



POB-JAVA Documentation

Table of contents

1	INTRODUCTION	4
2	INSTALLING POB-JAVA.....	5
▪	Installation of the GNUARM compiler	5
▪	Installing the Java Development Kit.....	7
▪	Installing of POB-Java.....	8
3	CONFIGURATION.....	9
▪	1 Manage a project with POB-JAVA	9
▪	2 Path to Java SDK.....	9
▪	3 Path to use the GNUARM compiler.....	10
▪	4 Select the serial port.....	10
▪	5 User native support	10
4	POB-JAVA PAGE	11
4.1	POB-Compiler	11
4.2	POB-Loader	13
4.3	POB-Bitmap	15
4.4	POB-Pattern.....	19
4.5	POB-Terminal.....	23
5	JAVA DEVELOPMENT WITH POB-EYE.....	24
▪	Java library help.....	24
▪	Java limitations	25
6	SAMPLE APPLICATION.....	27
▪	Real-Time display on POB-LCD128.....	28
▪	Pattern recognition and POB-Terminal display.....	31
▪	Display images on POB-LCD128	33
▪	Reconnaître des formes et manipuler deux tampons graphiques.....	35

Document management

Filename	Manuel POB-Java US.doc
Creation date	30.03.2006
Author	Pierre Séguin

POB-Technology contacts

Address	POB-TECHNOLOGY 4, rue nicéphore niépce 69 680 CHASSIEU, FRANCE
Mail	contact@pob-technology.com
Phone	+33 (0)4 72 43 02 36
Fax	+33 (0)4 78 58 04 92

1 Introduction

POB-Java is a set of tools for programming, loading and controlling the POBEYE module in the Java language. This tool is made up of 6 modules:



POB-TOOLS: Allows managing an application project.

POB-Compiler: Enables one-click source file compilation.

POB-Loader: Uploads an application to the POBEYE module.

POB-Bitmap: Produces a library of pictures for the POB-LCD128.

POB-Pattern: Creates a dictionary of pattern.

POB-Terminal: Helps debug your application with the serial port.

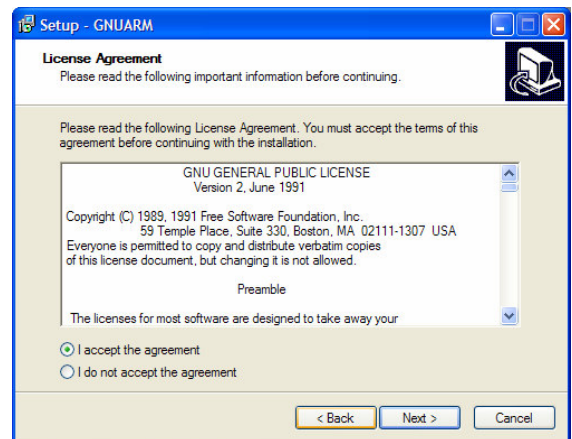
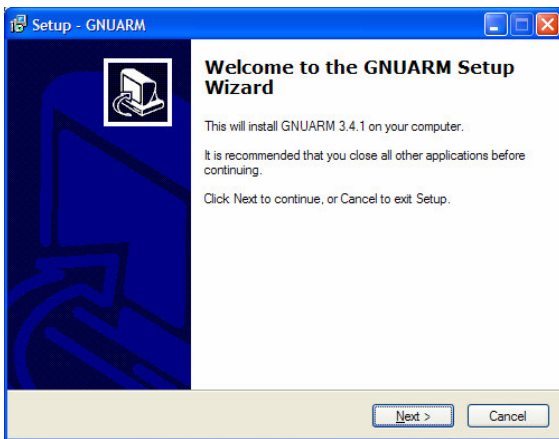
2 Installing POB-Java

On the CD supplied with POBEYE module, you will find 3 files.

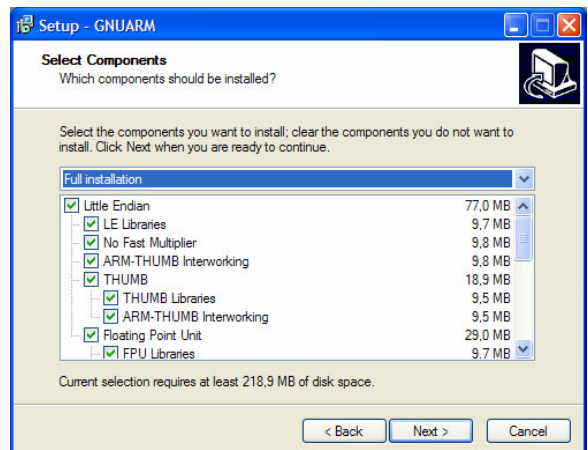
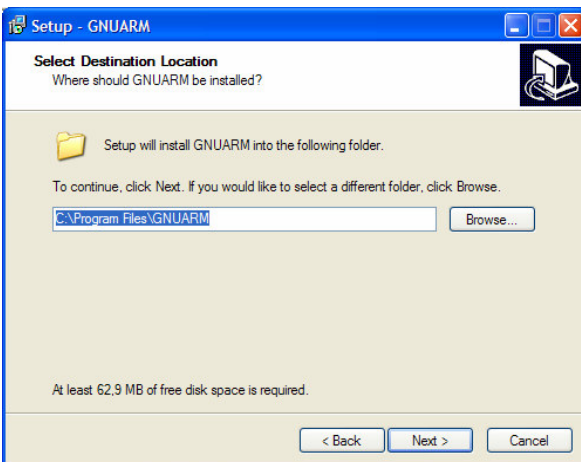
- *Installation of the GNUARM compiler*

The file « **bu-2.15_gcc-3.4.1-c-c++-java_nl-1.12.0_gi-6.0.exe** » is the GNUARM compiler. POB-Java uses this compiler to create your application.

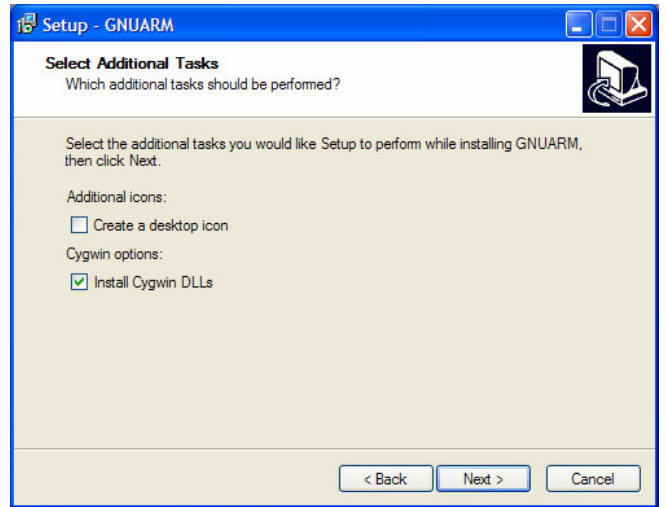
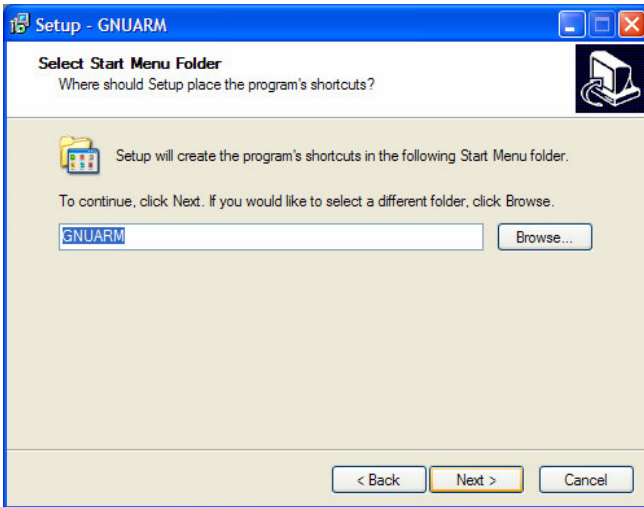
To install the GNUARM compiler, click on this file.



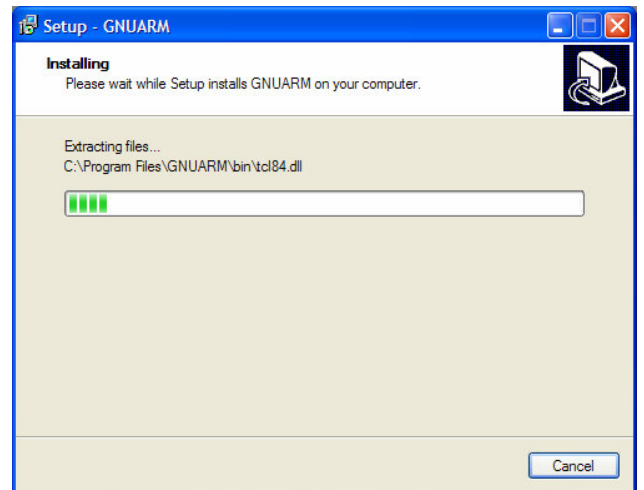
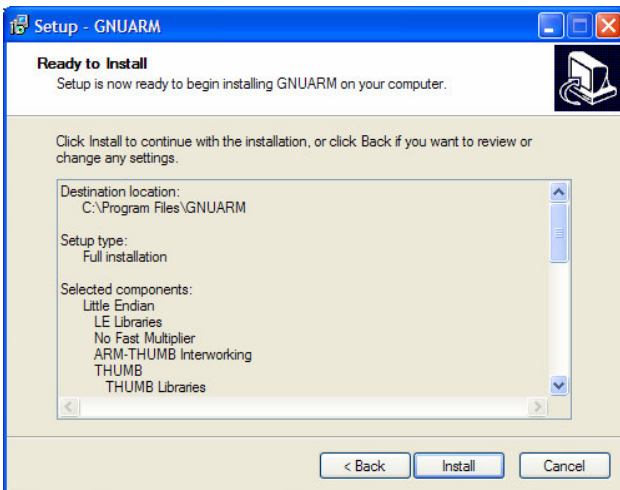
GNUARM introduction and software license: GNU GPL



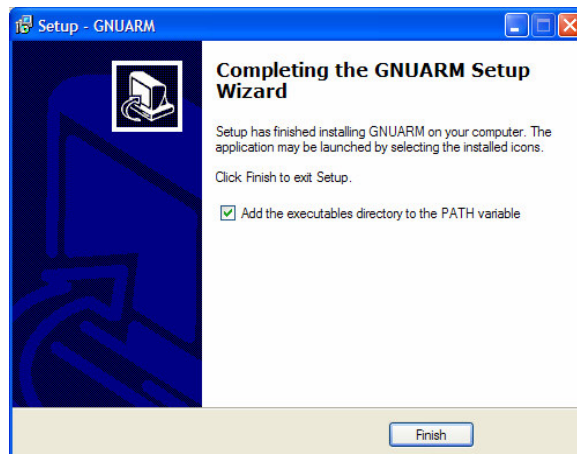
Install repertory and software options (**leave everything ticked**)



Start Menu and options: **Deselect 'Create a desktop icon' and leave 'Install Cygwin DLLs'**



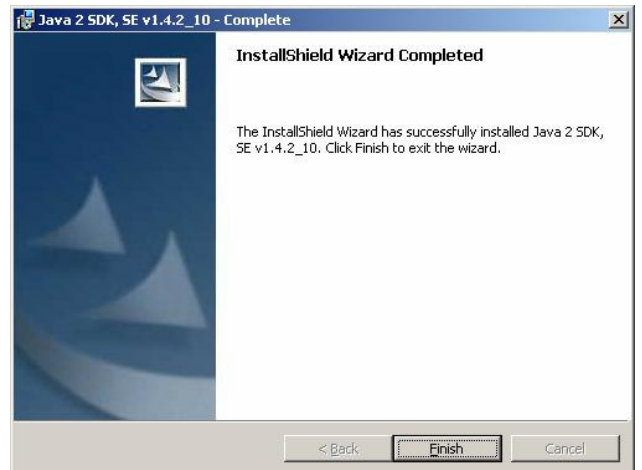
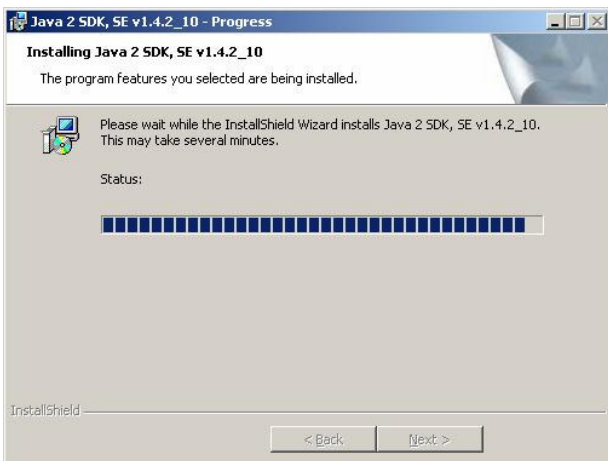
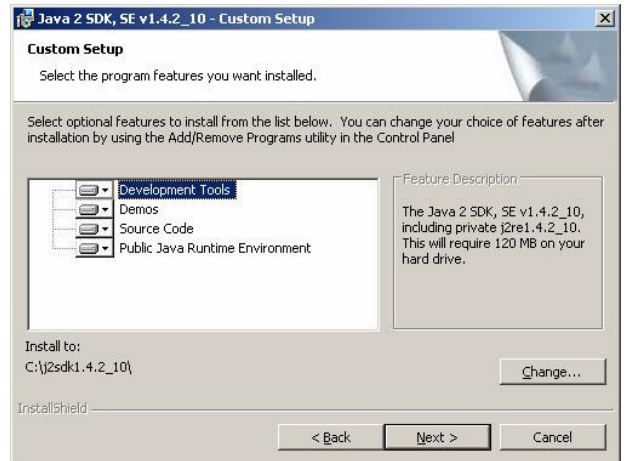
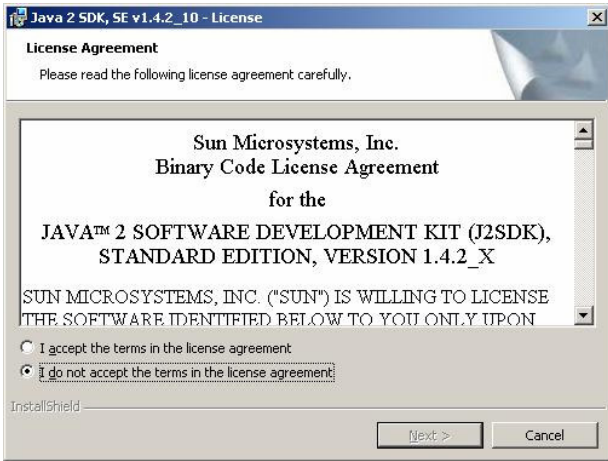
Summary of the components that will be installed and install begin



■ *Installing the Java Development Kit*

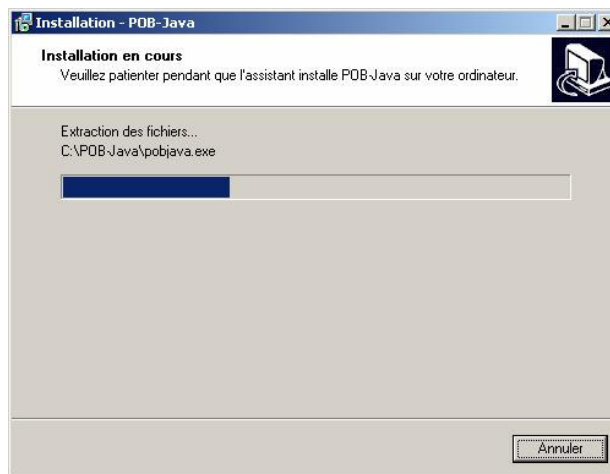
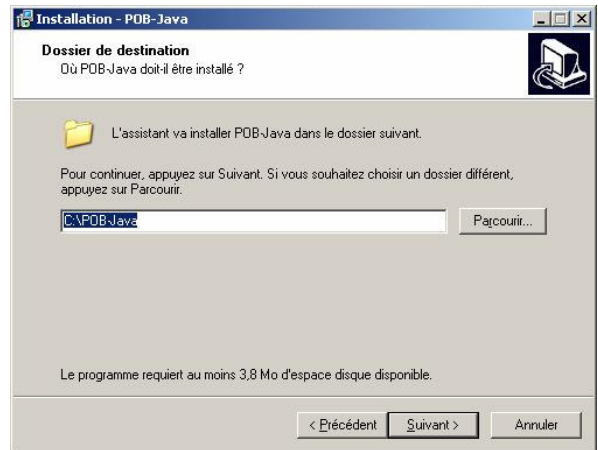
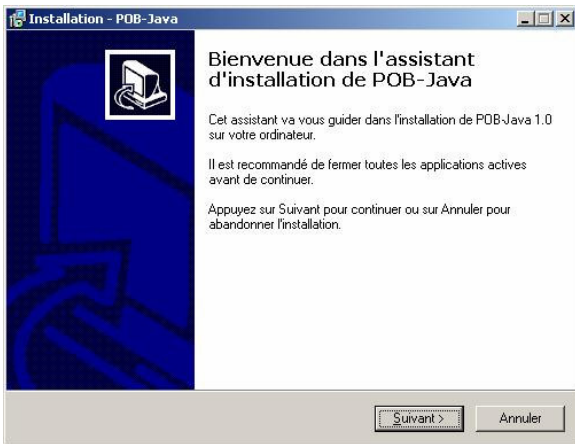
Second file to install is the java development kit « **j2sdk-1_4_2_10-windows-i586-p.exe** ».

Click on the file and follow the instructions to install the JDK on the computer.

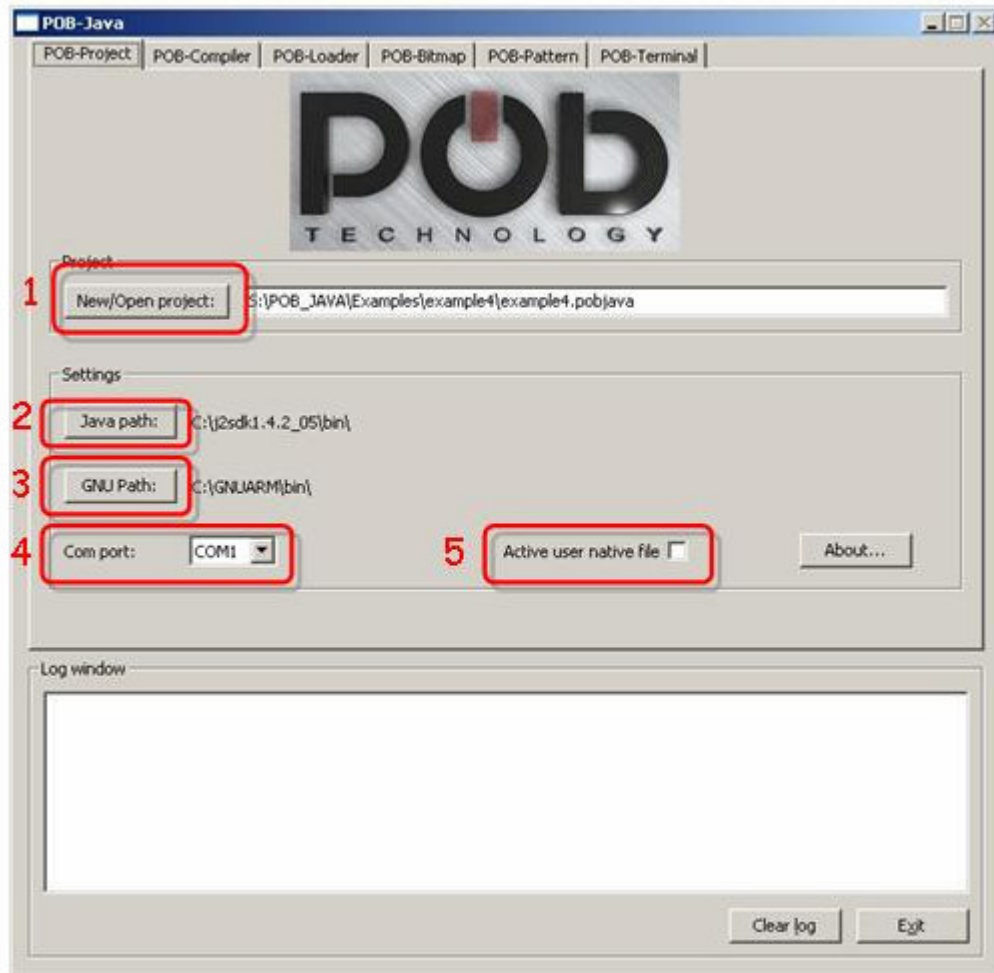


▪ *Installing of POB-Java*

Click on « **pobjava_setup.exe** » to install POB-Java.



3 Configuration



- **1 Manage a project with POB-JAVA**

POB-Project allows creating and opening project for POB-Java.

The “*new/open project*” button allows to create of a new project or opens an existing project. You must type the name of the new project or select an existing project (extension .pobjava)

- **2 Path to Java SDK**

You must click on the ‘**bin**’ directory of Java SDK.

If you have installed it using the default settings, the default path is: **C:\j2sdk1.4.2_10\bin**

▪ *3 Path to use the GNUARM compiler*

You must click on the 'bin' directory of GNUARM.

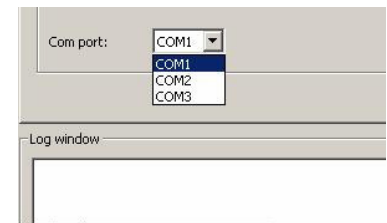
If you have installed it using the default settings, the default path is:

C:\Program Files\GNUARM\bin



▪ *4 Select the serial port*

Choose the port number on which POBEYE is connected to your computer.



▪ *5 User native support*

It is possible have access to the POB-EYE hardware and to mix both Java and C language through the native support.

This option will allow selecting a native source file for compilation from the « POB-Compiler» bookmark. If you choose to use this option you will have to restart the POB-JAVA software.

Warning: Using Native support requires a good knowledge in Java language.

To declare a native method, simply add the « natif » keyword in you method declaration. Example: **public natif void myMethodNatif(void);**

POB-JAVA creates a method prototype in « user_native.h » file. Then you need to create a source file in C and copy paste the prototype from « user_native.h »

Finally, add your file in the POB-COMPILER bookmark.

4 POB-Java Page

4.1 POB-Compiler

POB-EYE module is programmed in the Java language. The POB-Compiler allows you to create an application written in Java for the POB-EYE module. To create an application, follow the 5 following steps:



1 – Naming your application:

Click on « *Filename output* »: a dialog box will be displayed, allow you to set the name of your application (.hex extension).

If the file already exists, you only have to select it.

2 – Add the main java file:

Click on the « *Add your Main java* » button to add the main java file. This file must be the main file of your application. It must contains the method: « **public static void main(String []args** ».

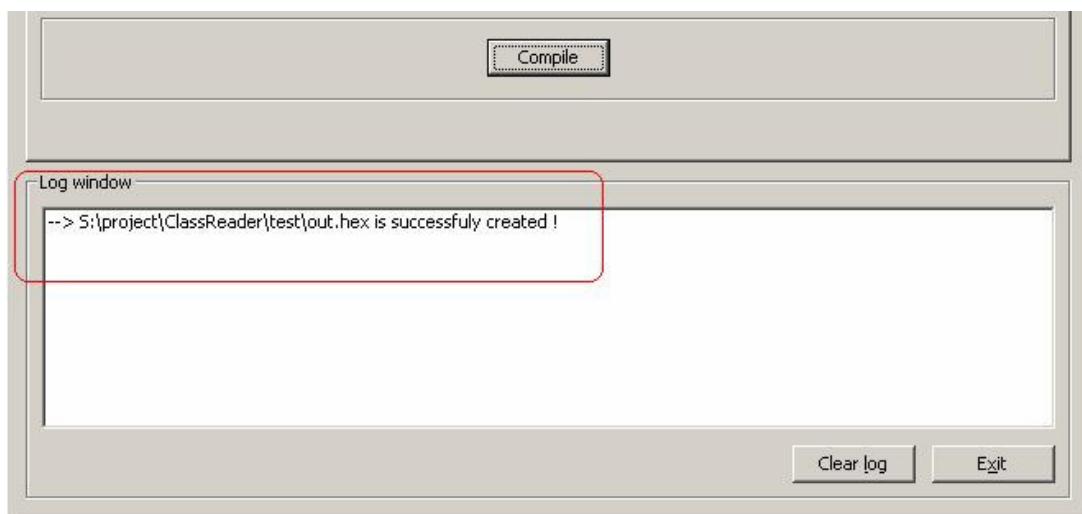
3 – Add a folder in the « classpath »:

You can add a folder in the « *classpath* » if your source code is set in different folders. By default, the folder containing your main file is added.

Remark: If your Java source code is located in the same folder as the main file you don't need to add any folders.

4 - Compiling:

Click on the « **Compile** » button to launch the compilation of your application. During the compilation, you will see traces being displayed in the window called « **Log window** ».



The last line that you will see appearing in the console will tell you if the application was compiled successfully. If the message « **... is successfully created!!!** » appears, your application is ready to be used in the POBEYE module.

If the compilation process failed, the message « **...is not created** » appears. You will then need to correct your application in relation to the message of the compiler.

4.2 POB-Loader

POB-Loader module allows you to load a program in the POBEYE memory.

Two steps are necessary to load a program:



1 – Selecting an application:

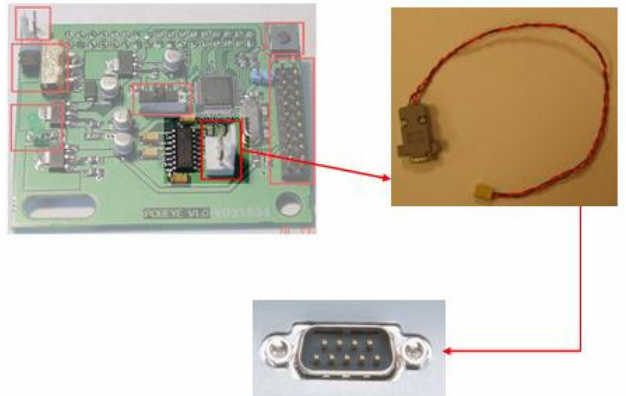
Click on the « *Filename to upload* » button to choose the application.

2 – Program uploads:

Remark: Before loading a program, POBEYE module must be on, in “programming” mode and link to your computer by the serial cable.

Preparing POBEYE module:

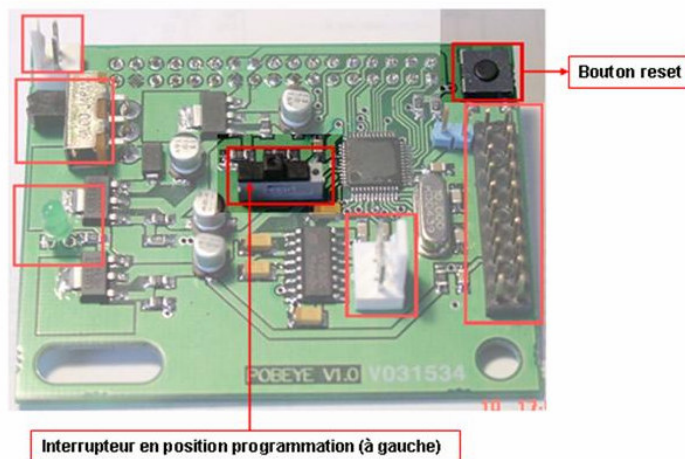
- POBEYE module must be link to your computer: see chapter 1.9
- POBEYE module must be in “programming” mode: see chapter 1.3.



To load a program, click on the “**Upload**” button. If the loading proceeds correctly, you should see the progress bar changing.

If this does not work, an error message will appear. You will have to follow the instructions to solve the problem. If the problem persists, do not hesitate to contact the POB-Technology support: support@pob-technology.com

Remark: If the modules are already on, simply switch to programming mode the switch “programming/execution” and press the reset button.

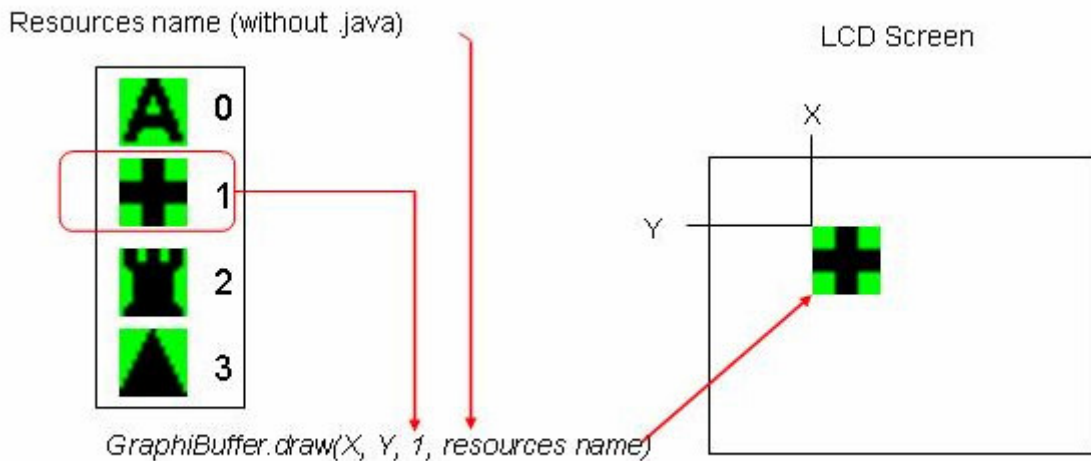


4.3 POB-Bitmap

POB-Bitmap generates graphic resources for the POB-LCD128. POB-LCD allows, for example, real time visualization of what the camera sees or to act as a user interface.

The graphics resources can be displayed using the library supplied with the POB-Java. The graphic functions allow you to manage the transparency of the images and to carry out the superposition of images on the LCD screen.

Images are displayed using the « *GraphicBuffer.draw* » method:

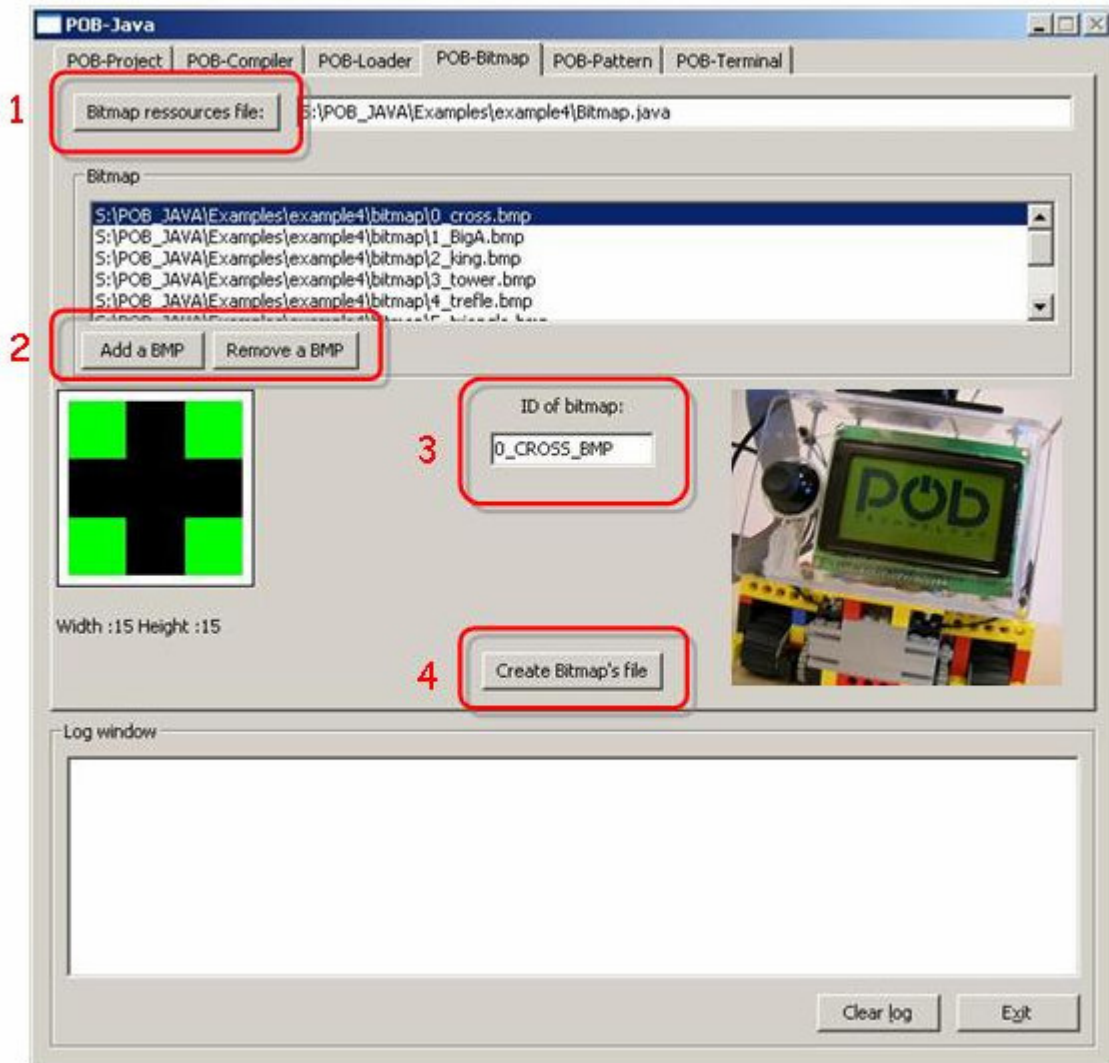


This function uses a number to display the desired image. For more clarity in your code, POB-Bitmap generates in a « .java » a series of constant. The call to « *GraphicBuffer.draw*» can be becomes (if the name of the file is *cross.bmp*):

```
GraphicBuffer.draw (X, Y, 1, Resource Name);
→ GraphicBuffer.draw(X, Y, 1_cross_bmp, Resource Name);
```


- *Graphic resources generation:*

There are 4 steps to create the file:



1 – Naming the file:

Click on « *Bitmap resources file* » to create your bitmap resources file (.java extension). Name of this file is used by the POB-Java library.

Warning: If you choose an existing file, the file content will be lost.

2 – Manage images:

To add or remove an image, press the “*Add a Bmp*” or “*Remove a Bmp*” button.

Remark: The images put in the library must respect the following format: Bitmap 256 colors, maximum size of 256 per 256 pixels.

POB-Bitmap uses 3 colors to draw the graphic resources:

- Black: black pixel is drawing on the LCD.
- White: white pixel is drawing on the LCD.
- Green (Red 0, green 255, Blue 0): transparency color (allow stack images).

3 – Edit the bitmap identifies:

You can rename the bitmap identify. This identify can be used in your code for use the bitmap.

4 – File creation:

Click on « *Create bitmap's file* » to create the java file.

- *About transparency*

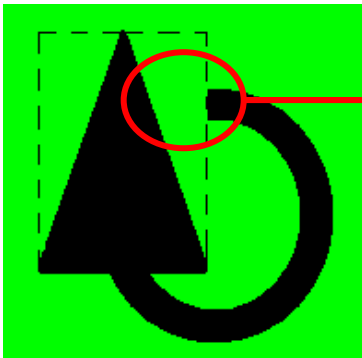
Definition of transparency color:

This color allows a pixel the possibility of not taking shape. Thus, by this means, one keeps what is already drawn.

For example, take's 2 images:

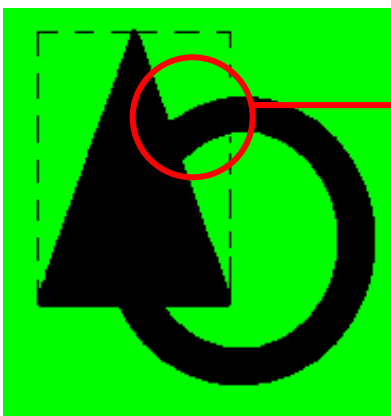


If you draw the triangle on the circle without the management of the color transparency, here is what happens:



The triangle frame erases the circle.

With the transparency management, the superposition of images is possible. Here is the result:

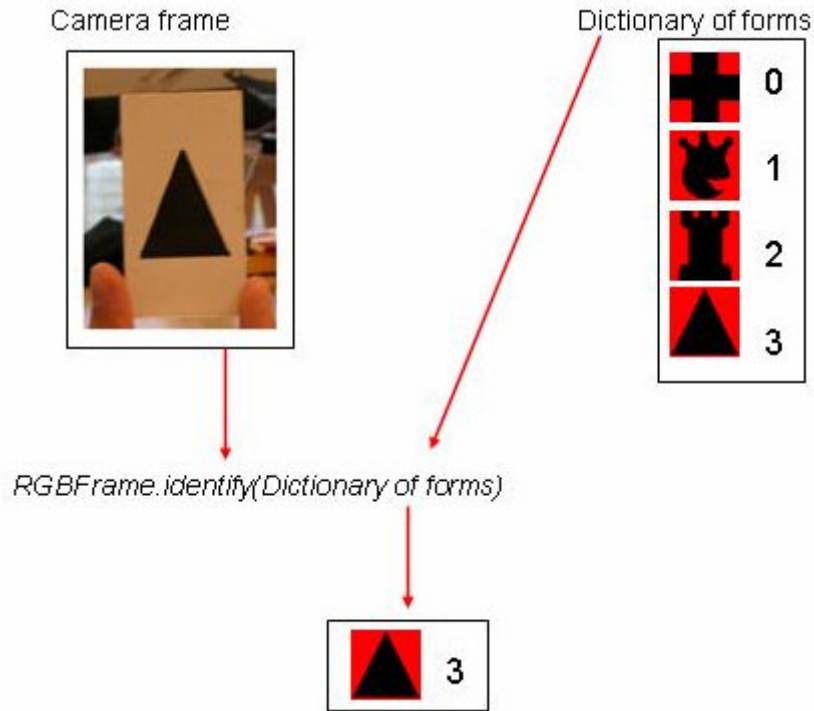


Transparency management allows the part of the circle under the triangle to be shown.

4.4 POB-Pattern

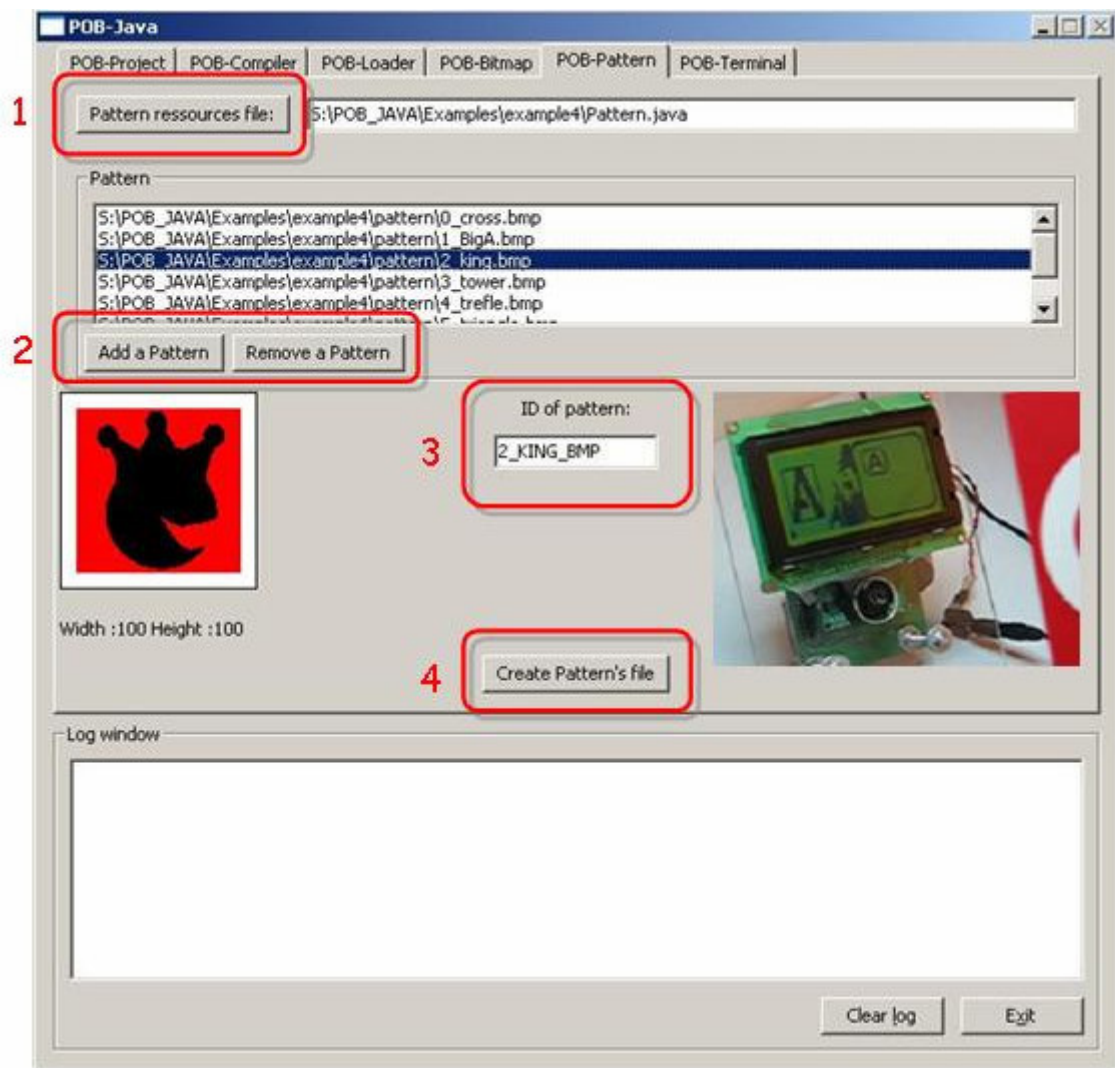
POB-Pattern allows you to create a dictionary of forms. This dictionary allows you to find forms starting from the images of the POB-EYE camera.

The pattern recognition is carried out using the “*RGBFrame.identify*” method.



The « *RGBFrame.identify* » uses the camera frame and the dictionary of forms. The method fills an array with the various forms recognized in the image.

Follow the 4 steps to build your pattern file:



1 – Naming the dictionary:

Click on « *Pattern resources file* » to create the pattern file (.java extension) or choose an existing file.

Warning: If you select an existing file, the file content will be lost.

2 – Manage images:

For add or remove an image, press the “*Add a Pattern*” or “*Remove a Pattern*” button.

Remark: Image must respect the following format: Bitmap 256 colors, **size of 100 per 100 pixels.**

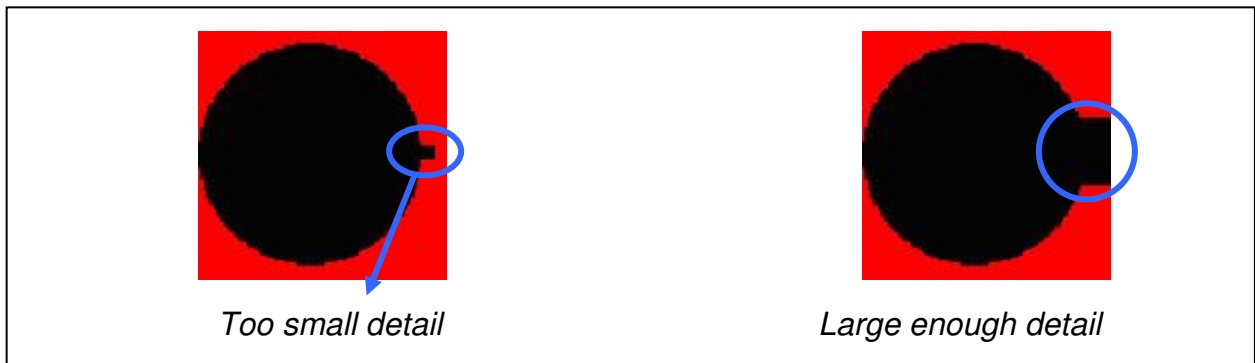
The forms must be drawn in black with a red background

To obtain the best possible result during the creation of the dictionary, certain conditions should be observed:

- The drawn image must take the maximum of space within the framework of 100 per 100 pixels.



- It is necessary to avoid the small details on the image.



- The form should not have any empty space.



3 – Editing the pattern identify:

You can edit the pattern identify.

4 – Dictionary creation:

Click on « *Create pattern's file* » to create the .java file.

4.5 POB-Terminal

POB-Terminal interface allows you to play the message of the POB-EYE module via the serial port. The aim of the POB-Terminal is to facilitate the development of your application.



1 - Connecting POB-Terminal:

The button « *Click to connect* » allows you to connect POB-Terminal to the POB-EYE module. When POB-Terminal is connected, you can disconnect it with the « *Click to disconnect* » button.

2 – Clearing messages:

You can clear the POB-Terminal messages with the « *Clear log* » button.

5 Java development with POB-EYE

- *Java library help*

To get the help on how to use the POB-Java libraries refer to the file « documentation LIBPOB JAVA.html ».


Nom	Taille	Type
Documentation		Dossier de fic
Examples		Dossier de fic
Ressources		Dossier de fic
Documentation LIBPOB JAVA.html	1 Ko	Fichier HTML
pobjava.dll	164 Ko	Extension de
pobjava.exe	1 112 Ko	Application
zlib1.dll	55 Ko	Extension de


Your web explorer will then display the help page.

[Main Page](#) | [Class List](#) | [Class Members](#)

POB-JAVA library Documentation

1.0

Generated by  1.4.1

This is a documentation from  All rights reserved.

From this page you have access to all the function's definitions used in POB-Java library.

▪ *Java limitations*

The POB-JAVA for POB-EYE uses an embedded Java virtual Machine. Nevertheless the one used in POB-EYE does not have all the functions like for a virtual machine in a computer. That's why they are a few constrains to you Java code.

- **Memory:**

The POB-EYE has 64 KB of RAM. 32 KB of this RAM is reserved to store an image from the camera (All three colors are stored).

27 KB of RAM are available for your program (5KB are used for the Java environment). Be aware when using Java objects.

- **64 bits calculus:**

A 64 bits operation is not possible with POB-JAVA: Do not use « **long** » or « **double** » in Java language.

Example of a working source code:

```
float a = (float)1.3456 ;
int b = 2006 + (int) a ;
```

Example of a NONE-working source code:

```
double a = 1.3456 ;
long b = 2006 + (long)a ;
```

- **Exception:**

For now, POB-JAVA does not support simple « *try –catch* » blocs.

Example of a working source code:

```
try {
    // Java code can throw an exception...
} catch( Exception e )
    { /* Exception management */ }
```

Example of a NONE-working source code:

```
try {
    // Java code can throw an exception...

} catch( VotreException e )
    { /* Exception management */ }
```

The instruction *catch* catches only the exceptions.

Example of a NONE-working source code:

```
try {
    // Java code can throw an exception...

} catch( Exception e )
    { /* exception Exception management */ }
catch( VotreException e )
    { /* exception VotreException management */ }
finally { /* instructions bloc finally */ }
```

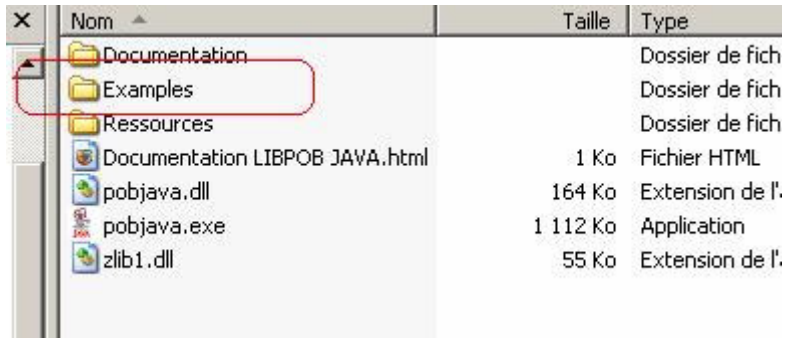
POB-JAVA does not support multiple « *catch* » blocs and the « *finally* » bloc.

- Interface/abstract:

POB-JAVA does not support Java interface.

6 Sample application

Examples are in the “Examples” folder of POB-Java.

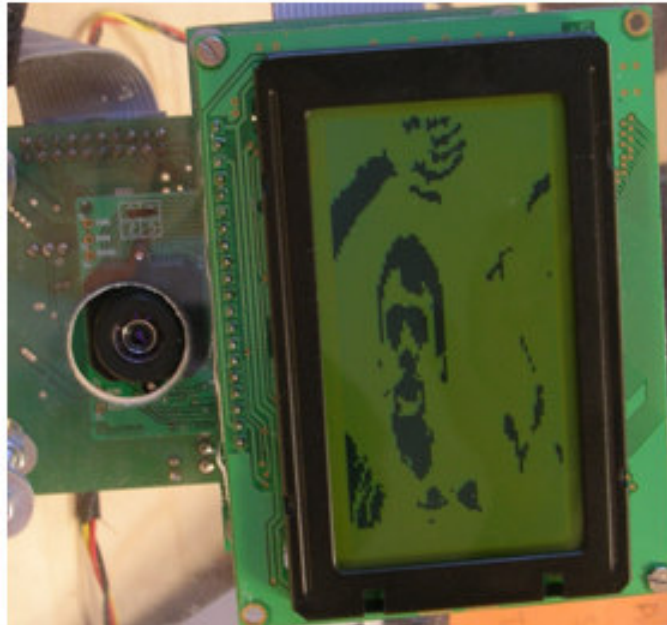


Nom	Taille	Type
Documentation		Dossier de fich
Examples		Dossier de fich
Ressources		Dossier de fich
Documentation LIBPOB JAVA.html	1 Ko	Fichier HTML
pobjava.dll	164 Ko	Extension de l'
pobjava.exe	1 112 Ko	Application
zlib1.dll	55 Ko	Extension de l'

In example repertory, you will find a pob-java project, source code and resources (bitmap or pattern).

- *Real-Time display on POB-LCD128*

This example shows how to display a real-time image on the LCD screen.



First of all you have to import packages to use the POB-JAVA class:

```
/* import all the pob package for use the pobeye/poblcd functions */
import com.pobtechnology.pobeye.* ;
import com.pobtechnology.poblcd.* ;
```

```
public class Example1
{
    public static void main( String args[])
    {
```

To initialize the POB-LCD128 use the following function.

```
/* lcd init */
lcd.init();
```

To draw on screen you need to create a « *GraphicBuffer* » object. This object is a graphic buffer on which all the drawing calculations will be done before being displayed.

The last field indicates if the buffer should be of 1 bit per pixel (true) or 8 bit per pixel (false). In our example we chose a 128 by 64 buffer using 8 bits per pixel.

```
/* create a new graphic buffer */
GraphicBuffer gb = new GraphicBuffer(128,64,false);
```

To grab images from the camera an « *RGBFrame* » object must be created.

```
/* create a new RGB frame */
RGBFrame aFrame = new RGBFrame() ;
```

Before drawing in the buffer you have to clear it.

```
/* clear the graphic buffer */
gb.clear();

int k=0,j=0,i=0 ;
```

The main loop grabs an image from the camera, binarizes it, saves the image in a graphic buffer then display the buffer on the LCD screen.

```
/* main loop */
while (true==true)
{
    /* grab the RGB components from CMOS sensor */
    aFrame.grabRGBFrame();
```

The captured image being in color, it has to be binarized before it can be displayed on the black and white screen.

```
/* binary the RGB frame */
aFrame.binary();
```

Ensuite, le tampon graphique est rempli avec une des composantes de l'image de la caméra. Dans cet exemple, la composante rouge de la caméra est utilisée mais une des autres composantes peut être utilisée (bleu ou vert), elles sont toutes égales : elles ont été binarisées par la méthode précédente.

```

/* put the pixel from the red component which is binary, to the lcd buffer */
for (k=0, i=64 ; i != 0 ; i-- )
{
    for( j=0 ; j<120 ; j++, k++ )
    {
        gb.buffer[k] = (byte)aFrame.getRedPixel( i+(j*88) );
    }
    k+=8;
}

```

Finalement, le tampon graphique est affiché sur l'écran LCD.

```

/* draw the graphic buffer on the screen */
lcd.draw(gb);
}
}
}

```

- *Pattern recognition and POB-Terminal display*

Le programme consiste à capturer une image, à identifier les formes présente dans l'image et à afficher le nom de la forme et les coordonnées dans le POB-Terminal. Ce programme est codé en langage Java et utilise les fonctions du POB-JAVA.

Pour utiliser la caméra et la reconnaissance de forme du POB-JAVA, il faut importer le package `com.pobtechnology.pobeye`.

```
/* import the pobeye package for use the RGBFrame and Form class */
import com.pobtechnology.pobeye.* ;
```

```
public class Example2
{
```

```
    public static void main( String []args)
    {
```

Les formes reconnues vont être placées dans un tableau de type « Form ».

```
        Form form_list[] ;
```

Pour récupérer l'image de la camera POB-EYE, il est nécessaire de créer un nouvel objet « RGBFrame ».

```
        RGBFrame aFrame = new RGBFrame();
```

```
        while(true == true )
        {
```

On capture une image de la camera.

```
            aFrame.grabRGBFrame();
```

Avant de reconnaître des formes, il est nécessaire de binariser l'image capturée.

```
            aFrame.binary();
```

Ensuite, on cherche à identifier les formes dans l'image. Le résultat est placé dans le tableau de formes. Le paramètre de « *identify* » est le nom du dictionnaire crée dans l'onglet POB-Pattern (dans l'exemple 2, ouvrez le projet dans POB-Java, et allez dans l'onglet POB-pattern pour voir le nom du fichier en enlevant l'extension .java et le chemin du fichier).

```
            form_list = aFrame.identify("Pattern");
```

Avant de travailler sur le tableau, il faut **toujours vérifier si l'objet n'est pas nul**.

```
            if( form_list != null )
            {
```

Ensuite, on affiche le résultat dans le POB-Terminal à l'aide de la méthode « System.print ».

```
System.print( "New form :" );

for(int i = 0 ; i < form_list.length ; i++ )
{
    System.print( System.toString(form_list[i].id ) );
    System.print( System.toString(form_list[i].x) );
    System.print( System.toString(form_list[i].y) );
}
}
}
}
```


- *Display images on POB-LCD128*

This program will display images on the LCD and uses all the graphical functions such a “draw line, draw point...”



Pour utiliser le POB-LCD, il faut importer le package « com.pobtechnology.poblcd ».

```
/* import the lcd package for use the graphic buffer and lcd class */
import com.pobtechnology.poblcd.* ;
```

```
public class Example3
{
```

```
    public static void main(String []args)
    {
```

Il est nécessaire d’initialiser l’écran POB-LCD avant d’être utilisé.

```
        /* init the pob-lcd */
        lcd.init();
```

Les opérations de dessin utilisent un objet « GraphicBuffer ». Toutes les opérations de dessin vont manipuler ce tampon avant d’être affiché.

```
        /* create a new graphic buffer of 128 x 64 pixel */
        GraphicBuffer gb = new GraphicBuffer(128,64,true);
```

```
        /* clear the graphic buffer */
        gb.clear();
```

On affiche une image dans le tampon en utilisant la ressource "Bitmap". Le nom de la ressource est crée dans POB-Bitmap lorsque vous sélectionnez le nom du fichier à créer.

Par exemple, le nom de votre fichier est **S:\POB_JAVA\Examples\example3\Bitmap.java**, alors le nom à utiliser dans la méthode « draw » est « **Bitmap** ».

```

/* draw picture */
gb.draw(30,10,0, "Bitmap");
gb.draw(45,10,1, "Bitmap");
gb.draw(65,10,2, "Bitmap");

```

Pour dessiner une ligne, il faut utiliser la méthode « drawLine ». Pour un point, la méthode « plotAPoint » permet d'afficher un point.

```

/* draw a line */
gb.drawLine(10,10,25,30);

/* draw a plot */
gb.plotAPoint(20,10);

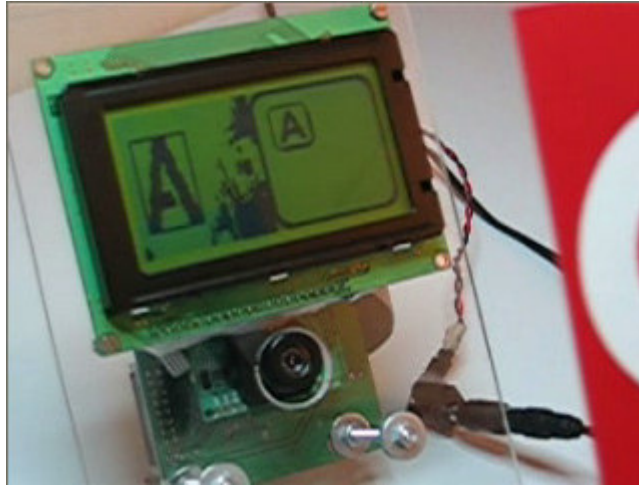
/* draw the graphic buffer on the pob-lcd */
lcd.draw(gb);
}
}

```

- *Reconnaître des formes et manipuler deux tampons graphiques*

Dernier exemple plus complexe, avec manipulation de deux tampons graphiques, reconnaissance de forme et interface graphique.

Le but du programme est de reconnaître des formes, d'afficher l'image capturée sur la partie gauche du POB-LCD et d'afficher sur la partie droite de l'écran les formes reconnues.



Il faut ajouter les packages « com.pobtechnology.pobeye » et « com.pobtechnology.poblcd » pour disposer de l'ensemble des fonctions du POB-EYE et du POB-LCD128.

```
import com.pobtechnology.pobeye.* ;
import com.pobtechnology.poblcd.* ;
```

```
public class Example4
{
    static final int BUFFER_WIDTH = 64;
    static final int BUFFER_HEIGHT = 64;

    public static void main(String []args)
    {
        int i , j ;
        int tmp ;
        int X_Pic, Y_Pic ;
        GraphicBuffer left ,right ;
        RGBFrame aFrame ;
        Form list_of_form[];
```

Le programme initialise l'écran LCD, crée un objet RGBFrame pour récupérer les images de la camera et initialise deux tampons graphiques.

```

lcd.init();

aFrame = new RGBFrame();
left = new GraphicBuffer(BUFFER_WIDTH,BUFFER_HEIGHT,false);
right = new GraphicBuffer(BUFFER_WIDTH,BUFFER_HEIGHT,true);

left.clear();
right.clear();

while(true==true)
{
    // Init the x,y of the first picture
    X_Pic=8;
    Y_Pic=8;
  
```

Dans le tampon droit, on affiche une fenêtre.

```

// draw a frame in right buffer
right.draw(0,0,8, "Bitmap" );
  
```

Depuis la camera, on capture des images et on binarise celle-ci.

```

// Grab the RGB components
aFrame.grabRGBFrame();

// Binary the three RGB Buffer
aFrame.binary();
  
```

On copie les pixels de la caméra dans le tampon graphique de gauche.

```

// copy the video buffer on the left Lcd Screen
for ( i=0 ; i < BUFFER_HEIGHT ; i++)
{
    for (j=1;j<BUFFER_WIDTH;j++)
    {
        tmp = ((i*480)>>8)*88; // manage Y
        tmp += ((j*352)>>8); // manage X

        if(aFrame.getRedPixel(tmp) != 0 )
            left.buffer[i*BUFFER_WIDTH+j]=(byte)0xFF;
        else
            left.buffer[i*BUFFER_WIDTH+j]=0;
    }
}
  
```

Enfin, on identifie les formes. A noter que l'image doit être binarisée avant toutes reconnaissances. Pour reconnaître les formes, on utilise le dictionnaire créé dans le POB-Java. Ce dictionnaire de forme a pour nom « Pattern ».

```
// try to identify the forms and make a list of it.
list_of_form = aFrame.identify("Pattern");
```

Remarque : avant de manipuler le tableau de formes, vérifier toujours si ce tableau n'est pas nul.

```
if( list_of_form != null )
{
    // Parse the list of the form
    for( i=0 ; i < list_of_form.length ; i++ )
    {
```

On copie dans le tampon droit un cadre et l'image reconnue :

```
right.draw(X_Pic, Y_Pic,9, "Bitmap");
// the picture of what is known in the frame
right.draw( X_Pic+3, Y_Pic+3, list_of_form[i].id -1, "Bitmap");

//manage the (x,y) To draw the next known form on the right lcd
screen
X_Pic += 28;
if(( X_Pic+28 ) > BUFFER_WIDTH )
{
    Y_Pic += 26;
    X_Pic = 8;
}
}
}
```

Enfin, on affiche les deux écrans sur le POB-LCD128.

```
// Draw the twis LCDs screens
lcd.drawLeft(left); //the left for the real time video
lcd.drawRight(right); //the right for the result
}

}

}
```


POB-Technology contacts

POB-TECHNOLOGY
4, rue nicéphore niépce
69 680 CHASSIEU
FRANCE

Web: www.pob-technology.com

Mail: contact@pob-technology.com

Phone: +33 (0)4 72 43 02 36
Fax: +33 (0)4 78 58 04 92