HTML documentation of the library and driver, generated by Doxygen
/ projects	
/ linux	
/ rpi_pico	
/ win32_mingw	
/ win32_vs2022	
/ src	
/ ccid	Source code for the CCID “driver”
/ hal	Hardware abstraction layer (one per platform)
/ sample	Source code for the sample application
/ scard	Source code for the PC/SC-Like library

4.3 Building the examples

4.3.1 Linux

To build this sample, you must have GCC and GNU make installed on your machine. There are plenty of tutorials explaining how to do so on Linux.

To build the sample,

- Enter the directory of the project

`cd projects/linux`

- Run `make` to build the project

`make`

When running the sample, specify the communication device after the `-d` command line flag.

`bin/ccid-serial -d /dev/ttyS5`

4.3.2 Windows, using MinGW and GCC

To build this sample, you must have MinGW, GCC and GNU make installed on your machine. There are plenty of tutorials explaining how to do so on Windows.

To build the sample,

- Enter the directory of the project

`cd projects/win32_mingw`

- Run `make` to build the project

`make`

When running the sample, specify the communication device after the `-d` command line flag.

`bin/ccid-serial.exe -d COM5`

4.3.3 Windows, using Visual Studio

To build this sample, you must have Microsoft Visual Studio 2022 and its C/C++ compiler installed on your machine.

- Open the `projects/win32_vs2022/ccid-serial.sln` solution that contains the project.

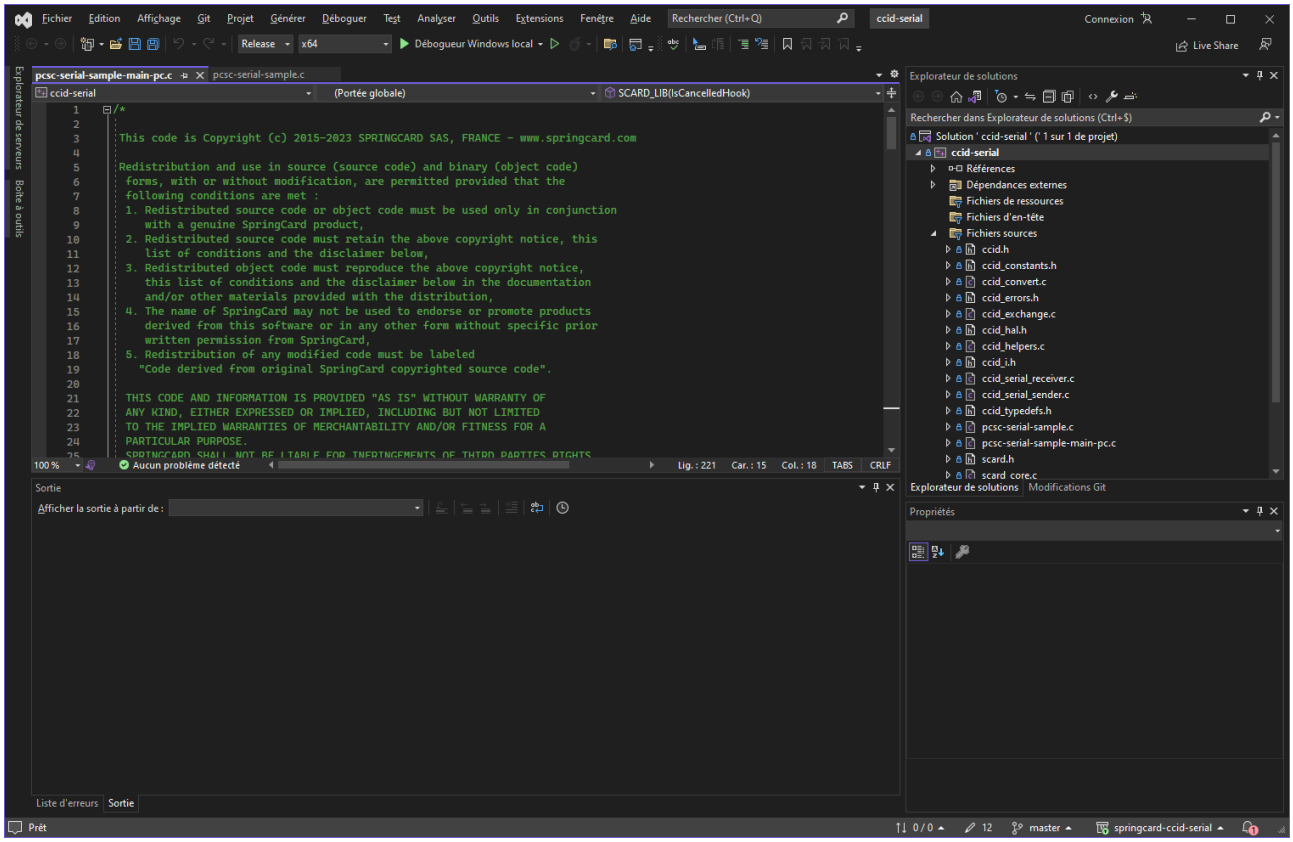

Illustration 8: The ccid-serial solution and project in Visual Studio 2022

- Select the target configuration you want to build: Debug or Release, x86 or x64,
- Build the software.

When running the sample, specify the communication device after the `-d` command line flag.

```
ccid-serial.exe -d COM5
```

4.3.4 Raspberry Pico

To build this sample, you must have the Raspberry Pico SDK and its toolchain (GCC for ARM, CMake, GNU make) installed on your machine. There are plenty of tutorials explaining how to do so on Linux.

For instance, you may follow this tutorial:

https://www.gibbard.me/using_the_raspberry_pi_pico_on_ubuntu/

On a Windows machine, easiest approach is to install or setup WSL (Windows Subsystem for Linux) to run an Ubuntu or Debian system directly inside Windows, then install the required SDK and toolchain under WSL.

To build the sample,

- Enter the directory of the project

```
cd projects/rpi_pico/build
```

- Run `cmake` to recreate the Makefile according to your actual setup

```
cmake ..
```

- Run `make` to build the project

```
make
```

To run the sample,

- Unplug the Raspberry Pico from the USB host, press the button on the Raspberry Pico and keep it pressed, plug the Raspberry Pico to the USB host again,
- Once the Raspberry Pico is seen as a disk drive, copy the ccid-serial.uf2 firmware file to this disk drive,
- The Raspberry Pico then resets and now enumerates as a USB serial communication device,
- Connect to this communication device with a terminal software (parameters are not important; you may use 115200bps, 8 data bits, 1 stop bit, no parity, no flow control) and press the `?` Key to see the banner and menu.

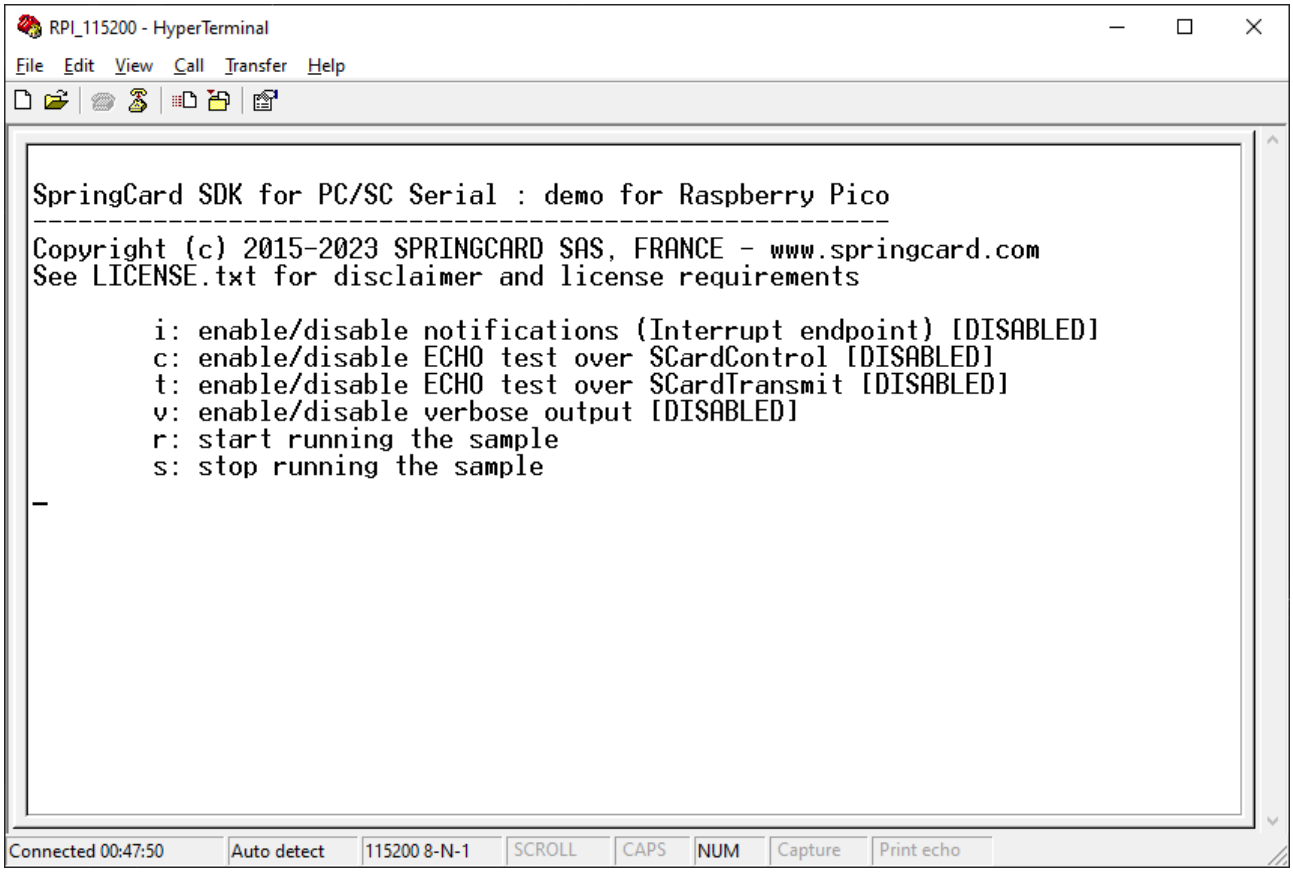

Illustration 9: Banner and menu of the sample application running in the Raspberry Pico

- Press the `r` key to run the example.

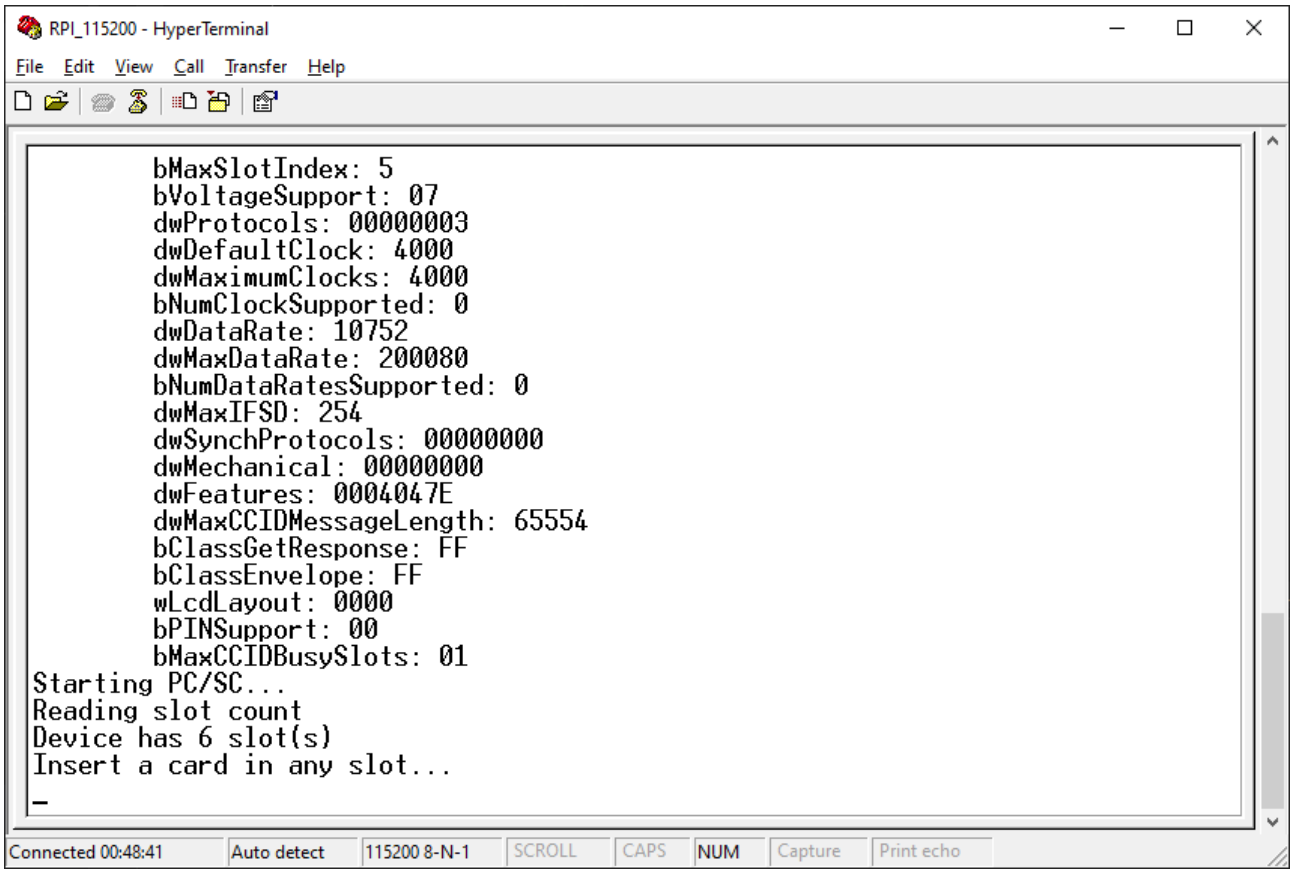

Illustration 10: Sample application running in the Raspberry Pico

4.4 Libraries and namespaces, documentation

The SDK contains two libraries that could be re-used in any project using the M519 hardware:

- The CCID over Serial “driver”,
- The PC/SC-Like library.

4.4.1 The CCID namespace

The public functions of the CCID “driver” are exposed in `ccid.h` .

Every function of this “driver” is prototyped as follow:

```
LONG CCID_LIB(FunctionName)(<Arguments>);
```

Thanks to this CCID_LIB macro in `pcsc-serial.h`, the prototype expands as:

```
LONG CCID_FunctionName(<Arguments>);
```

In the following pages of the documentation, we will write only CCID_FunctionName to ease the reading, but the application code should use CCID_LIB(FunctionName) to keep maximum flexibility.

The return type is generally a LONG, with a few exceptions:

- BOOL for CCID_IsValidDriver that tells whether the driver is up and running, or not,
- BYTE for CCID_GetSequence that is used internally to manage the sequence counter of the CCID messages.

4.4.2 The PC/SC namespace

The public functions of the PC/SC-Like stack are exposed in scard`.h` and the error codes are defined in `scard_errors.h` .

The PC/SC-Like library mimics the implementation of the PC/SC API (`WinsCard.h`) that Microsoft and others have created from the standard. Every function prototyped in `WinsCard.h` has a name starting with SCard (for instance, SCardStatus), and the error-code values have names starting with SCARD_ (SCARD_S_SUCCESS for no error).

To avoid name collisions when building on Windows/Linux, every function in the PC/SC-Like library is prototyped as follow:

```
LONG SCARD_LIB(FunctionName)(<Arguments>);
```

and the error-code values are defined as follow:

```
SCARD_ERR(ERROR_CODE) = <Value>
```

Thanks to the SCARD_LIB and SCARD_ERR macros in `pcsc-serial.h`, these expand respectively as:

```
LONG SCARD_FunctionName(<Arguments>);
```

and

```
SCARD_ERR_ERROR_CODE = <Value>
```

In the following pages of the documentation, we will write for convenience:

- SCardFunctionName (even if the actual name of the function generated by the preprocessor is SCARD_FunctionName),
- SCARD_ERROR_CODE (even if the actual name of the function generated by the preprocessor is SCARD_ERR_ERROR_CODE).

The application code should use SCARD_LIB(FunctionName) and SCARD_ERR(ERROR_CODE) to keep maximum flexibility.

The return type is generally a LONG, with the exceptions of functions with a “Is” in the name (SCardIsValidContext, SCardIsCancelledHook, SCardIsFatalError) that return a BOOL.

4.4.3 Documentation of the PC/SC-Like stack and CCID “driver”

The code is documented using the Doxygen syntax.

The corresponding documentation is available as HTML format in the `/docs` directory.

Pay attention that due to the CCID_LIB, SCARD_LIB and SCARD_ERR macros control the namespaces; the actual function and macro names may be different than the ones written in the documentation.

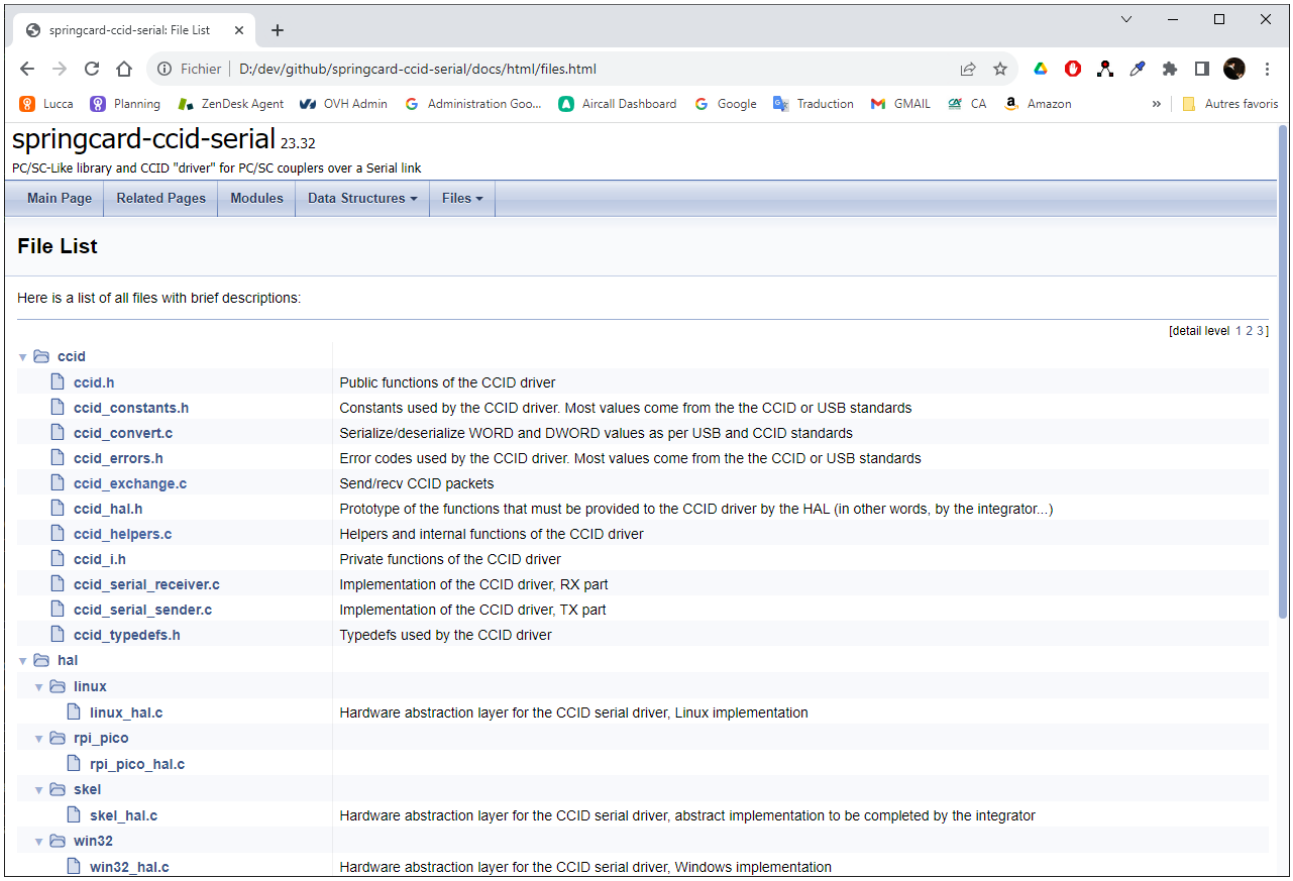

Illustration 11: HTML documentation of the libraries

4.5 The sample application at a glance

All the examples that can be compiled within the SDK are based on the same source code and run basically the same sequence:

- Initiate serial configuration, send a “ping” command and expect to receive a response from the M519.

The “ping” command is actually a GET STATUS command in the CCID over Serial protocol, as documented here:

https://docs.springcard.com/books/SpringCore/Host_Protocols/CCID_Protocol_(PCSC)/Non_USB_Control/GET_STATUS

Once a device has answered to the ping, the sequence continues with:

- Read the descriptors from the device.

This part is not necessary, but is interesting to confirm that the communication is working as expected, and that we are actually connected to a M519 or another SpringCard device using the same CCID over Serial protocol.

Reading the descriptor is documented here:

https://docs.springcard.com/books/SpringCore/Host_Protocols/CCID_Protocol_(PCSC)/Non_USB_Control/GET_DESCRIPTOR

The format of the descriptors is specified in [USB] and in [CCID] (and documented in the page mentioned above).

- Activate PC/SC operation.

By default, the device is idle to limit the power consumption when no host is ready to handle the arrival of a smart card. That's why the CCID_Start function must be invoked explicitly.

The CCID_Start function uses the SET CONFIGURATION command of the CCID over Serial protocol, as documented here:

https://docs.springcard.com/books/SpringCore/Host_Protocols/CCID_Protocol_(PCSC)/Non_USB_Control/SET_CONFIGURATION

The option byte tells the M519 whether the host application supports Notifications, or not *(more on that in § 5.4)*.

- Read the slot count from the device.

This is also optional; if we want to use only the contactless slot, we can consider that the M519 has only one slot.

Reading the slot count is done using SCardControl ($_H$58 $_H$20 $_H$80). See this page for reference:

https://docs.springcard.com/books/SpringCore/Host_Protocols/Direct_Protocol/CONTROL_class/Queries/GET_DATA

- If this test is enabled in the parameters, run an “echo” test using the SCardControl function (ESCAPE in [CCID]).
- Wait for a smart card to be inserted in any slot.

If the Notifications have been enabled, the application waits using SCardGetStatusChange until a Notification arrives.

Otherwise, the application polls the slots using SCardStatus.

When a smart card arrives:

- Invoke the SCardConnect function to get access to the card and retrieve its ATR.
- Invoke SCardTransmit ($_H$FF $_H$CA $_H$00 $_H$00 $_H$00) which is the APDU defined by PC/SC to get the “serial number” (protocol-level ID) of the card (this works only with contactless cards).
- If this test is enabled in the parameters, run an other “echo” test using the SCardTransmit function (XFR BLOCK in [CCID]).
- Invoke SCardDisconnect to reset and release the card.

A real-world application would complete the card transaction (between SCardConnect and SCardDisconnect) by getting authenticated on the card, selecting directories or file, reading and/or writing data.

This may sound far from this basic example at first, but any transaction, even the most complex one, would rely on exchanging APDUs with the smart card through the SCardTransmit function (XFR BLOCK in [CCID]).

Therefore, once we have enabled the trio SCardConnect / SCardTransmit / SCardDisconnect, we have built the foundation to develop absolutely any smart card-aware application.

5 Porting the SDK to another platform

Porting the SDK starts by adjusting the HAL (hardware abstraction layer). The functions that you shall implement are the ones listed in file `ccid_hal.h`.

5.1 UART functions

The minimal HAL provides the CCID_SerialSendByte function to send one byte to the M519, and calls the CCID_SerialRecvByteFromISR callback when one byte has been received.

There are only a few auxiliary functions that may be simplified or removed:

- CCID_SerialSendBytes: unless you want to use a DMA or alike, this function is nothing more than a loop over CCID_SerialSendByte,
- CCID_SerialInit, CCID_SerialOpen: selecting and enabling the UART has to be done in your own code, before activating the CCID driver (with function CCID_Init). Your initialization code shall configure the UART for 38400bps, 8 data bits, 1 stop bit, no parity, no flow control. CCID_SerialClose is virtually useless,
- CCID_SerialIsOpen: on a MCU, could return TRUE all the time, unless a physical error (framing violation or overrun) has been encountered by the UART. In this case, your upper-level code is remains of reinitializing the UART before activating the CCID driver again.

5.2 Synchronization functions

A single execution context runs the PC/SC-aware application, the PC/SC-Like stack and the CCID driver. Only the CCID_SerialRecvByteFromISR callback runs from a different context, which is the ISR (interrupt vector) of the UART RX interrupt. When waiting for a message from the M519, the execution flow has to be stopped until a complete buffer has arrived. This is done using the CCID_WaitWakeup function.

In turn, the CCID_SerialRecvByteFromISR callback will use the CCID_WakeupFromISR callback when an incoming message has been received, and expects that this callback resumes the execution flow.

The HAL shall then provide:

- CCID_WaitWakeup: this functions takes a timeout (in millisecond) as single parameter. It shall block until a message arrives or the timeout occurs, and return accordingly (TRUE if a message has been received, FALSE in case of a timeout);
- CCID_WakeupFromISR: this callback shall unblock the execution flow;
- CCID_ClearWakeup: this function shall reset the unblock state.

A minimal implementation could be done using a volatile flag and a free running timer with a 1-millisecond interval or a busy-wait function:

```
static volatile BOOL fWakeup;

BOOL CCID_WaitWakeup(DWORD timeout_ms)
{
	while (!fWakeup)
	{
		if (timeout_ms != (DWORD) -1) /* (DWORD) -1 is INFINITE */
			if (timeout_ms-- == 0) return FALSE;
		sleep_ms(1);
	}
	return TRUE;
}

void CCID_WakeupFromISR(void)
{
	fWakeup = TRUE;
}

void CCID_ClearWakeup(void)
{
	fWakeup = FALSE;
}
```

(where sleep_ms shall stop the execution for the specified number of milliseconds)

Of course if you have an OS or a real-time kernel, using system events would be definitively better.

5.3 Test suite

File `pcsc-serial-samples.c` provides everything implementers need to validate the behaviour of the PC/SC-Like stack and CCID driver after porting them to their own target.

Set fTestEchoControl and fTestEchoTransmit to TRUE to enable the tests. They both rely on this feature of the M519:

https://docs.springcard.com/books/SpringCore/PCSC_Operation/APDU_Interpreter/Vendor_instructions/ECHO

5.3.1 Timing considerations

The role of the SCardTransmit function (XFR BLOCK in [CCID]) is to send a command (C-APDU) to the smart card and to receive its response (R-APDU).

Generally speaking, the card is free to take any time it wants before answering. For instance, generating a RSA private key in a SAM or a secure element usually takes about 1 minute. Even some contactless card, like passports or eID cards, claim in their protocol bytes that they must be allowed up to 5 seconds to complete some operations.

If the host application opened a timeout-window of 60 seconds, or even 5 seconds, error detection and recovery would be dramatically slow. For this reason, the M519 always replies in 1 second or less. If the module is busy with a long card exchange, the reply is a time extension message. The host application shall allow a slightly longer timeout-window (our example uses 1200ms) and keep waiting for replies until the exchange with the card comes to an end.

Some complexity comes from the lack of control flow: the message that contains the response from the card may arrive “immediately” after the last time extension message. In this situation, a slow host is likely to be unable to process the time extension message before the next message arrives. To overcome this issue, a double-buffer structure is implemented in the CCID driver (see comments in `ccid_serial_receiver.c` for details).

The test_echo_transmit function performs an exhaustive test of this part, using the M519's ECHO ADPU through SCardTransmit to introduce a variable delay, up to 60 seconds. The test_echo_control function does the same through SCardControl[2].

NB: it shall also be noticed that there is no way for the host to interrupt a pending exchange between the M519 and the card. Asserting the /RESET pin of module is the only way to abort a long exchange.

5.3.2 Size considerations

According to ISO/IEC 7816-4, the longest short APDU that is possible is a case 4 (L_E provided) C-APDU having L_C = 255. This lead to a C-APDU size of 4 (CLA INS P1 P2) + 1 (L_C) + 255 (data) + 1 (L_E), i.e. 261 bytes. The longest R-APDU is 256 (data) + 2 (SW), i.e. 258 bytes.

The buffers of the CCID drivers are aligned with this limit of 261 bytes. If your application needs to use extended APDUs, increase the constant CCID_MAX_PAYLOAD_LENGTH in file pcsc-serial.h accordingly.

On the over hand, if you know for sure that the smart cards used by your application are limited to smaller APDUs[3], you may decrease the constant to reduce the RAM memory used by the driver.

5.4 Notification system

USB is a full-duplex communication bus, where the concept of endpoints makes it very easy to multiplex communication streams (more or less like the concept of sockets in TCP/IP). CCID, that is a protocol designed for USB, takes benefit of this concept by using one pair of endpoint for the main master/slave communication flow between the host and

2 This may appear very strange to send APDUs through SCardControl. Furthermore, the code does not even verify that there is a card in the slot before sending these APDUs... This is only possible because the ECHO APDU is internally wired to be used like that. No other APDU will behave the same.

3 For instance NXP Desfire EV0 and EV1 cards will never send, nor receive, an APDU that is larger than 64 bytes.

the device (Bulk Out & Bulk In endpoints) and another, virtually independent, endpoint, to let the device notify the host that its status has changed (Interrupt endpoint). A USB CCID device should send a Notification through the Interrupt endpoint every time a smart card is inserted or removed from one of the slots.

Plain-old serial communication does not have an equivalent concept; therefore, enabling device-to-host Notifications is not always possible, and, when possible, it may introduce too much complexity for a low-end host.

Not possible: if the serial link is based on RS-485, the communication medium is half-duplex (not full-duplex as it is with RS-232 or "RS-TTL"). In this situation, the Interrupt message coming from the M519 is likely to collide with a command coming from the host.

Too complex: the M519 may send a Notification message anytime. This could be while the host is transmitting a command, or immediately before or after sending a response. The host must therefore

- be listening all the time,
- have two buffers, to be able to receive at least two messages one after the other; the implementation shall also be able to handle them independently of the order they arrive[4].

If you are in one of these situation (RS-485 or low-end host), do not enable the Notifications when invoking the CCID_Start function (this is a flag in the SET CONFIGURATION message sent to the M519). In our sample application, the flag fCcidUseNotifications controls this behaviour.

If the Notifications are not enabled, the application must poll the device with SCardStatus instead of waiting for a Notification with SCardGetStatusChange.

4 In the implementation provided in the SDK, the CCID_Exchange function discards all Interrupt messages that arrive while waiting for a Bulk response. This is not much an issue with a single slot device since the status of the slot is also provided in the Bulk response itself, but on a multi-slot device, the application will remain blind to the changes concerning the other slots that the one in use.

6 Going further

The goal of the present document is only to let the developer get a basic system onto which he may start developing a smart-card aware application using the M519 as a Serial “PC/SC-Like” coupler.

Generally speaking, going further means:

1. Learning to use the PC/SC API — This is not complicated, and the sample application covers everything.

and

2. Learning how to use the smart card(s) that is(are) involved in the project.

Point 2 is virtually unlimited. The M519 may communicate with any contact or contactless smart cards, as well as with proprietary contactless chips (like the NXP Mifare family), RFID labels, NFC tags, remote applications running in NFC mobile phones, etc. How the card has to be used, how the transaction with the card should be implemented by the application developer is far outside the scope of this document. Always refer to the documentation of the card or NFC application that must be provided to you by the manufacturer or developer of the card or application.

Consider [PNA23207] as the reference for going deeper with PC/SC and the M519. Even though this application notes and its related SDK target USB applications, the concepts are the same and most of the source code can be adapted and reused in Serial applications.

Legal Information

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 SPRINGCARD and you. No information provided in this document shall be considered a substitute for your independent investigation. The information provided in the 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 SPRINGCARD 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. SPRINGCARD reserves the right to change the information at any time without notice.

SPRINGCARD doesn't warrant any results derived from the use of the products described in this document. SPRINGCARD 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 products may result in personal injury. SPRINGCARD customers using or selling these products for use in such applications do so on their own risk and agree to fully indemnify SPRINGCARD for any damages resulting from such improper use or sale.

INFORMATION ABOUT THE BRAND

SPRINGCARD, the SPRINGCARD logo are registered trademarks of SPRINGCARD SAS. All other brand names, product names, or trademarks belong to their respective holders. Information in this document is subject to change without notice. Reproduction without written permission of SPRINGCARD is forbidden.

COPYRIGHT NOTICE

All information in this document is either public information or is the intellectual property of SPRINGCARD 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 these documents, and you are not allowed to publish or reproduce this document, either on the web or by any means, without written permission of SPRINGCARD.

EDITOR'S INFORMATION

SPRINGCARD SAS company with a capital of 227 000 €
RCS EVRY B 429 665 482
Parc Gutenberg, 2 voie La Cardon
91120 Palaiseau – FRANCE

CONTACT

For more information and to locate our sales office or distributor in your country or area, please visit **www.springcard.com**