Introduction

This document describes the functions and the APIs offered by the Android **btdatakeymodule.aar** library. This is conceived as an interface between an Android application and the BTDataKey, by supplying an abstraction level for the "connection", "key identification", "operator code configuration" and "data extraction" functions.
The released library is supplied with a Demo application (***CogesBTDataKeyLibDemo***) which shows an example of calls to the different APIs of the library itself. This demo is a reference point for the development of a customized application which uses the library for the extraction of the accounting data by means of **Coges BTDataKey**.

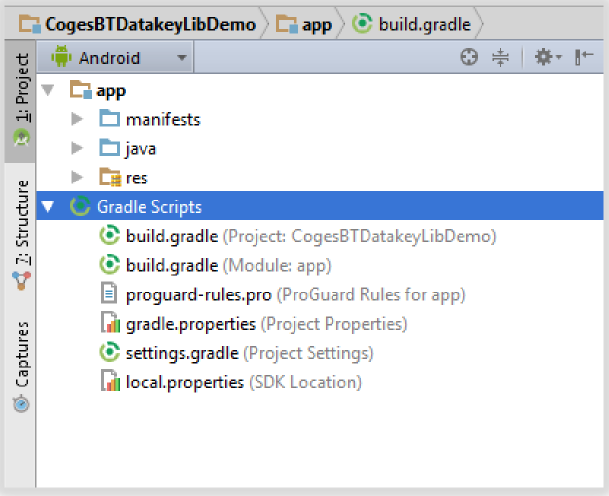
Integration of the library in a new project

This section describes a step by step guide for the creation of a new project in Android Studio, which integrates the library released as Android Archive (AAR) and described in these instructions.

First of all a new Android project shall be created. In the example it is called **CogesBTDatakeyLibDemo** and it offers a main module (**app**) with the following source files:

- MainActivity.java
- FragmentSelection.java
- ScanDevicesActivity.java

The project structure is shown on the window.

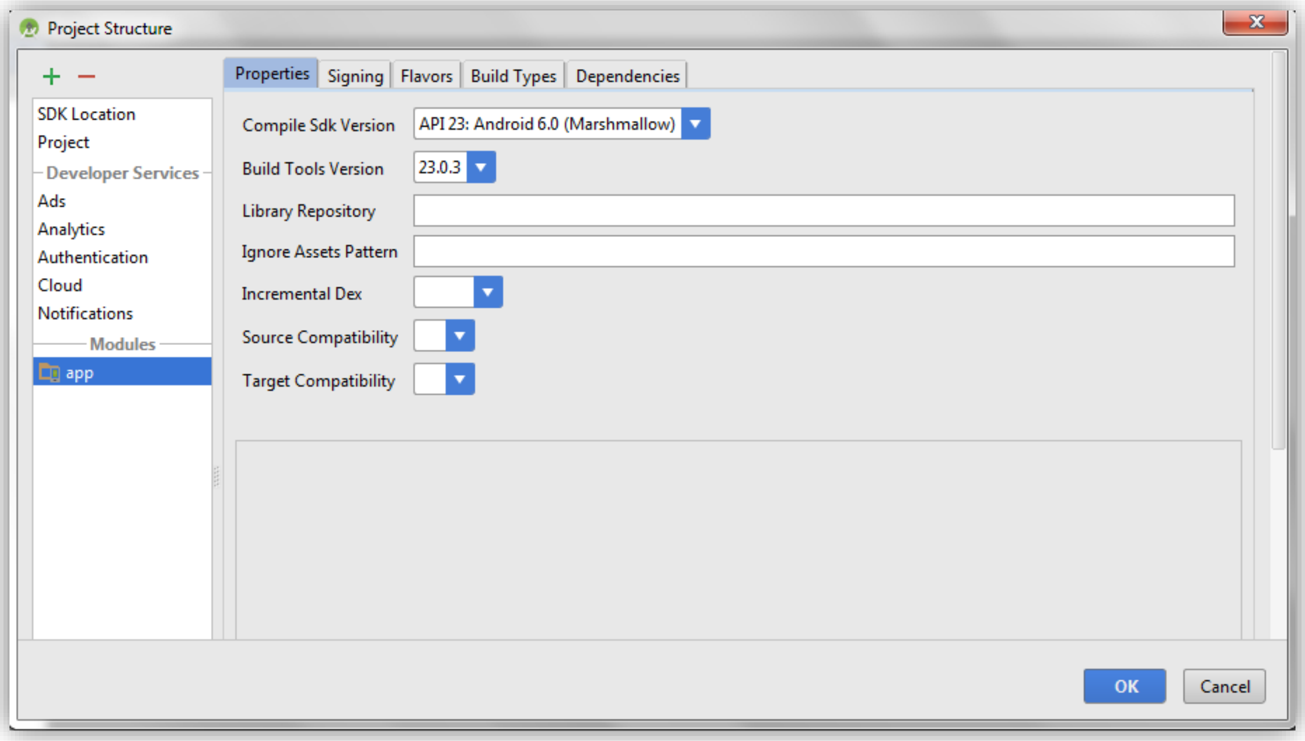

CogesBTDatakeyLibDemo
app
build.gradle
Android
1: Project
7: Structure
Captures
app
manifests
java
res
Gradle Scripts
build.gradle (Project: CogesBTDatakeyLibDemo)
build.gradle (Module: app)
proguard-rules.pro (ProGuard Rules for app)
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)

The .aar library is integrated in the new project by means of a new module containing the **btdatakeymodule.aar** file and from which the **app** module depends.

First of all save the .aar file in a new **libs/** folder in the project root, e.g. *C:\Users\Owner\AndroidStudioProjects\CogesBTDataKeyModule*.

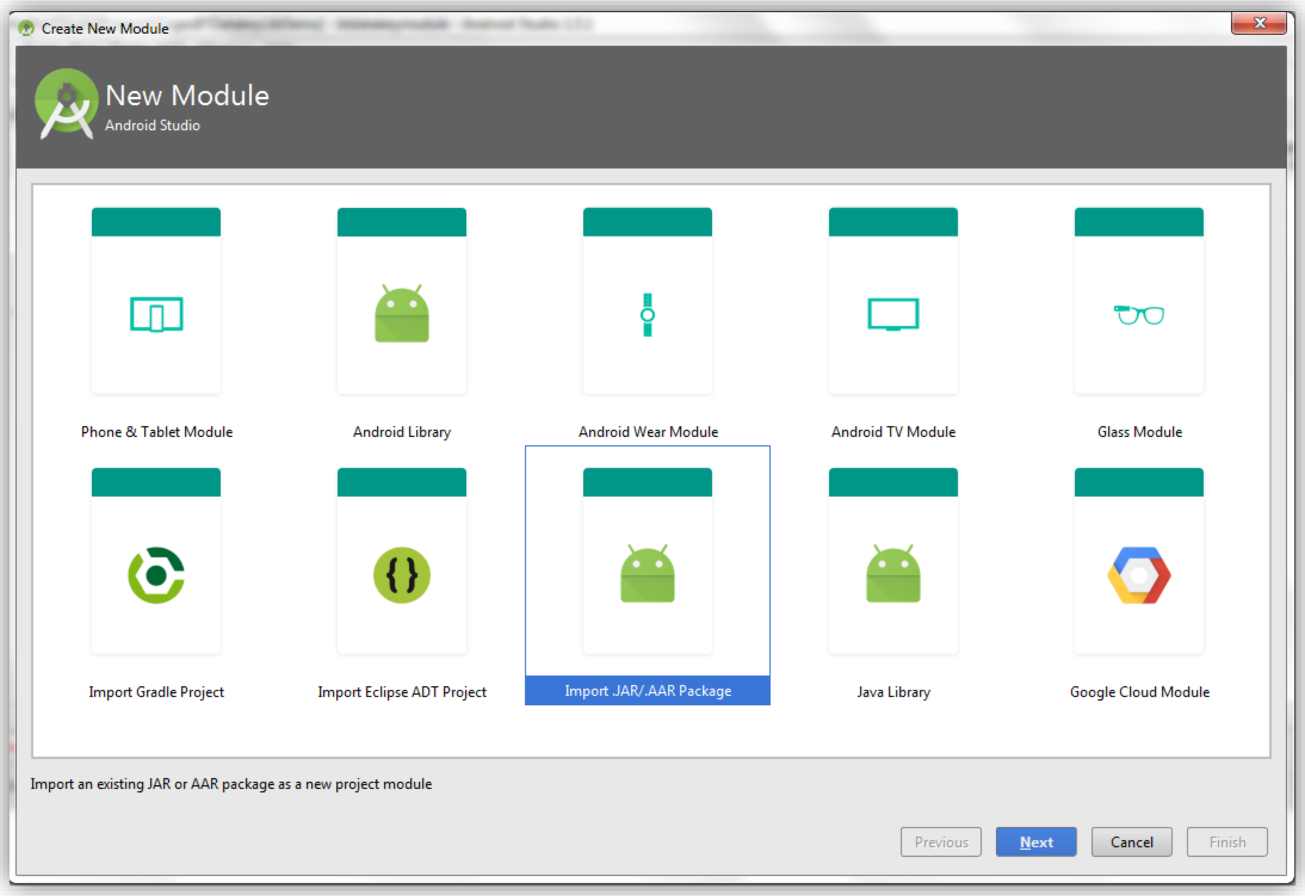
To create a new module which will contain the library, select the **app** from the tab of the project structure, press F4, or click on the right key, and select the *Open Module Settings* item.

The window here below opens:



Compile Sdk Version: API 23: Android 6.0 (Marshmallow)
Build Tools Version: 23.0.3
Library Repository:
Ignore Assets Pattern:
Incremental Dex:
Source Compatibility:
Target Compatibility:

High on the left click on the + key,

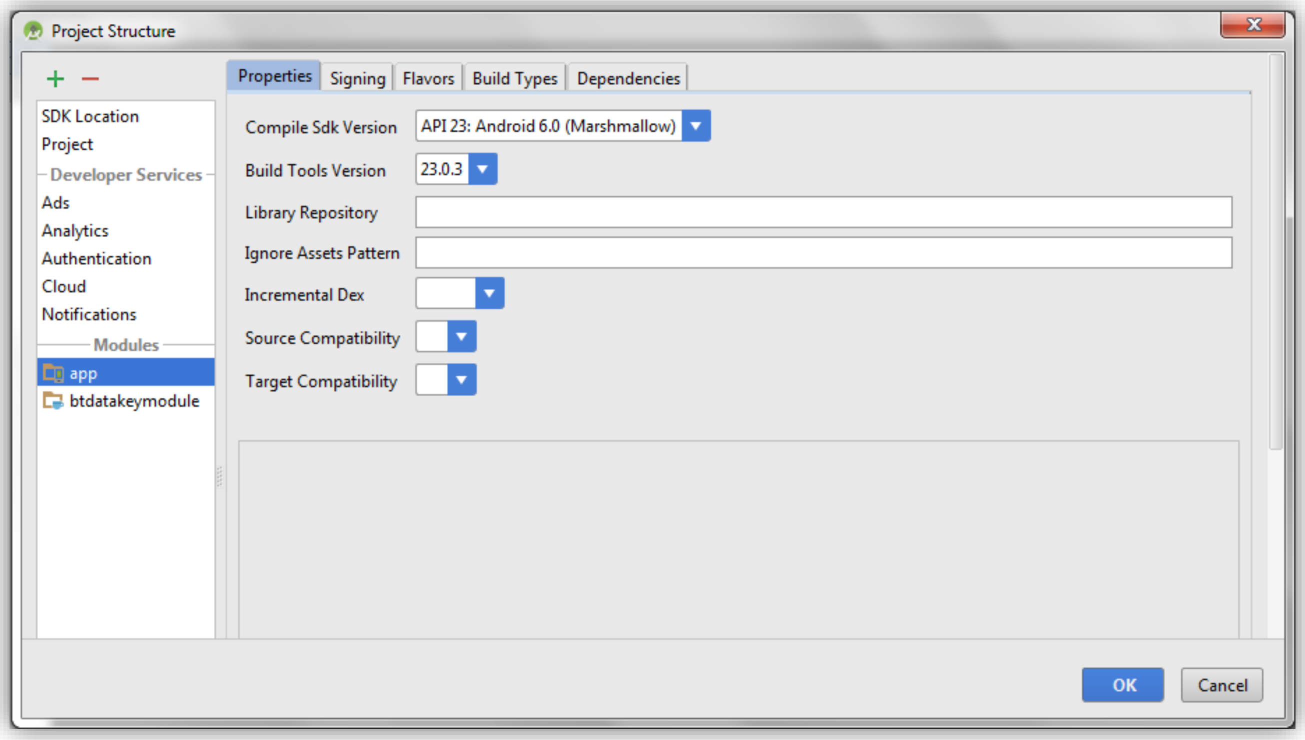


Import an existing JAR or AAR package as a new project module

And select the *Import .JAR / .AAR Package* item.

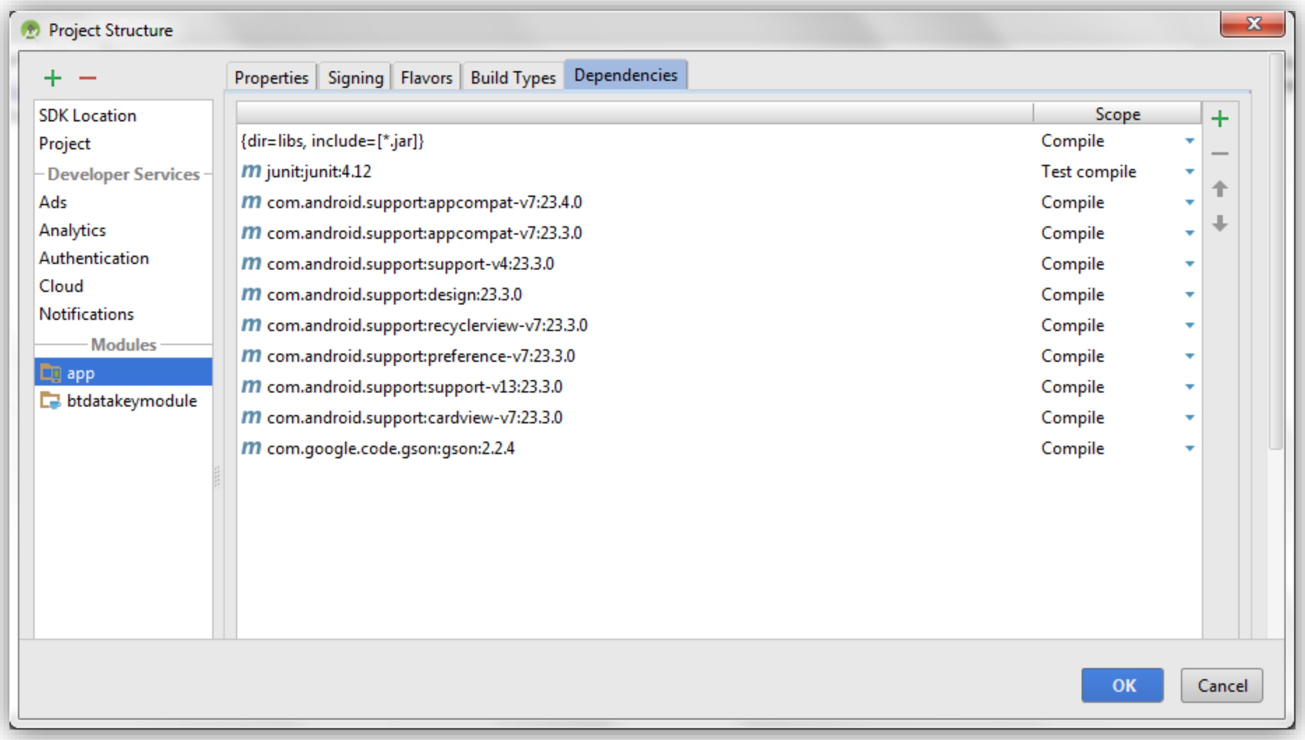
In the following window enter the path of the .aar file in the *Filename* field. It is easier to click on the … key and select the **btdatakeymodule.aar** file which was previously saved in the **libs/** folder.

At this point the list of the project modules will report something like that is shown on the window below:



Compile Sdk Version	API 23: Android 6.0 (Marshmallow)
Build Tools Version	23.0.3
Library Repository	
Ignore Assets Pattern	
Incremental Dex	
Source Compatibility	
Target Compatibility	

To end the integration it is necessary to add the new module as dependency of the **app** main module. To do this select the **app** module, and click on the *Dependencies* tab.

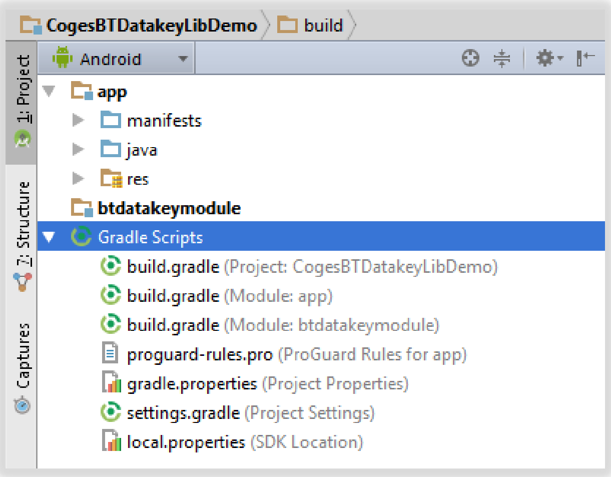


	Scope
{dir=libs, include=[*.jar]}	Compile
m junit:junit:4.12	Test compile
m com.android.support:appcompat-v7:23.4.0	Compile
m com.android.support:appcompat-v7:23.3.0	Compile
m com.android.support:support-v4:23.3.0	Compile
m com.android.support:design:23.3.0	Compile
m com.android.support:recyclerview-v7:23.3.0	Compile
m com.android.support:preference-v7:23.3.0	Compile
m com.android.support:support-v13:23.3.0	Compile
m com.android.support:cardview-v7:23.3.0	Compile
m com.google.code.gson:gson:2.2.4	Compile

High on the right click on the + key and select the M*odule Dependency* item. In case no other modules are defined, the only module which can be selected is the one just created (**btdatakeymodule**).

Now the structure of the demo project has got two modules:

- **app**: the main module which implements the user interface and the logic of the calls to the **btdatakeymodule.aar** library
- **btdatakeymodule**: the module which contains the interface library with the BTDataKey.

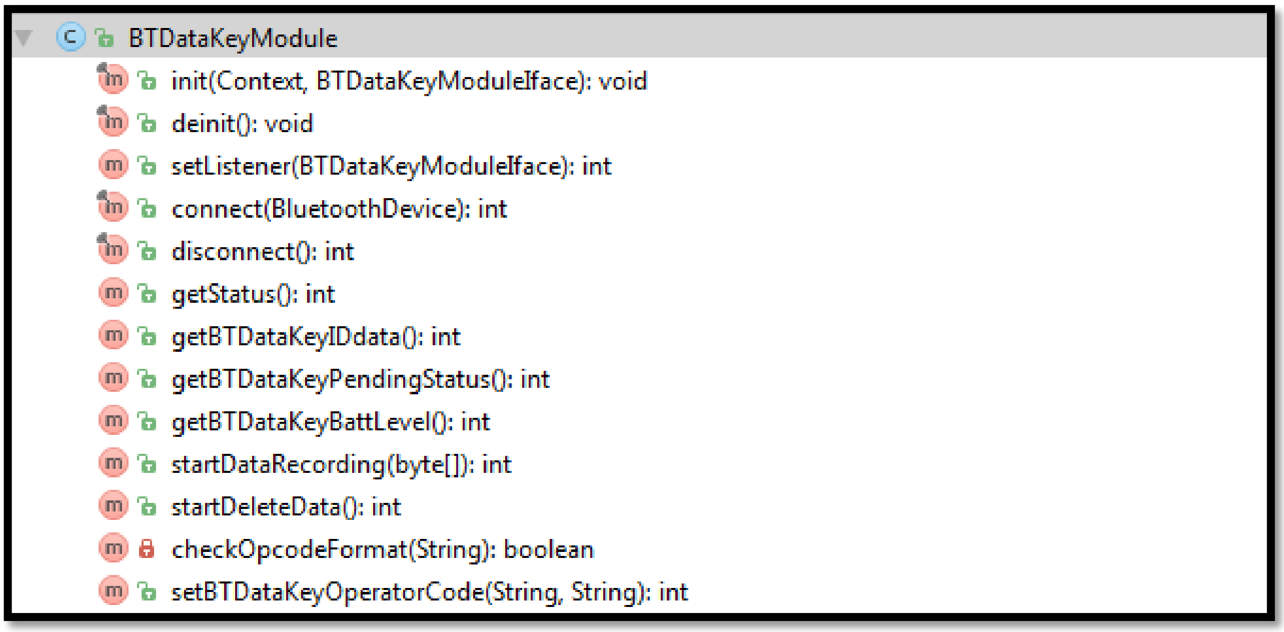

CogesBTDatakeyLibDemo
build
1: Project
2: Structure
Captures
Android
app
manifests
java
res
btdatakeymodule
Gradle Scripts
build.gradle (Project: CogesBTDatakeyLibDemo)
build.gradle (Module: app)
build.gradle (Module: btdatakeymodule)
proguard-rules.pro (ProGuard Rules for app)
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)

Library structure

The library consists of a class which supplies the library interface by means of some public methods (APIs) and a public interface for the callbacks. The component of the library which supplies the APIs is the **BTDataKeyModule** object, the callbacks instead are defined by the **BTDataKeyModuleIface** public interface.

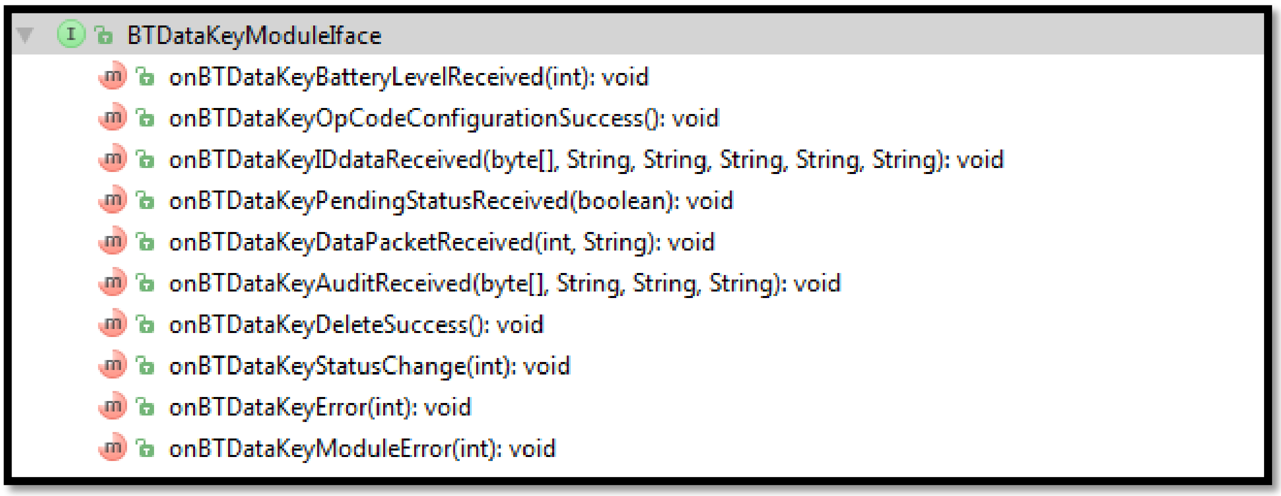
BTDataKeyModule API / Callbacks

The offered APIs are all public methods defined by the `BTDataKeyModule` object:



```
BTDataKeyModule
    init(Context, BTDataKeyModuleIface): void
    deinit(): void
    setListener(BTDataKeyModuleIface): int
    connect(BluetoothDevice): int
    disconnect(): int
    getStatus(): int
    getBTDataKeyIDdata(): int
    getBTDataKeyPendingStatus(): int
    getBTDataKeyBattLevel(): int
    startDataRecording(byte[]): int
    startDeleteData(): int
    checkOpcodeFormat(String): boolean
    setBTDataKeyOperatorCode(String, String): int
```

These methods are used to enable the different functions of the library. The answers to the different commands and the error notifications occurs by means of the callbacks which are defined by a `BTDataKeyModuleIface` public interface:



BTDataKeyModuleIface
onBTDataKeyBatteryLevelReceived(int): void
onBTDataKeyOpCodeConfigurationSuccess(): void
onBTDataKeyIDdataReceived(byte[], String, String, String, String, String): void
onBTDataKeyPendingStatusReceived(boolean): void
onBTDataKeyDataPacketReceived(int, String): void
onBTDataKeyAuditReceived(byte[], String, String, String): void
onBTDataKeyDeleteSuccess(): void
onBTDataKeyStatusChange(int): void
onBTDataKeyError(int): void
onBTDataKeyModuleError(int): void

The different statuses of the library are (on the window from `S_NO_INIT` to `S_BUSY_DELETING`):

```
// Errors returned by onBTDataKeyError()
final public static int ERR_BTDK_NONE                = 0;
final public static int ERR_BTDK_ILLEGAL_STATUS      = 1;
final public static int ERR_BTDK_NOT_RESPONDING      = 2;  // NO STACK RECEIVED
final public static int ERR_BTDK_DATA_REC_TIMEOUT    = 3;  // DDCMP TIMEOUT EXPIRED
final public static int ERR_BTDK_UNLOCKED            = 4;
final public static int ERR_BTDK_OPCODE_SIZE         = 5;
final public static int ERR_BTDK_OPCODE_FORMAT       = 6;
final public static int ERR_BTDK_OPCODE_UNLOCK_FAIL  = 7;
final public static int ERR_BTDK_OPCODE              = 8;
final public static int ERR_BTDK_IRCODE              = 9;
final public static int ERR_BTDK_SERVICE_FAILED      = 10;

// Errors returned by onBTDataKeyModuleError()
final public static int ERR_NONE                     = 0;
final public static int ERR_BLE_CONNECTION           = 1;
final public static int ERR_EVA_DATA_WRITE           = 2;
final public static int ERR_INTERRUPTED_PROCESS      = 3;
final public static int ERR_NOPATH                   = 4;

final public static int S_NO_INIT            = 0; // BLE Service not inited
final public static int S_READY              = 1; // GATT SERVICES DISCOVERED --> READY for Data Recording
final public static int S_BUSY               = 2; // Connecting to BLE Service or GATT Server
final public static int S_CONNECTED          = 3; // Connected to GATT Server
final public static int S_DISCONNECTED       = 4; // Disconnected from GATT Server
final public static int S_BUSY_COLLECTING    = 5; // BUSY during data recording
final public static int S_BUSY_OPCODE        = 6; // BUSY during operator code configuration
final public static int S_BUSY_UNIQUE        = 7; // BUSY during unique data query
final public static int S_BUSY_BATT          = 8; // BUSY during battery level query
final public static int S_BUSY_DELETING      = 9; // BUSY during process of deleting data
final public static int S_BUSY_PENDING       = 10;// BUSY during pending data status query
```

Callbacks of the BTDataKeyModuleIface

BTDataKeyModule Status Change notification

Method:

```
public void onBTDataKeyStatusChanged(int statusCode)
```

Notification method of the status change of the `BTDataKeyModule` module. The possible statuses are listed above.

BTDataKey error notification

Method:

`public void onBTDataKeyError(int errorCode)`

Method which notifies the errors due to the normal operation of the library and of the interface with the BTDataKey:

- `ERR_BTDK_ILLEGAL_STATUS`: it is not notified by callback, but it is one of the error codes returned by the APIs.
- `ERR_BTDK_NOT_RESPONDING`: it is notified when the BTDataKey stops communicating with the library, e.g., after sending a command which needs an answer, as calling `getBTDataKeyIDdata()`
- `ERR_BTDK_DATA_REC_TIMEOUT`: it is notified when the timeout of data receiving expires. The timeout starts when a new data recording begins and it stops when the success or the error is notified.
- `ERR_BTDK_UNLOCKED`: it is notified when the BTDataKey has got an unlocked operator code. In this situation the library will not allow the accounting data recording. First the operator code shall be configured with `setBTDataKeyOperatorCode()`.
- `ERR_BTDK_OPCODE_SIZE`: it is notified when the number of characters of one of the two operator codes with which the `setBTDataKeyOperatorCode()` method is called is not exactly 8.
- `ERR_BTDK_OPCODE_FORMAT`: it is notified when one of the operator codes with which the `setBTDataKeyOperatorCode()`method is called contains some characters which are not digits, e.g., alphabetical or special characters.
- `ERR_BTDK_OPCODE_UNLOCK_FAIL`: it is notified when the `setBTDataKeyOperatorCode()`method fails because the writing of the operator code of the BTDataKey cannot be unlocked. This occurs if the operator code entered for the unlock does not correspond to the code stored in the BTDataKey.
- `ERR_BTDK_OPCODE`: it is not notified by callback, but it is one of the error codes immediately returned by the APIs.
- `ERR_BTDK_IRCODE`: it is not notified by callback, but it is one of the error codes immediately returned by the APIs.
- `ERR_BTDK_SERVICE_FAILED`: it is notified when GATT Service discovery fails either immediately or after a timeout.

BTDataKeyModule error notification

Method:

`public void onBTDataKeyModuleError(int errorCode)`

Method which notifies the errors due to the `BTDataKeyModule` module and caused by the Android operating system:

- `ERR_BLE_CONNECTION`: error due to the use of the Bluetooth Adapter to manage the Bluetooth Low Energy communication.
- `ERR_EVA_DATA_WRITE`: error occurred during the saving of the EVA-DTS file in the **Download/Coges** folder.
- `ERR_INTERRUPTED_PROCESS`: error caused by Android which arbitrarily ended a task of the library while this task was in `wait()` status on a semaphore. This

error may occur when Android is short of resources and it decides to recover some of them by ending some tasks which are not prior for it.

- `ERR_NOPATH`: error caused by failing to create the path **Download/Coges** that is used to store the accounting data received by BTDataKey.

Battery Level notification

Method:

```
public void onBTDataKeyBatteryLevelReceived(int battLevel)
```

Method which notifies the result of the API asking the battery level `getBTDataKeyBattLevel()`.

Parameters:

battLevel – Battery level notified by the BTDataKey with a [0-100] range value.

BTDataKey ID notification

Method:

```
public void onBTDataKeyIDdataReceived(byte[] raw, String serialNum,
                                 String hwId, String hwVer,
                                 String productionDate, String fwVersion)
```

Method which notifies the result of the API asking the identification data of the BTDataKey, i.e., `getBTDataKeyIDdata()`.

Parameters:

raw – The byte stream received from Bluetooth Low Energy and sent by the Key.
serialNum – The serial number of the BTDataKey in string (8 digits) format.
hwId – Coges product identification ("5100" for BTDataKey).
hwVer – The version of the HW revision of Coges product.
productionDate – Date of production.
fwVersion – Version of the FW currently installed in the BTDataKey.

For a specific Key only the *fwVersion* can change in the device life. This value identifies the FW version installed in the Key and it is updated together with the FW, when a new FW is installed.

<u>This library does not allow the FW update of the BTDataKey</u>.

Operator Code notification

Method:

```
public void onBTDataKeyOpCodeConfigurationSuccess()
```

Method which notifies that the change of the operator code occurred and thus the API of operator code change, `setBTDataKeyOperatorCode()`, did not return any error.

Delete success notification

Method:

```
public void onBTDataKeyDeleteSuccess()
```

Method which notifies that the accounting data delete in the BTDataKey was successful. The delete process starts with `startDeleteData()`.

Pending Data Status notification

Method:

```
public void onBTDataKeyPendingStatusReceived(boolean isDataPending)
```

Method which notifies the presence or not of data in the memory of the BTDataKey to which the module is connected. In case there are some data in the Key, the `startDataRecording()` API will start the recovery process of accounting data instead of starting a new recording. If need be, to delete the data, the `startDeleteData()`API can be used.

Parameters:

isDataPending – Boolean value that if it is `true` it notifies the presence of data from a previous recording in the memory of the BTDataKey.

Data Packet notification

Method:

```
public void onBTDataKeyDataPacketReceived(int msgCounter, String msg)
```

Method which notifies the receiving of each DDCMP packet on the module's side. Unless there are setting/communication errors, the receiving of the first packet occurs after the `startDataRecording()`API call.

<u>To avoid the overcharge of the application using the library, the code to be entered in the implementation of this callback shall be as fast as possible.</u>

Parameters:

msgCounter – Index of the packet received. Each recording starts with the first packet (index 0), but the number of packets can be variable.
msg – The contents of the data packet; that is a partial fragment of the EVA-DTS file which the BTDataKey sends to the module.

Data Recording notification

Method:

```
public void onBTDataKeyAuditReceived(byte []raw, String dataKeyMode,
                                     String auditText, String auditPath)
```

Method which notifies the receiving of the accounting data from the BTDataKey. When this method is called, in the BTDataKey the data has been deleted and the Key is ready for a new recording.

Parameters:

raw – The byte stream contained in the DDCMP data packets received from Bluetooth Low Energy and sent to the Key.
dataKeyMode – Mode by which the BTDataKey recorded the data from the System: **CL** (Contactless), or **IR** (infrared port).
auditText – *raw* conversion in UTF-8 format string.
auditPath – String which identifies the path where the *auditText* text file was stored (**Download/Coges/**<*file name*>).

API of the BTDataKeyModule

Init / Deinit

Method:

```
public void init(Context aContext, BTDataKeyModuleIface listener)
```

It initializes the library, creates the **Download/Coges** folder if it does not already exist.

Parameters:

aContext – The Activity which instanciates the `BTDataKeyModule` object.
listener – The object which extends the `BTDataKeyModuleIface` interface. This object will permit the notification of the APIs results or of some asynchronous errors by the callbacks defined by the `BTDataKeyModuleIface` interface.

The **Download/Coges** folder is used to store the recording file extracted from the BTDataKey.

<u>At the end of recording the file is not automatically deleted</u>.

Method:

```
public void deinit()
```

It frees the resources used by the library, by disconnecting the communication with the Bluetooth Low Energy device, if active.

Method:

```
public int setListener(BTDataKeyModuleIface listener)
```

It registers the new object which will answer to the library notifications. This method can be called in all statuses, but `S_READY` and `S_CONNECTED`.

Parameters:

listener – The object which extends the `BTDataKeyModuleIface`.

The first operation to be executed by the user class (i.e. `ScanDevicesActivity`) of the library to connect to the Key is to initialize the library. The initialization requires the

“Context” of who is calling, in order to get a `BluetoothAdapter`, and manage the Bluetooth communication by means of an internal service of the library.

The initialization requires the listener too, that is the object which implements the `BTDataKeyModuleIface` interface, the listener anyway can be initialized at `null`, and configured afterwards with the `setListener()` method. Many of the library APIs however return `ERR_BTDK_ILLEGAL_STATUS` if they are called while the listener is not configured.

As soon as the `BTDataKeyModule` object is instantiated, the library finds itself in the `S_NO_INIT` status. In this status it is possible to call only the `init()`method. After calling `init()`, the module enters the `S_BUSY` status till the Bluetooth service of the library does not succeed in connecting and the library enters the `S_DISCONNECTED` status.
Now the user can start a connection to the BTDataKey.

Viceversa `deinit()` unregister the service.

Get Status

Method:

```
public int getStatus()
```

It immediately returns to the current status of the Key. The possible statuses are reported in the Library structure section.

Connect / Disconnect

Method:

```
public int connect(BluetoothDevice aDevice)
```

It starts a connection to a Coges BTDataKey. This method can be called only in the `S_READY` status.

Parameters:

aDevice – Bluetooth Low Energy device to be connected to.

Method:

```
public int disconnect()
```

It ends the connection to the Bluetooth device which is connected. This method can be called only in the `S_READY`, or `S_CONNECTED` statuses.

These methods require the connection/ disconnection to a BLE device. The connection method can be called only when the library is in `S_DISCONNECT` status. If the connection request starts successfully, the library enters the `S_BUSY` status. When the connection to a compatible BTDataKey occurs, the module changes the status and enters the `S_CONNECTED` one. This status is not notified, as the module is enabled to recording only after receiving the operator code status (locked/ unlocked) which is stored in the Key.

When the connection is carried out, that is when module passing to the `S_CONNECTED` status, the request of the Operator Code occurs automatically.

If the operator code of the Key is UNLOCKED, an error is notified by callback, otherwise the `S_READY` status is notified, which enables all the Key functions.

Get Key Identification data / Get Battery Level

Method:

`public int getBTDataKeyIDdata()`

It requests the identification data of the connected BTDataKey. The Key enters the `S_BUSY_UNIQUE` status and the result is notified by `onBTDataKeyIDdataReceived()`, when the status returns to `S_READY`. This method can be called only in the `S_READY` status.

Method:

`public int getBTDataKeyBattLevel()`

It requests the battery level of the connected BTDataKey. The Key enters the `S_BUSY_BATT` status and the result is notified by `onBTDataKeyBatteryLevelReceived()`, when the status returns to `S_READY`. This method can be called only in the `S_READY` status.

By means of the `getBTDataKeyIDdata()` and `getBTDataKeyBattLevel()`methods, in the `S_READY` status the user class of the module can ask the identification data to the Key (two different BLE services are necessary to get them: Unique Data Char of Coges Configurations service and DFU frame Char of Coges DFU service).

Set BTDataKey Operator Code

Method:

`public int setBTDataKeyOperatorCode(String opcodeOld, String opcodeNew)`

Parameters:

opcodeOld – The operator code currently stored in the BTDataKey.
opcodeNew – The new operator code which will replace the *opcodeOld* in the Key memory.

It changes the operator code in the BTDataKey. If successful `onBTDataKeyOpCodeConfigurationSuccess()`will be notified, otherwise an error is notified. This method can be called only in the `S_READY` status.

Pending Data Check

Method:

`public int getBTDataKeyPendingStatus()`

This API permits to request the status of the data memory to the BTDataKey currently connected. In case the Key answers to the command, the status of the data in the Key is asynchronously reported by a boolean parameter with the callback `onBTDataKeyPendingStatusReceived()`. If there are data in the memory of the Key, the parameter will be valued `true`, otherwise it will be `false`. This method can be called only in the `S_READY` status.

Delete Data

Method:

```
public int startDeleteData()
```

This API starts the delete process of data in the memory of the BTDataKey which is currently connected. This method can be called only in the `S_READY` status.

Remember that the accounting data in the Key are not maintained in its memory in case the sending process ends successfully. For a detailed guide of how to use the BTDataKey see the instructions of the device.

Data Recording

Method:

```
public int startDataRecording(byte passwordIR[])
```

Parameters:

passwordIR – The security code used for extracting the data in IR mode. This code is ignored if the recording mode is Contactless (CL).

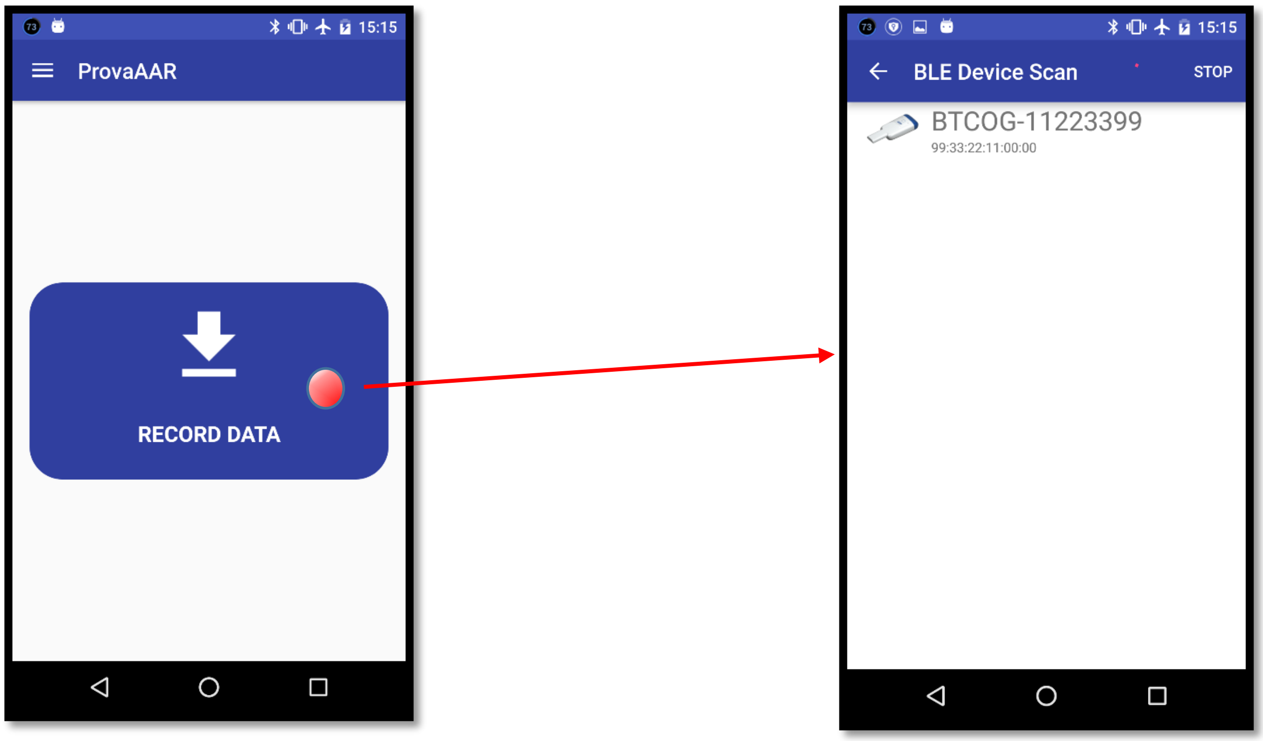
This is the main API of the library, in other words it enables the recording process from the Key. If the recording process is successful, that is a EVA-DTS file with the accounting data sent by the system to which the BTDataKey is connected is received, the file is saved in the **Download/Coges** folder and the success notification is sent by callback: `onBTDataKeyAuditReceived()`.

Remember that the notified data in case of success are: the byte stream of the data contained in the DDCMP packets sent by the BTDataKey, the recording mode by which the data are extracted (**IR** or **CL**), the conversion of the byte stream into a string, the path where the EVA-DTS file was stored (**Download/Coges**/<*file name*>).

In case of failure, a corresponding error is sent by the specific callbacks.

Demo User Interface

The demo application is very simple and austere, as its aim is to explain how to use the APIs in the `BTDataKeyModule` object to get the identification information from the Key and carry out the data collection. The Activity which implements the user interface which uses the library for the data collection is the one executing the device selection (`ScanDevicesActivity`). This Activity starts by an "Intent" as soon as the RECORD DATA button is clicked.



The library APIs are called by directly acting on the `BTDataKeyModule` object, while the answers are managed in asynchronous way by callbacks. To react to the callbacks it is necessary to register a Listener, that is an object implementing a specific interface: `BTDataKeyModuleIface`.

`BTDataKeyModule` is a private object of the `ScanDevicesActivity` class, therefore as soon as the RECORD DATA button is clicked, this object is instantiated.
The connection to the Key occurs later, when the device to be connected to is selected.

The listener of the `BTDataKeyModule` object is the `ScanDevicesActivity`, that is the same Activity which implements the `BTDataKeyModuleIface` interface.

As soon as the device is selected from the list obtained by the `ScanDevicesActivity`, in the `onItemClick()` method, the following:

`mBTDataKeyModule.init(this, ScanDevicesActivity.this)` is called;

On the callback for managing the status change of the module (`onBtDataKeyStatusChange`), when the status becomes `S_READY`, the request of the Key identification data is entered. This API call IS NOT COMPULSORY, as the module

`S_READY` status already enables all the library functions, among which the accounting data collection API: `startDataRecording()`.

In our example, in case of error a message is displayed, in successful case, instead, in the implementation of the `onBTDataKeyIDdataReceived()`callback, the Key battery level is requested. This request too is made only for information purposes, and it is not necessary for collecting the data.

In case of error a message is displayed, in successful case, instead, the library calls `onBTDataKeyBatteryLevelReceived()`. In the implementation of this last callback a request of accounting data reading is entered, that is the call to the `startDataRecording()`method.

In case the data arrive, the `onBTDataKeyAuditReceived()`callback is called, to which is connected the display of an alert dialog notifying the data saving. In case of error a message is reported.
This is only an example to show how to use the library functions. The user is free to use the library as he prefers.