TERTIUM Technology's RFID readers management APP

BLE RFID readers by TERTIUM Technology

TERTIUM Technology manufactures and markets, under the **`BlueBerry`** name, some HF (ISO) and UHF (EPC) portable RFID reader devices with a BLE (*Bluetooth Low Energy*) interface that can be used by a smartphone or a tablet.

TERTIUM Technology customers who purchase **`BlueBerry`** RFID readers, usually develop management APPs for the most popular operating systems for smartphones and tablets that support the BLE communication standard, in particular Google Android and Apple iOS.

The low-level management native libraries for TERTIUM Technology BLE devices

TERTIUM Technology has developed 3 native libraries called **`TxRxLib`** for the low level management of its devices equipped with BLE interface:

- library for Android operating system in Java language;
- library for iOS operating system in Objective-C language;
- library for iOS operating system in Swift language.

The **`TxRxLib`** libraries, in addition to the BLE devices' scanning and detection functions, expose an API for sending commands and receiving replies in the form of strings composed of ASCII characters; as a result of the fact that all BLE devices made by TERTIUM Technology expose a protocol of commands and responses using two characteristics called *Tx* and *Rx* respectively, the low-level libraries **`TxRxLib`** allow you to manage any device with BLE interface made by TERTIUM Technology, not only **`BlueBerry`**[1] HF/UHF RFID readers.

TERTIUM Technology's low-level management libraries for BLE devices are documented according to the standards set by the IDEs used for development.

The test apps for the low-level native libraries of TERTIUM Technology's BLE devices

In order to test the native libraries of low-level management of their devices equipped with BLE interface, TERTIUM Technology has created three APPs:

- Native APP for Android operating system implemented in Java language;
- Native APP for iOS operating system implemented in Objective-C language;
- Native APP for iOS operating system implemented in Swift language.

All the test apps expose a GUI consisting of 2 views:

- one view with the list of detected BLE devices and possibility of selection and connection to a specific device;
  - one view with the list of commands sent to a TERTIUM Technology BLE device, the related replies and the error conditions generated, input control for typing and sending a command string.

The native test APPs also expose on the Wi-Fi network a TCP *socket* to which you can connect to send command strings to a connected BLE device and receive the related replies and generated

[1] In particular, the **`TxRxLib`** low-level native libraries allow the management of the so-called "active" devices: BLE sensors and *gateway* for accessing *wireless* sensor networks.

error conditions.

The figure below illustrates the overall software architecture of the native libraries and test apps for low-level management of TERTIUM Technology BLE devices:

Native test APP (Java)	Native test APP (Objective-C)	Native test APP (Swift)
native Tx/Rx library (Java)	native Tx/Rx library (Objective-C)	native Tx/Rx library (Swift)
Android operating system	**iOS operating system**	

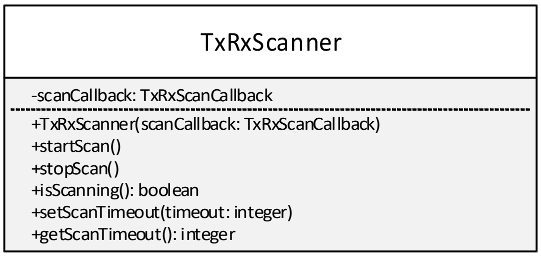
The functionalities of the **TxRxLib** libraries and the related native test APPs are described in the following attached documents:

- “TERTIUM_TxRxApp for Android operating system”
- “TERTIUM_TxRxApp for iOS operating system”

TERTIUM Technology's high-level management API for RFID readers

For HF/UHF **`BlueBerry`** BLE RFID readers by TERTIUM Technology has been designed and implemented a high level asynchronous management API, both for Android operating system and for iOS operating system. This API encapsulates the features of the low level **`TxRxLib`** library based on the following classes:

- ***TxRxScanner*** – exposes the methods for BLE devices scanning

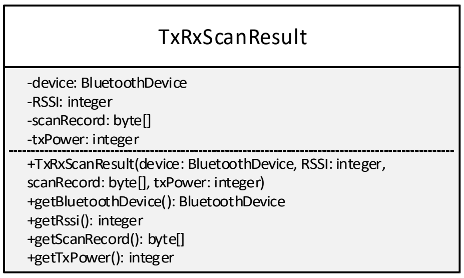


TxRxScanner

-scanCallback: TxRxScanCallback

+TxRxScanner(scanCallback: TxRxScanCallback)
+startScan()
+stopScan()
+isScanning(): boolean
+setScanTimeout(timeout: integer)
+getScanTimeout(): integer

- ***TxRxScanResult*** – encapsulates an object, which is an instance of the *BluetoothDevice* class, representing a detected device plus other information connected to the detection (radio signal RSSI, beacon transmission power, beacon advertisement content, …)



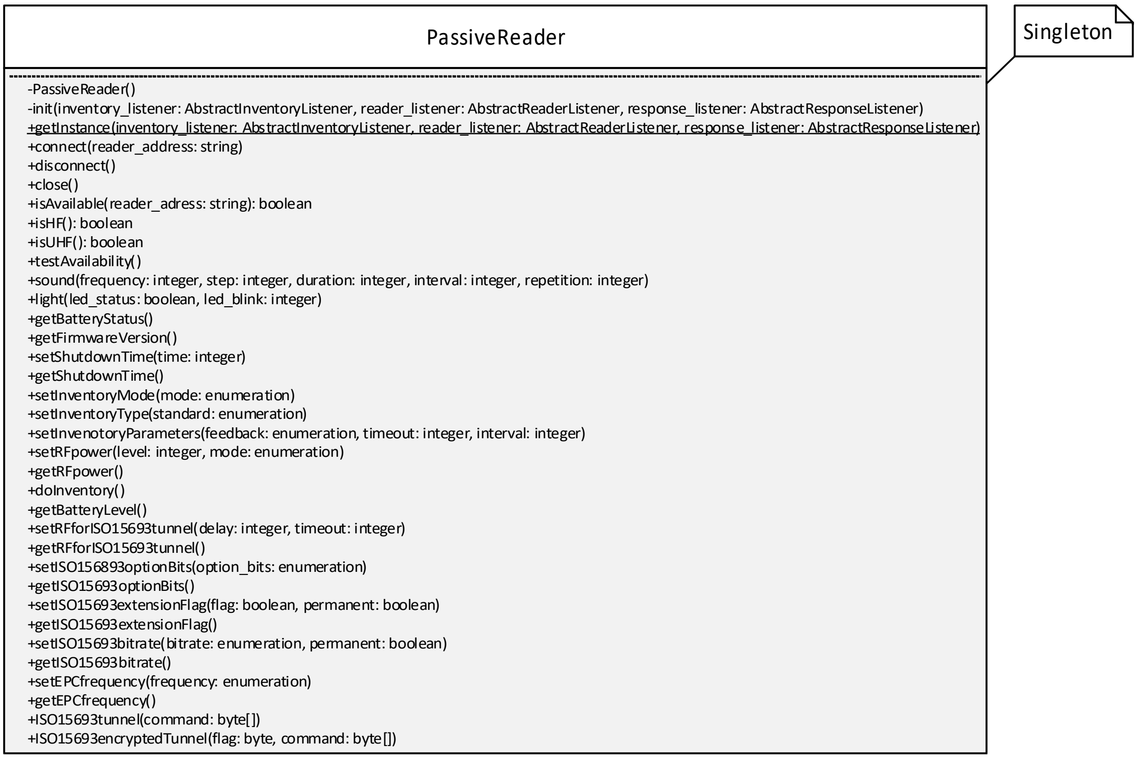
TxRxScanResult

-device: BluetoothDevice
-RSSI: integer
-scanRecord: byte[]
-txPower: integer

+TxRxScanResult(device: BluetoothDevice, RSSI: integer, scanRecord: byte[], txPower: integer)
+getBluetoothDevice(): BluetoothDevice
+getRssi(): integer
+getScanRecord(): byte[]
+getTxPower(): integer

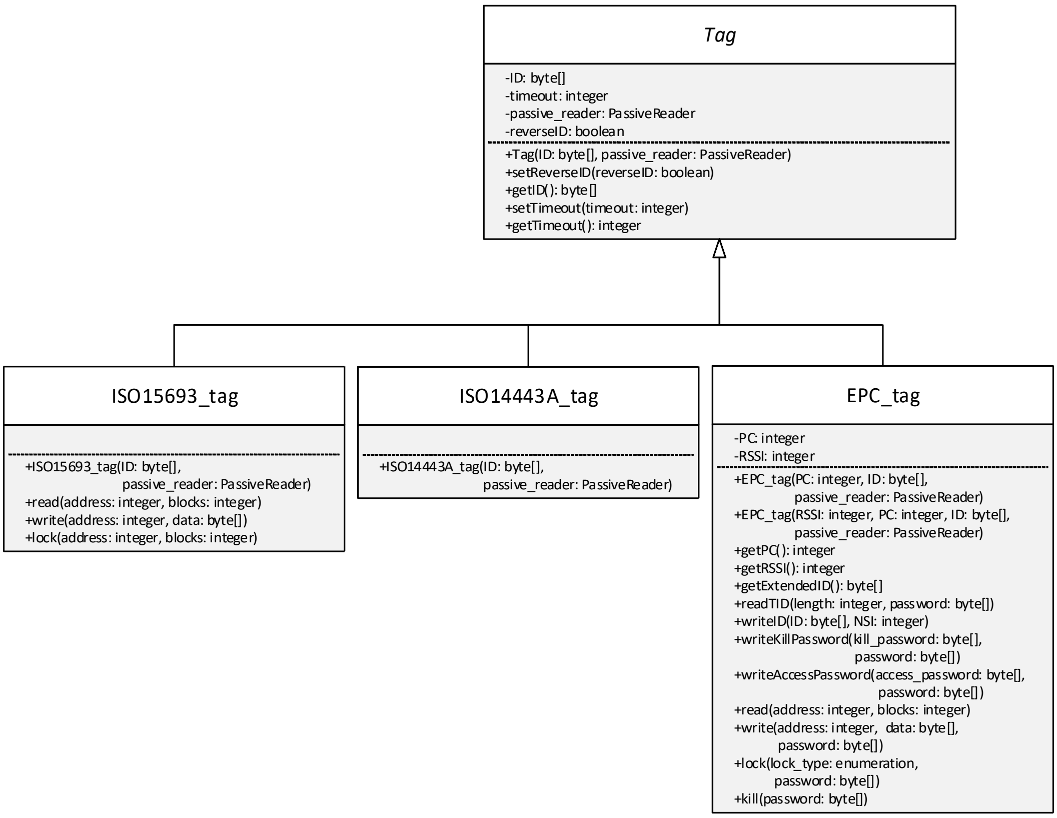
-

- ***PassiveReader*** – exposes the methods for HF/UHF RFID reader management, yielding an abstraction level which is independent from the protocol command strings



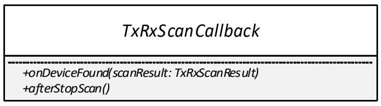
```
PassiveReader

-PassiveReader()
-init(inventory_listener: AbstractInventoryListener, reader_listener: AbstractReaderListener, response_listener: AbstractResponseListener)
+getInstance(inventory_listener: AbstractInventoryListener, reader_listener: AbstractReaderListener, response_listener: AbstractResponseListener)
+connect(reader_address: string)
+disconnect()
+close()
+isAvailable(reader_adress: string): boolean
+isHF(): boolean
+isUHF(): boolean
+testAvailability()
+sound(frequency: integer, step: integer, duration: integer, interval: integer, repetition: integer)
+light(led_status: boolean, led_blink: integer)
+getBatteryStatus()
+getFirmwareVersion()
+setShutdownTime(time: integer)
+getShutdownTime()
+setInventoryMode(mode: enumeration)
+setInventoryType(standard: enumeration)
+setInvenotoryParameters(feedback: enumeration, timeout: integer, interval: integer)
+setRFpower(level: integer, mode: enumeration)
+getRFpower()
+doInventory()
+getBatteryLevel()
+setRFforISO15693tunnel(delay: integer, timeout: integer)
+getRFforISO15693tunnel()
+setISO156893optionBits(option_bits: enumeration)
+getISO15693optionBits()
+setISO15693extensionFlag(flag: boolean, permanent: boolean)
+getISO15693extensionFlag()
+setISO15693bitrate(bitrate: enumeration, permanent: boolean)
+getISO15693bitrate()
+setEPCfrequency(frequency: enumeration)
+getEPCfrequency()
+ISO15693tunnel(command: byte[])
+ISO15693encryptedTunnel(flag: byte, command: byte[])
```

- ***Tag*** – stands for a RFID *tag*; derived classes such as ***ISO15693_tag***, ***ISO14443_tag*** or ***EPC_tag*** expose methods for reading, writing and managing specific type tags; in this situation too, they yield an abstraction level that is independent from the protocol command strings

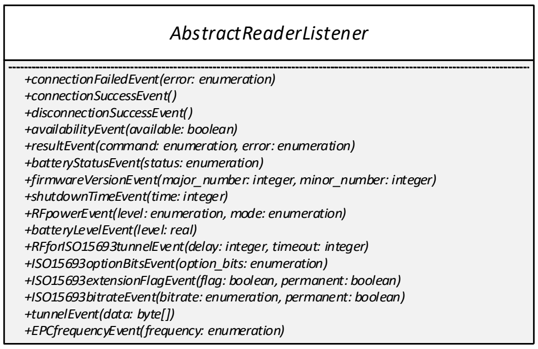

Tag
-ID: byte[]
-timeout: integer
-passive_reader: PassiveReader
-reverseID: boolean
+Tag(ID: byte[], passive_reader: PassiveReader)
+setReverseID(reverseID: boolean)
+getID(): byte[]
+setTimeout(timeout: integer)
+getTimeout(): integer
ISO15693_tag
+ISO15693_tag(ID: byte[], passive_reader: PassiveReader)
+read(address: integer, blocks: integer)
+write(address: integer, data: byte[])
+lock(address: integer, blocks: integer)
ISO14443A_tag
+ISO14443A_tag(ID: byte[], passive_reader: PassiveReader)
EPC_tag
-PC: integer
-RSSI: integer
+EPC_tag(PC: integer, ID: byte[], passive_reader: PassiveReader)
+EPC_tag(RSSI: integer, PC: integer, ID: byte[], passive_reader: PassiveReader)
+getPC(): integer
+getRSSI(): integer
+getExtendedID(): byte[]
+readTID(length: integer, password: byte[])
+writeID(ID: byte[], NSI: integer)
+writeKillPassword(kill_password: byte[], password: byte[])
+writeAccessPassword(access_password: byte[], password: byte[])
+read(address: integer, blocks: integer)
+write(address: integer, data: byte[], password: byte[])
+lock(lock_type: enumeration, password: byte[])
+kill(password: byte[])

and on the following interfaces:

- ***TxRxScanCallback*** – defines the callback methods for BLE devices detection

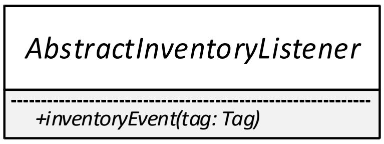


- ***AbstractReaderListener*** – defines the callback methods related to the methods of *PassiveReader* class



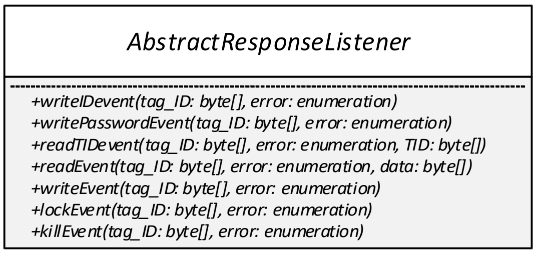
```
AbstractReaderListener
+connectionFailedEvent(error: enumeration)
+connectionSuccessEvent()
+disconnectionSuccessEvent()
+availabilityEvent(available: boolean)
+resultEvent(command: enumeration, error: enumeration)
+batteryStatusEvent(status: enumeration)
+firmwareVersionEvent(major_number: integer, minor_number: integer)
+shutdownTimeEvent(time: integer)
+RFpowerEvent(level: enumeration, mode: enumeration)
+batteryLevelEvent(level: real)
+RFforISO15693tunnelEvent(delay: integer, timeout: integer)
+ISO15693optionBitsEvent(option_bits: enumeration)
+ISO15693extensionFlagEvent(flag: boolean, permanent: boolean)
+ISO15693bitrateEvent(bitrate: enumeration, permanent: boolean)
+tunnelEvent(data: byte[])
+EPCfrequencyEvent(frequency: enumeration)
```

- ***AbstractInventoryListener*** – defines the callback method for RFID tag detection



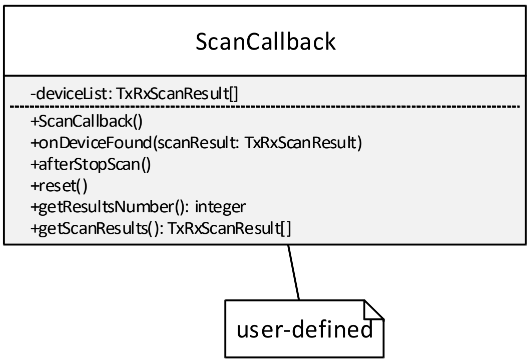
```
AbstractInventoryListener
+inventoryEvent(tag: Tag)
```

- ***AbstractResponseListener*** - defines the callback methods related to the methods of classes derived from *Tag* class

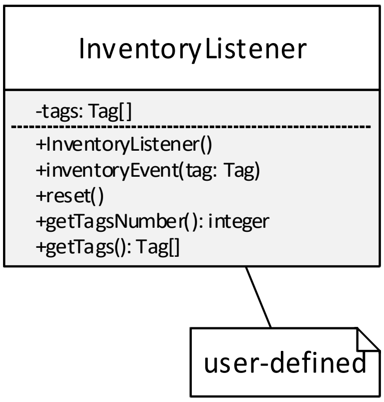


```
AbstractResponseListener
+writeIDevent(tag_ID: byte[], error: enumeration)
+writePasswordEvent(tag_ID: byte[], error: enumeration)
+readTIDevent(tag_ID: byte[], error: enumeration, TID: byte[])
+readEvent(tag_ID: byte[], error: enumeration, data: byte[])
+writeEvent(tag_ID: byte[], error: enumeration)
+lockEvent(tag_ID: byte[], error: enumeration)
+killEvent(tag_ID: byte[], error: enumeration)
```

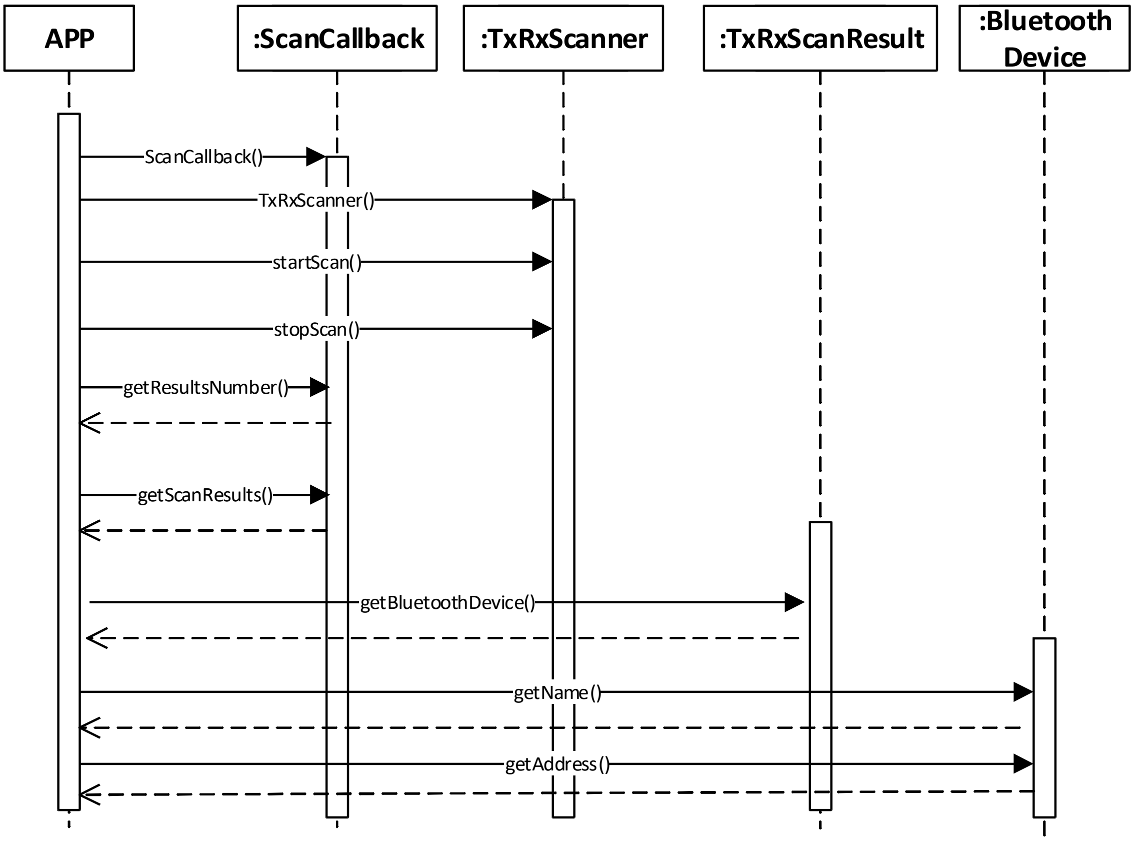
If you choose to implement the *TxRxScanCallback* interface with the *ScanCallback* class, exposing methods for the management of the collection of detected RFID *tags*


ScanCallback
-deviceList: TxRxScanResult[]
+ScanCallback()
+onDeviceFound(scanResult: TxRxScanResult)
+afterStopScan()
+reset()
+getResultsNumber(): integer
+getScanResults(): TxRxScanResult[]
user-defined

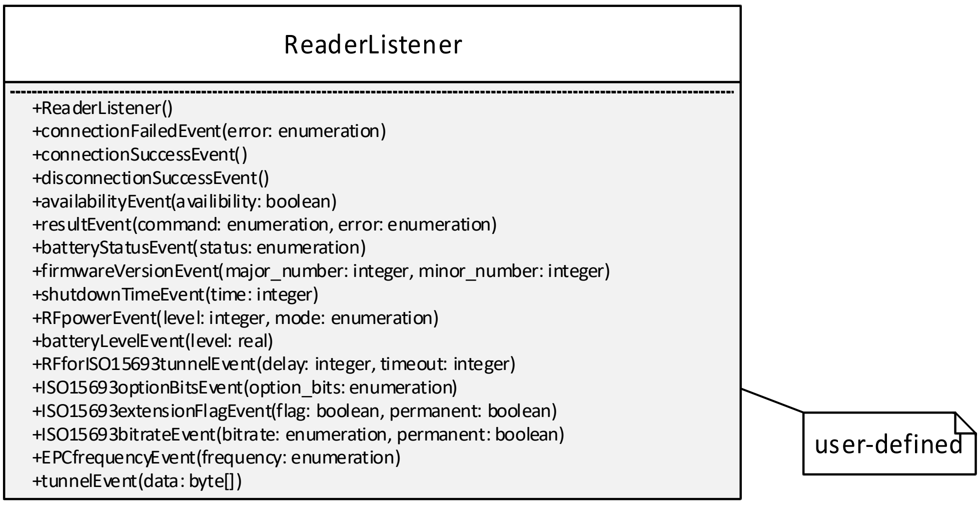
and to implement the *AbstractInventoryListener* interface with the *InventoryListener* class, exposing methods for the management of the collection of detected RFID *tags,*



the typical interaction between an APP and the APIs starts with a scan for a BLE RFID reader and it is documented by the following UML sequence diagram:



If you choose to implement the *AbstractReaderListener* interface with the *ReaderListener* class

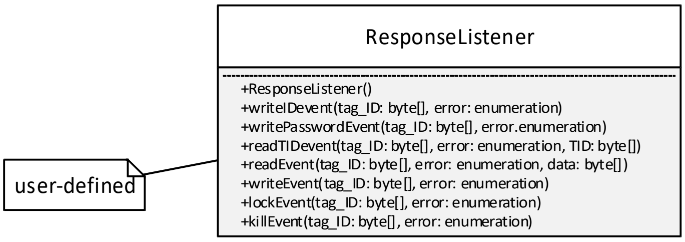


ReaderListener

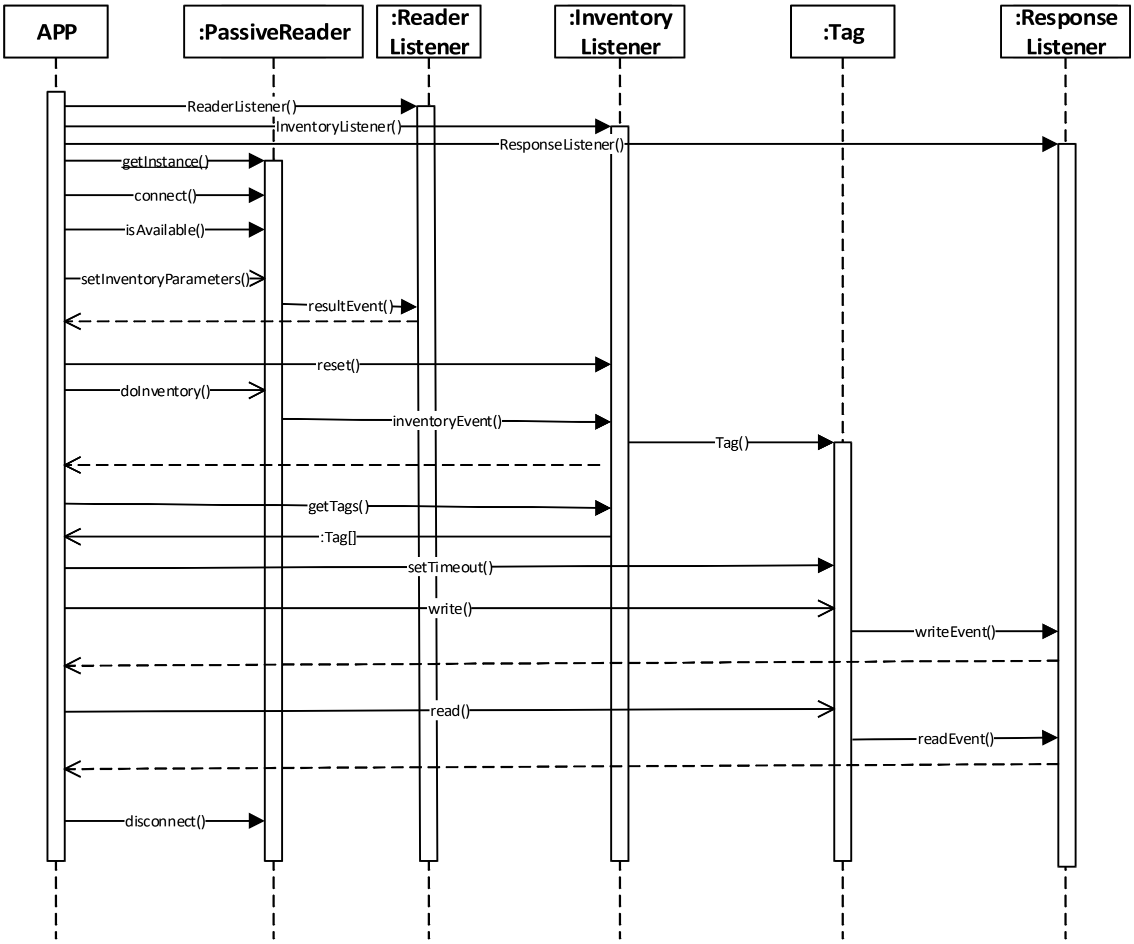
+ReaderListener()
+connectionFailedEvent(error: enumeration)
+connectionSuccessEvent()
+disconnectionSuccessEvent()
+availabilityEvent(availibility: boolean)
+resultEvent(command: enumeration, error: enumeration)
+batteryStatusEvent(status: enumeration)
+firmwareVersionEvent(major_number: integer, minor_number: integer)
+shutdownTimeEvent(time: integer)
+RFpowerEvent(level: integer, mode: enumeration)
+batteryLevelEvent(level: real)
+RFforISO15693tunnelEvent(delay: integer, timeout: integer)
+ISO15693optionBitsEvent(option_bits: enumeration)
+ISO15693extensionFlagEvent(flag: boolean, permanent: boolean)
+ISO15693bitrateEvent(bitrate: enumeration, permanent: boolean)
+EPCfrequencyEvent(frequency: enumeration)
+tunnelEvent(data: byte[])

user-defined

and the *AbstractResponseListener* class with the *ResponseListener* class,


ResponseListener
+ResponseListener()
+writeIDevent(tag_ID: byte[], error: enumeration)
+writePasswordEvent(tag_ID: byte[], error.enumeration)
+readTIDevent(tag_ID: byte[], error: enumeration, TID: byte[])
+readEvent(tag_ID: byte[], error: enumeration, data: byte[])
+writeEvent(tag_ID: byte[], error: enumeration)
+lockEvent(tag_ID: byte[], error: enumeration)
+killEvent(tag_ID: byte[], error: enumeration)
user-defined

the typical interaction between an APP and the APIs continues with the detection of RFID tags (inventory) and the execution of operations on one of them, both phases documented in the following UML sequence diagram:



Le librerie native di implementazione delle API di gestione ad alto livello dei lettori RFID BLE di TERTIUM Technology

TERTIUM Technology has developed three native libraries (**`RfidPassiveApiLib`**) implementing the high level management APIs for its RFID readers equipped with a BLE interface:

- library for Android operating system implemented in Java language;
- library for iOS operating system implemented in Objective-C language;
- library for iOS operating system implemented in Swift language.

The implementation libraries of TERTIUM Technology's high-level RFID reader management APIs are documented according to the standards set by the IDEs used for development; in particular for the library in Java language the documentation was generated in HTML format according to the `JavaDoc` standard.

The test apps for the high-level native libraries of TERTIUM Technology's BLE devices

In order to test the native libraries of high-level management of their devices equipped with BLE interface, TERTIUM Technology has created three APPs:

- Native APP for Android operating system implemented in Java language;
- Native APP for iOS operating system implemented in Objective-C language;
- Native APP for iOS operating system implemented in Swift language.

All the test apps expose a GUI consisting of 2 views:

- a view with the list of detected BLE devices and with the possibility of selection and connection to a specific device;
- other view, divided in three sections:

  - display of the outcome of the invocation of the API methods for the initialization of the RFID reader carried out at the time of connection,
  - display of the outcome of the invocation of the API methods for querying the status of the RFID reader carried out periodically,

- a control for selection[2] and invocation with predefined parameters of an RFID reader management API; a control for the interaction with a detected *tag* and the display of its outcome (includes the display of the IDs of the tags detected following the selection of the *doInventory* method of the API).

The figure below illustrates the overall software architecture of the native libraries and API test APPs for high-level management of TERTIUM Technology BLE devices:

Native test APP (`Java`)	Native test APP (`Objective-C`)	Native test APP (`Swift`)
Native API library (`Java`)	Native API library (`Objective-C`)	Native API library (`Swift`)
Native Tx/Rx library (`Java`)	Native Tx/Rx library (`Objective-C`)	Native Tx/Rx library (`Swift`)
Android operating system	**iOS operating system**	

[2] you can choose among a meaningful subset of the API methods.

The TERTIUM Technology RFID reader library/APP code *repository*

The code of the libraries and of the RFID reader management APPs produced by TERTIUM Technology is released with an *open-source* MIT license (opensource.org/licenses/MIT) that allows its use in both open-source and proprietary software projects, without the need to redistribute the source code.

The source code of the libraries and management APPs is distributed on the github.com platform in the following TERTIUM Technology repositories (github.com/tertiumtechnology/):

Library/APP	***Repository***
TxRxLib native library for Java/Android	tt-txrx-lib-android
TxRxLib native library for Objective-C/iOS	tt-txrx-lib-ios-objc
TxRxLib native library for Swift/iOS	tt-txrx-lib-ios-swift
TxRxLIB native test APP for Java/Android	tt-txrx-demoapp-android
TxRxLib native test APP for Objective-C/iOS	tt-txrx-demoapp-ios-objc
TxRxLib native test APP for Swift/iOS	tt-txrx-demoapp-ios-swift
RfidPassiveApiLib native library for Java/Android	tt-rfid-passive-api-lib-android
RfidPassiveApiLib native library for Objective-C/iOS	tt-rfid-passive-api-lib-ios-objc
RfidPassiveApiLib native library for Swift/iOS	tt-rfid-passive-api-lib-ios-swift
RfidPassiveApiLib native test APP for Java/Android	tt-rfid-passive-api-testapp-android
RfidPassiveApiLib native test APP for Objective-C/iOS	tt-rfid-passive-api-testapp-ios-objc
RfidPassiveApiLib native test APP for Swift/iOS	tt-rfid-passive-api-testapp-ios-swift