



PMD19004-AG
DRAFT - PUBLIC

SPRINGCARD PC/SC-LIKE OVER BLE LIBRARY

Developer's Manual

DOCUMENT IDENTIFICATION

Category	Specification		
Family/Customer	CCID PC/SC Couplers		
Reference	PMD19004	Version	AG
Status	Draft	Classification	Public
Keywords	SpringCore, Puck, PC/SC, CCID, BLE, Android, iOS, Java, Kotlin, Objective C, Swift		
Abstract			

File name	V:\Dossiers\SpringCard\A-Notices\PCSC\PCSC over BLE\Mobiles\[PMD19004-AG] BLE CCID library for Mobiles-Developer Manual.odt		
Date saved	07/06/19	Date printed	

REVISION HISTORY

Ver.	Date	Author	Valid. by		Approv. by	Details
			Tech.	Qual.		
AA	07/01/19	JDA				Initial draft
AB	21/01/19	JDA				First official release
AC	23/01/19	JDA				Added the getting started pages for Android
AD	28/01/19	CRA				Specification of Android part is final
AE	18/02/19	HTH				Specification of iOs part is final
AF	25/02/19	CRA				Authentication specifications
AG	17/04/19					Sleep and wake-up

CONTENTS

1. INTRODUCTION.....	6	5.4. SECURE COMMUNICATION.....	32
1.1. ABSTRACT.....	6		
1.1.1. Context.....	6		
1.1.2. Document identification.....	6		
1.1.3. Architecture overview.....	7		
1.2. SUPPORTED PRODUCTS.....	7		
1.3. AUDIENCE.....	7		
1.4. SUPPORT AND UPDATES.....	8		
1.5. RELATED DOCUMENTS.....	8		
1.5.1. PC/SC standard.....	8		
1.5.2. Reference documents.....	8		
2. GETTING STARTED WITH THE LIBRARY.....	9		
2.1. FOR IOS.....	9		
2.1.1. Download the Library and the sample project.....	9		
2.1.2. Compile and test the sample project.....	9		
2.1.3. Use the Library in your own project.....	10		
2.2. FOR ANDROID.....	13		
2.2.1. Download the Library and the sample project.....	13		
2.2.2. Compile and test the sample project.....	13		
2.2.3. Use the Library in your own project.....	14		
2.3. LICENSE POLICY.....	18		
2.4. GETTING SUPPORT.....	19		
3. FLOWCHART OF A TYPICAL APPLICATION.....	20		
3.1. OPEN A CONNECTION TO A SPRINGCARD BLE SMARTCARD READER.....	20		
3.2. OPEN A CONNECTION TO A SMARTCARD.....	21		
3.3. PERFORM THE TRANSACTION WITH THE SMARTCARD.....	22		
3.4. DISCONNECT FROM THE SPRINGCARD BLE SMARTCARD READER.....	23		
3.5. THE CARD'S STATE-MACHINE.....	23		
4. PER-SYSTEM REFERENCE PAGE.....	24		
4.1. IOS IMPLEMENTATION.....	24		
4.1.1. Application-driven actions and callbacks.....	24		
4.1.2. Callback fired following a device-initiated event.....	25		
4.1.3. Error handling.....	26		
4.2. ANDROID IMPLEMENTATION.....	27		
4.2.1. Application-driven actions.....	27		
4.2.2. Callback invoked on device-initiated events.....	28		
4.2.3. Error handling.....	29		
5. ADVANCED FEATURES.....	30		
5.1. PROPERTIES AND CONSTANTS OF THE READERLIST OBJECT.....	30		
5.1.1. Static properties (constants).....	30		
5.1.2. Dynamic properties (from the BLE device).....	30		
5.2. CONTROL METHODS.....	31		
5.2.1. iOS implementation.....	31		
5.2.2. Android implementation.....	31		
5.3. READ DEVICE POWER STATE & BATTERY LEVEL.....	32		
5.3.1. iOS implementation.....	32		
5.3.2. Android implementation.....	32		

1. INTRODUCTION

1.1. ABSTRACT

1.1.1. Context

SpringCard offers a wide range of contactless (NFC/RFID @13.56MHz) couplers and of contact (smartcard) couplers. All these devices are designed with PC/SC compliance in mind. PC/SC, short for “personal computer / smartcard” is a standard that eases the developer’s job by hiding most of the specificities of any given smartcard reader behind a high-level API, a complex middleware architecture and an interoperable device driver. Unfortunately, this high-end approach is virtually limited to the Windows and Linux worlds, and to USB-based devices.

When it comes to mobile development (Android, iOS) and to smartcard readers that are Bluetooth Smart devices (or BLE, Bluetooth Low Energy), it is more efficient to operate them directly.

SpringCard contactless or contact couplers using BLE as main communication channel are not actually “PC/SC compliant” for they don’t come with an interoperable device driver, yet they are still designed with the standard in mind, to ease the developer’s job.

More than that, **SpringCard** offers a software Library to operate these BLE devices through a high-level API, that provides a counterpart for all key PC/SC functions (SCardListReaders, SCardConnect, SCardTransmit, SCardDisconnect).

1.1.2. Document identification

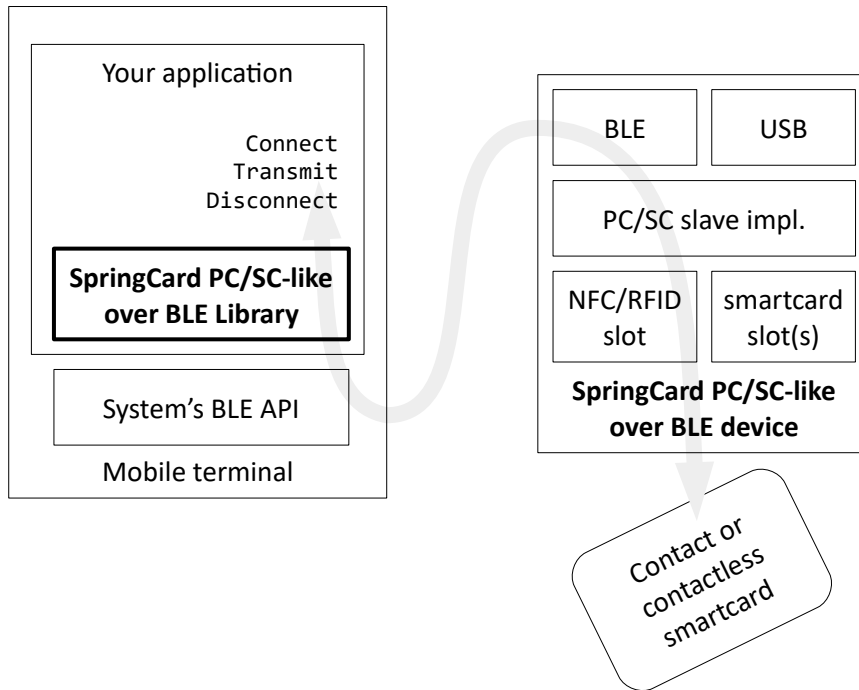
This document is the **Developer’s Manual** of the **SpringCard PC/SC-like over BLE Library**, targeting **iOS** and **Android** systems.

It shall be considered as an introduction and an overall guide when using this Library; the detailed documentation of the API is generated from its source code thanks to Doxygen, and only made online at:

- <https://docs.springcard.com/apis/iOS/PCSC-Like/> for the iOS Library,
- <https://docs.springcard.com/apis/Android/PCSC-Like/> for the Android Library.

Note: *developers who do not intend to use the Library and aim to work with the BLE device directly shall refer to doc. PMD15282 “SpringCard PC/SC couplers – Zero-driver CCID low-level implementation” instead.*

1.1.3. Architecture overview



This document covers the integration and the use the **SpringCard PC/SC-like over BLE Library** (bold box) into a mobile application.

1.2. SUPPORTED PRODUCTS

At the date of writing, the products covered by this document are:

Device name	Description	Platform
SpringCard Puck Blue	Desktop USB+BLE contactless coupler	SpringCore'18
AF Care dual	Mobile dual-slot contact coupler	SpringCore'18

1.3. AUDIENCE

This manual is designed for use by application developers. It assumes that the reader has expert knowledge of computer development and a basic knowledge of PC/SC, of the ISO 7816-4 standard for smart-cards, and of the NFC Forum's specifications for contactless cards.

Note: *Beginners are advised to read doc. PMD17041 "Smartcards and contactless smartcards – Integrator's and Implementer's Guide" as an introduction.*

1.4. SUPPORT AND UPDATES

Useful related materials (product datasheets, application notes, sample software, HOWTOs and FAQs...) are available at SpringCard's web site:

www.springcard.com

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

For technical support enquiries, please refer to SpringCard support page, on the web at

www.springcard.com/support

1.5. RELATED DOCUMENTS

1.5.1. PC/SC standard

Reference	Publisher	Title
PC/SC	PC/SC Workgroup	Interoperability Specification for ICCs and Personal Computer Systems Revision 2 Download link: https://www.pcscworkgroup.com/specifications/download/

1.5.2. Reference documents

Reference	Publisher	Title
PMD17041	SpringCard	Smartcards and contactless smartcards Integrator's and Implementer's Guide
PMD15282	SpringCard	PC/SC couplers Zero-driver CCID low-level implementation

2. GETTING STARTED WITH THE LIBRARY

2.1. FOR iOS

2.1.1. Download the Library and the sample project

The sample iOS project can be download on Github:

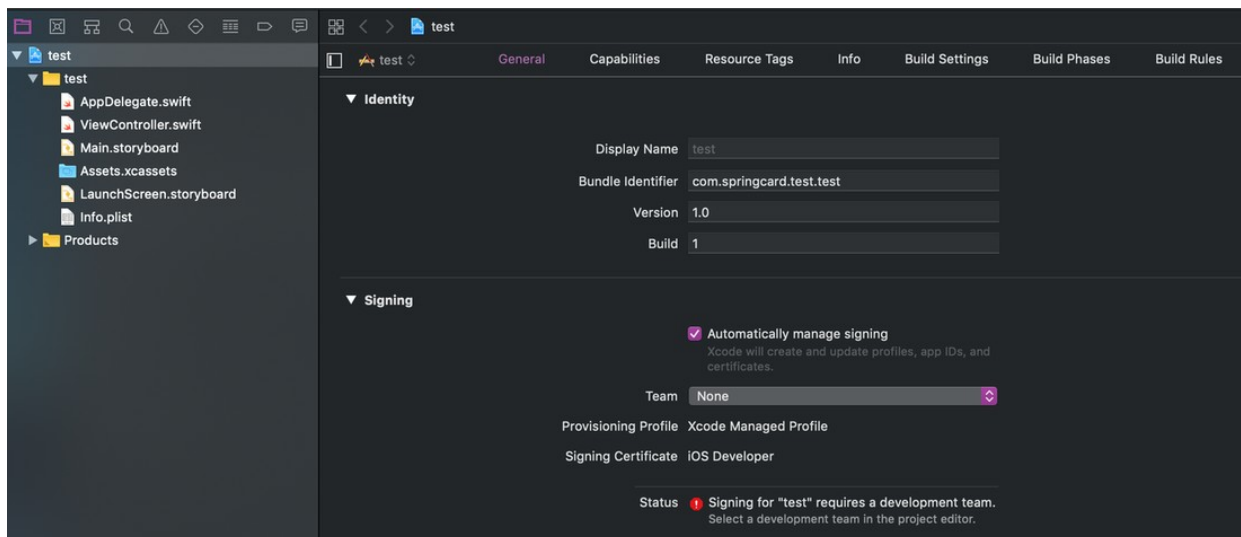
<https://github.com/springcard/ios-pcsclike-sample-ble>

And the library can be found here:

<https://github.com/springcard/ios-pcsclike>

2.1.2. Compile and test the sample project

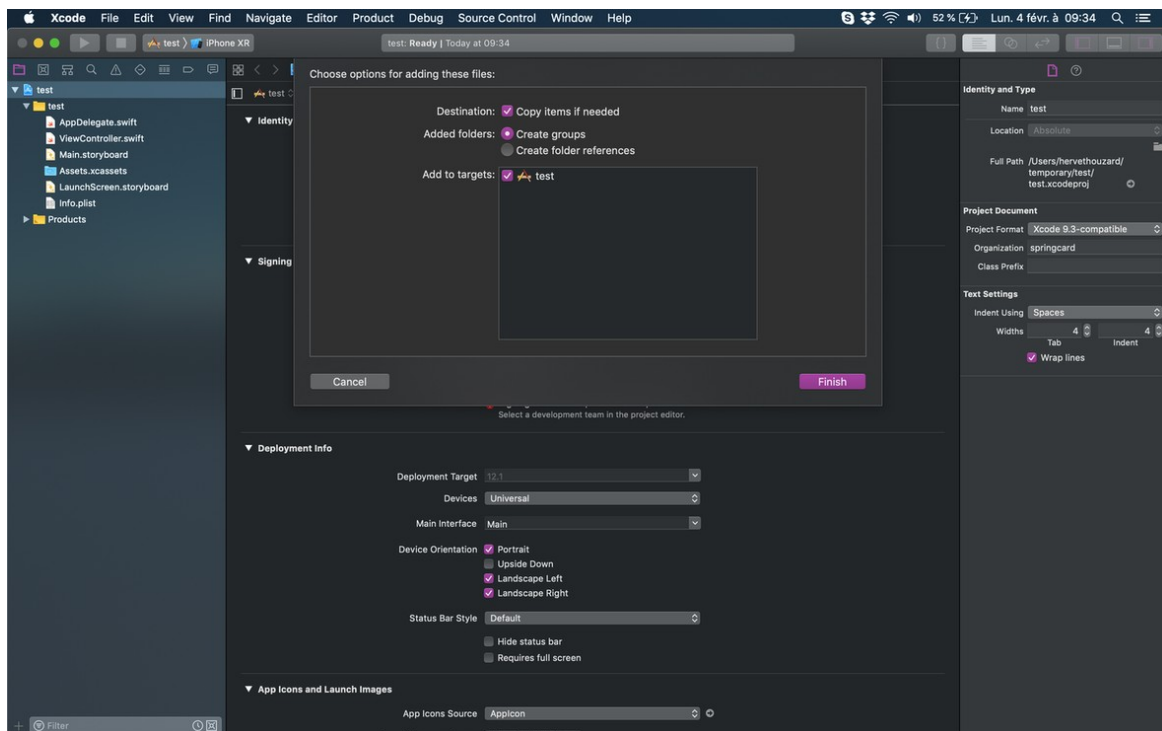
- From the **Finder** locate the project and double click on **test.xcodeproj** it will open Xcode with the project,
- Change the **Signing** part to use your own identifier,



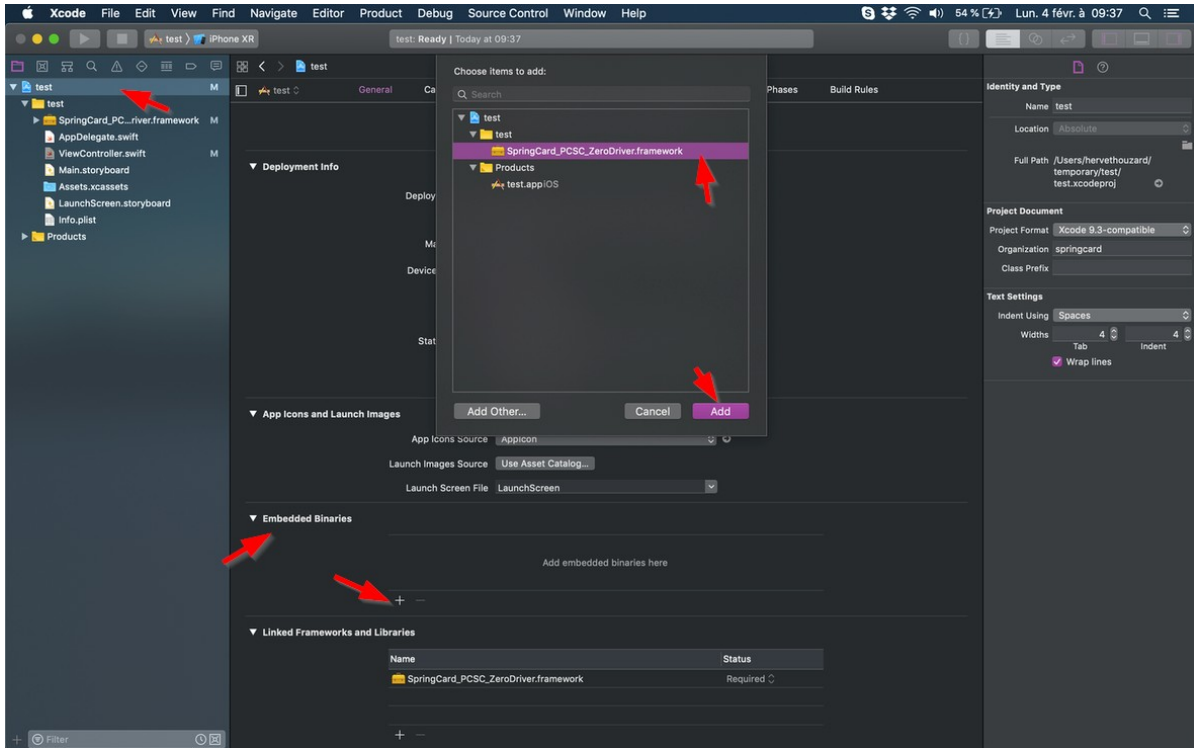
- Go in the **Build** menu, select **Clean Build Folder** then, in the same menu **Build**,
- Connect an iPhone or an iPad to the computer, then launch the project.

2.1.3. Use the Library in your own project

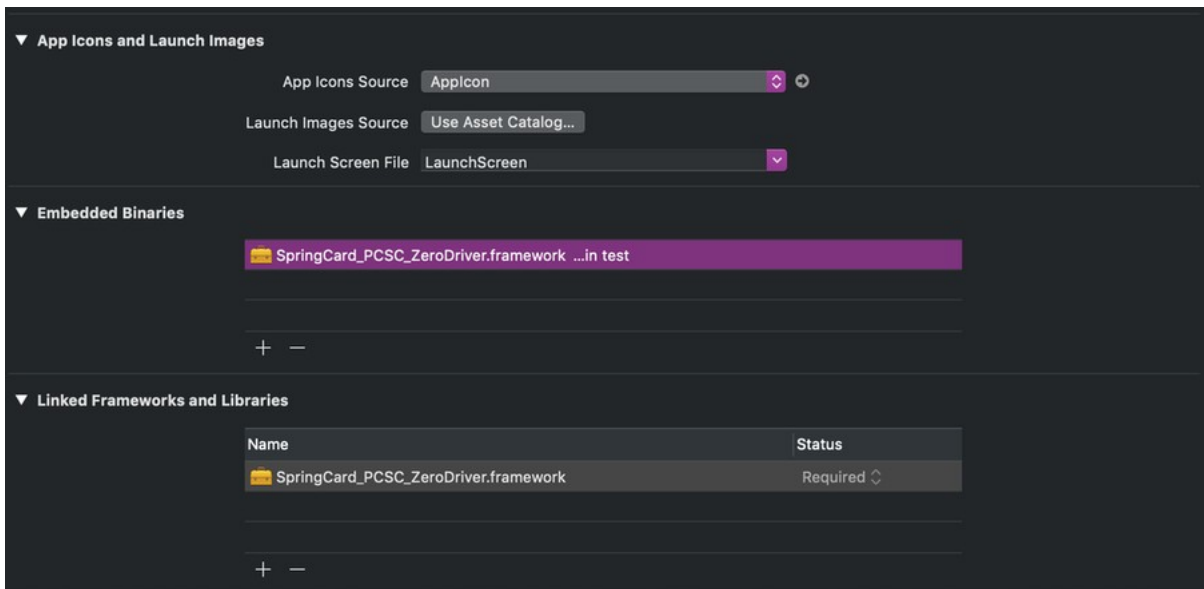
- Create a project with Xcode or open an existing one,
- From the Finder locate the file *SpringCard_PCSC_ZeroDriver.framework*,
- Drag and drop it inside your project's directory (inside Xcode),
- A new window will open, select the required options:



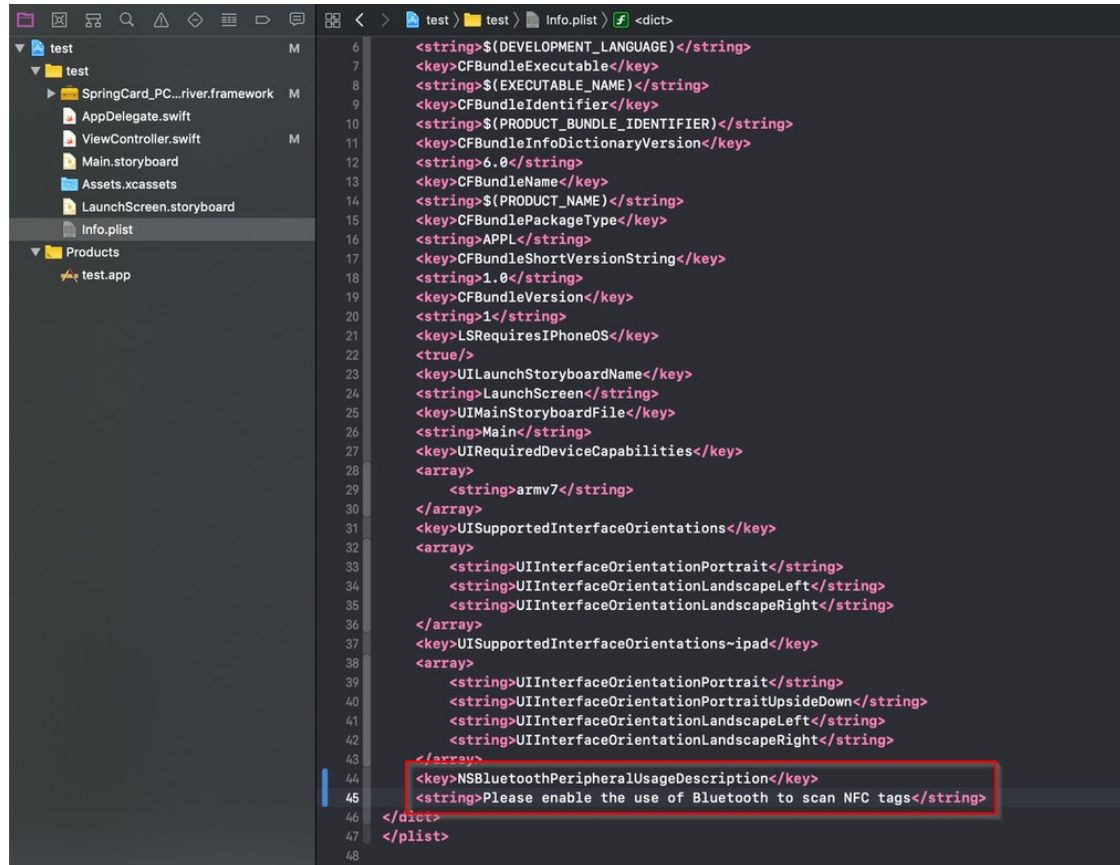
- Select the project and its target, go in the **Embedded Binaries** section, click on **+** then select the framework and click **Add**:



- The project shall now look like this:



- To use the Library from your source code, add the following import in your Controller(s):
import SpringCard_PcSc_Like
- Don't forget to update the project's **info.plist** file to add the required key to allow the application to use Bluetooth:



```
6 <string>$(DEVELOPMENT_LANGUAGE)</string>
7 <key>CFBundleExecutable</key>
8 <string>$(EXECUTABLE_NAME)</string>
9 <key>CFBundleIdentifier</key>
10 <string>$(PRODUCT_BUNDLE_IDENTIFIER)</string>
11 <key>CFBundleInfoDictionaryVersion</key>
12 <string>6.0</string>
13 <key>CFBundleName</key>
14 <string>$(PRODUCT_NAME)</string>
15 <key>CFBundlePackageType</key>
16 <string>APPL</string>
17 <key>CFBundleShortVersionString</key>
18 <string>1.0</string>
19 <key>CFBundleVersion</key>
20 <string>1</string>
21 <key>LSRequiresIPhoneOS</key>
22 <true/>
23 <key>UILaunchStoryboardName</key>
24 <string>LaunchScreen</string>
25 <key>UIMainStoryboardFile</key>
26 <string>Main</string>
27 <key>UIRequiredDeviceCapabilities</key>
28 <array>
29 <string>armv7</string>
30 </array>
31 <key>UISupportedInterfaceOrientations</key>
32 <array>
33 <string>UIInterfaceOrientationPortrait</string>
34 <string>UIInterfaceOrientationLandscapeLeft</string>
35 <string>UIInterfaceOrientationLandscapeRight</string>
36 </array>
37 <key>UISupportedInterfaceOrientations~ipad</key>
38 <array>
39 <string>UIInterfaceOrientationPortrait</string>
40 <string>UIInterfaceOrientationPortraitUpsideDown</string>
41 <string>UIInterfaceOrientationLandscapeLeft</string>
42 <string>UIInterfaceOrientationLandscapeRight</string>
43 </array>
44 <key>NSBluetoothPeripheralUsageDescription</key>
45 <string>Please enable the use of Bluetooth to scan NFC tags</string>
46 </dict>
47 </plist>
48
```

2.2. FOR ANDROID

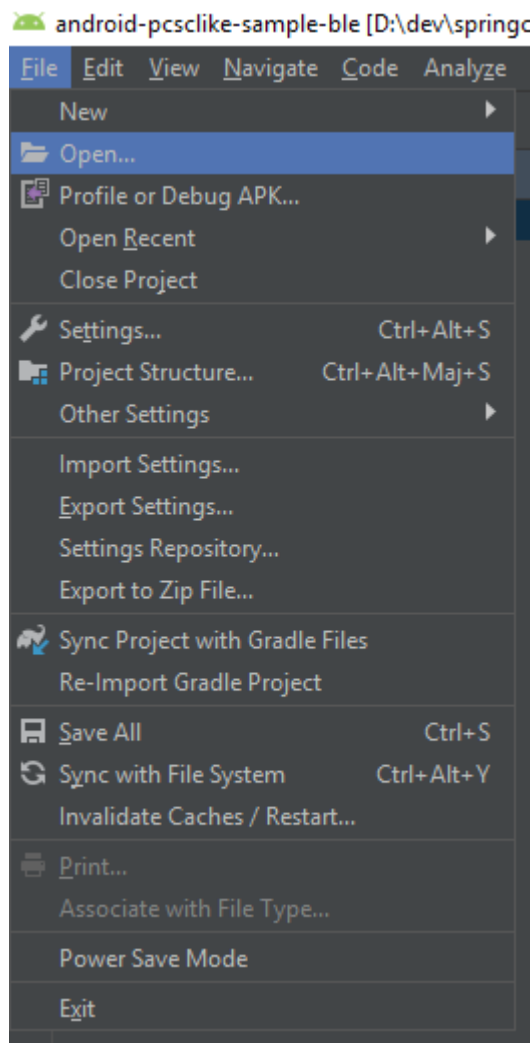
2.2.1. Download the Library and the sample project

The sample Android project and library source code can be downloaded on Github:

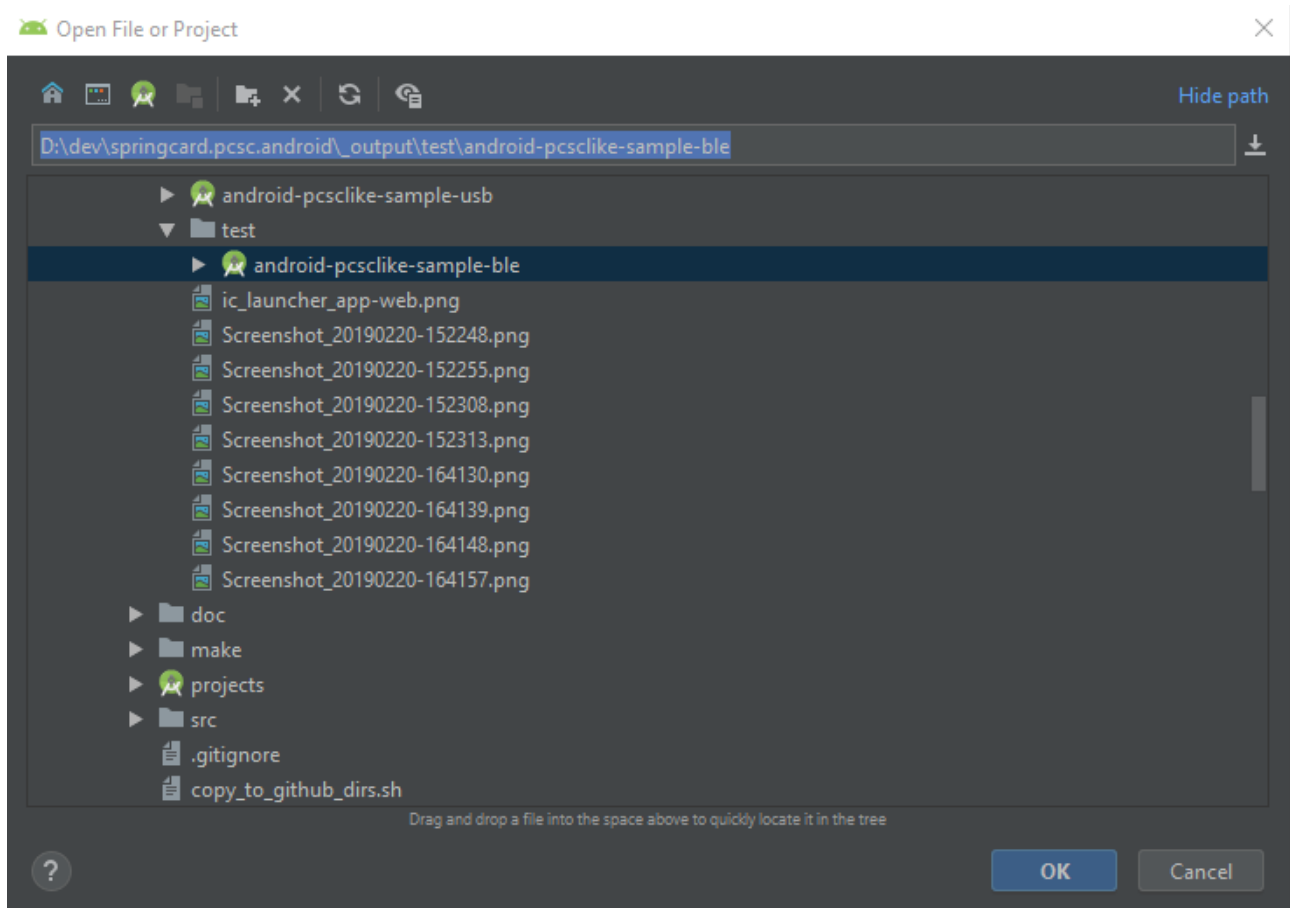
<https://github.com/springcard/android-pcslike>

2.2.2. Compile and test the sample project

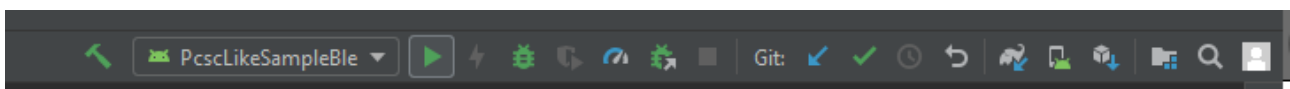
Once you got the source repository, open Android Studio, click on **File** then **Open...**



Then navigate to the directory where the code source of the project is located and click **Ok**.

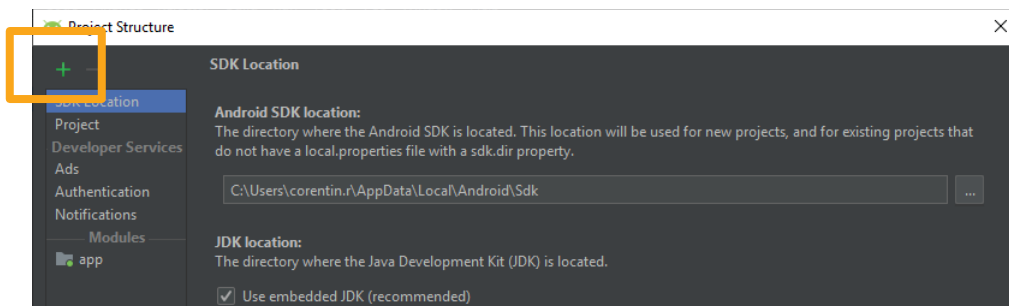


Once the project is opened and Gradle have synced, it's possible to launch the application by clicking on the green "Start" button (or press "**Maj+F10**")

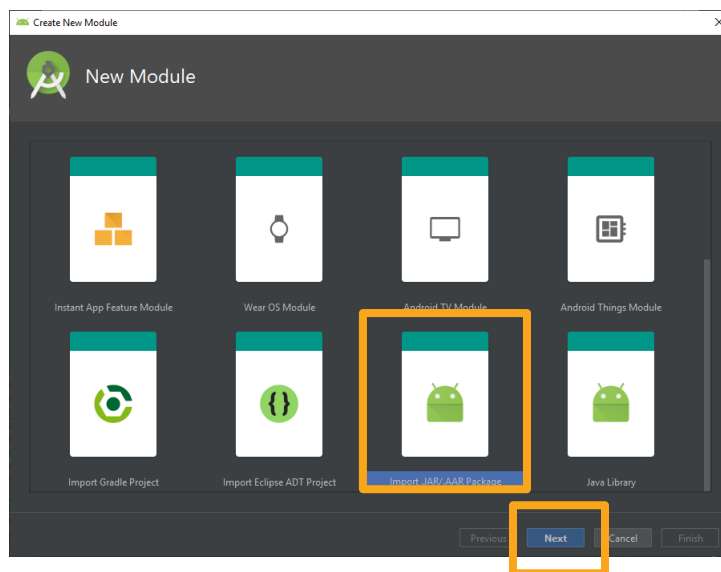


2.2.3. Use the Library in your own project

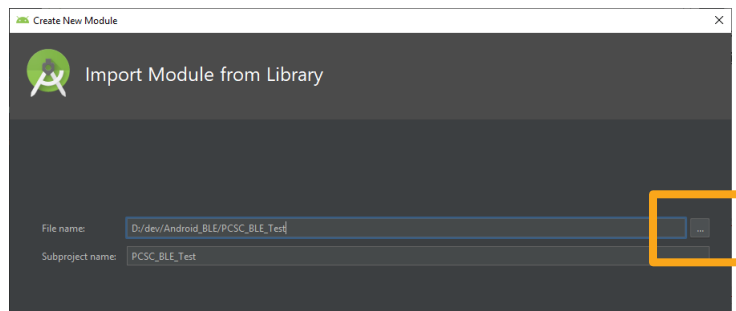
- Create a project with Android Studio or open an existing project.
- In the **File** menu, click **Project Structure**.
- In the **Project Structure** screen, click the green “+” button situated at the top-left corner, to add a new module.



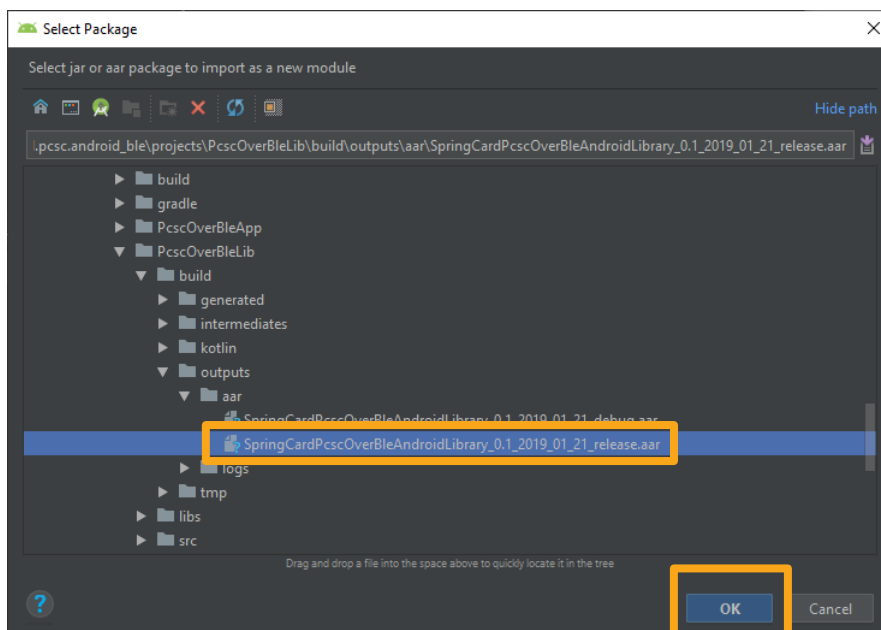
- In the **New Module** screen, choose **Import .JAR or .ARR Package** and click **Next**.



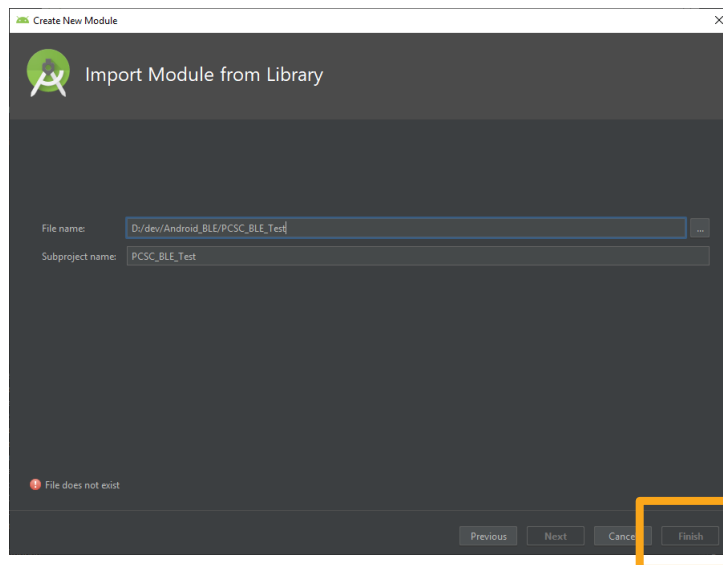
- In the *Import Module from Library* screen, click the “...” button near the *File name* field. Browse to locate the Library file.



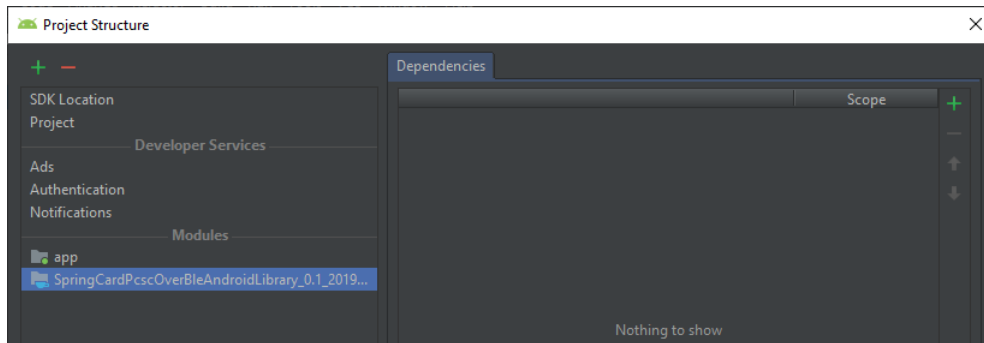
- Browse to locate and select the Library .AAR file, and click **Ok**.



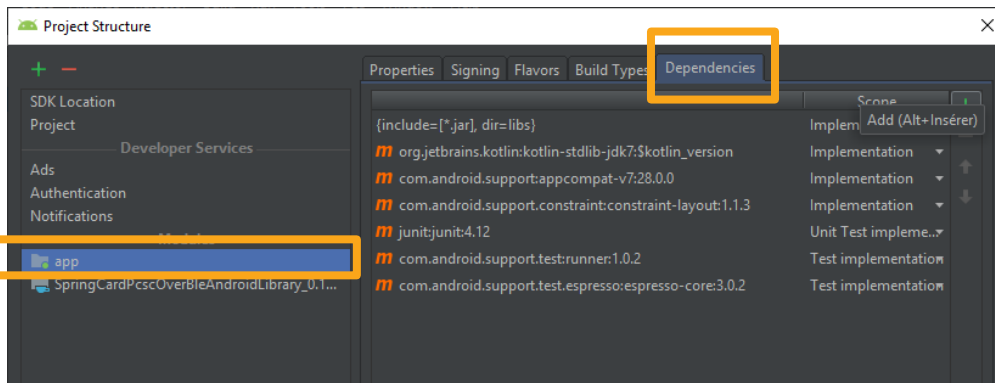
- Android Studio assigns the *Subproject name* field automatically. Click **Finish**.



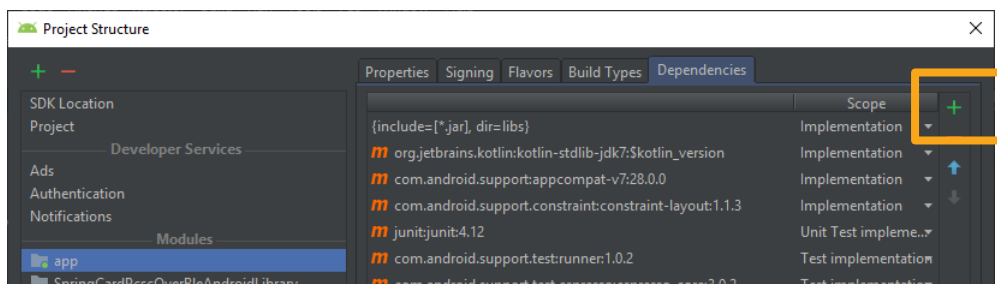
- Wait until Gradle synchronizes the solution.
- Back in the *Project Structure* screen, a new module has been added, representing the Library.



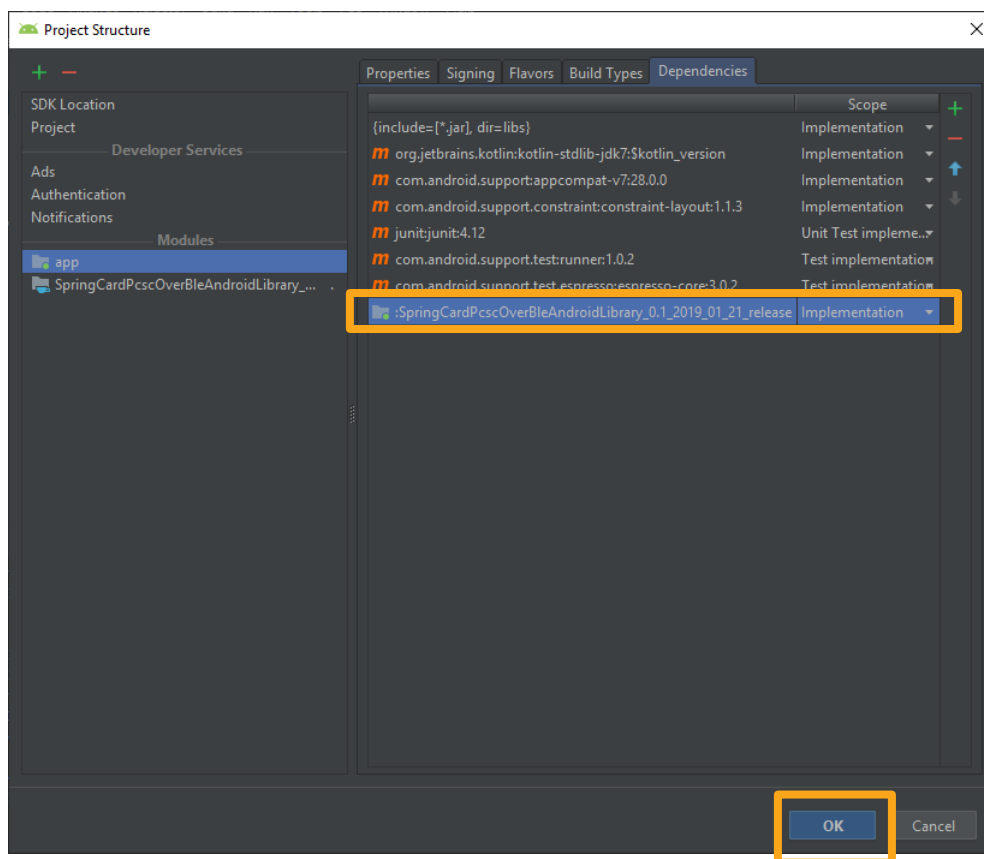
- Select the **app** module in the left navigation panel, and select the **Dependencies** tab in the right panel.



- In the *Dependencies* tab, click the green “+” button, to add a new dependency.



- Select the Library module and click **Ok**.



- Wait until Gradle synchronizes the solution.

To use the Library from your source code, add the following import in your modules:

```
import com.springcard.pcsclike.*
```

Warning, if you use the debug version of the library, you must use another plugin (aspectj).

You'll need to do the following modifications:

Add these lines to the build.gradle of the project:

```
repositories {  
    ...  
    mavenCentral()  
    ...  
}  
dependencies {  
    ...
```

```
classpath 'com.archinamon:android-gradle-aspectj:3.3.11'  
...  
}
```

Also, at the beginning of the build.gradle file of the modules using the library:

```
apply plugin: 'com.archinamon.aspectj'
```

2.3. LICENSE POLICY

On both platforms, the Library is made available in 3 forms:

- as a binary file without debug symbols (release version),
- as a binary file with debug symbols (debug version),
- as source code.

The **SpringCard SDK License**, reproduced below, grants you an unlimited right to redistribute the binary of the release version together with your application.

You shall not redistribute the binary of the debug version.

SPRINGCARD SOFTWARE DEVELOPMENT KIT (SDK) LICENSE AGREEMENT

This software is part of the SPRINGCARD SDK FOR PC/SC Redistribution and use in source (source code) and binary (object code) forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributed source code or object code shall be used only in conjunction with products (hardware devices) either manufactured, distributed or developed by SPRINGCARD,
2. Redistributed source code, either modified or un-modified, must retain the above copyright notice, this list of conditions and the disclaimer below,
3. Redistribution of any modified code must be clearly identified "Code derived from original SPRINGCARD copyrighted source code", with a description of the modification and the name of its author,
4. Redistributed object code must reproduce the above copyright notice, this list of conditions and the disclaimer below in the documentation and/or other materials provided with the distribution,
5. The name of SPRINGCARD may not be used to endorse or promote products derived from this software or in any other form without specific prior written permission from SPRINGCARD.

THIS SOFTWARE IS PROVIDED BY SPRINGCARD "AS IS".
SPRINGCARD SHALL NOT BE LIABLE FOR INFRINGEMENTS OF THIRD PARTIES RIGHTS BASED ON THIS SOFTWARE.
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
SPRINGCARD DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THIS SOFTWARE WILL MEET THE USER'S REQUIREMENTS OR THAT THE OPERATION OF IT WILL BE UNINTERRUPTED OR ERROR-FREE.
IN NO EVENT SHALL SPRINGCARD BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL

DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

2.4. GETTING SUPPORT

SpringCard provides a free support service over the Library and its use together with SpringCard products. The support team is reachable only through email at support@springcard.com.

In order to get an accurate and efficient help, please always identify precisely the host device (manufacturer, product name and version, operating system version), the development environment and language you are using and the **SpringCard** device you are working with.

If you are reporting an issue with a given device or think you have found a bug, please reproduce the issue with the debug version of the Library and send us the detailed execution log, pointing out the line(s) showing the issue. It is a good practice to have also the application log its execution flow and the encountered errors or exceptions.

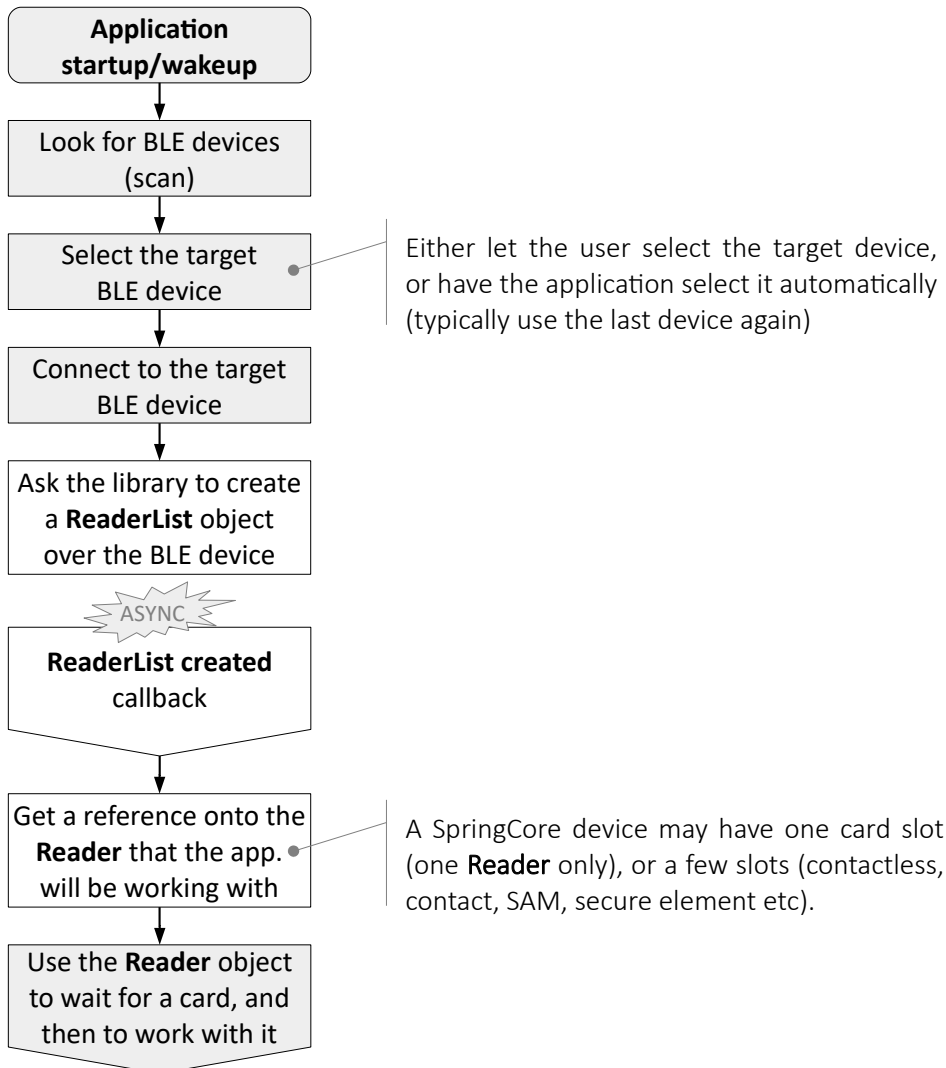
Sorry, but we are not able to provide assistance to an end-user or to a developer who is compiling his own version of the Library instead of using the provided binaries.

3. FLOWCHART OF A TYPICAL APPLICATION

3.1. OPEN A CONNECTION TO A SPRINGCARD BLE SMARTCARD READER

In this first step the application shall

1. Enumerate the available BLE devices and initiate a BLE connection with the target device,
2. Initialize the **SpringCard PC/SC-like over BLE Library** onto the said device, and get an access to the target smartcard reader (slot).

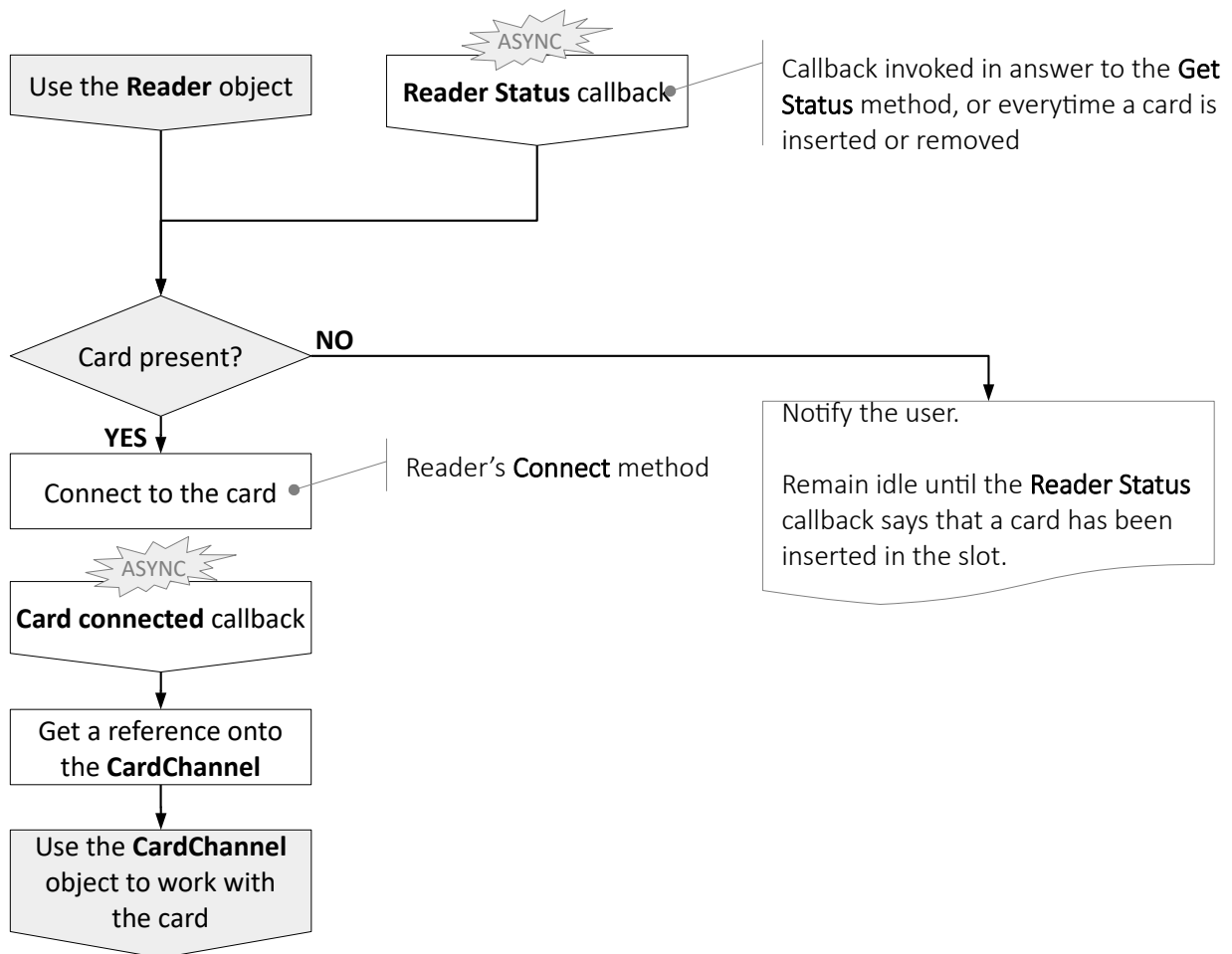


Note: *in this diagram and the following, the greyed boxes refer to some actions/code that has no direct relationship with the Library. White boxes show the interactions with the Library.*

3.2. OPEN A CONNECTION TO A SMARTCARD

In this second step the application shall

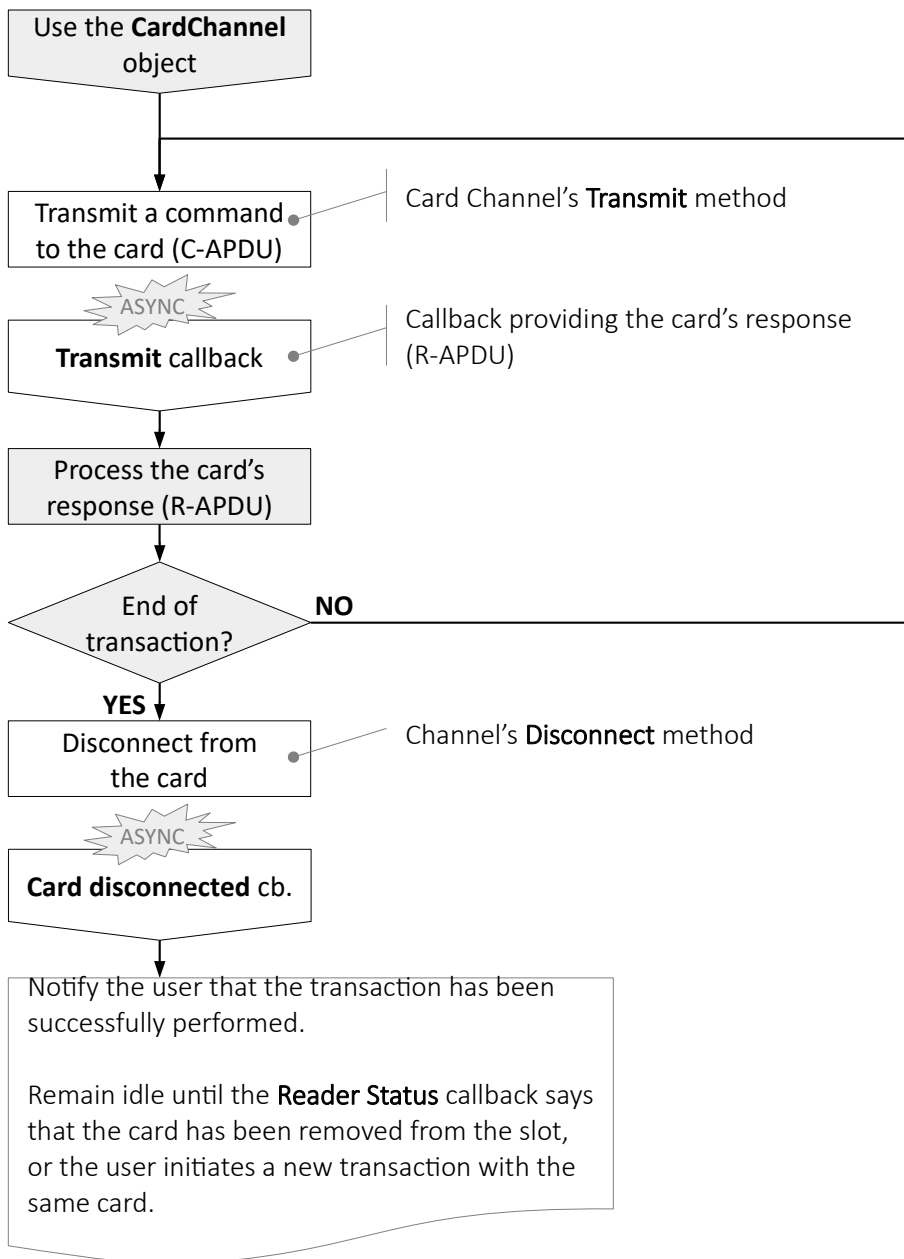
1. Verify that a smart-card is present in the reader, or wait until then,
2. Power up the smart-card and get a communication channel with it.



3.3. PERFORM THE TRANSACTION WITH THE SMARTCARD

In this last step, the application runs its key algorithm by sending commands (C-APDU) to the card, and processing its responses (R-APDU).

At the end of the algorithm, the application shall typically disconnect from the card, so the reader may power it down (at least the card is reset to clean up its transaction context).



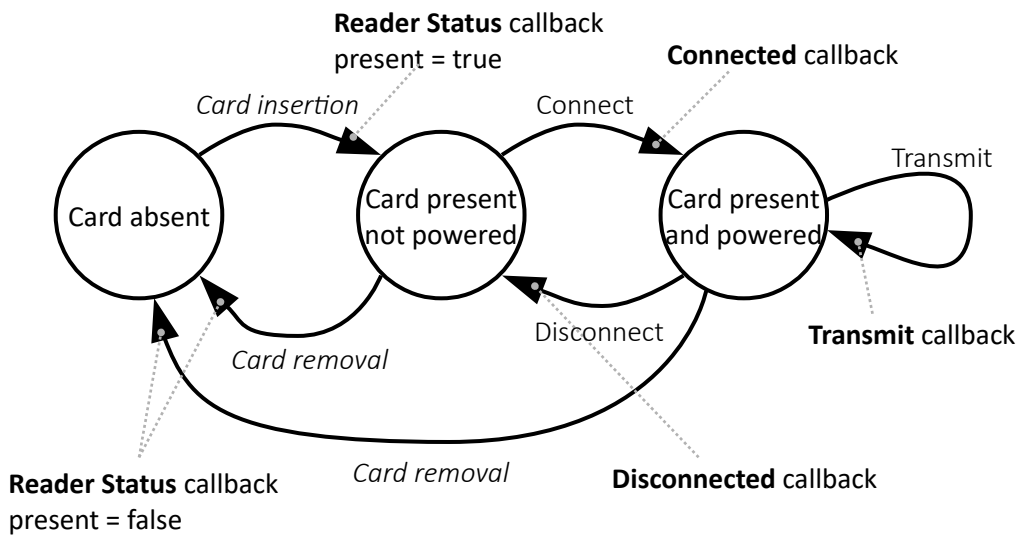
3.4. DISCONNECT FROM THE SPRINGCARD BLE SMARTCARD READER

The application should disconnect from the device as soon as the user does not need/want to use smartcards anymore. This let the device power down the smartcards (if some are still present) and to enter a low-power mode, until the application initiates a connection again.

The application may also instruct the device to shut-down.

3.5. THE CARD'S STATE-MACHINE

The following state machine should be observed to understand when the callbacks are fired.



4. PER-SYSTEM REFERENCE PAGE

4.1. IOS IMPLEMENTATION

This chapter lists the API's objects, methods and callbacks, but does not provide any detail regarding their parameters and usage precautions.

Please refer to the reference documentation, available online at:

<https://docs.springcard.com/apis/iOS/PcscLikeOverBle/>

4.1.1. Application-driven actions and callbacks

Type	Method or callback	Remark
Ask the library to create a ReaderList object over the BLE device		
async method	<code>SCardReaderList.create(...)</code>	Counterpart to PC/SC's SCardListReaders
async callback	<code>onReaderListDidCreate(readers, error)</code>	
Enumerate the readers in the ReaderList object		
property : int	<code>readers.slotCount</code>	Number of slots
property : string[]	<code>readers.slots[]</code>	Name of every slot
Get a reference onto one reader (i.e. a slot of a multi-slot device)		
method	<code>reader = readers.getReader(slot)</code>	slot could be either the index or the name
Query the status of the reader		
property: bool	<code>reader.cardPresent()</code>	Counterpart to PC/SC's SCardStatus
property: bool	<code>reader.cardPowered()</code>	
Connect to the card (power up + open a communication channel with the card)		
async method	<code>reader.cardConnect()</code>	Counterpart to PC/SC's SCardConnect
async callback	<code>onCardDidConnect(channel, error)</code>	
property : bytes[]	<code>channel.atr</code>	The ATR of the card

Transmit a C-APDU to the card, receive a R-APDU in response

async method	channel.transmit(command)	Counterpart to PC/SC's SCardTransmit
async callback	onTransmitDidResponse(channel, response, error)	

Disconnect from the card (close the communication channel + power down)

async method	channel.cardDisconnect()	Counterpart to PC/SC's SCardDisconnect
async callback	onCardDidDisconnect(channel)	

Connect to the card again (re-open an existing communication channel)

async method	channel.cardReconnect()	Counterpart to PC/SC's SCardReconnect
async callback	onCardDidConnect(channel)	Same as after reader.cardConnect()

Disconnect from the BLE device and release the library

async method	readers.close()	
async callback	onReaderListDidClose(readers)	

4.1.2. Callback fired following a device-initiated event

<i>Type</i>	<i>Callback</i>	<i>Remark</i>
The BLE device has been lost		
async callback	onReaderListDidClose(readers)	Same callback as after readers.close()
A card is inserted into, or removed from an active reader		
async callback	onReaderStatus(reader, present, powered)	Counterpart to PC/SC's SCardGetStatusChange
The reader fails to connect (or reconnect) to the card		
async callback	onCardDidConnect(channel, error)	The error object is not null in case of an error.

The card is removed during a communication

async callback

```
onTransmitDidResponse(channel,  
response, error)
```

The **error** object is not null is case of an error. The `onReaderStatus()` callback fires afterwards to notify that the card has been removed.

4.1.3. Error handling

On iOS, all the callbacks provide an **error**: **Error** parameter.

If error is `nil`, the execution is successful.

If error is not `nil`, the execution has failed. The error object contains the reason of the error. The application shall notify the user and, given the context and the reason, take appropriate decisions to either retry the last action, or to report that it is not able to go any further.

The error's domain is set to **Bundle.main.bundleIdentifier** and you can use the following properties:

- **code**
- **debugDescription**

4.2. ANDROID IMPLEMENTATION

This chapter lists the API's objects, methods and callbacks, but does not provide any detail regarding their parameters and usage precautions.

Please refer to the reference documentation, available online at:

<https://docs.springcard.com/apis/Android/PcscLikeOverBle/>

4.2.1. Application-driven actions

Type	Method or callback	Remark
Ask the library to create a ReaderList object over the BLE device		
async method	<code>SCardReaderList.create(...)</code>	Counterpart to PC/SC's SCardListReaders
async callback (success)	<code>onReaderListCreated(readers)</code>	
Enumerate the readers in the ReaderList object		
property: int	<code>readers.slotCount</code>	Number of slots
property: string[]	<code>readers.slots[]</code>	Name of every slot
Get a reference onto one reader (i.e. a slot of a multi-slot device)		
method	<code>reader = readers.getReader(slot)</code>	slot could be either the index or the name
Query the status of the reader		
property: bool	<code>reader.cardPresent</code>	Card present in slot
property: bool	<code>reader.cardPowered</code>	Card powered by the reader
property: bool	<code>reader.cardConnected</code>	Card connected (ICC Pwr On) or disconnected (ICC Pwr Off) or card gone
Connect to the card (power up + open a communication channel with the card)		
async method	<code>reader.cardConnect()</code>	Counterpart to PC/SC's SCardConnect
async callback	<code>onCardConnected(channel)</code>	
property: bytes[]	<code>channel.atr</code>	The ATR of the card

Get info on the reader

property: string	reader.name	Name of the slot
property: int	reader.index	Index of the slot in the SCardReaderList
property: SCardReaderList	reader.parent	Point to the SCardReaderList parent
property: SCardChannel	reader.channel	Get the channel associated to this slot

Transmit a C-APDU to the card, receive a R-APDU in response

async method	channel.transmit(command)	Counterpart to PC/SC's SCardTransmit
async callback	onTransmitResponse(channel, response)	

Disconnect from the card (close the communication channel + power down)

async method	channel.disconnect()	Counterpart to PC/SC's SCardDisconnect
async callback	onCardDisconnected(channel)	

Connect to the card again (re-open an existing communication channel)

async method	channel.reconnect()	Counterpart to PC/SC's SCardReconnect
async callback	onCardConnected(channel)	Same as after reader.cardConnect()

Disconnect from the BLE device and release the library

async method	readers.close()	
async callback	onReaderListClosed(readers)	

4.2.2. Callback invoked on device-initiated events

Type	Callback	Remark
------	----------	--------

The BLE device has been lost

async callback	onReaderListError(readers, error)	
----------------	--	--

A card is inserted into, or removed from an active reader

async callback	onReaderStatus(reader, present, connected)	Counterpart to PC/SC's SCardGetStatusChange
----------------	---	---

The card is removed during a connect or a transmit

async callback

`onReaderOrCardError(readerOrCard,
error)`

The `onReaderStatus()` callback fires afterwards to notify that the card has been removed.

4.2.3. Error handling

On Android, every async method may raise either a success callback or an error callback.

There are two error callbacks:

- `onReaderListError(readerOrCard, error)` is invoked for all device-level errors, e.g. BLE error, protocol error, etc. When this callback is invoked, the connection to the device is closed.
- `onReaderOrCardError(readers, error)` is invoked for all “recoverable” errors, e.g. invalid slot number, card absent, card removed or mute, etc. When this callback is invoked, the connection to the device is kept open.

In both callbacks, the error object contains the reason of the error. The application shall notify the user and, given the context and the reason, take appropriate decisions to either retry the last action, or to report that it is not able to go any further.

5. ADVANCED FEATURES

5.1. PROPERTIES AND CONSTANTS OF THE **ReaderList** OBJECT

5.1.1. Static properties (constants)

The **BuildConfig** object exposes the following constants properties to identify the Library:

<i>Type</i>	<i>Name</i>	<i>Description</i>
string	libraryName	“PC/SC-Like over BLE Library”
string	librarySpecial	(free string)
bool	libraryDebug	false: release version true: debug version
string	libraryVersion	Version of the Library, in the form “MM.mm-bb-gXXXXX”
int	libraryVersionMajor	The MM part of libraryVersion
int	libraryVersionMinor	The mm part of libraryVersion
int	libraryVersionBuild	The bb part of libraryVersion

5.1.2. Dynamic properties (from the BLE device)

The **ReaderList** object exposes the following properties that are retrieved from the BLE device:

<i>Type</i>	<i>Name</i>	<i>Description</i>
string	vendorName	Manufacturer name of the BLE device
string	productName	Product name of the BLE device
string	serialNumber	Serial number of the BLE device, expressed in hex.
byte[]	serialNumberRaw	Serial number of the BLE device
string	firmwareVersion	Firmware version of the device, in the form “MM.mm-bb-gXXXXX”
int	firmwareVersionMajor	The MM part of firmwareVersion
int	firmwareVersionMinor	The mm part of firmwareVersion

int `firmwareVersionBuild` The bb part of `firmwareVersion`

5.2. CONTROL METHODS

5.2.1. iOS implementation

Type	Method or callback	Remark
Send a direct command to the device, using the Reader object		
async method	<code>reader.control(command)</code>	Counterpart to PC/SC's <code>SCardControl</code>
async callback	<code>onControlDidResponse(readers, response, error)</code>	Warning: the callback targets the parent <code>ReaderList</code> , not the <code>Reader</code> itself
Send a direct command to the device, using the ReaderList object		
async method	<code>readers.control(command)</code>	Same as <code>readers.getReader(0).control()</code>
async callback	<code>onControlDidResponse(readers, response, error)</code>	

5.2.2. Android implementation

Send a direct command to the device, using the ReaderList object		
async method	<code>readers.control(command)</code>	Counterpart to PC/SC's <code>SCardControl</code>
async callback	<code>onControlResponse(readers, response)</code>	

5.3. READ DEVICE POWER STATE & BATTERY LEVEL

5.3.1. iOS implementation

Type	Method or callback	Remark
Read device's power state & battery level		
async method	readers.getPowerInfo()	
async callback	onPowerInfo(powerState: Int?, batteryLevel: Int?, error: Error?)	powerState: see https://docs.springcard.com/books/SpringCore/Host_interfaces/BLE/Standard_Services#page_Device-s-power-capabilities-and-states batteryLevel: 0-100%

5.3.2. Android implementation

Read device's power state & battery level		
async method	readers.getPowerInfo()	
async callback	onPowerInfo(readers, powerState, batteryLevel)	powerState: 0 : unknown 1 : on USB/5V power supply 2 : on battery batteryLevel: 0-100%

5.4. SLEEP AND WAKE-UP

5.4.1. iOS implementation

Type	Method or callback	Remark
Read device's state		
async method	readers.wakeUp()	Wake-up device if it was sleeping

async callback	onReaderListState(readers: SCardReaderList, isLowPowerMode: Bool)	<p>Invoked when the device is going to sleep or waking up</p> <p>isLowPowerMode: true: going to sleep false = waking-up</p>
----------------	--	---

5.4.2. Android implementation

<i>Type</i>	<i>Method or callback</i>	<i>Remark</i>
Read device's state		
async method	readers.wakeUp()	Wake-up device if it was sleeping
async callback	onReaderListState(readers, isLowPowerMode)	<p>Invoked when the device is going to sleep or waking up</p> <p>isLowPowerMode: true: going to sleep false = waking-up</p>

5.5. SECURE COMMUNICATION

The secure communication can be used with this SDK by using the object CcidSecureParameters to the method SCardReaderList.create(...).

The CcidSecureParameters class comport the following elements:

- AuthenticationMode : 0x00: none or 0x01: AES128
- AuthenticationKeyIndex : 0x00: User or 0x01: Admin
- AuthenticationKey: The value of the key on 16 bytes for AES128
- CommunicationMode : This field specify how the communication will be secured once the authentication is passed (Plain, MACed, Cipher and MACed)

The details of the secure communication can be found here:

[https://docs.springcard.com/books/SpringCore/Host_interfaces/BLE/CCID_\(PCSC\)](https://docs.springcard.com/books/SpringCore/Host_interfaces/BLE/CCID_(PCSC))

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 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 product 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.

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 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 SPRINGCARD.

Copyright © SPRINGCARD SAS 2019, all rights reserved.

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 INFORMATION

For more information and to locate our sales office or distributor in your country or area, please visit

www.springcard.com