Creating a new Processing Library using the Library Builder

Created by: André Faesen

1. Content

[2 Initialisation 3](#_Toc15292456)

[3 Folder structure 4](#_Toc15292457)

[3.1 <<library>> 4](#_Toc15292458)

[3.2 Output 4](#_Toc15292459)

[3.3 Input 5](#_Toc15292460)

[4 Adding a new function 6](#_Toc15292461)

[4.1 Example 6](#_Toc15292462)

[5 Error handling 8](#_Toc15292463)

[5.1 Cannot find symbol 8](#_Toc15292464)

[5.2 Other errors 8](#_Toc15292465)

[6 Design rules 9](#_Toc15292466)

[7 Contact 10](#_Toc15292467)

# Initialisation

The application does not have to be installed on its own but requires other applications to be installed.

Java

The java JDK has to be installed and the correct system variables have to be set.  
The user and system variable JAVA\_HOME has to point to the folder where the JDK is installed, usually this is something like C:\Program Files\Java\jdk1.8.0\_211.

ANT

For the installation of ANT please refer to its installation guide <https://ant.apache.org/manual/install.html>

Python

# Folder structure

The library builder uses a specific file structure to build the required library, as is shown in the picture below.



This diagram shows the different files and folders that should be present in the library builder.

## <<library>>

The <<library>> folder will contain all the files that are mandatory for creating the basic Processing library. These files can be found on the Processing GitHub.

## Output

This folder will contain all the automatic generated files and folders. The java folder will contain the entire java file that is required for the creation of the final library.

## Input

The input folder will contain all the files that have to be modified by the users themselves. Within the input folder a file named functionList.txt exists, which will contain a list of functions that will be added in the library. Besides this file, the folder contains three more folders, the vars folder will contain a file that lists all used variables in the library. The replace folder contains three files that list all functions that are library dependent and have to be replaced in the library. Finally the functions folder contains all functions that have to be added to the library sorted in every type.

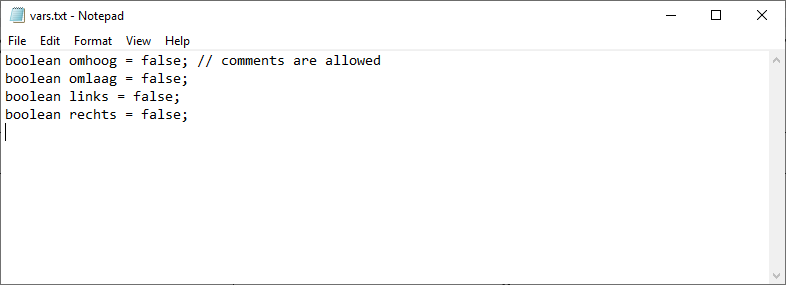
# Adding a new function

After downloading the entire repo of the library builder, new functions can be added immediately. To add these functions, specific steps have to be followed. These steps will be described below.

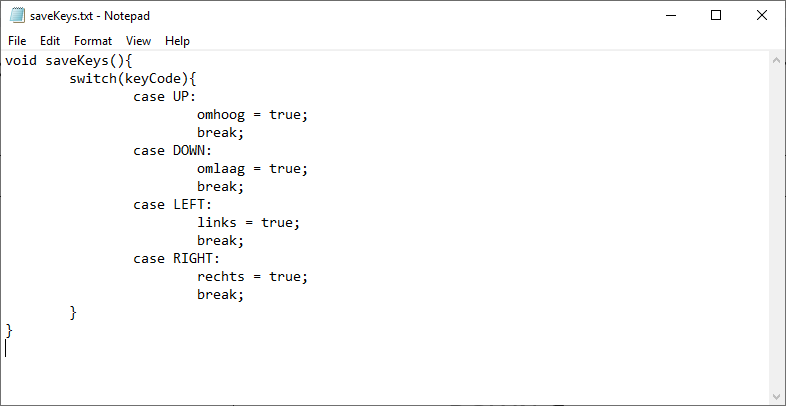
## Example

Lets say we want to add the function named saveKeys. This function will contain multiple variables which it uses internally.

|  |  |
| --- | --- |
| vars | function |
| boolean omhoog = false;  boolean omlaag = false;  boolean links = false;  boolean rechts = false; | void saveKeys(){  switch(keyCode){  case UP:  omhoog = true;  break;  case DOWN:  omlaag = true;  break;  case LEFT:  links = true;  break;  case RIGHT:  rechts = true;  break;  }  } |

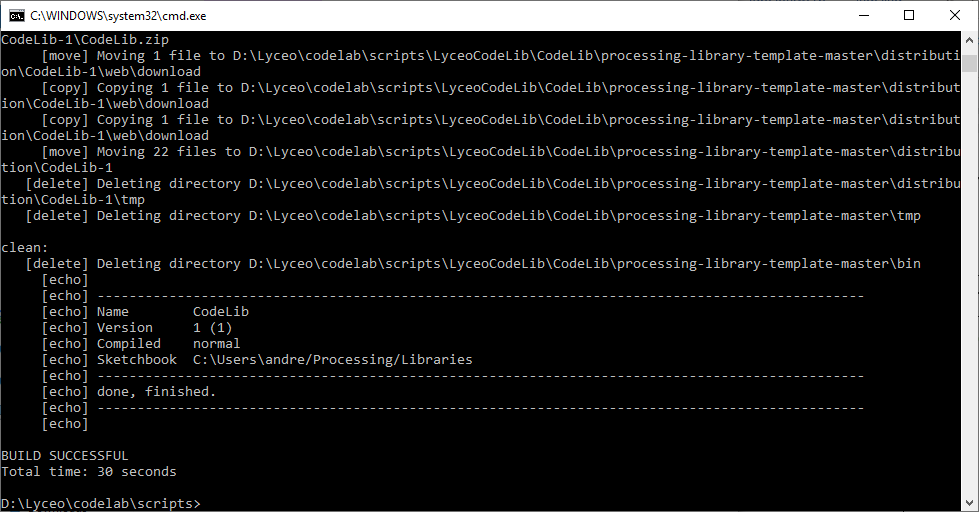
1. Add the function to the function list, as shown below.  
     
   As seen here, the function is added with the right function prefix “void”. This prefix corresponds to the right function folder in the input folder.   
   When adding a new function, make sure the prefix is correct and the name is spelled the same way as the actual function is named.
2. Add the variables to the vars file.  
     
   As shown in the example, the variables are just added in the file and comments are allowed. This file will contain all variables for all the library functions, so you have to make sure that no two variables are named the same.
3. Add the function to the function folder.

In this example we used a function with prefix *void*, therefor a file for the function has to be created in the scripts/input/functions/void folder. This file has to have the same name as the function that will be added, so in this case the file will be named saveKeys.txt.

The entire function will be added to this file, as shown below.  


After this, the library can be build using the libBuilder.cmd file located in the script main directory. This will change and merge all the files, create the output and java files and finally build the entire library.

If the build was successful the output will look like this.



The build can contain warnings but errors are not allowed, handling errors will be explained in the corresponding chapter.

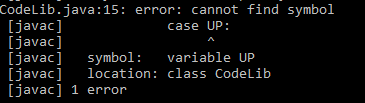
After building the library, it can be found in the Processing sketchbook and in the IDE.

# Error handling

There are multiple types of Errors that can be encountered during the building phase. Most errors are caused by wrongly installed files, or wrong paths. If all files are installed correctly, as described in the first chapter, the only errors that can be encountered during the building phase are input errors.

## Cannot find symbol

If a symbol is missing from one of the replace files, input/replace/\*.txt the builder will output the following error message.



In this case, the UP symbol was missing so it has to be added in one of the replacement files. All standard symbols, have a standard class that they belong to. The three most used classes are Math, PApplet and myParent.

These files can be found in the input/replace/ folder.

The Math class uses mostly math based functions like max(). So if these functions have to be added in the Math.txt file as max(, the ‘(‘ is added so it won’t replace variables with the name *max* in it.

The PApplet class will contain all the functions that are Processing native and do not depend on the instance of the game itself. This includes functions like print(), sq() and enums like UP and DOWN

The myParent class will contain all the functions and vars that are instance specific. This includes functions like line(), rect(), fill() and cursor(), and variables as mouseX, mouseY and key. These functions will use the active instance of the processing application and have a direct link to it.

If a function, enum or variable is missing it can be added freely in one of the corresponding files. Please contact the creator if a new file has to be added to the library builder, since this has to be adapted in the code and then this can be changed on the GitHub for future use.

## Other errors

Please send a message to the creator of the Library builder with a log dump and a zip of your project so he can check what has caused the error.

If an error is java or Ant based, please refer to corresponding installation guides. If these guides are followed by the letter, no errors of these kind should be encountered.

# Design rules

A few rules have to be followed when designing your own functions for the library.

* The int() method is not allowed, just use standard recasts with (int).
* Don’t use recasts to the same datatype, like (int)(x/y) when x and y are already ints.
* Floats should be written in the form of 0.05f not 0.05.
* Don’t use words like *key,* in variable names otherwise the find and replace functions will destroy them.

# Contact

For contact, send a mail to: [a.faesen@gmail.com](mailto:a.faesen@gmail.com) with the prefix “LB:” in the subject.