Processing To C

translator with library

Version 2021.10.07

Table of Contents

Goal

At our Robert Zajonc Institute of Social Studies, we have been dealing with simulations of social processes for over 30 years. Initially, we used for this many variants of Pascal language (Turbo Pascal, Borland Pascal, Delphi and ALGO for didactic purposes) and, of course, C/C++.

As the Pascal language began to go down in history, we looked for another language that could be used both for educating students and for prototyping scientific simulations. We needed the same source code to be used on Windows and MacOS computers, and possibly also on Linux workstations, although that was our main use of C++ there. We wanted the language to have modern syntax with strong typing and classes, and to be as fast as possible. We also wanted the language to be easy to learn for beginners, but also very flexible as our students gain experience.

So, our choice fell on Processing, which we have been using for almost ten years.

The only problem we have struggled with over the years is the need to create two parallel applications for each model: one in Processing as a prototype, and the other in C++ for massive model testing on workstations. It seemed like a waste of time since the syntax of these languages is quite similar.

Processing to C translator & library is an attempt to solve this dilemma.

Translation mechanism

Processing uses a simplified JAVA language syntax, which makes it easier to translate than JAVA itself. The translator does not need to parse the entire code, it only modifies fragments in the context of one line.

At the moment, for translation we use a script with several dozen **sed** calls combined into a long pipe, which in the linux system turned out to be quite efficient and effective solution (*processing2cpp.sh*). Only in some cases it requires taking care of the format in Processing - e.g. headers, class and function declarations must be on one line. In addition, sometimes additional directives are needed, which are simply comments in Processing.

The second script takes care of the translation of the entire Processing project and the preparation of auxiliary files, in particular the ***CMakeLists.txt*** file that allows for automatic compilation of the application (*makeCPPproject.*sh)

To ensure the general visibility of global variables and functions, which are visible in Processing also BEFORE DEFINING, we use a generated header file containing all variables and functions commented with a one-line comment starting with a triple '/' sign go. The third script takes care of it (*prepare_local_h.sh*)

Create a common header file

All lines matching the following ***regex*** patterns shown below go to ***local.h***.

Of course, they are properly translated.

```
'^\s*(final\s+int|final\s+float|final\s+double|final\s+String|final\s+boolean|
final\s+\w+)\s+(\w+)\s*[;=].*///'
```

```
'^\s*(int|float|double|String|boolean|\w+)\s+(\w+)\s*[;=].*///'
```

```
 '^\s*(void|int|float|double|String|boolean)\s+(\w+)\s*\(.*\)\s*\{*\s*///'
```

Specific translations rules

Each translation rule is described with a conversion pattern in sed syntax. The order is the same as in the real pipeline, and it matters.

- Enumeration declarations are copied to the global head, and cannot appear twice in C ++ code, so everywhere else they are turned into comments:

```
's|enum([^\{]*)\{([^\}]*)\}|//enum\1 : \2|'
```

- Constructor body that begins with the keyword super can be automatically parsed if that beginning is on a single line:

```
's|\{(\s*)super\/\*(\w+)\*\/\((.*)\)(s*);|\:\1\2\(\3\)\4{|'
```

- Adding line ends after {} for functions but not for "enum":

```
's|\{(.*)}|{\n\t\1\n\t}|'
```

```
's|(\s*)([;}])(\s*)return([^;]+);|\1\2\n\3\treturn \4;|'
```

- Public / private declarations before functions and variables, fortunately rarely used in Processing, because their context-free translation does not always give the right results:

```
"s/^(\s*)public/\1public:\n\t/g"

"s/^(\s*)private/\1private:\n\t/g"
```

- Converting class declarations only works for single-line declarations with '{' on the same line! There is no equivalent to the abstract classes in C++, so they are changed to normal.

```
's|\s*abstract\s*class(.+)\{|//abstract\nclass\1{|'

's|class(\s+)(\w+)(\s*)extends|class\1\2\3: public |g'
```

- Class representing JAVA interfaces. Inheritance can only be virtual. Processing assumes each class inheriting from the Object class. In C++, we have to say this explicitly:

```
's|\s*interface(.+)\{|//interface\nclass\1\{|'

's|class(.+)\{|class\1\, public virtual Object{|'

's|extends|: public virtual |g'
```

- Also, interface implementations can only be virtual:

```
's|implements|: public virtual|g'
```

- For more complex declarations, you must explicitly prompt the */*_pubext*/* DIRECTIVE

```
's|\/\*_pubext\*\/|public virtual|g'
```

- Change the excess ':' characters to commas

```
's|class(.+)(:\s*public\s*\s+)(\w+)\s*\:(\s*public\s*virtual)(.*)\{|class\1\2\3,\4\5\{|g'

's|class(.+)(:\s*public\s*virtual\s+)(\w+)\s*\:(\s*public\s*virtual)(.*)\{|class\1\2\3,\4\5\
{|g'
```

- When there is no base class, Processing assumes inheriting from the Object class. In C++, we have to say this explicitly:

```
's|class\s+(\w+)\s*\,\s*public|class \1: public|'
```

- By default, class members are public in Processing, and private in C++. Therefore, each definition of class members must begin with '**public**:'

```
's|(\s*)class(\s+)(\w+)(.*)\{|&\n\1  public:|'
```

- Changing the syntax for **array**s of simple values:

```
's/(int|float|double|boolean|String)(\s*)(\[\s*]\s*\[\s*]\s*\[\s*])/scuboid<\1>/'

's/(int|float|double|boolean|String)(\s*)(\[\s*]\s*\[\s*])/smatrix<\1>/'

's/(int|float|double|boolean|String)(\s*)(\[\s*])/sarray<\1>/'

's/new(\s+)(int|float|double|boolean|String)(\s*)\[(.+)](\s*)\[(.+)](\s*)\[(.+)]/new\
1cuboid<\2>(\4,\6,\8)/'
```

```
's/new(\s+)(int|float|double|boolean|String)(\s*)\[(.+)](\s*)\[(.+)]/new\1matrix<\2>(\4,\6)/'

's/new(\s+)(int|float|double|boolean|String)(\s*)\[(.+)]/new\1array<\2>(\4)/'
```

- The other cases are **array**s of objects:

```
's|(\w+)(\s*)(\[\s*]\s*\[\s*]\s*\[\s*])|scuboid<p\1>|g'

's|(\w+)(\s*)(\[\s*]\s*\[\s*])|smatrix<p\1>|g'

's|(\w+)(\s*)(\[\s*])|sarray<p\1>|g'

's|new(\s+)(\w+)(\s*)\[(.+)](\s*)\[(.+)](\s*)\[(.+)]|new\1cuboid<p\2>(\4,\6,\8)|'

's|new(\s+)(\w+)(\s*)\[(.+)](\s*)\[(.+)]|new\1matrix<p\2>(\4,\6)|'

's|new(\s+)(\w+)(\s*)\[(.+)]|new\1array<p\2>(\4)|'
```

- Lists of objects are of special importance, e.g. **ArrayList**<Link> connections; connections=new **ArrayList**<Link>();

```
's|ArrayList<(\s*)(\w+)(\s*)>(\s+)(\w+)(\s*)([\),;:=])|sArrayList<p\2>\4\5\6\7|g'

's|new(\s+)ArrayList<(\s*)(\w+)(\s*)>|new ArrayList<p\3>|g'
```

- **Assert**ion syntax modification:

```
's|assert (.+):|assert \1; // |g'

's|assert ([^;]+);|assert(\1);\t//|g'
```

- Modifying **print** and **println** calls for files:

```
's|(\w+)\.print\(|print(\1,|g'

's|(\w+)\.println\(|println(\1,|g'
```

- More general modifications, little dependent on the context:

```
's/boolean([ >])/bool   \1/g'

's/this\./this->/g'

's|super\/\*(\w+)\*\/\.|\1::|g'

's/super\./super::/'

's/(\w+)(\s+)instanceof(\s+)(\w+)/instanceof< \4 >( \1 )/g'

"s/frameRate(/setFrameRate(/"

's/null/nullptr/g'

's/final /const /g'
```

- Modifications specifically for MATH & FLOATs:

```
's/Float.MAX_VALUE/FLT_MAX/g'
```

```
's/Float.MIN_VALUE/FLT_MIN/g'

's/Integer.parseInt/std::stoi/g'

's/Float.parseFloat/std::stof/g'

's/Math\./std::/g'
```

- Provisional, context-free operator translation >>>:

```
's|>>>(\s*)([0-9]+)|>>\1\2/*UNSIGNED SHIFT EXPECTED*/|g'
```

- DIRECTIVES to improve C ++ translation. Sometimes it is impossible to do without them:

```
's|\/\*_OnlyCppBlockBegin|/*_OnlyCppBlockBegin*/|'

's|_OnlyCppBlockEnd\*\/|/*_OnlyCppBlockEnd*/|'

's|\/\*_interfunc\*\/|virtual|g'

's|\/\*_forcebody\*\/|=0|g'

's|\/\*_downcast\*\/\((\w+)\)|std::dynamic_pointer_cast\<\1\>|g'

's|\/\*_dncast\*\/\((\w+)\)|std::dynamic_pointer_cast\<\1\>|g'

's|\/\*_upcast\*\/\((\w+)\)|static_cast\<p\1\>|g'

's|\/\*_tmpfree\*\/\((\w+)\)|_free_ptr_to\<\1\>|g'

's|\/\*_reference\*\/|\& |g'

's|\s*\/\*_tmpptr\*\/|\* |g'
```

- Wrapping character and string literals in string concatenation operations:

```
"s|\+(\s*)('[^']')|\+\1String(\2)|g"

's|(\"[^"]*\")(\s*)\+|String(\1)\2\+|g'

's|\+(\s*)(\"[^"]*\")|+\1 String(\2)|g'
```

- Translate some JAVA class imports:

```
's/import java.util.Map;/#include "processing_map.hpp"/'

's/import java.util.Arrays;/#include "processing_lists.hpp"/'

's/import java.util.Collections;/#include\<algorithm\>/'
```

- Translation of COLLECTIONS ALGORITHMS (only one for now):

```
's/Collections.sort\((\w+)\);/std::sort(\1.begin(),\1.end());/g'
```

- Methods for class 'processing_window':

```
's/void setup()/void processing_window::setup()/'

's/void draw()/void processing_window::draw()/'
```

```
's/void exit()/void processing_window::exit()/'

's/super::exit()/processing_window_base::exit()/g'
```

- Events handled by the 'processing_window' class:

```
's/void keyPressed()/void processing_window::onKeyPressed()/'

's/void keyReleased()/void processing_window::onKeyReleased()/'

's/void mouseClicked()/void processing_window::onMouseClicked()/'

's/void mousePressed()/void processing_window::onMousePressed()/'

's/void mouseReleased()/void processing_window::onMouseReleased()/'

's/void mouseMoved()/void processing_window::onMouseMoved()/'

's/void mouseDragged()/void processing_window::onMouseDragged()/'
```

- Translation of some JAVA exceptions (http://www.cplusplus.com/reference/ios/ios/exceptions/)

```
's/IOException/std::ifstream::failure/g'

's/Exception/std::runtime_error/g'

's/throw(\s+)new/throw/g'
```

- REPLACEMENT OF "AAA" USER TYPES INTO intelligent "pAAA" type pointers. The file '*userclasses.sed*' is created by the script '*prepare_local_h.sh*':

```
-f userclasses.sed
```

- CONVERTING DOT CALLS TO ARROW CALLS:
  - First, we try to protect the names of files enclosed in "" - this mainly applies to inclusions, but may also cause problems in long text concatenations.

    ```
    's/"(.+)\.(.+)"/"\1@@@\2"/'
    ```

  - NOW A REAL SWITCH:

    ```
    's/([_a-zA-Z][_a-zA-Z0-9]*)\.([_a-zA-Z][_a-zA-Z0-9]*)/\1->\2/g'

    's/\]\.([_a-zA-Z][_a-zA-Z0-9]*)/]->\1/g'

    's/\)\.([_a-zA-Z][_a-zA-Z0-9]*)/)->\1/g'

    's/([_a-zA-Z][_a-zA-Z0-9]*)\.([_a-zA-Z][_a-zA-Z0-9]*)/\1->\2/g'
    ```

  - Finally, restore the filenames:

    ```
    's/"(.+)\@\@\@(.+)"/"\1.\2"/'
    ```

- CLEANING. If by any chance any of the base types are treated as an object in the <> template brackets:

```
's/\<p(bool|int|long|float|double|String)(\s*)\>/\1\2/g'
```

- CLEANING. Combining redundant comment marks.

```
's|//\s+//|//|g'
```

Compilation using cmake

The main script that translates the project is called '*processing2cpp.sh*'.

It must be used in the Processing project directory containing all its '*pde*' files.

As a result of its operation, the 'cppsrc' directory, the 'local.h' file and the '*CMakeList.txt*' file are created, which enables the compilation of the entire project. Then, the following commands make the application:

```
$ cmake .

$ make
```

For now, as in Processing itself, compilation is done from one file, called here '*all_in_one.cpp*', which includes the actual source files resulting from translation with include directives.

We recommend QtCreator as an IDE as it can handle '*CMakeList.txt*' files directly.

Library

Processing has a very extensive "standard library" with numerous functions and classes focused primarily on graphics programming. Much of it is based on JAVA libraries.

The translator is so far a one-man work, so creating a replica of the entire Processing library is beyond the author's ability.

Only a modest subset used in our simulation programs has been implemented. The following chapters detail what is implemented.

Structure

() (parentheses) – work like in C++
, (comma)– work like in C++ (probably)
. (dot) – in most cases translated into ->
/* */ (multiline comment) – work like in C++
/** */ (doc comment)
// (comment) – work like in C++
; (semicolon) – work like in C++, but for the end of class declaration should be added manually. Such modification work both in Processing and in C++
= (assign) – work like in C++
[] (array access) – work like in C++, because of library object **array**, **sarray**, **matrix**, **smatrix** (and others)
{} (curly braces) – work like in C++
catch – same syntax in Processing/Java/C++ but different exception names!

class – syntax and semantics are different. Translator always try to make as many translation as possible, but often special directives and/or manual changes are needed.
draw() – function with special meaning translated into **processing_window::draw()**
exit() – function with special meaning translated into **processing_window::exit()**
extends – translated into "**: public**"
false – work like in C++
final - translated into "**const**"
implements - translated into "**: public**" (redundancy with **extend** are removed)
import – sometimes translated into #include
loop() - as library **function**
new – work like similar C++ because of implementation of Processing::ptr<T>, and sarray etc...
noLoop() - library **function**
null - - translated into "**nullptr**"
pop() - NOT IMPLEMENTED
popStyle() - NOT IMPLEMENTED

private public - syntax and meaning is different. Translator always try to make as many translation as possible, but special directives and/or manual changes may be needed (see → "class")

push() - NOT IMPLEMENTED
pushStyle() - NOT IMPLEMENTED
redraw() - library function, still not used
return – work like in C++
setup() – function with special meaning translated into **processing_window::setup()**
static – remain in code, but may not work properly!
super – it can be translated automatically in some contexts, but in many cases requires manual translation.!
this. – All ***this.*** are replaced by **this->**
thread() - there is different philosophy in C++, so **NOT IMPLEMENTED**!
true – work like in C++
try - same syntax in Processing/Java/C++ but different exception names!
void – work like in C++

Control

Relational Operators

!= (inequality) – work like in C++

< (less than) – work like in C++

<= (less than or equal to) – work like in C++

== (equality) – work like in C++

> (greater than) – work like in C++

>= (greater than or equal to) – work like in C++

Iteration

for while – work like in C++

Conditionals

?: (conditional) – work like in C++

break – work like in C++

case – work like in C++

continue – work like in C++

default – work like in C++

else – work like in C++

if – work like in C++

switch – work like in C++

Logical Operators

! (logical NOT) – work like in C++
&& (logical AND) – work like in C++
|| (logical OR) – work like in C++

Environment

cursor() - library **function. Only makes a cursor visible if already hidden**
delay() - library **function**
displayDensity() - dummy implementation, which always return 1
focused - NOT IMPLEMENTED! Confirms if a Processing program is "focused," meaning that it is active and will accept mouse or keyboard input. This variable is "true", if it is focused and "false", if not.
frameCount - library variable with **const**
frameRate() - translated into **setFrameRate()** library function
frameRate - library variable with **const**
fullScreen() - library **function**
height - library variable with **const**
noCursor() - library **function**
noSmooth() - library **function**
pixelDensity() - dummy implemented. IGNORED
pixelHeight – same value as height
pixelWidth -same value as width
settings() - The **settings()** function is new with Processing 3.0. It's not needed in most sketches. NOT IMPLEMENTED
size() - library **function**
smooth() - library **function**
width - library variable with **const**

Event handling

Mouse

mouseButton - library **variable**

mouseClicked() - empty library **function for reiplementation by user.** Renamed for **processing_window::onMouseClicked()**

mouseDragged() - NOT IMPLEMENTED

mouseMoved() - NOT IMPLEMENTED

mousePressed() - empty library **function for reiplementation by user.** Renamed for **processing_window::onMousePressed()**

mousePressed - library **variable**

mouseReleased() - empty library **function for reiplementation by user.** Renamed for **processing_window::onMouseReleased()**

mouseWheel() - NOT IMPLEMENTED

mouseX - library **variable**

mouseY - library **variable**

pmouseX - library **variable**

pmouseY - library **variable**

Keyboard

key - library **variable**

keyCode - NOT IMPLEMENTED YET

keyPressed() - empty library **function for reiplementation by user.** Renamed for **processing_window::onKeyPressed()**

keyPressed - library **variable**

keyReleased() - empty library **function for reiplementation by user.** Renamed for **processing_window::onKeyReleased**

keyTyped()- NOT IMPLEMENTED YET

Data

Primitive

boolean - translated into **bool**

byte - NOT IMPLEMENTED YET

char – work like in C++

color – implemented as class

double – work like in C++

float – work like in C++

int – work like in C++

long – work like in C++

String - translated into **String, _param_string** library classes derived from **std::string**

Object – In Processing, like in JAVA, objects are instances of classes accessed by some kind of reference with counting (“.” operator, but managed heap with garbage collection is used). Similar meaning in C++ have **std::shared_ptr**s, but they have different interface. So we translate such references into **Processing::ptr<T>** templates opaquing **shared_ptr**s. It saves compatibility, but it is not very efficient. In many cases, especially as function parameters, such **ptr**s could be replaced with **Processing::ptr<T>&** or even **T&**. But this should be done manually and very carefully.

Composite

Array - translated into **array, matrix** library classes

ArrayList – implemented as class based on std::vector. Only most important methods!

DoubleDict - NOT IMPLEMENTED YET

DoubleList - NOT IMPLEMENTED YET

FloatDict - NOT IMPLEMENTED YET

FloatList implemented as class based on std::vector. Only most important methods!

HashMap - NOT IMPLEMENTED YET

IntDict - NOT IMPLEMENTED YET

IntList implemented as class based on std::vector. Only most important methods!

JSONArray - NOT IMPLEMENTED YET

JSONObject - NOT IMPLEMENTED YET

LongDict - NOT IMPLEMENTED YET

LongList - NOT IMPLEMENTED YET

StringDict - NOT IMPLEMENTED YET

StringList implemented as class based on std::vector. Only most important methods!

Table - NOT IMPLEMENTED YET

TableRow - NOT IMPLEMENTED YET

XML - NOT IMPLEMENTED YET

Conversion

binary() - implemented

boolean() - NOT IMPLEMENTED YET, but may work when is C++ correct.

byte() - NOT IMPLEMENTED YET, but may work when is C++ correct.

char() - NOT IMPLEMENTED YET, but may work when is C++ correct.

float() - for float(String) need to be replaced by **Float.parseFloat** and translated from this syntax.

hex() - implemented

int() - - for int(String) need to be replaced by **Integer.parseInt** and translated from this syntax.

str() - NOT IMPLEMENTED YET, but may work when is C++ correct.

unbinary() - NOT IMPLEMENTED YET

unhex() - NOT IMPLEMENTED YET

String Functions

join() - NOT IMPLEMENTED YET

match() - NOT IMPLEMENTED YET

matchAll() - NOT IMPLEMENTED YET

nf() nfc() nfp() nfs() - implemented as library **functions**

split() - implemented as library **functions**

splitTokens() - NOT IMPLEMENTED YET

trim() - NOT IMPLEMENTED YET

Array Functions

append() - NOT IMPLEMENTED YET
arrayCopy() - NOT IMPLEMENTED YET
concat() - NOT IMPLEMENTED YET
expand() - NOT IMPLEMENTED YET
reverse() - NOT IMPLEMENTED YET
shorten() - NOT IMPLEMENTED YET
a.sort() - translated int std::sort(a);
splice() - NOT IMPLEMENTED YET
subset() - NOT IMPLEMENTED YET

Math

PVector - NOT IMPLEMENTED YET

Operators

% (modulo) – work like in C++

* (multiply) – work like in C++

*= (multiply assign) – work like in C++

+ (addition) – work like in C++

For Strings work +- like in Processing but there are many warnings from compiler!
Manualy replacing + with & may remove some warnings.

++ (increment) – work like in C++

+= (add assign) – work like in C++

- (minus) – work like in C++

-- (decrement) – work like in C++

-= (subtract assign) – work like in C++

/ (divide) – work like in C++

/= (divide assign) – work like in C++

Bitwise Operators

& (bitwise AND) – work like in C++

<< (left shift) – work like in C++

>> (right shift) – work like in C++

>>> - translated into >> with comment:

```
a >>> b;  ==>  a >> b; /*UNSIGNED SHIFT EXPECTED*/
```

| (bitwise OR) – work like in C++

Calculation

abs() – work like in C++

ceil() – work like in C++

constrain() - ???

dist() - ???

exp() – work like in C++

floor() – work like in C++

lerp() - IMPLEMENTED AS INLINE FUNCTION

log() – work like in C++

mag() - ???

map() - IMPLEMENTED AS INLINE FUNCTION

max() – implemented but mostly work like in C++

min() – implemented but mostly work like in C++

norm() - IMPLEMENTED AS INLINE FUNCTION

pow() – work like in C++

round() - ???

sq() - ???

sqrt() – work like in C++

Trigonometry

acos() – work like in C++

asin() – work like in C++

atan() – work like in C++

atan2() – work like in C++

cos() – work like in C++

degrees() - IMPLEMENTED AS INLINE FUNCTION

radians() - IMPLEMENTED AS INLINE FUNCTION

sin() – work like in C++

tan() – work like in C++

Random

noise() NOT IMPLEMENTED YET
noiseDetail() NOT IMPLEMENTED YET
noiseSeed() NOT IMPLEMENTED YET
random() - IMPLEMENTED AS LIBRARY FUNCTION
randomGaussian() NOT IMPLEMENTED YET
randomSeed() - IMPLEMENTED AS LIBRARY FUNCTION

Constants

HALF_PI - IMPLEMENTED
PI - IMPLEMENTED
QUARTER_PI - IMPLEMENTED
TAU - IMPLEMENTED
TWO_PI - IMPLEMENTED

Color

Setting

background() - library **function**

clear() - NOT IMPLEMENTED YET

colorMode() - NOT IMPLEMENTED

fill() - library **function**

noFill() - library **function**

noStroke() - library **function**

stroke() - library **function**

Creating & Reading

alpha() - NOT IMPLEMENTED
blue() - library **function**
brightness() - NOT IMPLEMENTED
color() - library **function**
green() - library **function**
hue() - NOT IMPLEMENTED
lerpColor() - NOT IMPLEMENTED
red() - library **function**
saturation() - NOT IMPLEMENTED

Shape

createShape() - **NOT IMPLEMENTED (dummy only)**
loadShape() - **NOT IMPLEMENTED (dummy only)**

PShape - **NOT IMPLEMENTED (dummy only)**

Attributes

ellipseMode() - library **function**

rectMode() - library **function**

strokeCap() - library **function**

strokeJoin() - library **function**

strokeWeight() - library **function**

2D Primitives

arc() - library **function**

circle() - NOT IMPLEMENTED

ellipse() - library **function**

line() - library **function**

point() - library **function**

quad() - NOT IMPLEMENTED

rect() - library **function**

square() - NOT IMPLEMENTED

triangle() - NOT IMPLEMENTED

Curves

bezier() - NOT IMPLEMENTED

bezierDetail() - NOT IMPLEMENTED

bezierPoint() - NOT IMPLEMENTED

bezierTangent() - NOT IMPLEMENTED

curve() - NOT IMPLEMENTED

curveDetail() - NOT IMPLEMENTED

curvePoint() - NOT IMPLEMENTED

curveTangent() - NOT IMPLEMENTED

curveTightness() - NOT IMPLEMENTED

3D Primitives - NOT IMPLEMENTED

box() - 3D GRAPHIX NOT IN MY PLAN

sphere() - 3D GRAPHIX NOT IN MY PLAN

sphereDetail() - 3D GRAPHIX NOT IN MY PLAN

Vertex - NOT IMPLEMENTED

beginContour() - NOT IMPLEMENTED

beginShape() - NOT IMPLEMENTED (dummy present)

bezierVertex() - NOT IMPLEMENTED

curveVertex() - NOT IMPLEMENTED

endContour() - NOT IMPLEMENTED

endShape() - NOT IMPLEMENTED (dummy present)

quadraticVertex() - NOT IMPLEMENTED

vertex() - NOT IMPLEMENTED (dummy present)

Loading & Displaying

shape() - NOT IMPLEMENTED
shapeMode() - NOT IMPLEMENTED

Input - NOT IMPLEMENTED

Files

BufferedReader – library class wraping std::ifstream

createInput() - NOT IMPLEMENTED

createReader() - library function

launch() - NOT IMPLEMENTED

loadBytes() - NOT IMPLEMENTED

loadJSONArray() - NOT IMPLEMENTED

loadJSONObject() - NOT IMPLEMENTED

loadStrings() - NOT IMPLEMENTED

loadTable() - NOT IMPLEMENTED

loadXML() - NOT IMPLEMENTED

parseJSONArray() - NOT IMPLEMENTED

parseJSONObject() - NOT IMPLEMENTED

parseXML() - NOT IMPLEMENTED

selectFolder() - NOT IMPLEMENTED

selectInput() - NOT IMPLEMENTED

Time & Date

day() - library function
hour() - library function
millis() - library function
minute() - library function
month() - library function
second() - library function
year() - library function

Output

Text Area

print() - library **functions**

printArray() - NOT IMPLEMENTED

println() - library **functions**

Image

save() - library **functions (not fully compatible)**

saveFrame() - library **functions (not fully compatible)**

Files

beginRaw() - NOT IMPLEMENTED
beginRecord() - NOT IMPLEMENTED
createOutput() - NOT IMPLEMENTED
createWriter() - library **function**
endRaw() - NOT IMPLEMENTED
endRecord() - NOT IMPLEMENTED
PrintWriter - library **class masking std::fstream**
saveBytes() - NOT IMPLEMENTED
saveJSONArray() - NOT IMPLEMENTED
saveJSONObject() - NOT IMPLEMENTED
saveStream() - NOT IMPLEMENTED
saveStrings() - NOT IMPLEMENTED
saveTable() - NOT IMPLEMENTED
saveXML() - NOT IMPLEMENTED
selectOutput() - NOT IMPLEMENTED

Transform - NOT IMPLEMENTED

applyMatrix() - 3D GRAPHIX NOT IN MY PLAN
popMatrix() - 3D GRAPHIX NOT IN MY PLAN
printMatrix() - 3D GRAPHIX NOT IN MY PLAN
pushMatrix() - 3D GRAPHIX NOT IN MY PLAN
resetMatrix() - 3D GRAPHIX NOT IN MY PLAN
rotate() - 3D GRAPHIX NOT IN MY PLAN
rotateX() - 3D GRAPHIX NOT IN MY PLAN
rotateY() - 3D GRAPHIX NOT IN MY PLAN
rotateZ() - 3D GRAPHIX NOT IN MY PLAN
scale() - 3D GRAPHIX NOT IN MY PLAN
shearX() - 3D GRAPHIX NOT IN MY PLAN

shearY() - 3D GRAPHIX NOT IN MY PLAN
translate() - 3D GRAPHIX NOT IN MY PLAN

Lights, Camera - NOT IMPLEMENTED

Lights - NOT IMPLEMENTED

ambientLight() - 3D GRAPHIX NOT IN MY PLAN

directionalLight() - 3D GRAPHIX NOT IN MY PLAN

lightFalloff() - 3D GRAPHIX NOT IN MY PLAN

lights() - 3D GRAPHIX NOT IN MY PLAN

lightSpecular() - 3D GRAPHIX NOT IN MY PLAN

noLights() - 3D GRAPHIX NOT IN MY PLAN

normal() - 3D GRAPHIX NOT IN MY PLAN

pointLight() s- 3D GRAPHIX NOT IN MY PLAN

potLight() - 3D GRAPHIX NOT IN MY PLAN

Camera - NOT IMPLEMENTED

beginCamera() - 3D GRAPHIX NOT IN MY PLAN

camera() - 3D GRAPHIX NOT IN MY PLAN

endCamera() - 3D GRAPHIX NOT IN MY PLAN

frustum() - 3D GRAPHIX NOT IN MY PLAN

ortho() - 3D GRAPHIX NOT IN MY PLAN

perspective() - 3D GRAPHIX NOT IN MY PLAN

printCamera() - 3D GRAPHIX NOT IN MY PLAN

printProjection() - 3D GRAPHIX NOT IN MY PLAN

Coordinates - NOT IMPLEMENTED

modelX() - NOT IMPLEMENTED

modelY() - NOT IMPLEMENTED

modelZ() - 3D GRAPHIX NOT IN MY PLAN

screenX() - NOT IMPLEMENTED

screenY() - NOT IMPLEMENTED

screenZ() - 3D GRAPHIX NOT IN MY PLAN

Material Properties - NOT IMPLEMENTED

ambient() - 3D GRAPHIX NOT IN MY PLAN
emissive() - 3D GRAPHIX NOT IN MY PLAN
shininess() - 3D GRAPHIX NOT IN MY PLAN
specular() - 3D GRAPHIX NOT IN MY PLAN

Image - NOT IMPLEMENTED

createImage() - NOT IMPLEMENTED

PImage - NOT IMPLEMENTED

Loading & Displaying - NOT IMPLEMENTED

image() - NOT IMPLEMENTED

imageMode() - NOT IMPLEMENTED

loadImage() - NOT IMPLEMENTED

noTint() - NOT IMPLEMENTED

requestImage() - NOT IMPLEMENTED

tint() - NOT IMPLEMENTED

Textures- NOT IMPLEMENTED

texture() - 3D GRAPHIX NOT IN MY PLAN

textureMode() - 3D GRAPHIX NOT IN MY PLAN

textureWrap() - 3D GRAPHIX NOT IN MY PLAN

Pixels - NOT IMPLEMENTED

Blend() - - NOT IMPLEMENTED
copy() - NOT IMPLEMENTED
filter() - NOT IMPLEMENTED
get() - NOT IMPLEMENTED
loadPixels() - NOT IMPLEMENTED
pixels[] - NOT IMPLEMENTED
set() - NOT IMPLEMENTED
updatePixels() - NOT IMPLEMENTED

Rendering - NOT IMPLEMENTED

blendMode() - 3D GRAPHIX NOT IN MY PLAN

clip() - 3D GRAPHIX NOT IN MY PLAN

createGraphics() - 3D GRAPHIX NOT IN MY PLAN

noClip() - 3D GRAPHIX NOT IN MY PLAN

PGraphics - 3D GRAPHIX NOT IN MY PLAN

Shaders- NOT IMPLEMENTED

loadShader() - 3D GRAPHIX NOT IN MY PLAN
PShader - 3D GRAPHIX NOT IN MY PLAN
resetShader() - 3D GRAPHIX NOT IN MY PLAN
shader() - 3D GRAPHIX NOT IN MY PLAN

Typography

PFont - NOT IMPLEMENTED

Loading & Displaying

createFont() - NOT IMPLEMENTED

loadFont() - NOT IMPLEMENTED

text() - implemented as library functions

textFont() - NOT IMPLEMENTED

Attributes

TextAlign() - **implemented**

textLeading() - NOT IMPLEMENTED

textMode() - dummy implementation

textSize() - dummy implementation

textWidth() - **implemented**

Metrics

textAscent() - NOT IMPLEMENTED
textDescent() - NOT IMPLEMENTED

Final remarks

The list of unimplemented but expected functions can be found in the file 'not_implemented.h'

Please send comments and implementation suggestions to my e-mail:
wborkowski@uw.edu.pl