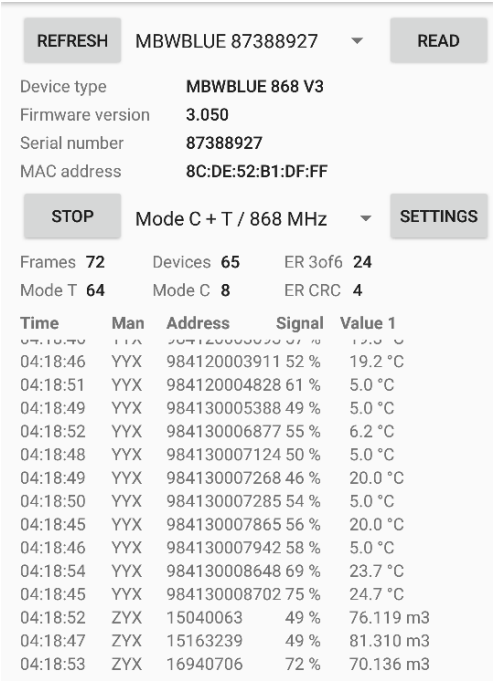
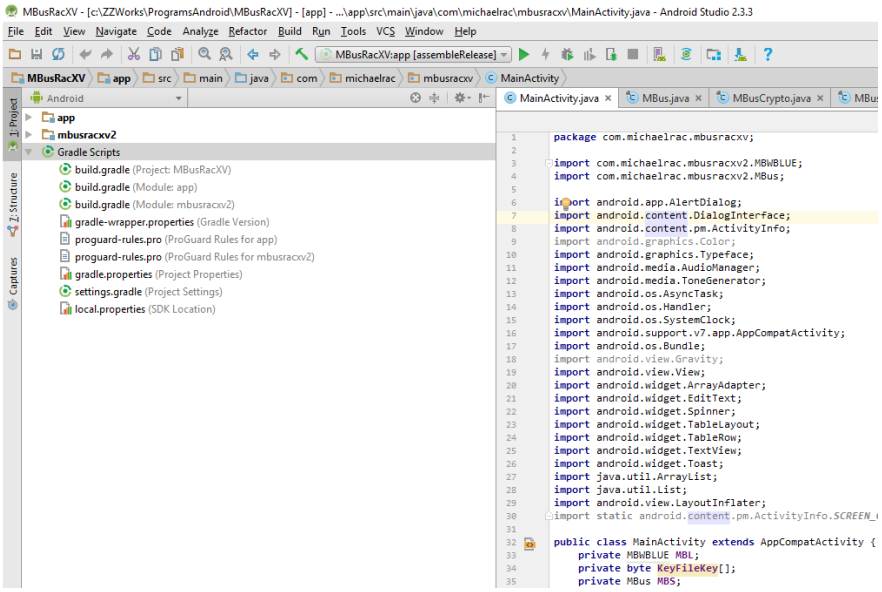
MBusRacXV2
For Android





Manual

01/2022

Michael Rac GmbH
Am Hirtenfeld 51
91522 Ansbach
GERMANY

Email: mrg@michaelrac.com

Table of Contents

MBusRacXV2 for Android

Introduction

MBusRacXV2 for Android is a function library for reading and interpreting wireless M-Bus radio frames according to EN13757-3, EN13757-4 and OMS, respectively.

It is written using Android Studio and is distributed in the form of an Android Archive file (mbusracxv2.aar). For integration of AAR files in your Android project please check the respective internet pages.

The library is compiled for Android versions 4.03 (API15) and higher for phones and tablets.

To demonstrate the use of the library a small demo application (MBWBLUE_DEMO.apk) with source code is available.

The library is mainly developed for reading MBWBLUE wireless M-Bus receiver devices but it is also possible to only use the M-Bus radio frame interpretation part of it with other receivers. For more information about the MBWBLUE: http://www.michaelrac.com/download/MBWBLUE_DeviceManual.pdf

The mbusracxv2.aar library can be royalty free integrated in your projects and distributed with your projects, as long as the mbusracxv2.aar file is not modified, decompiled or reverse engineered.

It is in general assumed that the user is familiar with wireless M-Bus radio frames and the definitions used therein. More information about wireless M-Bus can be found in EN13757-x, on the OMS web page (http://oms-group.org) and other sources of the internet.

Items Supplied

The MBusRacXV2 package contains the following files:

- MBusRacXV2_Manual.pdf	This documentation
- mbusracxv2.aar	The Android Archive to integrate in your projects
- MBWBLUE_DEMO.apk	The ready compiled demo application
- MBWBLUE_DEMO.ZIP	The demo application Android Studio project including the source code

Acronyms and Abbreviations

Abbreviation	Explanation
AAR	Android Archive file extension
ACC	Access Number
AES	Advanced Encryption Standard
AES CBC	AEC Cipher Block Chaining
AES CTR	AES Counter Mode
AFL	Authentication and Fragmentation Layer
APK	Android Package file extension
C Field	Control Field
CI Field	Control Information Field
CRC	Cyclic Redundancy Check
DIB	Data Information Block
DIF	Data Information Field
DIFE	Data Information Field Extension
IV	Initialization Vector
ELL	Extended Link Layer
LL	Link Layer
M-Bus	Meter-Bus (wired or wireless)
MAC	Message Authentication Code
RSSI	Received Signal Strength Indication
TPL	Transport Layer
USB	Universal Serial Bus
VIB	Value Information Block
VIF	Value Information Field
VIFE	Value Information Field Extension

Available Classes

The MBusRacXV2 library contains two main public classes:

- MBWBLUE methods and properties reading MBWBLUE devices
- MBus methods and properties for radio frame interpretation

The MBWBLUE is a wireless M-Bus receiver with Bluetooth interface. The respective class contains methods and properties for connecting and reading this device. In general, the user must establish the Bluetooth pairing manually using the Android operating system. Once the MBWBLUE is paired with the Android device the MBWBLUE class provides everything necessary to get information about the paired device, establish a connection and read it out.

The MBus class provides methods and properties for deciphering and interpreting wireless M-Bus radio frames. It contains also a radio key handling database for storing radio frame keys on the Android device (in ciphered form).

An example for invoking the two classes is shown below:

```
package com.michaelrac.mbwblue_demo;

import com.michaelrac.mbusracxv2.MBWBLUE;          // import class MBWBLUE
import com.michaelrac.mbusracxv2.MBus;             // import class MBus


public class MainActivity extends AppCompatActivity {

    private MBWBLUE MBL;                           // MBWBLUE class declaration -> MBL
    private MBus MBS;                              // MBus class declaration -> MBS
    private byte KeyFileKey[];                     // KeyFileKey is used to hold the
                                                   // ciphering key for storing the radio key
                                                   // database on the Android device


    @Override
    protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);

       MBL = new MBWBLUE("");              // invoke MBWBLUE class (parameter must be “”)

       KeyFileKey=new byte[16];            // define you own key for storing radio frame keys
        KeyFileKey[0]=(byte)0x36;KeyFileKey[1]=(byte)0x98;KeyFileKey[2]=(byte)0xBF;
        KeyFileKey[3]=(byte)0x98;KeyFileKey[4]=(byte)0xAC;KeyFileKey[5]=(byte)0xED;
        KeyFileKey[6]=(byte)0x4E;KeyFileKey[7]=(byte)0xC3;KeyFileKey[8]=(byte)0xD9;
        KeyFileKey[9]=(byte)0x72;KeyFileKey[10]=(byte)0x21;KeyFileKey[11]=(byte)0xAE;
        KeyFileKey[12]=(byte)0x6B;KeyFileKey[13]=(byte)0x80;KeyFileKey[14]=(byte)0xFA;
        KeyFileKey[15]=(byte)0x23;
        MBS = new MBus(getApplicationContext(),KeyFileKey);
```

MBWBLUE class: methods

This chapter lists all available MBWBLUE class methods. All methods are blocking methods; they return only after having finished the respective task. **It is advised to invoke all Bluetooth communication methods (BTConnect, BTCMDxxxxx) only in AsyncTasks to not block the Android main UI task.**

MBWBLUE(String DTypeName)

Constructor of the MBWBLUE class.

DTypeName must be set to ""

void Destroy()

Destructor of the MBWBLUE class. Should be called after having finished using the MBWBLUE class since some clean-up work is done.

void AvailableDevicesList()

Searches for and creates a list of all paired MBWBLUE devices.

int AvailableDevicesCount()

After having invoked AvailableDevicesList() this method returns the number of paired MBWBLUE devices.

String AvailableDevicesName(int Indx)

After having invoked AvailableDevicesList() this method returns the device name (e.g. MBWBLUE 12345678) of the MBWBLUE devices at list index Indx. If Indx is not valid the method returns "".

Indx: Entry index of the available MBWBLUE device list.

String AvailableDevicesAddress(int Indx)

After having invoked AvailableDevicesList() this method returns the Bluetooth MAC address (e.g. 8C:DE:52:B1:DF:FF) of the MBWBLUE devices at list index Indx. If Indx is not valid the method returns "".

Indx:	Entry index of the available MBWBLUE device list.

String AvailableDevicesGetAddressFromName(String BTName)

After having invoked AvailableDevicesList() this method searches the available MBWBLUE device list to get the MAC address of a device from the given device name.

BTName:	device name (e.g. MBWBLUE 12345678) of the MBWBLUE device to get the MAC address (e.g. 8C:DE:52:B1:DF:FF) from.

boolean BTConnect (String MACAdr)

Establishes the connection to the MBWBLUE device with the given MACAdr. Return value is true if successful and false if not successful.

MACAdr:	MAC address (e.g. 8C:DE:52:B1:DF:FF) of the MBWBLUE

void BTDisConnect ()

Shuts down the connection to the MBWBLUE device.

boolean BTIsConnected()

Checks if a Bluetooth connection is established. Return value is true if the connection is established, false otherwise.

boolean BTCMDRequestFirmwareVersionLeg()

Sends the command to request the firmware version of the MBWBLUE device. This method may only be invoked if a Bluetooth connection is established. If the return value is true (successful) the following properties are updated:

BTFreq:	434 = MBWBLUE is a 434 MHz device 868 = MBWBLUE is a 868 MHz device
BTGen3:	0 = MBWBLUE is a generation 2 device 1 = MBWBLUE is a generation 3 device
BTFWVersion	Firmware version of the device (e.g. "3.050")
BTDevType	MBWBLUE device type, may contain: "MBT1BLUE 868" "MBT1BLUE 434" "MBWBLUE 868 V2" "MBWBLUE 434 V2" "MBWBLUE 868 V3" "MBWBLUE 434 V3"

boolean BTCMDReadROMLeg()

Sends the command to request the serial number of the MBWBLUE device. This method may only be invoked if a Bluetooth connection is established. If the return value is true (successful) the following property is updated:

BTSerialNumber	Serial number of the MBWBLUE (e.g. "87388927”)

boolean BTCMDReceiverModeLeg()

Sends the command to set the receiver mode of the MBWBLUE device and clears additionally all buffered radio frames and statistical values, if available. This method may only be invoked if a Bluetooth connection is established. The return value is true if the command has been successfully sent.
Before invoking this method the following properties must be set correctly:

BTCurReceiverFreq:	434	set reception mode to 434 MHz
	868	set reception mode to 868 MHz
BTCurReceiveMode	0	set reception to receiving mode T and C frames
	1	set reception to receiving mode S frames

boolean BTCMDClearRadioFrameBuffer()

Sends the command to clear all buffered radio frame and statistical values, if available. This method may only be invoked if a Bluetooth connection is established. The return value is true if the command has been successfully sent.

boolean BTCMDReadRadioLeg()

Sends the command to read the next received radio frame from the MBWBLUE. This method may only be invoked if a Bluetooth connection is established. The return value is true if the command has been successfully sent.

This method must be invoked if the MBWBLUE is a generation 2 device: **BTGen3 = 0**

To decipher and interpret the radio frame the MBus class's Interpret() method must be invoked in the following way:

MBS.Interpret(MBL.BTInBuffer,**12,241**,false);

boolean BTCMDReadRadio()

Sends the command to read the next received radio frame from the MBWBLUE. This method may only be invoked if a Bluetooth connection is established. The return value is true if the command has been successfully sent.

This method must be invoked if the MBWBLUE is a generation 3 device: **BTGen3 = 1**

To decipher and interpret the radio frame the MBus class's Interpret() method must be invoked in the following way:

MBS.Interpret(MBL.BTInBuffer,**5,256**,false);

boolean BTCMDSwitchOff()

Sends the command to switch the MBWBLUE off. This method may only be invoked if a Bluetooth connection is established. The return value is true if the command has been successfully sent.

boolean BTCMDMBWBLUEFirmwareUpdate()

Sends the command to start a firmware update of the MBWBLUE. This method may only be invoked if a Bluetooth connection is established. The return value is true if the command has been successfully sent.

This method takes about 120 seconds to complete. By checking the property BTProgress (0 … 100) it is possible to track the progress.

With the MBWBLUE it is only possible to write a new firmware to the device. Even though you may invoke this method writing the same or an older firmware version to the MBWBLUE, the MBWBLUE is ignoring older or same versions of the firmware. Therefore, it is advised to first check the firmware version contained in this library and the firmware version of the MBWBLUE device:

BTUpdateFWVersion	this property contains the firmware version of the firmware contained in this library.
BTFWVersion	after having invoked the method BTCMDRequestFirmwareVersionLeg() this property contains the firmware version of the MBWBLUE device.

If the firmware version in BTUpdateFWVersion is higher than the firmware version in BTFWVersion you may start the firmware update.

The firmware update is only available for generation 3 devices: **BTGen3 = 1**

MBWBLUE class: properties

boolean BTAdapterOK

After construction of the MBWBLUE class this property contains:

false:	Bluetooth not available on Android device
true:	Bluetooth available and initialised

String BTUpdateFWVersion

Contains the firmware version (e.g. "3.050") of the firmware contained in this library (for firmware updates).

byte BTOutBuffer[]
int BTOutBufferLen
byte BTInBuffer[]
int BTInBufferLen

Internal buffers and buffer lengths for communicating with the MBWBLUE device. It is not necessary to use these buffers directly since all communication is done by methods. However, BTInBuffer is a parameter to be passed to the MBWBLUE class Interpretation method.

int BTGen3

After having invoked BTCMDRequestFirmwareVersionLeg() this property contains:

BTGen3	0 = MBWBLUE is a generation 2 device 1 = MBWBLUE is a generation 3 device

int BTFreq

After having invoked BTCMDRequestFirmwareVersionLeg() this property contains:

BTFreq	434 = MBWBLUE is a 434 MHz device 868 = MBWBLUE is a 868 MHz device

int BTCurReceiveMode

This property must be set correctly before invoking BTCMDReadRadioLeg() or BTCMDReadRadio().

BTCurReceiveMode	0 = mode T + C 1 = mode S

int BTCurReceiverFreq

This property must be set correctly before invoking BTCMDReadRadioLeg() or BTCMDReadRadio().

BTCurReceiverFreq	434 = 434 MHz receive mode 868 = 868 MHz receive mode

int BTTelgRSSI

After having invoked BTCMDReadRadioLeg() or BTCMDReadRadio() this property contains the signal strength (RSSI) of the received radio frame in percent (0…100).

int BTStatisticsGoodFrameTS

After having invoked BTCMDReadRadio() this property contains the number of correctly received radio frames in either mode T or mode S. This statistical value is reset to zero after having invoked BTCMDReceiverModeLeg() or BTCMDClearRadioFrameBuffer().
This property is only valid for generation 3 MBWBLUE devices (BTGen3=1).

int BTStatisticsGoodFrameCA

After having invoked BTCMDReadRadio() this property contains the number of correctly received radio frames in mode C format A. This statistical value is reset to zero after having invoked BTCMDReceiverModeLeg() or BTCMDClearRadioFrameBuffer().
This property is only valid for generation 3 MBWBLUE devices (BTGen3=1).

int BTStatisticsGoodFrameCB

After having invoked BTCMDReadRadio() this property contains the number of correctly received radio frames in mode C format B. This statistical value is reset to zero after having invoked BTCMDReceiverModeLeg() or BTCMDClearRadioFrameBuffer().
This property is only valid for generation 3 MBWBLUE devices (BTGen3=1).

int BTStatisticsError36

After having invoked BTCMDReadRadio() this property contains the number of received radio frames with 3 of 6 coding error. This statistical value is reset to zero after having invoked BTCMDReceiverModeLeg() or BTCMDClearRadioFrameBuffer().
This property is only valid for generation 3 MBWBLUE devices (BTGen3=1).

int BTStatisticsErrorCRC

After having invoked BTCMDReadRadio() this property contains the number of received radio frames with CRC error. This statistical value is reset to zero after having invoked BTCMDReceiverModeLeg() or BTCMDClearRadioFrameBuffer().
This property is only valid for generation 3 MBWBLUE devices (BTGen3=1).

String BTFWVersion

After having invoked BTCMDRequestFirmwareVersionLeg() this property contains:

BTFWVersion	Firmware version of the device (e.g. "3.050")

String BTDevType

After having invoked BTCMDRequestFirmwareVersionLeg() this property contains:

BTDevType	MBWBLUE device type; may contain: "MBT1BLUE 868" "MBT1BLUE 434" "MBWBLUE 868 V2" "MBWBLUE 434 V2" "MBWBLUE 868 V3" "MBWBLUE 434 V3"

String BTSerialNumber

After having invoked BTCMDReadROMLeg() this property contains:

BTSerialNumber: Serial number of the MBWBLUE (e.g. "87388927”)

int BTProgress

This property is only used during firmware update and contains the progress of the firmware update in percent (0% to 100%).

MBus class: methods

This chapter lists all general MBus class methods. There is a separate chapter for the methods of handling the radio frame deciphering keys.

MBus (Context MAContext, byte KFKey[])

Constructor of the MBus class.

MAContext	the context of the MainActivity of the application (use e.g. getApplicationContext())
KFKey	a 16 byte array containing a developer specific ciphering key. This key is used to cipher the database of stored radio keys (see next chapter) before it is written to the Android application storage space. Since the radio key database is stored on the Android device it can be read by unauthorized persons. Use a strong secret key to protect the radio key database from unauthorized access.

boolean Interpret(byte BIn[], int BInStart, int BInLen, boolean DoNotDecipher)

This method interprets the radio frame in BIn[] starting from byte index BInStart. BInLen gives the total number of bytes in BIn[]. If you do not want to decipher the radio frame the parameter DoNotDecipher must be set to true; usually it should always contain false.

BIn[]	byte array containing the received radio frame	
BInStart	byte index in BIn[] at which the received radio frame starts (index of length byte of the radio frame)	
BInLen	total number of bytes in BIn[]	
DoNotDecipher	false	try to decipher radio frame
	true	do not try to decipher radio frame

If the return value is true all properties of the MBus class are updated and are containing all information about the radio frame. If the return value is false the properties of the MBus class are invalid.
This is equivalent to checking the property IsValid (IsValid = true, radio frame properties are valid, IsValid = false, radio frame properties are not valid).

MBus class: crypto methods

The MBus class does not only contain the deciphering and interpretation of radio frames but is also capable of managing a set of radio frame deciphering keys. Depending on how ciphering is used there are four different types of radio keys. Radio keys are stored in a database within the library.

It is possible to write this database to the application specific Android storage space on the Android device. This is done by ciphering it using the KFKey[] parameter of the MBus class. However, if you do not want the MBusRacXV2 library to manage the radio key storage you may also always set all radio keys at the start of the library and do not save the keys to the Android device.

Since there are different strategies on how to cipher radio frames there are four radio key types.

KeyType	3	AES 128 bit keys together with the address of the radio frame. In this case the MBus class searches for the radio frame address in the key database. If there is an entry with the radio frame address the respective key is used to decipher the radio frame. Multiple entries with the same radio frame address are possible. A maximum of 933 keys of KeyType=3 can be stored.
KeyType	2	AES 128 bit keys together with a manufacturer code. In this case MBus class searches for the manufacturer code of the radio frame in the key database. If there is an entry with the manufacturer code the respective key is used to decipher the radio frame. Multiple entries with the same manufacturer code are possible. A maximum of 50 keys of KeyType=2 can be stored.
KeyType	1	General 64 bit keys. If the radio frame is ciphered using an older ciphering scheme with 64 bit keys, the library tries all 64 bit keys one after another until the radio frame is deciphered. A maximum of 5 keys of KeyType=1 can be stored.
KeyType	0	General AES 128 bit keys. The library tries all AES 128 bit keys one after another until the radio frame is deciphered. A maximum of 10 keys of KeyType=0 can be stored.

Since the radio frame deciphering is a time consuming process care must be taken on how and how many radio keys should be configured in the library.

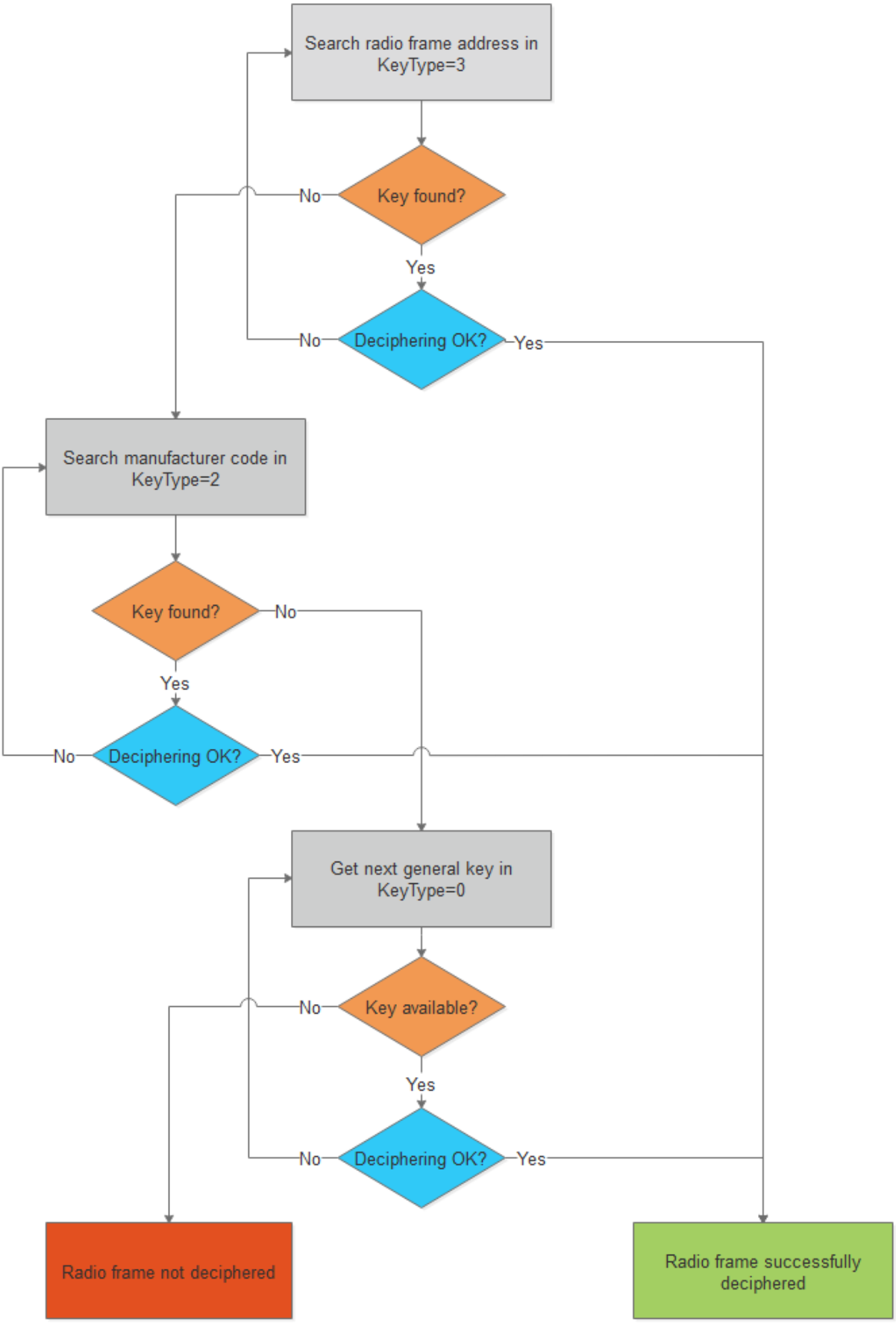
If you are using radio device specific keys (KeyType=3) you should e.g. configure the radio keys project wise. That is only those radio device keys used for the current reading tour are configured. For the next tour you reconfigure all radio keys.

If specific manufacturers use specific keys for all their radio devices it is advised to configure manufacturer specific keys (KeyType=2).

If you are configuring general keys (KeyType=0) you should configure as few keys as possible since the library is trying to decipher radio frames using one general key after another until a radio frame was successfully deciphered or until there are no more keys available. With many general keys, this process may consume a lot of time (~ seconds per radio frame, depending on Android device).

The general sequence on how the library tries to find the correct key for a received radio frame is shown in the next picture.

First the radio address specific radio keys are searched for a match, then the manufacturer code specific radio keys are searched for a match and at last general keys are tried one after another.


Search radio frame address in KeyType=3
Key found?
No
Yes
Deciphering OK?
No
Yes
Search manufacturer code in KeyType=2
Key found?
No
Yes
Deciphering OK?
No
Yes
Get next general key in KeyType=0
Key available?
No
Yes
Deciphering OK?
No
Yes
Radio frame not deciphered
Radio frame successfully deciphered

void CryptoKeysWrite()

Writes the configured radio keys to the application specific storage space on the Android device. In the constructor of the MBus class these storage space is read and all keys imported to the library.
If you want to keep the configured radio key stored on the Android device you should call this method after any modification on the radio keys.
If you do not want to store the configured radio keys on the Android device you should not call this method.

int CryptoKeysGetKeyCount(int KeyType)

Returns the number of configured keys for the specified key type.

KeyType	0,1,2,3

String CryptoKeysGetKey(int KeyType, int KeyIndx)

Returns the configured key.

KeyType	0,1,2,3
KeyIndx	0 to (CryptoKeysGetKeyCount(int KeyType)-1)

boolean CryptoKeysDelete(int KeyType, int KeyIndx)

Deletes a configured key.

KeyType	0,1,2,3
KeyIndx	0 to (CryptoKeysGetKeyCount(int KeyType)-1)

boolean CryptoKeysAddOrModify(int KeyType, int KeyIndx, String KeyStr, String Key2Str)

Adds or modifies an entry of the radio key database. If you want to add a key, set the KeyIndx to -1. If you want to edit a key, set the KeyIndx to the KeyIndx to modify.

KeyType	0,1,2,3	
KeyIndx	-1 (add) or 0 to (CryptoKeysGetKeyCount(int KeyType)-1) (edit)	
KeyStr	the radio key (e.g. "000102030405060708090A0B0C0D0E0F“)	
Key2Str	depending on KeyType	
	KeyType=0	"" (empty)
	KeyType=1	"" (empty)
	KeyType=2	3 characters manufacturer code, e.g. "XYZ"
	KeyType=3	8 characters address, e.g. "12345678"

MBus class: properties

The following properties contain information about the interpreted radio frame. They are only valid after having invoked the Interpret method and if the property IsValid is true. Otherwise, these properties are containing arbitrary values.

String LibVersion

Contains the program version of the library, e.g. "1.00". This property is always valid (independent from IsValid).

boolean IsValid

Contains true after having invoked Interpretation if the radio frame was successfully interpreted, otherwise false. The properties of MBus class are only valid if IsValid is true.

int RecSecond
int RecMinute
int RecHour
int RecDay
int RecMonth
int RecYear

Contain date and time of radio frame reception.

int LField

Contains the length field of the radio frame.

int CField

Contains the C-field of the radio frame.

String ManFieldLL
String ManFieldTPL

Contain the manufacturer field of the link layer, respectively transport layer of the radio frame. If there is only a link layer manufacturer field, ManFieldTPL is set equal to ManFieldLL. Usually, only ManFieldTPL is of interest.

String AdrFieldLL
String AdrFieldTPL

Contain the address field of the link layer, respectively transport layer of the radio frame. If there is only a link layer address field, AdrFieldTPL is set equal to AdrFieldLL. Usually, only AdrFieldTPL is of interest.

int VerFieldLL
int VerFieldTPL

Contain the version field of the link layer, respectively transport layer of the radio frame. If there is only a link layer version field, VerFieldTPL is set equal to VerFieldLL. Usually, only VerFieldTPL is of interest.

int DevFieldLL
int DevFieldTPL

Contain the device type field of the link layer, respectively transport layer of the radio frame. If there is only a link layer device type field, DevFieldTPL is set equal to DevFieldLL. Usually, only DevFieldTPL is of interest. For an interpretation of the device type field please check EN13757-3.

int ELLType

Contains the type of the extended link layer.

0	no extended link layer
1	extended link layer with CI=0x8C
2	extended link layer with CI=0x8D
3	extended link layer with CI=0x8E
4	extended link layer with CI=0x8F

int ELLCC

Contains the communication control field of the extended link layer as complete byte. (only if ELLType = 1, 3).

int ELLCC_RepAccess

Contains the repeated access field of the communication control field of the extended link layer (only if ELLType = 1, 3).

int ELLCC_Accessib

Contains the accessibility field of the communication control field of the extended link layer (only if ELLType = 1, 3).

int ELLCC_Priority

Contains the priority field of the communication control field of the extended link layer (only if ELLType = 1, 3).

int ELLCC_HopCtr

Contains the hop counter field of the communication control field of the extended link layer (only if ELLType = 1, 3).

int ELLCC_Synchr

Contains the synchronous field of the communication control field of the extended link layer (only if ELLType = 1, 3).

int ELLCC_Delay

Contains the delay field of the communication control field of the extended link layer (only if ELLType = 1, 3).

int ELLCC_BiDi

Contains the bidirectional communication field of the communication control field of the extended link layer (only if ELLType = 1, 3).

int ELLACC

Contains the access number field of the extended link layer as complete byte (only if ELLType = 1, 3).

int ELLSN

Contains the session number field of the extended link layer as complete integer (only if ELLType = 2, 4).

int ELLSN_Enc

Contains the encryption field of the session number field of the extended link layer (only if ELLType = 2, 4).

int ELLSN_Time

Contains the time field of the session number field of the extended link layer (only if ELLType = 2, 4).

int ELLSN_Session

Contains the session field of the session number field of the extended link layer (only if ELLType = 2, 4).

int ELLSN_CRC

Contains the CRC field of the extended link layer (only if ELLType = 2, 4).

String ELLMan

Contains the manufacturer field of the extended link layer (only if ELLType = 3, 4).

String ELLAdr

Contains the address field of the extended link layer (only if ELLType = 3, 4).

int ELLVer

Contains the version field of the extended link layer (only if ELLType = 3, 4).

int ELLDev

Contains the device type field of the extended link layer (only if ELLType = 3, 4). For an interpretation of the device type field please check EN13757-3.

int AFLCI

Contains the CI field of the authentication and fragmentation layer. If this property contains 0x90, there is an authentication and fragmentation layer, otherwise this property contains 0x00 and there is no authentication and fragmentation layer.

int AFLLen

Contains the length field of the authentication and fragmentation layer (only if AFLCI=0x90).

int AFLFCL

Contains the fragmentation control field of the authentication and fragmentation layer as complete word (only if AFLCI=0x90).

int AFLFCL_FID

Contains the fragment ID field of the fragmentation control field of the authentication and fragmentation layer (only if AFLCI=0x90).

int AFLFCL_KIP

Contains the key information present field of the fragmentation control field of the authentication and fragmentation layer (only if AFLCI=0x90).

int AFLFCL_MACP

Contains the MAC present field of the fragmentation control field of the authentication and fragmentation layer (only if AFLCI=0x90).

int AFLFCL_MCRP

Contains the message counter field present field of the fragmentation control field of the authentication and fragmentation layer (only if AFLCI=0x90).

int AFLFCL_MLP

Contains the message length field present field of the fragmentation control field of the authentication and fragmentation layer (only if AFLCI=0x90).

int AFLFCL_MCLP

Contains the message control field present field of the fragmentation control field of the authentication and fragmentation layer (only if AFLCI=0x90).

int AFLFCL_MF

Contains the more fragments field of the fragmentation control field of the authentication and fragmentation layer (only if AFLCI=0x90).

int AFLMCL

Contains the message control field of the authentication and fragmentation layer as complete byte (only if AFLCI=0x90).

int AFLMCL_AT

Contains the authentication type field of the message control field of the authentication and fragmentation layer (only if AFLCI=0x90).

int AFLMCL_KIMP

Contains the key information field present field of the message control field of the authentication and fragmentation layer (only if AFLCI=0x90).

int AFLMCL_MCMP

Contains the message counter field present field of the message control field of the authentication and fragmentation layer (only if AFLCI=0x90).

int AFLMCL_MLMP

Contains the message length field present field of the message control field of the authentication and fragmentation layer (only if AFLCI=0x90).

int AFLKI

Contains the key information field of the authentication and fragmentation layer as complete byte (only if AFLCI=0x90).

int AFLKI_KID

Contains the key ID field of the key information field of the authentication and fragmentation layer (only if AFLCI=0x90).

int AFLKI_KDFS

Contains the key derivation function field of the key information field of the authentication and fragmentation layer (only if AFLCI=0x90).

int AFLKI_KVer

Contains the key version field of the key information field of the authentication and fragmentation layer (only if AFLCI=0x90).

int AFLMCR

Contains the message counter field of the authentication and fragmentation layer as complete integer (only if AFLCI=0x90).

byte AFLMAC[]

Contains the message authentication code of the authentication and fragmentation layer (only if AFLCI=0x90). The number of bytes is given in AFLMACLen.

int AFLMACLen

Contains the number of bytes of the message authentication code of the authentication and fragmentation layer (only if AFLCI=0x90).

int AFLML

Contains the message length field of the message authentication code of the authentication and fragmentation layer as word (only if AFLCI=0x90).

int TPLCI

Contains the CI field of the transport layer.

int TPLAcc

Contains the access number field of the transport layer.

int TPLStatus

Contains the status field of the transport layer.

int TPLConfig

Contains the configuration field of the transport layer as word.

int TPLConfig_Mode

Contains the mode field of the configuration field of the transport layer.

int TPLConfig_EncBytes

Contains the number of encrypted bytes of the radio frame from the configuration field of the transport layer.

int TPLConfig_HopCtr

Contains the hop counter field of the configuration field of the transport layer.

int TPLConfig_RepAccess

Contains the repeated access field of the configuration field of the transport layer.

int TPLConfig_ContentMessage

Contains the content of message field of the configuration field of the transport layer.

int TPLConfig_ContentIndex

Contains the content index field of the configuration field of the transport layer.

int TPLConfig_Synchr

Contains the synchronous field of the configuration field of the transport layer.

int TPLConfig_Accessib

Contains the accessibility field of the configuration field of the transport layer.

int TPLConfig_BiDi

Contains the bidirectional communication field of the configuration field of the transport layer.

int TPLConfig_CtrSize

Contains the counter size field of the configuration field of the transport layer.

int TPLConfigExt

Contains the configuration field extension of the transport layer as complete byte.

int TPLConfigExt_KID

Contains the key ID field of the configuration field extension of the transport layer.

int TPLConfigExt_KDFS

Contains the key derivation function field of the configuration field extension of the transport layer.

int TPLConfigExt_Ver

Contains the version field of the configuration field extension of the transport layer.

int TPLConfigExt_PType

Contains the protocol type field of the configuration field extension of the transport layer.

int DRCount

Contains the number of data records within the radio frame.

String DRDIB[]

Array of strings containing the data information blocks of all data records in the radio frame. The array index is between 0 and (DRCount-1).

int DRDIF_Dat[]

Array of integers containing the data field of the data information blocks of all data records in the radio frame. The array index is between 0 and (DRCount-1).

int DRDIF_Func[]

Array of integers containing the function field of the data information blocks of all data records in the radio frame. The array index is between 0 and (DRCount-1).

int DRDIF_Unit[]

Array of integers containing the unit field of the data information blocks of all data records in the radio frame. The array index is between 0 and (DRCount-1).

int DRDIF_Tariff[]

Array of integers containing the tariff field of the data information blocks of all data records in the radio frame. The array index is between 0 and (DRCount-1).

int DRDIF_Storage[]

Array of integers containing the storage field of the data information blocks of all data records in the radio frame. The array index is between 0 and (DRCount-1).

String DRVIB[]

Array of strings containing the value information blocks of all data records in the radio frame. The array index is between 0 and (DRCount-1).

String DRVIFTxt[]

Array of strings containing the physical units of all data records in the radio frame. The array index is between 0 and (DRCount-1).

The physical units of the data records are recalculated to base units (e.g. volume is always given as m3). The property DRValusStr[] containing the value of the data record is automatically corrected to fit the physical unit.

int DRVIF[]

Array of integers containing the indexes of the text entries in the value information field tables of EN13757-3 of all data records in the radio frame. The array index is between 0 and (DRCount-1).

Depending on the value DRVIF the following tables must be used:

DRVIF = 0 … 127	EN 13757-3, primary VIF code table
DRVIF = 128 … 255	EN 13757-3, alternative VIF code table (VIF=0xFB) (DRVIF – 128)
DRVIF = 256 … 383	EN 13757-3, alternative VIF code table (VIF=0xFD) (DRVIF – 256)
DRVIF = 384 … 511	EN 13757-3, primary VIF code table with non metric units (DRVIF – 384)

String DRValueStr[]

Array of strings containing the values of all data records in the radio frame. The array index is between 0 and (DRCount-1).

boolean CIPDeciphOK

This property contains true if the radio frame was successfully deciphered, false if it was not deciphered.

byte Frame[]

This byte array contains the non-deciphered, original radio frame.

byte FrameDeCiphered[]

This byte array contains the deciphered radio frame.

Code Example

The following lines of pseudo-code show the principal usage of the MBusRacXV2 library. It should be noted that this is simplified and incomplete code just to show the principles.
It is especially advised to invoke all Bluetooth communication methods (BTConnect, BTCMDxxxxx) only in AsyncTasks to not block the Android main UI task.

```
//*********************************************************************************************
// Initialization
//*********************************************************************************************

import com.michaelrac.mbusracxv2.MBWBLUE;    // import class MBWBLUE
import com.michaelrac.mbusracxv2.MBus;       // import class MBus


MBWBLUE MBL;                                 // MBWBLUE class declaration -> MBL
MBus MBS;                                    // MBus class declaration -> MBS
byte KeyFileKey[];                           // KeyFileKey is used to hold the
                                             // ciphering key for storing the radio key
                                             // database on the Android device


MBL = new MBWBLUE("");                       // invoke MBWBLUE class (parameter must be “”)
if (!MBL.BTAdapterOK) {error handling}       // if the bluetooth adapter of the Android device
                                             // is not available MBWBLUE cannot be used

KeyFileKey=new byte[16];                     // define you own key for storing radio frame keys
KeyFileKey[0]=(byte)0x36;KeyFileKey[1]=(byte)0x98;KeyFileKey[2]=(byte)0xBF;
KeyFileKey[3]=(byte)0x98;KeyFileKey[4]=(byte)0xAC;KeyFileKey[5]=(byte)0xED;
KeyFileKey[6]=(byte)0x4E;KeyFileKey[7]=(byte)0xC3;KeyFileKey[8]=(byte)0xD9;
KeyFileKey[9]=(byte)0x72;KeyFileKey[10]=(byte)0x21;KeyFileKey[11]=(byte)0xAE;
KeyFileKey[12]=(byte)0x6B;KeyFileKey[13]=(byte)0x80;KeyFileKey[14]=(byte)0xFA;
KeyFileKey[15]=(byte)0x23;
                                             // invoke MBus class with MainActivity context and
                                             // ciphering key for key data base
MBS = new MBus(getApplicationContext(),KeyFileKey);




//*********************************************************************************************
// Configure radio frame deciphering keys
//*********************************************************************************************

                                             // Configure AES 128 bit general keys
MBS.CryptoKeysAddOrModify(0, -1, "000102030405060708090A0B0C0D0E0F", "");
MBS.CryptoKeysAddOrModify(0, -1, "0F0E0D0C0B0A09080706050403020100", "");

                                             // Configure AES 128 bit address specific keys
MBS.CryptoKeysAddOrModify(3, -1, "00000000000000000000000000000000", "12345678");
MBS.CryptoKeysAddOrModify(3, -1, "11111111111111111111111111111111", "87654321");

MBS.CryptoKeysWrite();                       // Write key database to application storage space
                                             // on Android device.
                                             // In this case the configured keys are
                                             // automatically loaded on invoking the MBus
                                             // class.
                                             // However, it is also possible to not save the
                                             // radio keys and configure them every time the
                                             // library is started.
```

```
//*****************************************************************************************
// Get available, paired MBWBLUE devices
//*****************************************************************************************

MBL.AvailableDevicesList();                 // Retrieve list of available, paired MBWBLUE
                                            // If no paired MBWBLUE -> error
if (MBL.AvailableDevicesCount()<=0) {error handling}

                                            // Put available, paired MBWBLUE into a list for
                                            // user selection
for (DSCounter=0; DSCounter<MBL.AvailableDevicesCount(); ++DSCounter)
{
       List[DSCounter]=MBL.AvailableDevicesName(DSCounter);
}




//*****************************************************************************************
// Read out selected MBWBLUE to get general information (firmware version etc.)
//*****************************************************************************************

MBL.BTDeviceName=List[0];                   // Set MBWBLUE name to first entry of List
                                            // Establish Bluetooth connection to MBWBLUE
if (MBL.BTConnect(MBL.AvailableDevicesGetAddressFromName(MBL.BTDeviceName)))
{                                           // If successful request firmware version
       if (MBL.BTCMDRequestFirmwareVersionLeg())
       {                                    // If successful request serial number
              if (MBL.BTCMDReadROMLeg())
              {
                     X1=MBL.BTDevType;      // These four properties are now set correctly
                     X2=MBL.BTFWVersion;    // and can be e.g. displayed to the user
                     X3=MBL.BTSerialNumber;
                     X4=MBL.BTMACAddress;
              }
       }
        MBL.BTDisConnect();                 // Shutdown Bluetooth connection to MBWBLUE
}
```

```
//*********************************************************************************************
// Reading radio frames
//*********************************************************************************************

if (MBL.BTConnect(MBL.AvailableDevicesGetAddressFromName(MBL.BTDeviceName)))
{
	MBL.BTCurReceiveMode=0;			// Set MBWBLUE receive mode to mode T + C
	MBL.BTCurReceiverFreq=868;		// Set MBWBLUE receive frequency to 868
					// Configure MBWBLUE receiver
	if (!MBL.BTCMDReceiverModeLeg()) {error handling}

	while (!ReadingEnd)
	{				// Generation 1,2 and generation 3
		if (MBL.BTGen3==1)		// devices have different reading commands
		{				// Get next received radio frame
			if (!MBL.BTCMDReadRadio()) {error handling}
			else
			{			// Interpret radio frame
				if (MBS.Interpret(MBL.BTInBuffer,5,256,false))
				{		// Use properties of MBus class
					Y1=MBS.AdrFieldTPL;
					Y2=MBS.ManFieldTPL;
					Y3=MBS.VerFieldTPL;
					Y4=MBS.DevFieldTPL;
					for (Counter=0;Counter<MBS.DRCount;++Counter)
					{
						Y5[Counter]=MBS.DRValueStr[Counter];
						Y6[Counter]=MBS.DRVIFTxt[Counter];
					}
				}
			}
		}
		else	// Generation 2 devices
		{				// Get next received radio frame
			if (!MBL.BTCMDReadRadioLeg()) {error handling}
			else
			{			// Interpret radio frame
				if (MBS.Interpret(MBL.BTInBuffer,12,241,false))
				{		// Use properties of MBus class
					Y1=MBS.AdrFieldTPL;
					Y2=MBS.ManFieldTPL;
					Y3=MBS.VerFieldTPL;
					Y4=MBS.DevFieldTPL;
					for (Counter=0;Counter<MBS.DRCount;++Counter)
					{
						Y5[Counter]=MBS.DRValueStr[Counter];
						Y6[Counter]=MBS.DRVIFTxt[Counter];
					}
				}
			}
		}
	}
	MBL.BTDisConnect();			// Shutdown Bluetooth connection to MBWBLUE
}



//*********************************************************************************************
// Clean up MBWBLUE and MBus classes after usage
//*********************************************************************************************

MBL.Destroy();
MBS.Destroy();
```

MBWBLUE_DEMO application

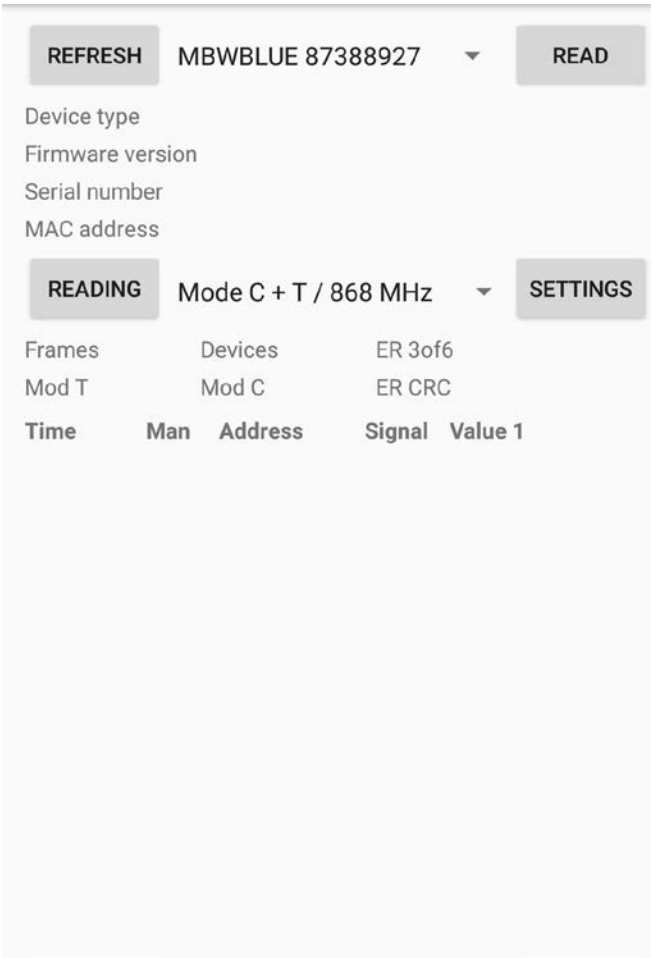
MBWBLUE_DEMO is a small and simple Android application to demonstrate the usage of the mbusracxv2.aar library. It is provided “as is” and it is not intended to be used in productive environments.
The MBusRacXV2 library package contains the source code of the MBWBLUE_DEMO application as ZIP file as well as the compiled APK file.
For general information on how integrating Android Archives in you project please check the respective internet pages.

Installing and Starting

Copy the file MBWBLUE_DEMO.apk to your Android device using a network or USB connection. Use a file explorer on your Android device to install and start the application.

Before using the application with your MBWBLUE device, it is necessary to perform a Bluetooth pairing using the Android operating system.

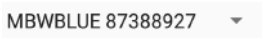
The started MBWBLUE_DEMO.apk looks like the screenshot below:



Usage

There are four user buttons:

REFRESH

Updates the drop-down list 

with the already paired MBWBLUE devices. If the drop-down list contains no MBWBLUE please perform a Bluetooth pairing.

READ

Reads out the general parameter of the selected MBWBLUE device, may also be used to test the connection.
Additionally, if the MBWBLUE_DEMO application contains a newer version of the MBWBLUE firmware the selected MBWBLUE device is automatically updated.
After a successful read, the parameters below are shown:

Device type	**MBWBLUE 868 V3**
Firmware version	**3.050**
Serial number	**87388927**
MAC address	**8C:DE:52:B1:DF:FF**

Device type: MBWBLUE type (868 or 434, V2 or V3)

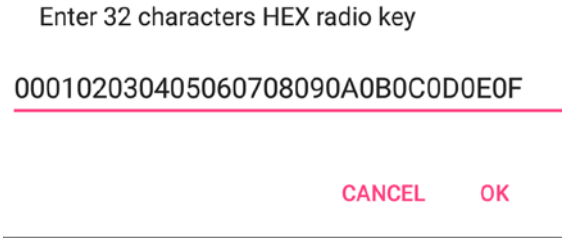
Firmware version: Version of the firmware

Serial number: Serial number of the MBWBLUE

MAC address: Bluetooth MAC address of the MBWBLUE

SETTINGS

Opens a dialog to enter one radio frame deciphering key (32 characters). The 32 characters key (128 bit) is used as AES128 deciphering key; the first 16 characters (64 bit) are additionally used as 64-bit key. Within the demo application, it is only possible to enter one key even though the library is able to handle multiple keys.



READING

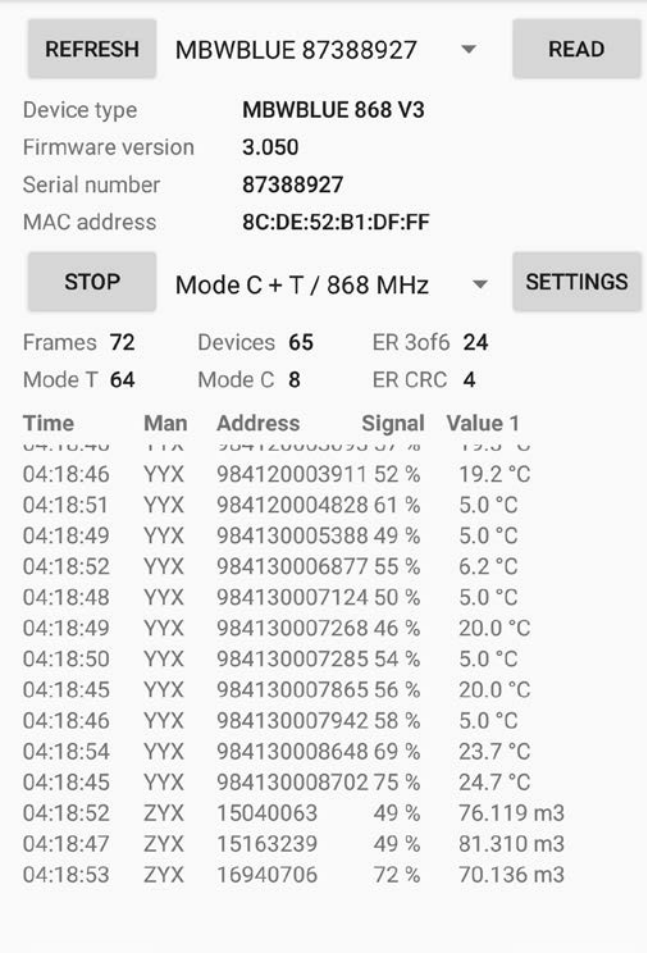
Starts the radio frame reading of the MBWBLUE in the selected mode.

Available modes are:

Mode C + T / 868 MHz
Mode S / 868 MHz
Mode C + T / 434 MHz
Mode S / 434 MHz

Please note that it is possible for 868 MHz MBWBLUE devices to receive at 434 MHz and vice versa. However, this is only for testing purposes and the performance is very bad.

All received radio frames are listed at the bottom of the application window. The list is sorted by manufacturer codes first and radio addresses afterwards. Only the latest received radio frame of each device is shown in the list.



Additionally, statistical values of received radio frames are shown during radio frame reading (only V3 MBWBLUE devices).

Frames:	The overall number of correctly received radio frames
Devices:	The number of different devices received
Mode T:	The overall number of correctly received mode T frames
Mode C:	The overall number of correctly received mode C frames
ER 3of6:	Number of frames with 3 of 6 coding error received
ER CRC:	Number of frames with CRC error received