B4Rlibrary - A tool to help at wrapper creation

What it is :

It is a script to create B4R library from an Arduino library
It is a script running under PSPad editor because it is easy to add features in vbs script on this one. From h file Arduino, it will help at wrapper creation for B4R by cpp file creation, h file creation and xml file creation

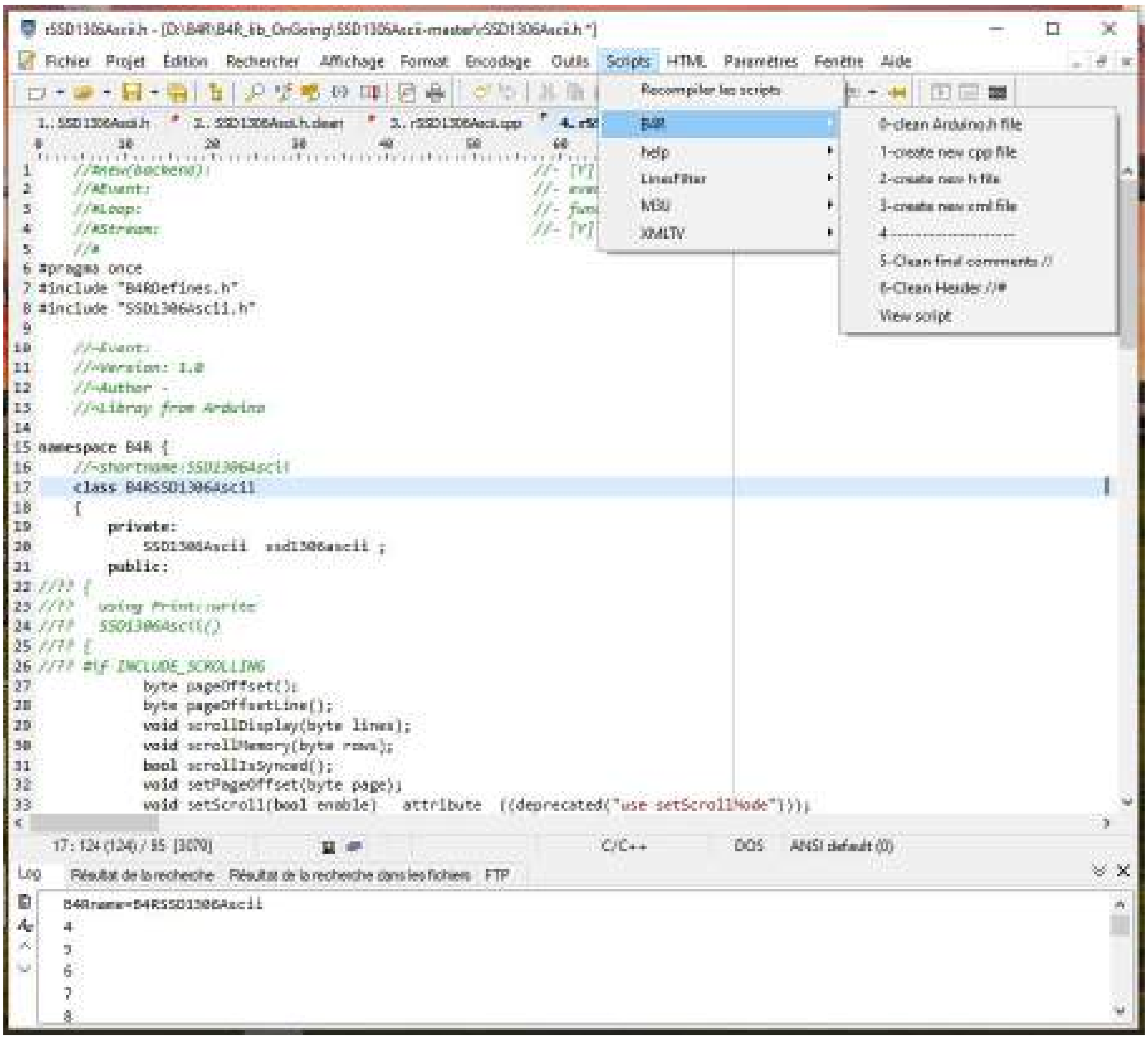
Installation :

1. First solution: a new install

⇨ Download PSPad , (freeware) and install it in c:\PSPad editor or another root area.
⇨ Activate vbs scripts in PSPad parameters
⇨ In PSPad parameters, Add PSPad at Windows 10 in contextual menu, it will help at files loading
⇨ Install this vbs script in PSPad editor\script\VBSscript (zip of this thread)
⇨ In PSPad menu, now you have « scripts » menu and under « recompile scripts » => run it
⇨ After that, in script menu you have a sub-menu « B4R » and « Help »

2. Second solution: download a full portable version PSPad with script installed and vbs helps.

And you can install it directly in c:\PSPad editor (or another root area)
https://drive.google.com/file/d/1qmn8NL7fpkbLUpBqpzXoO9Pt0uEQRDwz/view?usp=sharing
And install last version of script from here.



```
    //#new(backend):
    //#Event:
    //#Loop:
    //#Stream:
    //#
#pragma once
#include "B4RDefines.h"
#include "SSD1306Ascii.h"

    //~Event:
    //~Version: 1.0
    //~Author -
    //~Libray from Arduino

namespace B4R {
    //~shortname:SSD1306Ascii
    class B4RSSD1306Ascii
    {
        private:
            SSD1306Ascii  ssd1306ascii ;
        public:
//?? {
//??   using Print::write
//??   SSD1306Ascii()
//?? {
//?? #if INCLUDE_SCROLLING
            byte pageOffset();
            byte pageOffsetLine();
            void scrollDisplay(byte lines);
            void scrollMemory(byte rows);
            bool scrollIsSynced();
            void setPageOffset(byte page);
            void setScroll(bool enable)  attribute  ((deprecated("use setScrollMode")));
```

```
B4Rname=B4RSSD1306Ascii
4
5
6
7
8
```

How it is working :

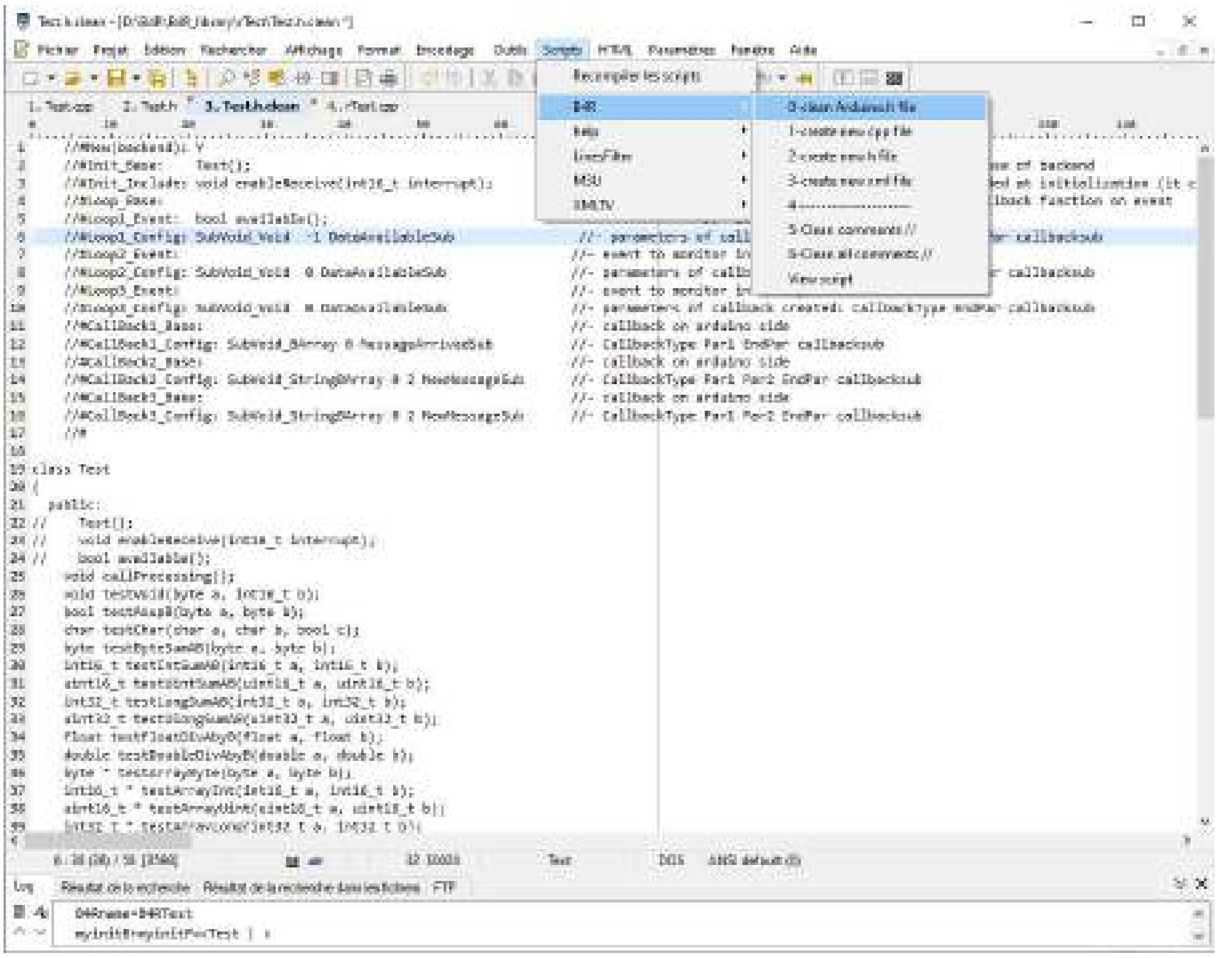
First you have to load xxx.h file arduino library in PSPad
After run this script in 4 steps:

- Creation of a file xxx.h.clean file
- Creation a rxxx.cpp file for B4R library
- Creation of rxxx.h file for B4R library
- Creation of rxxx.xml file for B4R library

- And when you think your library finished, when tests are OK, you can clean all comment lines.

1) **Creation of a file xxx.h.clean file**

With xxx.h file active, launch "Clean Arduino h file"



```
    //#New(backend): Y                                                      ...se of backend
    //#Init_Base:    Test();                                                ...ded at initialization (it c
    //#Init_Include: void enableReceive(int16_t interrupt);                 ...lback function on event
    //#Loop_Base:
    //#Loop1_Event:  bool available();
    //#Loop1_Config: SubVoid_Void  -1 DataAvailableSub         //- parameters of call...         ...for callbacksub
    //#Loop2_Event:                                            //- event to monitor in...
    //#Loop2_Config: SubVoid_Void  0 DataAvailableSub          //- parameters of callb...        ...r callbacksub
    //#Loop3_Event:                                            //- event to monitor in...
    //#Loop3_Config: SubVoid_Void  # DataAvailableSub          //- parameters of callback created: callbacktype endPar callbacksub
    //#CallBack1_Base:                                         //- callback on arduino side
    //#CallBack1_Config: SubVoid_BArray 0 MessageArrivedSub    //- CallbackType Par1 EndPar callbacksub
    //#CallBack2_Base:                                         //- callback on arduino side
    //#CallBack2_Config: SubVoid_StringBArray # 2 NewMessageSub  //- CallbackType Par1 Par2 EndPar callbacksub
    //#CallBack3_Base:                                         //- callback on arduino side
    //#CallBack3_Config: SubVoid_StringBArray # 2 NewMessageSub  //- CallbackType Par1 Par2 EndPar callbacksub
    //#

class Test
{
  public:
//    Test();
//    void enableReceive(int16_t interrupt);
//    bool available();
    void callProcessing();
    void testVoid(byte a, int16_t b);
    bool testAnd8(byte a, byte b);
    char testChar(char a, char b, bool c);
    byte testByteSumAB(byte a, byte b);
    int16_t testIntSumAB(int16_t a, int16_t b);
    uint16_t testUIntSumAB(uint16_t a, uint16_t b);
    int32_t testLongSumAB(int32_t a, int32_t b);
    uint32_t testULongSumAB(uint32_t a, uint32_t b);
    float testFloatDivAbyB(float a, float b);
    double testDoubleDivAbyB(double a, double b);
    byte * testArrayByte(byte a, byte b);
    int16_t * testArrayInt(int16_t a, int16_t b);
    uint16_t * testArrayUInt(uint16_t a, uint16_t b);
    int32_t * testArrayLong(int32_t a, int32_t b);
```

```
B4Rname=B4RTest
myInitArrayInitForTest | ;
```

This script first will load h file, will remove comments, will select all lines after a « class » and between « public » and end of class. Also « private » and « protected » areas are ignored.
At end, a new file h.clean is created
This script adds a header on top of the file for configuration of wrapper to build

- After this script, you have a new file with only lines used to create B4R wrapper and 18 lines added at top for configuration.

On header we have several parameters: we have to provide manually a configuration to indicate what type of wrapper we want to create

- **//#New(backend): Y** => it will create a wrapper with new(backend) and all options possible

- **//#New(backend**): => if <> Y it is direct wrapper without code added

- **//#Init_Base:** => launch of the arduino library with parameters or without parameters
  => if parameter is mandatory at launch, wrapper with backend is mandatory

all options after are for wrapper with backend

- **//#Init_Include:** function added here will be added in initialize part of wrapper

  (it is a function from arduino library in h file)

- **//#Loop_Base:** function launching the loop in wrapper (if you want a loop in wrapper)

  in loop we can monitor 3 event with activation of 3 sub in B4X with parameters selected here

  if empty and event declared, loop is started with initialize()
  if no event declared, no loop created

- **//#Loop1_Event**: event to monitor in the loop

  (it is a function from arduino library in h file)
- **//#Loop1_Config**: it is configuration for action in case of event , parameters sent to B4R interface

  (sub_type, parameter selected list, sub_name for B4R)

  Parametersselected are parameters from event function and sent to B4R

  First can be -1 if no parameter provided to B4R
- **//#Loop2_Event:**
- **//#Loop2_Config:**

- **//#Loop3_Event:**
- **//#Loop3_Config:**

- **//#CallBack1_Base:** it is function from h file launching callback to B4R (with or without parameter)

  (it is a function from arduino library in h file)
- //#CallBack1_Config: it is configuration of event for B4R interface creation(similar to loop)

  (sub_type, parameters selected list to handle, sub_name for B4R)

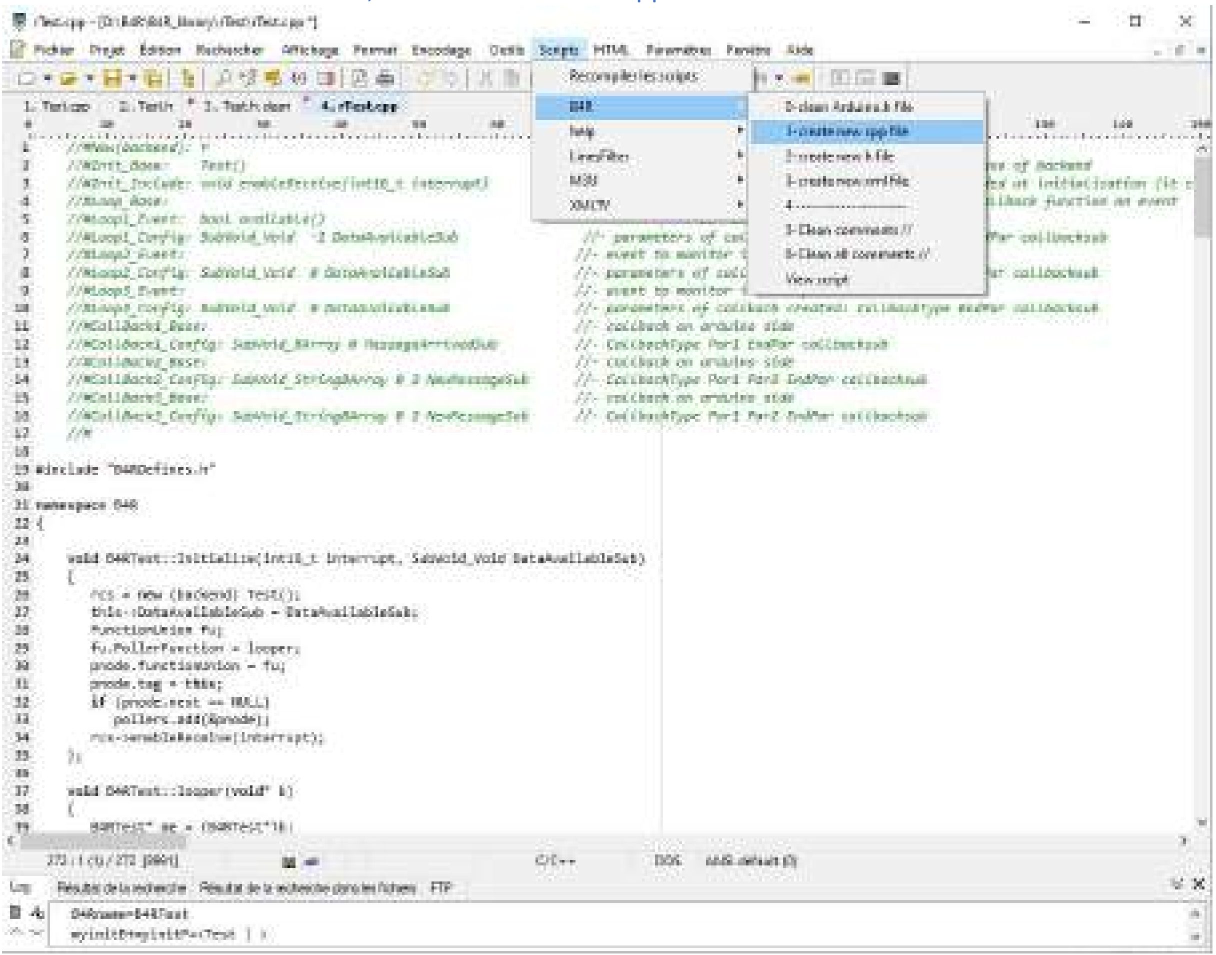
- **//#CallBack2_Base:**
- **//#CallBack2_Config:**

- **//#CallBack3_Base:**
- **//#CallBack3_Config:**

all functions added in parameters above must be blocked in h.clean file by addition of "//" on head of the line

2) <u>Creation of a rxxx.cpp file for B4R library</u>

With xxx.h.clean file active, launch “Create new cpp file”


rTest.cpp - [D:\B4R\B4R_library\rTest\rTest.cpp *]
Fichier Projet Edition Rechercher Affichage Format Encodage Outils Scripts HTML Paramètres Fenêtre Aide
1. Test.cpp 2. Test.h 3. Test.h.clean 4. rTest.cpp
Recompiler les scripts
B4R
help
LinesFilter
M30
XMLTV
0-clean Arduino.h file
1-create new cpp file
2-create new h file
3-create new xml file
4-----------------
5-Clean comments //
6-Clean All comments //
View script
272 : 1 (1) / 272 [9801]
C/C++
DOS
ANSI default (0)
Log Résultat de la recherche Résultat de la recherche dans les fichiers FTP

```
    //#New(backend): ?                                              // [illegible] of backend
    //#Init_Base:    Test()                                         // [illegible] at initialization (it [illegible]
    //#Init_Include: void enableReceive(int16_t interrupt)          // [illegible] callback function on event
    //#Loop_Base:
    //#Loop1_Event:  bool available()
    //#Loop1_Config: SubVoid_Void  1 DataAvailableSub        //- parameters of [illegible] Ptr callbacksub
    //#Loop2_Event:                                          //- event to monitor [illegible]
    //#Loop2_Config: SubVoid_Void # DataAvailableSub         //- parameters of [illegible] Ptr callbacksub
    //#Loop3_Event:                                          //- event to monitor [illegible]
    //#Loop3_Config: SubVoid_Void # DataAvailableSub         //- parameters of callback created: callbacktype #adPtr callbacksub
    //#CallBack1_Base:                                       //- callback on arduino side
    //#CallBack1_Config: SubVoid_BArray # MessageArrivedSub  //- CallbackType Par1 EndPtr callbacksub
    //#CallBack2_Base:                                       //- callback on arduino side
    //#CallBack2_Config: SubVoid_StringBArray # 2 NewMessageSub   //- CallbackType Par1 Par2 EndPtr callbacksub
    //#CallBack3_Base:                                       //- callback on arduino side
    //#CallBack3_Config: SubVoid_StringBArray # 2 NewMessageSub   //- CallbackType Par1 Par2 EndPtr callbacksub
    //#

#include "B4RDefines.h"

namespace B4R
{

    void B4RTest::Initialize(int16_t interrupt, SubVoid_Void DataAvailableSub)
    {
        rcs = new (backend) Test();
        this->DataAvailableSub = DataAvailableSub;
        FunctionUnion fu;
        fu.PollerFunction = looper;
        pnode.functionUnion = fu;
        pnode.tag = this;
        if (pnode.next == NULL)
            pollers.add(&pnode);
        rcs->enableReceive(interrupt);
    };

    void B4RTest::looper(void* b)
    {
        B4RTest* me = (B4RTest*)b;
```

```
B4Rname=B4RTest
myinitB4tmplinit#=rTest | ?
```

This script makes in first some changes needed by parameters not supported by B4R.
1. Const, static, inline, virtual, typedef, volatile, constexpr => are removed
2. Unsigned => u
3. ulong => uint32_t
4. long => int32_t
5. Size_t, uint, word => uint16_t
6. int, short => int16_t
7. uint8_t, ubyte , int8_t => byte
8. uchar => char
9. Boolean => bool

Normally, this script takes in account a large part of variable types.
We have 2 types of management of variable:
- Variable from library returned to B4R in return:
  Void, bool, char, byte, int16_t, uint16_t, int32_t, uint32_t, float, double,
  IPAdress, string, char*, byte*, int16_t*, uint16_t*, int32_t*, uint32_t*, float*, double* => only with new(backend)
- Variable in parameters received by library and sent by B4R
  Void, bool, char, byte, int16_t, uint16_t, int32_t, uint32_t, float, double,
  Char*, byte*, int16_t*, uint16_t*, int32_t*, uint32_t*, float*, double*

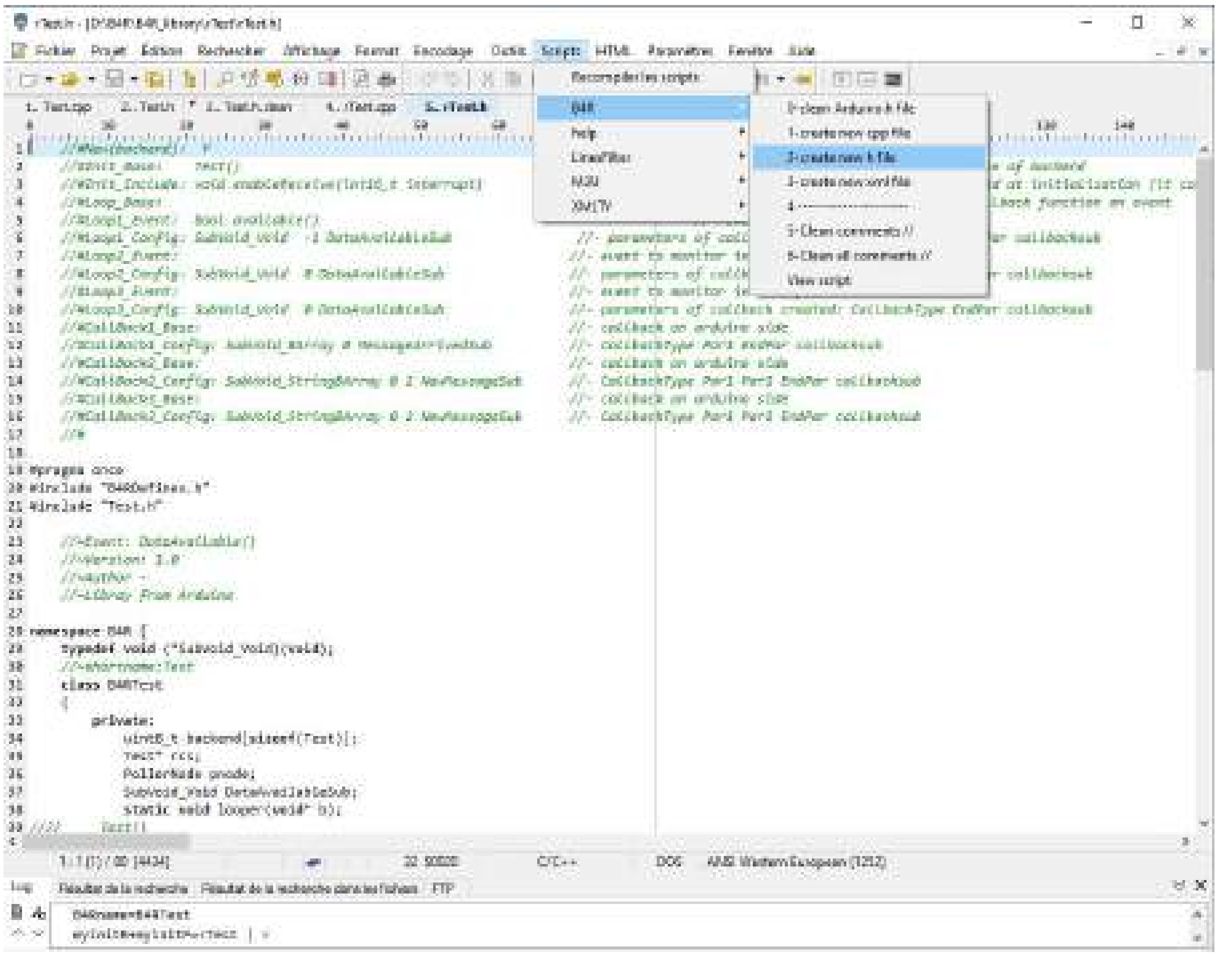
This script takes in account public variables created in Arduino library.
This script take in account “enum” and creates a class but in comments with “//enum “ in cpp file and it will be added at h file
⇨ Don’t remove comment lines starting by “//enum” before h file creation
“struct” are not managed and are moved to comments.
All parameters with a type corresponding to a “struct” will not be managed by this script,
⇨ some codes are needed to adapt to B4R

All parameters types not managed will include “??”
All lines not managed will be moved to comments starting by “//??”

- ⇨ By a check of “??” you will find parameters not managed by this script, and you have to adapt it manually or to remove the function
- ⇨ By a check of all lines starting by “//??” you will find all lines not managed by the script

3) Creation of rxxx.h file for B4R library

With rxxx.cpp file active, launch “Create new h file”



```
#pragma once
#include "B4RDefines.h"
#include "Test.h"

    //~Event: DataAvailable()
    //~Version: 1.0
    //~Author: -
    //~Library From Arduino

namespace B4R {
    typedef void (*SubVoid_Void)(void);
    //~shortname:Test
    class B4RTest
    {
        private:
            uint8_t backend[sizeof(Test)];
            Test* cs;
            PollerNode pnode;
            SubVoid_Void DataAvailableSub;
            static void looper(void* b);
////        Test()
```

This file is created from cpp file
This file is created in a format depending of parameter “//#new(backend)”
Lines starting by “//enum” parameters will be added and will add new class at h file

⇨ We have to check all lines in comment starting by “//??” because it is lines not managed by script

⇨ For complex cases, you have to check similar functions in available libraries B4R and to add similar code

⇨ You can check also code examples provided with Arduino library to understand how to use parameters of function and to add similar code in your wrapper.

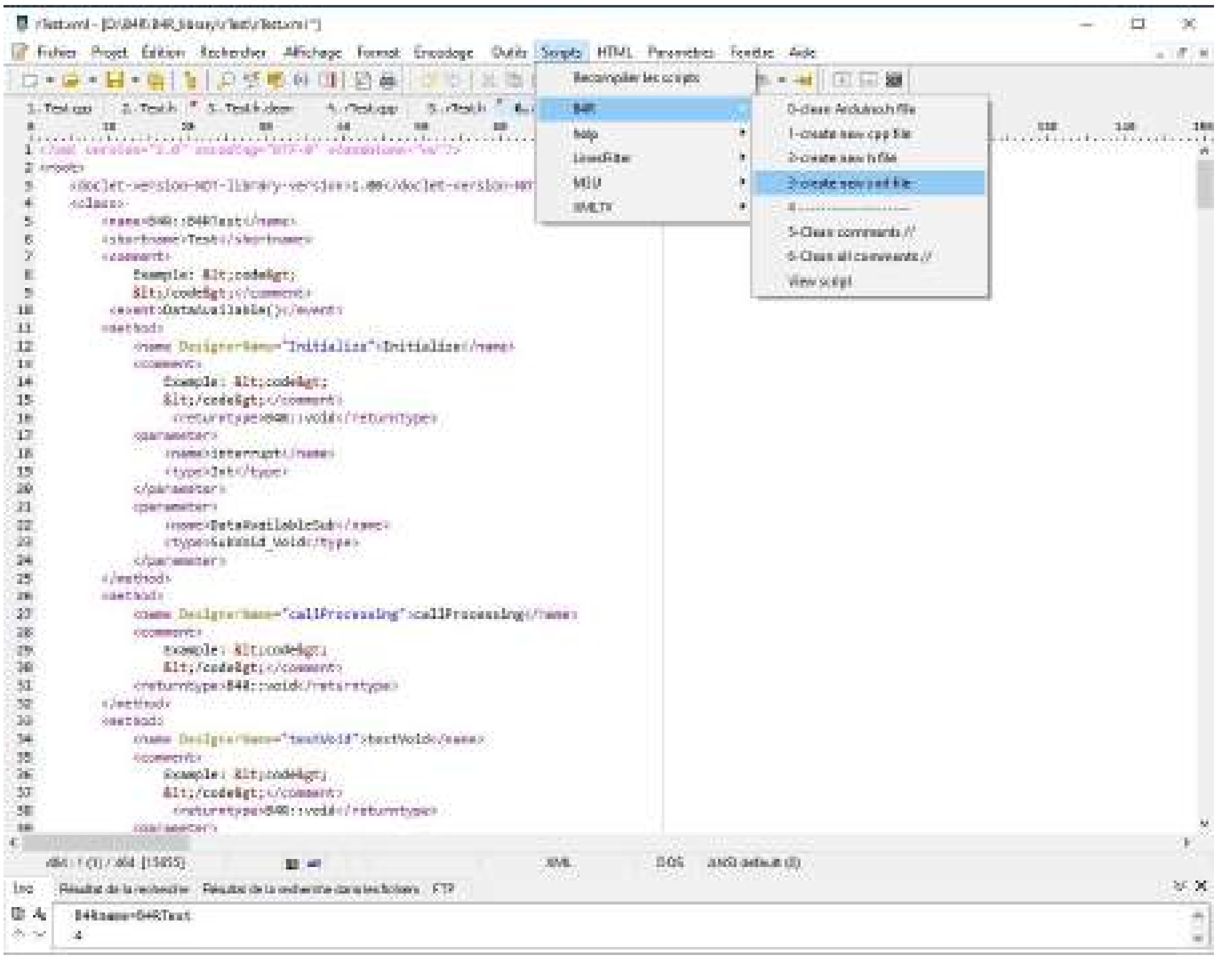
All parameters types not managed will include “??”
All lines not managed will be moved to comments starting by “//??”

⇨ By a check of “??” you will find parameters not managed by this script, and you have to adapt it manually or to remove the function

⇨ By a check of all lines starting by “//??” you will find all lines not managed by the script

4) <u>Creation of rxxx.xml file for B4R library</u>

With rxxx.h file active, launch “Create new xml file”



```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<root>
    <doclet-version-NOT-library-version>1.00</doclet-version-NOT
    <class>
        <name>B4R::B4RTest</name>
        <shortname>Test</shortname>
        <comment>
            Example: &lt;code&gt;
            &lt;/code&gt;</comment>
         <event>DataAvailable()</event>
        <method>
            <name DesignerName="Initialize">Initialize</name>
            <comment>
                Example: &lt;code&gt;
                &lt;/code&gt;</comment>
                 <returntype>B4R::void</returntype>
            <parameter>
                <name>interrupt</name>
                <type>Int</type>
            </parameter>
            <parameter>
                <name>DataAvailableSub</name>
                <type>SubVoid_Void</type>
            </parameter>
        </method>
        <method>
            <name DesignerName="callProcessing">callProcessing</name>
            <comment>
                Example: &lt;code&gt;
                &lt;/code&gt;</comment>
            <returntype>B4R::void</returntype>
        </method>
        <method>
            <name DesignerName="testVoid">testVoid</name>
            <comment>
                Example: &lt;code&gt;
                &lt;/code&gt;</comment>
                 <returntype>B4R::void</returntype>
            <parameter>
```

```
B4Rname=B4RTest
4
```

Xml file is created from h file

It will create “class”, “method” or “property” and “field” depending of h file.

We can force “method” creation if we add “//#Meth” at end of line of function declaration in h file

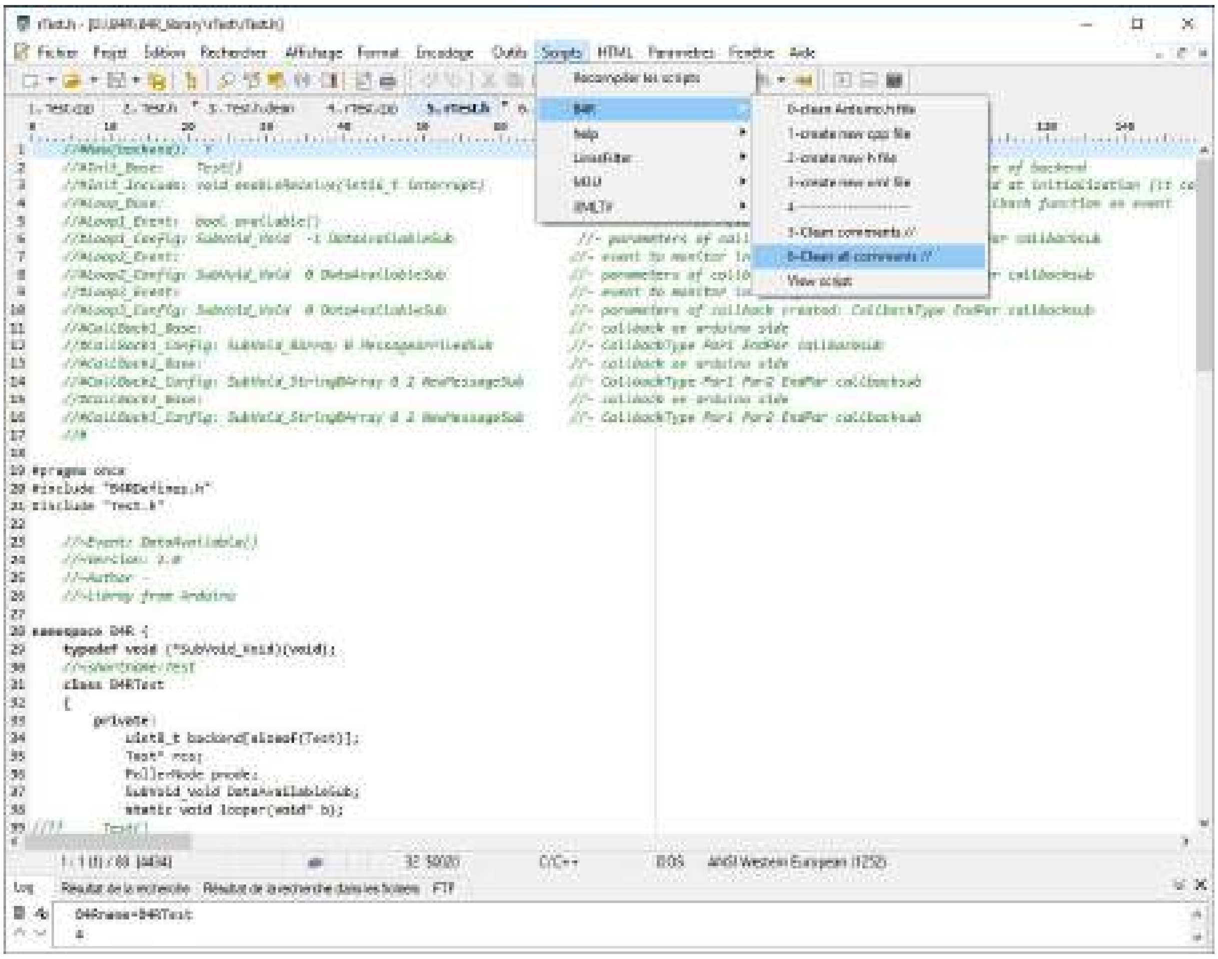
All lines not managed will be in comment starting by “//??” and after a check of xml format it can be removed if not needed

5) <u>When your library is tested and when you think your library finished you can clean all comment</u>

“Clean final comments” will remove all comments except header

“Clean Header” will remove header

⇨ You are ready to test your library with B4R



```
    //#Name(backend): T
    //#Init_Base:    Test()
    //#Init_Include: void enableReceive(uint8_t interrupt)
    //#Loop_Base:
    //#Loop1_Event:  bool available()
    //#Loop1_Config: SubVoid_Void  -1 DataAvailableSub
    //#Loop2_Event:
    //#Loop2_Config: SubVoid_Void  0 DataAvailableSub
    //#Loop3_Event:
    //#Loop3_Config: SubVoid_Void  0 DataAvailableSub        //- parameters of callback created: CallbackType EndPar callbacksub
    //#CallBack1_Base:                                       //- callback on arduino side
    //#CallBack1_Config: SubVoid_BArray 0 MessageArrivedSub  //- CallbackType Par1 EndPar callbacksub
    //#CallBack2_Base:                                       //- callback on arduino side
    //#CallBack2_Config: SubVoid_StringBArray 0 2 NewMessageSub  //- CallbackType Par1 Par2 EndPar callbacksub
    //#CallBack3_Base:                                       //- callback on arduino side
    //#CallBack3_Config: SubVoid_StringBArray 0 2 NewMessageSub  //- CallbackType Par1 Par2 EndPar callbacksub
    //#

#pragma once
#include "B4RDefines.h"
#include "Test.h"

    //~Event: DataAvailable()
    //~Version: 1.0
    //~Author -
    //~Library from Arduino

namespace B4R {
    typedef void (*SubVoid_Void)(void);
    //~shortname:Test
    class B4RTest
    {
        private:
            uint8_t backend[sizeof(Test)];
            Test* rcs;
            PollerNode pnode;
            SubVoid_Void DataAvailableSub;
            static void looper(void* b);
//!!!     Test()
```

```
B4Rname=B4RTest
4
```

6) <u>How to debug your wrapper</u>

First you have to make a copy of xml file and install it in directory of libraries for B4R

Now start B4R (if new xml file added with BAR running, you have to restart B4R to take it in account)

- you can create a new project on B4R
- add the new library at your project
- compile your project.
- check errors at compilation.
- in case of errors, you can make modifications on files opened in PSPAD and save it
- if modifications done in xml file unload and reload xml file in B4R
- run again compilation
- and continue until compilation done without error
- after we can add functions one by one and compile each time

at last, you can remove header and comments on files created

- with rTEST1.cpp active, run Scripts => BAR => clean all comments

  => and save cpp file.
- with rTEST1.h active, run Scripts => BAR => clean all comments

  => and save h file.

  => and save xml file.

7) <u>To help at debugging of library and wrapper :</u>

- first to start with just library added at your project without function added
- you can add a function in arduino library to make a serial.println
- you can add wrapper for this fonction
- after you can add some traces (some print) on some functions in library and in wrapper to check what is done at each level.

8) 2 examples of library “test” and “test1” were created to investigate wrapper interface with B4R

the 2 libraries for test are providing some examples of interfaces library ⇔ B4R to check how to manage in wrapper

B4R	===========>	Wrapper	==========>	Library arduino
parameter byte	=>	byte	=>	byte
byte	<=	byte	<=	byte RETURN
parameter int	=>	int16_t	=>	int16_t
int	<=	int16_t	<=	int16_t RETURN
parameter uint	=>	uint16_t	=>	uint16_t
uint	<=	uint16_t	<=	uint16_t RETURN
parameter long	=>	int32_t	=>	int32_t
long	<=	int32_t	<=	int32_t RETURN
parameter ulong	=>	uint32_t	=>	uint32_t
ulong	<=	uint32_t	<=	uint32_t RETURN
parameter double	=>	float	=>	float
double	<=	float	<=	float RETURN

parameter arraybyte	=>	byte*	=>	byte*
arraybyte	<=	byte*	<=	byte* RETURN
parameter arrayint	=>	int16_t*	=>	int16_t*
arrayint	<=	int16_t*	<=	int16_t* RETURN
parameter arrayuint	=>	uint16_t*	=>	uint16_t*
arrayuint	<=	uint16_t*	<=	uint16_t* RETURN
parameter arraylong	=>	int32_t*	=>	int32_t*
arraylong	<=	int32_t*	<=	int32_t* RETURN
parameter arrayulong	=>	uint32_t*	=>	uint32_t*
arrayulong	<=	uint32_t*	<=	uint32_t* RETURN
parameter arraydouble	=>	float*	=>	float*
arraydouble	<=	float*	<=	float* RETURN

No parameter	=>		=>	No parameter
B4RString	<=	char*	<=	char* RETURN
parameter B4RString	=>	char*	=>	char*
B4RString	<=	char*	<=	char* RETURN
parameter B4RString	=>	string*	=>	string*
B4RString	<=	string*	<=	string* IP RETURN
parameter arraybyte	=>	IPaddress	=>	IPaddress
B4RString	<=	IPaddress	<=	IPaddress RETURN

only for array, we can modify parameters in library arduino and take modified parameter in B4R

parameter arraybyte	=>	byte*	=>	byte* -> parameter from B4R
parameter arraybyte	<=	byte*	<=	byte* <- parameter modified

in test1.b4r we have an example of callback Library => wrapper => B4R

parameter arraybyte	<=	byte*	<=	byte* <- function with parameters launched by library
B4Rstring	<=	char*	<=	char*
int	<=	int	<=	int

in test.b4r we have an example of monitoring by loop in wrapper

loop in Wrapper

1) check => by a function(with parameters)

function B4R activated <= 2) if True

9) summary of typedef included in B4R

SubVoidArray	typedef void (*SubVoidArray)(Array* barray) ;
SubVoidBool	typedef void (*SubVoidBool)(bool b) ;
SubVoidByte	typedef void (*SubVoidByte)(Byte b) ;
SubVoidVoid	typedef void (*SubVoidVoid) (void);
SubByteVoidP	typedef void (*SubByteVoidP)(Byte, void*)
SubVoidVoidP	typedef void (*SubVoidVoidP)(void*);

10) example of typedef added in some wrappers

SubVoidStringByteArray	typedef	void (*SubVoidStringByteArray)(B4RString* str, Array* barray) ;
SubByteArray	typedef	void (*SubByteArray)(Array* barray) ;
SubByteArrayInt	typedef	void (*SubByteArrayInt)(Array* barray, int* id, bool* ext) ;
SubVoidStringArray	typedef	void (*SubVoidStringArray)(B4RString* func, Array* barray) ;

11) example of Subname used in several libraries

MessageArrivedSub	=> MessageArrived
DisconnectedSub	=> Disconnected
NewMessageSub	=> NewMessage
NewMessageSub	=> NewMessage

12) <u>In case of crash of script</u>

We can have crash of script on a specific line not fully managed by the script, or when some data are not expected at this place, or if some data are missing…

During script execution, each line number is added at log file when it is managed by the script

- ⇨ If crash, you have to look in log at last line managed and to check what is wrong on this one.
- ⇨ You can remove analysis of this line by addition of "//" at start of the line before more analysis