TertiumBleScan and *TertiumBleScanSwift* (TxRx) APP for iOS operating system

1. APP purpose and functionalities

The *TertiumBleScan* and *TertiumBleScanSwift* APPs for iOS Operating System have been released by TERTIUM Technology with a "MIT License", which is an *open source* license, with the specific purpose of simplifying, for its partners and clients, the development of APPs that interact with *TxRx Bluetooth Low Energy* (BLE) devices by TERTIUM Technology. The two APP have the same functionalities and display the same user interface, they differ just for the programming language used for their development: Objective-C and Swift, respectively.

TxRx devices by TERTIUM Technology expose a BLE service with two main characteristics – named respectively "Tx" e "Rx" – simulating a bidirectional ASCII *stream*. With this stream the device uses "Rx" characteristic to receive commands from a central device, usually an iPhone smartphone or an iPad tablet with iOS operating system; it uses the "Tx" characteristic, instead, to send the corresponding replies.

The APP offers functionalities for showing all the detectable BLE devices and for connecting to one of those devices (if it is a "*TxRx*" one) and a CLI; with this textual interface the users are allowed to write the commands they want to issue to the device and to read the received replies.

The core of the *TertiumBleScan e TertiumBleScanSwift* APP source code, named "TxRxLibrary", has been written and documented in order to be reused in each and every APP interacting with *TxRx* BLE devices by TERTIUM Technology; the GUI is meant as a useful example of "TxRxLibrary"[1] module functionalities.

[1] An APP for a specific application has obviously to manage the commands and text replies documented by TERTIUM Technology for the specific *TxRx* BLE device used.

2. Architecture of the source code of *TertiumBleScan* e *TertiumBleScanSwift* APPs

The source code of *TertiumBleScan* e *TertiumBleScanSwift* APP is entirely included in the "TxRxLibrary" library project, both for the Objective-C version and for the Swift one. The *Core* class, contained in the "Core" project folder, implements a *singleton* type *proxy* that interfaces the library with the remaining part of the code; this code implements the user interfaces of the two APPs and a delegate object transforming into messages to be shown every asynchronous notifications generated by the code of the library and reflected by the *Core* class (receptions of replies from the device, *timeout* events, errors, …). The usage of the *Core* class, though, is not mandatory for using the library itself: the developer can choose a different management *pattern* for the "TxRxLibrary" delegates.

3. Functionalities of the *Core* classes for the *TertiumBleScan* e *TertiumBleScanSwift* APPs

For the Objective-C *TertiumBleScan* APP, the *Core* class contains the following methods:

Method	Functionality
`getCore`	returns the *singleton* instance of the *Core* class
`isScanning`	returns *true* if the BLE devices scanning phase is active, *false* otherwise
`startScan`	starts the scanning phase for the BLE devices: every single device found yields a notification, even if it is not a *TxRx* device
`stopScan`	stops the scanning phase for the BLE devices
`getScannedDevices`	returns the list of the BLE devices found in the scanning phase (includes also the non *TxRx* devices)
`connectDevice`	connects the specified device between the ones found in the scanning phase: you can have a connection only with *TxRx* devices
`sendData`	sends a string of data to the connected device: the reception of the response yields a notification
`disconnectDevice`	disconnects the specified device

Moreover, the Objective-C *Core* class reflects the following notifications:

Notification	Description
`TxRxScanBegan`	start of the scanning phase
`TxRxScanError`	error in the scanning phase
`TxRxScanEnded`	end of the scanning phase
`TxRxDeviceError`	error on the device
`TxRxDeviceFound`	BLE device found
`TxRxDeviceConnectError`	connection error with the device[2] (includes *timeout* error)
`TxRxDeviceConnected`	connection with the device done
`TxRxDeviceReady`	the *TxRx* type device has been correctly detected: the device is ready to receive commands
`TxRxDeviceDisconnected`	disconnection from device done
`TxRxDeviceDataSent`	data correctly sent to device
`TxRxDeviceDataSendError`	error in data sending to the device
`TxRxDeviceDataSendTimeout`	data sending to the device timed out
`TxRxDeviceDataReceived`	data received from device
`TxRxDeviceDataReceiveError`	error receiving data from the device (includes *timeout* error*)*
`TxRxDeviceInternalError`	error in device management

Objective-C *Core* class does not intercept the following utility methods exposed by *TxRxManager* class:

Method	Functionality
`deviceWithIndexedName`	returns a reference to a device found by *indexed-name* (device name followed by the symbol "_" and by the position of the device itself in the list of the devices found in the scanning phase)
`getDeviceIndexedName`	returns the *indexed-name* of specified device
`setTimeOutDefaults`	set the default timeout time values
`getTimeOutValue`	return the specified timeout time value (connection, reception of the first fragment of response data, reception of successive fragments of response data, command data transmission)
`setTimeOutValue`	set the specified timeout time value (connection, reception of the first fragment of response data, reception of successive fragments of response data, command data transmission)
`deviceFromDeviceName`	returns a reference to a device found by device name (device name in the list of the devices found in the scanning phase)
`getDeviceName`	returns the device name

For the *TertiumBleScanSwift* Swift APP, the *Core* class includes the following methods:

[2] Error yielded even in case of non *TxRx* type device.

Method	Functionality
`getCore`	returns the *singleton* instance of the *Core* class
`isScanning`	returns *true* if the BLE devices scanning phase is active, *false* otherwise
`startScan`	starts the scanning phase for the BLE devices: every single device found yields a notification, even if it is not a *TxRx* device
`stopScan`	stops the scanning phase for the BLE devices
`getScannedDevices`	returns the list of the BLE devices found in the scanning phase (includes also the non *TxRx* devices)
`connectDevice`	connects the specified device between the ones found in the scanning phase: you can have a connection only with *TxRx* devices
`sendData`	sends a string of data to the connected device: the reception of the response yields a notification
`disconnectDevice`	disconnects the specified device

Moreover, the Swift *Core* class reflects the following notifications:

Notification	Description
`TxRxScanBegan`	start of the scanning phase
`TxRxScanError`	error in the scanning phase
`TxRxScanEnded`	end of the scanning phase
`TxRxDeviceError`	error on the device
`TxRxDeviceFound`	BLE device found
`TxRxDeviceConnectError`	connection error with the device[3] (includes *timeout* error)
`TxRxDeviceConnected`	connection with the device done
`TxRxDeviceReady`	the *TxRx* type device has been correctly detected: the device is ready to receive commands
`TxRxDeviceDisconnected`	disconnection from device done
`TxRxDeviceDataSent`	data correctly sent to device
`TxRxDeviceDataSendError`	error in data sending to the device
`TxRxDeviceDataSendTimeout`	data sending to the device timed out
`TxRxDeviceDataReceived`	data received from device
`TxRxDeviceDataReceiveError`	error receiving data from the device (includes *timeout* error*)*
`TxRxDeviceInternalError`	error in device management

[3] Error yielded even in case of non *TxRx* type device.

The Swift *Core* class does not intercepts the following utility methods exposed by the *TxRxManager* class:

Method	**Functionality**
`deviceWithIndexedName`	returns a reference to a device found by *indexed-name* (device name followed by the symbol "_" and by the position of the device itself in the list of the devices found in the scanning phase)
`getDeviceIndexedName`	returns the *indexed-name* of specified device
`setTimeOutDefaults`	set the default timeout time values
`getTimeOutValue`	return the specified timeout time value (connection, reception of the first fragment of response data, reception of successive fragments of response data, command data transmission)
`setTimeOutValue`	set the specified timeout time value (connection, reception of the first fragment of response data, reception of successive fragments of response data, command data transmission)
`deviceFromDeviceName`	returns a reference to a device found by device name (device name in the list of the devices found in the scanning phase)
`getDeviceName`	returns the device name

4. *Communication management algorithm with BLE TxRx devices*

First of all, you have to establish a connection by invoking the *connectDevice* method of the *Core* class and receiving positive confirmation by the generation of the notifications *TxRxDeviceConnected and TxRxDeviceReady*, which guarantee that the BLE device is of type *TxRx* (in case of connection failure, the *TxRxDeviceConnectError* notification is yielded).

Once the connection has been made, the communication between the iPhone smartphone or the iPad tablet and the BLE *TxRx* device is carried out following the following algorithm:

- invocation of *sendData* method from *Core* class – whose parameter is the string containing the text command to be sent to the BLE device – writes the "Rx" characteristic of the *TxRx* service: if the string to be written has a length greater than the declared size for the characteristic, then multiple sending of fragments of the string are made, each of length equal or inferior to the dimension of the characteristic[4];
- the generation of the *TxRxDeviceDataSent* notification confirms the correct writing of the command in the "Rx" characteristic; in case of error or timeout the *TxRxDeviceSendError* or *TxRxDeviceSendTimeout* call-back methods are invoked respectively (the timeout length is a configurable parameter);
- with the sending of the last fragment of the command string (coinciding with the first in the case of a string of length equal to or less than the size of the "Rx" characteristic), a timer initialized with the timeout time for the start of the reply is started;
- the reception of the possible reply to the command sent is received by successive BLE notifications of changing of the content of the "Tx" characteristic: the string fragments notified in succession are recomposed in a single response string; if no notification is yielded before the timer is reset, the TxRxDeviceReceiveError notification is yielded;
- after the reception of every single fragment of the response string, a timer initialized with the timeout time for the end of the reply is started or restarted: if the timer is reset, the reading of the answer is considered finished and the string that contains is provided as a parameter of the TxRxDeviceDataReceived notification.

With the exception of starting the timer for the timeout, the code of a command for writing and the code of a command for reading a reply are completely asynchronous: any unsolicited variation of the "Tx" characteristic by the BLE device generates the reception of the string in the same way as receiving a reply to a command.

[4] The BLE protocol involves the transmission of packets with a maximum *payload* of 20 bytes: if the length of the "Rx" characteristic is greater than 20, the transmission of the value to be written in the characteristic is divided into packets each with *payload* equal to or less than 20 in a way that is transparent for the Java code that invokes the Android BLE API.

5. Documentation of the *TertiumBleScan and TertiumBleScanSwift* APP library code

The Objective-C and Swift code of the classes included in the library for the *TertiumBleScan* and *TertiumBleScanSwift* APP has been commented with the style provided for the Apple development environment for iOS operating system APP (*Quick-help*).