

ArduinoWidgets library

Version 1.0

Pierre Molinaro, Jean-Luc Bechenec

October 26th, 2017

Contents

1	Versions	2
2	Introduction	3
3	An example	4
4	A quick overview of the AW_doc_example	11
4.1	How to use the code lines of the following sections of ArduinoWidgets	15
5	The fondation of ArduinoWidgets	17
5.1	Context	18
5.2	The Views	20
5.3	The action	24
5.4	The coordinate plane	25
5.5	Points and Pixels	26
5.6	Rectangle	28
5.7	Region	33
5.8	Colors	36
5.9	Fonts	38
6	The Views of ArduinoWidgets	40
6.1	Lines	40
6.2	Label and AutoLabel	42
6.3	PushButton	47
6.4	SegmentedControl	50
6.5	Slider	52
6.6	DynamicSlider	56
6.7	TabView	57
6.8	Switch	59
6.9	ArrowPushButton	61
6.10	KeyButton	63
6.11	Keyboard	66

7 Custom View Examples	67
7.1 The Target custom view	68
7.2 The Keyboard and List	69
8 The next steps of ArduinoWidgets	70
9 The calibration of the Touch screen	71

List of Figures

1 The AW_doc_example	4
2 The coordinate plane	25
3 Points and Pixels	26
4 A Rectangle	30
5 Two Rectangles	32
6 Region with clipping rectangle	35
7 Fonts	39
8 Points and Pixels	41
9 Lines and Pixels	42
10 Labels and AutoLabel	45
11 A PushButton	47
12 Aie !! missing figure !	50
13 Four Sliders to control the brightness and the color of a Rectangle	52
14 Aie !! missing figure !	56
15 Aie !! missing figure !	57
16 Aie !! missing figure !	59
17 Aie !! missing figure !	61
18 Aie !! missing figure !	63
19 The keyboard view	66
20 "Target" custom view	68
21 Keyboard and List	69
22 The List	69

1 Versions

Version	Date	Comment
0.1	October 15, 2017	Initial version
0.2	October 18, 2017	Description of an example of Arduino sketch
0.3	October 21, 2017	Description of the first half of the Views
0.4	October 26, 2017	Section 5 completed

2 Introduction

This document describes the `ArduinoWidgets` library, an Arduino library that allow Arduino programmers to perform highly complex graphic operations on a wide variety of touchscreens, such as building graphical interfaces and allowing interactivity. It handles a collection of widgets of user interfaces (UI) such as:

- Views and regions;
- Labels, text and lists;
- Points, Lines and Rectangles;
- PushButtons (rectangle and arrows);
- Tabs and Controls;
- Sliders, Switches;
- Keys and Keyboard;
- More to come later.

The `ArduinoWidgets` library is built above the `UTFT` library which take care of the physical interface between various Arduino hardware (ARM, AVR, Teensy, etc..) and a variety of LCD displays. In addition to `UTFT`, `ArduinoWidgets` include a physical interface with the Touch technology of the screen.

No other library is necessary to work with `ArduinoWidgets`.

In the next section, an example of usage of the `ArduinoWidgets` library is described.

3 An example

The example is fully contained in the following Arduino sketch "**AW_doc_example**":

This sketch is contained in the "**Examples**" folder of the `ArduinoWidgets` library. It can be easily found in the Arduino IDE : clic on the "Examples" item of the File menu and scroll to the "ArduinoWidgets" line. A submenu show some examples. Choose "**AW_doc_example**".

It displays a collection of graphic objects, some of them are "ready to use" in the library, and some others are custom objects which are created from the library. In addition, a label is displayed with the name each object :

- The line is a 5 pixel width black line.
- The `roundRect` is a green rectangle with round corners.
- The half circle is a red circle with has been clipped by an invisible rectangle to let visible only a half circle.
- The button include a numeric label inside. When clicking the button the number in the button is incremented and the built-in led is switched on or off
- The slider control the backlight of the screen (via a PWM pin of the micro-controller).

The last 2 objects are interactive objects : they allow actions from the user, which can control either output pins of the micro-controller (backlight, led) or specific properties of another object (a label value for instance).

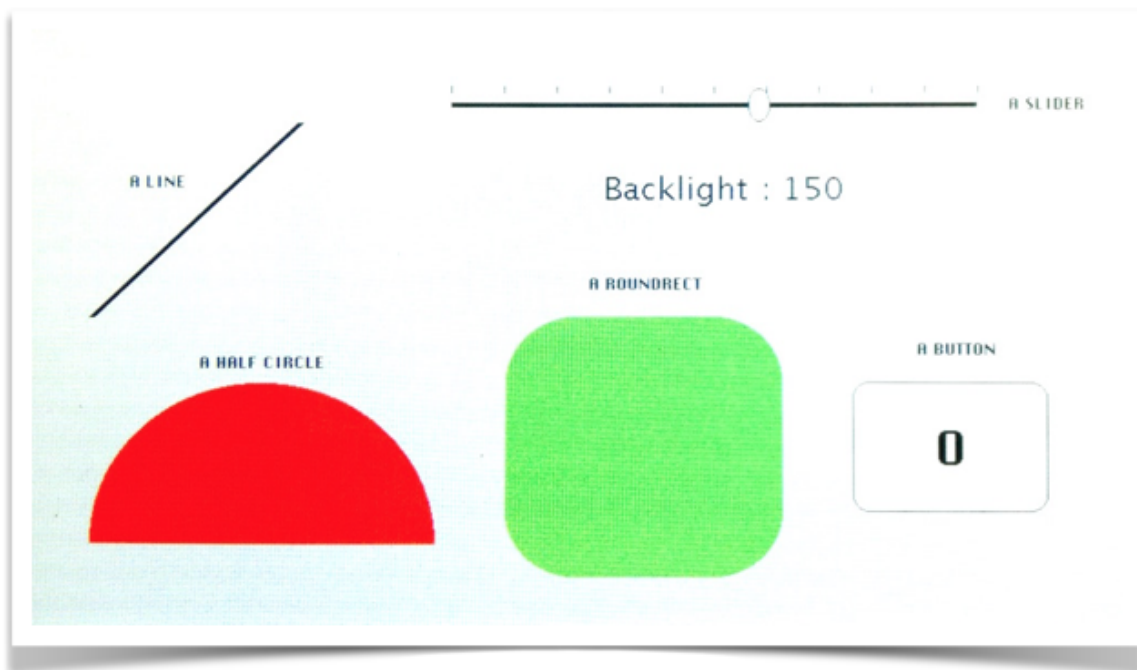


Figure 1: The AW_doc_example

Here is the complete listing :

```

1 //--- Project : ArduinoWidgets library
2 //--- Authors : Pierre Molinaro & Jean-Luc Bechennec
3 //--- Description : ArduinoWidgets library multiple views example
4
5 #include <ArduinoWidgets.h>
6 #include <UTFT.h>
7
8 //--- This part is hardware dependent :
9 //--- Teensy 3.6 + LCD 7" with Touch, SSD1963 controler,
    resolution 800x480
10 //--- LCD type (see /Applications/Arduino.app/Contents/Java/
    hardware/teensy/avr/libraries/UTFT/UTFT.h)
11
12 static const byte RS      = 23 ;
13 static const byte WR      = 22 ;
14 static const byte CS      = 15 ;
15 static const byte RESET = 33 ;
16
17 //--- Warning D33 of Teensy 3.1:
18 //--- https://forum.pjrc.com/threads/24823-Teensy-3-1-Tying-Pin-33-\(pta4\)-low-freezes-teensy
19
20 //--- Do not change 'myGLCD' name; it is declared as extern in
    AWContext.cpp
21
22 UTFT myGLCD (SSD1963_800ALT, RS, WR, CS, RESET) ;
23
24 static const byte T_CLK  = 11 ;
25 static const byte T_CS   = 12 ;
26 static const byte T_DIN  = 25 ;
27 static const byte T_DOUT = 24 ;
28 static const byte T_IRQ  = 28 ;
29
30 static const byte BACKLIGHT = 9 ;
31
32 //--- Do not change 'myTouch' name; it is declared as extern in
    AWContext.cpp
33 AWTouch myTouch (T_CLK, T_CS, T_DIN, T_DOUT, T_IRQ) ;
34
35 //--- end of hardware dependent part
36
37 ///////////////////////////////////////////////////////////////////
38 //--- Definitions of behaviours of existing Views in the library
39 ///////////////////////////////////////////////////////////////////
40

```

```

41 /////////////////////////////////////////////////// BUTTON ///////////////////////////////////
42
43 //--- button related global variable
44 int buttonValue = 0 ;
45
46 //--- button action
47 void bigButtonAction (AWView * inSender)
48 {
49     AWPushButton * sendingButton = (AWPushButton *) inSender ;
50     buttonValue ++ ;
51     sendingButton->setTitle (String(buttonValue)) ;
52     digitalWrite (LED_BUILTIN, !digitalRead(LED_BUILTIN));
53 }
54
55 /////////////////////////////////////////////////// SLIDER ///////////////////////////////////
56
57 //--- Slider global variables
58 AWSlider *backlightSlider;
59 AWLabel * label1;    // constant label
60 AWLabel * label2;    // variable label
61 //--- Slider action
62 void sliderAction (AWView * inSender)
63 {
64     AWSlider * sendingSlider = (AWSlider *) inSender ;
65     AWInt pos = sendingSlider->knobPosition ();
66     if (sendingSlider == backlightSlider) {
67         analogWrite (BACKLIGHT, pos);
68         label2->setTitle(pos);
69     }
70 }
71
72 ///////////////////////////////////////////////////
73 //--- Definitions of new View classes
74 ///////////////////////////////////////////////////
75
76 /////////////////////////////////////////////////// ROUND CORNERS RECTANGLE ///////////////////////////////////
77
78 class CustomView1 : public AWView
79 {
80     CustomView1 (const AWRect & inViewFrame);
81     virtual void drawInRegion (const AWRegion & inRegion) const;
82 };
83 CustomView1::CustomView1 (const AWRect & inViewFrame) :
84 AWView(inViewFrame,
85     AWCOLOR ()),          // let the corners opaque,
86     Color2 (AWColor::green ())
87 { }

```

```

88 void CustomView1::drawInRegion (const ARegion & inRegion) const
89 {
90     ARect viewFrame = absoluteFrame () ;
91     AContext::setColor (Color2) ;
92     viewFrame.fillRoundRectInRegion (AInt (50), inRegion) ; //
        radius of corners is 50
93 }
94 //--- Global variable ROUNDRECT
95 CustomView1 * roundRectView ;
96
97 /////////////// CLIPPING VIEW : A HALF CIRCLE ///////////////
98
99 class ClippingView : public AView
100 {
101     ClippingView (const ARect & inViewFrame);
102     virtual void drawInRegion (const ARegion & inRegion) const;
103 };
104 ClippingView::ClippingView (const ARect & inViewFrame) :
105 AView(inViewFrame,
106     AColor ())          // outside the drawing region is opaque
107 { }
108 void ClippingView::drawInRegion (const ARegion & inRegion) const
109 {
110     ARegion drawingRegion = inRegion ;
111     ARect viewFrame = absoluteFrame () ;
112     ARect clipRectangle = viewFrame ;
113     clipRectangle.size.width /= 1 ;
114     clipRectangle.size.height /= 2 ;    // the clip rectangle hide
        the low half of the circle
115     drawingRegion -= clipRectangle ;
116     AContext::setColor (AColor::red ());
117     viewFrame.fillOvalInRegion (drawingRegion) ;
118 }
119 //--- Global variable CLIPPING VIEW
120 ClippingView * crossView ;
121
122
123 /////////////// SETUP ///////////////
124
125 void setup() {
126     //--- This part is hardware dependent
127     //--- set up the backlight
128     analogWrite (BACKLIGHT, 150);
129     pinMode (LED_BUILTIN, OUTPUT);
130     digitalWrite (LED_BUILTIN, HIGH);
131
132     AContext::begin (kOrientationLandscape,
133                     800,          // Screen width

```

```

134         480,        // Screen height
135         true,        // true : X is flipped
136         false) ;    // false : Y is not flipped
137
138 //--- end of hardware dependent part
139
140 //--- create a button on screen
141 AWPushButton * bigButton = new AWPushButton(AWRect (600, 100,
142         140, 100), String (buttonValue), AWFFont (ChicagoDigit36)) ;
143 bigButton->setAction (bigButtonAction) ;
144 addView (new AWWLabel (AWPoint (625, 220), 100,
145         kAWAlignmentCenter, "A_BUTTON")) ;
146
147 //--- create 2 labels, one constant and one that can be changed
148 //      by the slider
149 label1 = new AWWLabel(AWPoint ( 300, 340), AWWInt (250),
150         AWWAlignment (kAWAlignmentRight), String ("Backlight_:_" ),
151         AWWFont (Lucida_Grande24)) ;
152 addView (label1) ;
153 label2 = new AWWLabel(AWPoint ( 550, 340), AWWInt (100),
154         AWWAlignment (kAWAlignmentLeft), String ("150"), AWWFont (
155         Lucida_Grande24)) ;
156 addView (label2) ;
157
158 //--- create the slider with its label
159 backlightSlider = new AWWSlider (AWPoint (300,400), 400,
160         kHorizontal, true) ;
161 backlightSlider->setMaxKnobPosition (255);
162 backlightSlider->setKnobPosition (150);
163 backlightSlider->setAction (sliderAction);
164 addView (new AWWLabel (AWPoint (690, 410), 100,
165         kAWAlignmentCenter, "A_SLIDER")) ;
166 addView (backlightSlider) ;
167
168 //--- create a black line of 5 pixel of thickness
169 AWWPoint lineOrigin ;
170 lineOrigin.x = 50 ;
171 lineOrigin.y = 250 ;
172 AWWPoint lineEnd;
173 lineEnd.x = 200 ;
174 lineEnd.y = 400 ;
175 for (int i=0 ; i < 5 ; i++) {
176     addView (new AWWLine (lineOrigin, lineEnd)) ;
177     lineOrigin.x++;
178     lineEnd.x++;
179 }
180 addView (new AWWLabel (AWPoint (50, 350), 100,

```



```

173         kAWAlignmentCenter, "A_LINE")) ;
174 //--- create the roundRect
175 roundRectView = new CustomView1(AWRect (350, 50, 200, 200)) ;
176 addView (roundRectView) ;
177 addView (new AWLabel (AWPoint (400, 270), 100,
178         kAWAlignmentCenter, "A_ROUNDRECT")) ;
179 //--- create the clipping view (half-circle)
180 crossView = new ClippingView(AWRect (50, -50, 250, 250)) ;
181 addView (crossView) ;
182 addView (new AWLabel (AWPoint (100, 210), 150,
183         kAWAlignmentCenter, "A_HALF_CIRCLE")) ;
184 }
185
186 //////////////////////////////////// LOOP ////////////////////////////////////
187
188 void loop() {
189     AWContext::handleTouchAndDisplay () ;
190 }

```

To run this program, it is recommended to open the sketch in the example folder of the `ArduinoWidgets` library, because a small file must be present in the same folder as the sketch. Its name must be "touch-calibration-values.h" and it must contain :

```

#include "touch-calibration-values.h"

const float xA = 310.0 ; // 571.0 ;
const float yA = 116.0 ; // 963.0 ;

const float xB = 3811.0 ; // 488.0 ;
const float yB = 113.0 ; // 3001.0 ;

const float xC = 3814.0 ; // 3526.0 ;
const float yC = 3936.0 ; // 3349.0 ;

const float xD = 362.0 ; // 3519.0 ;
const float yD = 3938.0 ; // 698.0 ;

```

This .cpp file contain the values of the calibration of your screen. If this file is not present in the sketch folder, a compilation error will occur

```

.../AWContext.cpp:617: undefined reference to `yA'
.../AWContext.cpp:642: _undefined_reference_to_ `xA'
.../AWContext.cpp:643: undefined reference to `yB'
.../AWContext.cpp:643: _undefined_reference_to_ `yC'
.../AWContext.cpp:643: undefined reference to `xB'

```

```
.../AWContext.cpp:643:_undefined_reference_to_`yD'  
.../AWContext.cpp:643: undefined reference to `xC'  
.../AWContext.cpp:634:_undefined_reference_to_`xD'
```

Go to the section "Calibration" to know how to calibrate your touch screen.

4 A quick overview of the AW_doc_example

This section is not an in depth description on how to use each object of the library but just a global introduction to how to build a program which use the `ArduinoWidgets` library. A detailed description of the library is given in the next sections of this document.

The program is divided in five parts :

- The first part is to include the necessary libraries.
- The second part is the necessary adaptation to your hardware.
- The third part is the declaration of new graphic objets, the specific behaviors of existing objects of the library and the declarations of constants and variables.
- The fourth part is the `setup()` function in which there also a part for your specific hardware
- The fifth part is the `loop()` function.

The first step is to import the two necessary libraries `ArduinoWidgets` and `UTFT`.

```
#include <ArduinoWidgets.h>
#include <UTFT.h>
```

Before using your specific Touch LCD display, please read the following documents which are located inside the `UTFT/documentation` folder :

- "UTFT_Requirements.pdf" for describing the pins which are used to connect your micro-controller to your LCD
- "UTFT_Supported_display_modules_&_controllers.pdf" for finding the right declaration of the controller which is used in your screen

Then adapt the hardware dependent part 1 in the listing to your specific hardware :

```
//--- This part is hardware dependent, such as here :
//--- Teensy 3.6 + LCD 7" with Touch,
//--- SSD1963 controler, resolution 800x480

static const byte RS      = 23 ;
static const byte WR      = 22 ;
static const byte CS      = 15 ;
static const byte RESET   = 33 ;

//--- Do not change 'myGLCD' name;
//--- it is declared as extern in AWContext.cpp

UTFT myGLCD (SSD1963_800ALT, RS, WR, CS, RESET) ;
```

```
static const byte T_CLK  = 11 ;
static const byte T_CS   = 12 ;
static const byte T_DIN  = 25 ;
static const byte T_DOUT = 24 ;
static const byte T_IRQ  = 28 ;

static const byte BACKLIGHT = 9 ;

//--- Do not change 'myTouch' name;
//--- it is declared as extern in AWContext.cpp

AWTouch myTouch (T_CLK, T_CS, T_DIN, T_DOUT, T_IRQ) ;

//--- end of hardware dependent part
```

At this stage, don't forget to add the file "touch-calibration-values.h" which must be present in the same folder as the sketch (see the end of section 3).

Then add the specific customizations and creations of the objects to display on the screen, with the desired interactivity :

- a value to display inside a button and an action when clicking in the button, which increment the button's value;
- some variables and an action to be attached to a slider which control the brightness of the backlight of the screen;
- a new `AWView` object to display a rectangle with round corners;
- a new `AWView` object to display a half circle which is the combination of a full circle with a clipping rectangle.

Then the `setup()` function initialize the backlight of the LCD to the value 150 (the maximum is 255). Then it initialize the LCD with the proper parameters (orientation, size, X and Y directions). Please note that the origine (0,0) of the screen is in the lower-left corner, exactly as in a Cartesian coordinate system.

The objects are then created :

- the button and its label;
- two labels, one of which is associated to the action of the slider;
- the slider;
- a black line;
- the `roundRect`;

- the half circle;

```
void setup() {
//--- This part is hardware dependent
//--- set up the backlight
    analogWrite (BACKLIGHT, 150);
    pinMode (LED_BUILTIN, OUTPUT);
    digitalWrite (LED_BUILTIN, HIGH);

    AWContext::begin (kOrientationLandscape,
                      800,          // Screen width
                      480,          // Screen height
                      true,         // true : X is flipped
                      false) ;     // false : Y is not flipped

//--- end of hardware dependent part

//--- create a button on screen
    AWPushButton * bigButton = new AWPushButton(AWRect (600, 100,
        140, 100), String (buttonValue), AWFFont (ChicagoDigit36)) ;
    bigButton->setAction (bigButtonAction) ;
    addView (new AWLabel (AWPoint (625, 220), 100,
        kAWAlignmentCenter, "A_BUTTON")) ;
    addView (bigButton) ;

//--- create 2 labels, one constant and one that can be changed
    by the slider
    label1 = new AWLabel(AWPoint ( 300, 340), AWWInt (250),
        AWWAlignment (kAWAlignmentRight), String ("Backlight:_:_"),
        AWWFont (Lucida_Grande24)) ;
    addView (label1) ;
    label2 = new AWLabel(AWPoint ( 550, 340), AWWInt (100),
        AWWAlignment (kAWAlignmentLeft), String ("150"), AWWFont (
        Lucida_Grande24)) ;
    addView (label2) ;

//--- create the slider with its label
    backlightSlider = new AWWSlider (AWPoint (300,400), 400,
        kHorizontal, true) ;
    backlightSlider->setMaxKnobPosition (255);
    backlightSlider->setKnobPosition (150);
    backlightSlider->setAction (sliderAction);
    addView (new AWLabel (AWPoint (690, 410), 100,
        kAWAlignmentCenter, "A_SLIDER")) ;
    addView (backlightSlider) ;

//--- create a black line of 5 pixel of thickness
    AWPoint lineOrigin ;
```

```
lineOrigin.x = 50 ;
lineOrigin.y = 250 ;
AWPoint lineEnd;
lineEnd.x = 200 ;
lineEnd.y = 400 ;
for (int i=0 ; i < 5 ; i++) {
    addView (new AWLine (lineOrigin, lineEnd)) ;
    lineOrigin.x++;
    lineEnd.x++;
}
addView (new AWLabel (AWPoint (50, 350), 100,
    kAWAlignmentCenter, "A_LINE")) ;

//--- create the roundRect
roundRectView = new CustomView1(AWRect (350, 50, 200, 200)) ;
addView (roundRectView) ;
addView (new AWLabel (AWPoint (400, 270), 100,
    kAWAlignmentCenter, "A_ROUNDRECT")) ;

//--- create the clipping view (half-circle)
crossView = new ClippingView(AWRect (50, -50, 250, 250)) ;
addView (crossView) ;
addView (new AWLabel (AWPoint (100, 210), 150,
    kAWAlignmentCenter, "A_HALF_CIRCLE")) ;

}
```

The `loop()` function of the sketch is very simple since it contain only one instruction which do everything: get events and call actions: `handleTouchAndDisplay()`

```
void loop() {
    AWContext::handleTouchAndDisplay () ;
}
```

4.1 How to use the code lines of the following sections of **Ardui**noWidgets

In the following sections of this document, there is three pieces of Arduino code that will not be repeated.

The first one is :

```
#include <ArduinoWidgets.h>
#include <UTFT.h>
//--- This part is hardware dependent, such as here :
//--- Teensy 3.6 + LCD 7" with Touch,
//--- SSD1963 controler, resolution 800x480

static const byte RS      = 23 ;
static const byte WR      = 22 ;
static const byte CS      = 15 ;
static const byte RESET = 33 ;

//--- Do not change 'myGLCD' name;
//--- it is declared as extern in AWContext.cpp

UTFT myGLCD (SSD1963_800ALT, RS, WR, CS, RESET) ;

static const byte T_CLK   = 11 ;
static const byte T_CS    = 12 ;
static const byte T_DIN   = 25 ;
static const byte T_DOUT  = 24 ;
static const byte T_IRQ   = 28 ;

static const byte BACKLIGHT = 9 ;

//--- Do not change 'myTouch' name;
//--- it is declared as extern in AWContext.cpp

AWTouch myTouch (T_CLK, T_CS, T_DIN, T_DOUT, T_IRQ) ;

//--- end of hardware dependent part

//--- add here you constants, classes, functions, variables, ...
```

The second one is the `setup()` :

```
void setup() {
//--- This part is hardware dependent
//--- set up the backlight
  analogWrite (BACKLIGHT, 150);
  pinMode (LED_BUILTIN, OUTPUT);
  digitalWrite (LED_BUILTIN, HIGH);
}
```

4.1 How to use the code lines of the following sections of QUICK OVERVIEW OF THE AW_DOC_EXAMPLE

```
AWContext::begin (kOrientationLandscape,
                  800,          // Screen width
                  480,          // Screen height
                  true,         // true : X is flipped
                  false) ;     // false : Y is not flipped

//--- end of hardware dependent part

//--- add here your lines of code for setup

} //--- end of setup
```

The third part is the `loop()` function of the sketch.

```
void loop() {
    AWContext::handleTouchAndDisplay ();
}
```

You can create an Arduino sketch by inserting first the 3 parts above, then the lines of code found in the various examples below.

If you want to test the examples which are described in this document, the best way is to use the "example" menu of the Arduino's IDE. But you can also assemble the lines of code of each subsection with the above parts to be adapted first to your specific hardware.

5 The fondation of ArduinoWidget s

The `ArduinoWidgets` library is objet oriented (C++). It is made of a collection of objets which can be expanded later in future versions.

The consequence is the naming and the syntax to use in your Arduino program :

- Objectcts are named "classes".
- Each class embed its own variables which are hidden to your sketch.
- Functions are named "methods".

To create an instance of a class `AWClass` :

```
AWClass * myObject; // myObject is a pointer in memory  
myObject = new AWClass(parameters of the constructor);
```

To call a method of `myObject` :

If `myObject` is a pointer :

```
myObject->method(parameters);
```

If `myObject` is a not a pointer or if it is simply a structure :

```
myObject.method(parameters);
```

5.1 Context

The *Context* is the entire Screen you are using. You must know about *Context* before seeing something on your screen and touch it.

To create a context, just call the `begin` function of class `AWContext`.

The class `AWContext` is defined by :

```
typedef enum {kOrientationLandscape, kOrientationPortrait}
    tOrientation ;

class AWContext {
    static void begin (const tOrientation inOrientation,
        const AWInt inScreenWidth,
        const AWInt inScreenHeight,
        const bool inHorizontalFlip,
        const bool inVerticalFlip) ;
```

In the example sketch, it is written :

```
AWContext::begin (kOrientationLandscape,
    800,          // Screen width
    480,          // Screen height
    true,         // true : X is flipped
    false) ;     // false : Y is not flipped
```

This describe the specific display which is in use : landscape orientation, dimensions : 800x600, horizontal axis is flipped, not the vertical axis.

There is a method in the `loop` for dealing with all events and actions of the touch screen and the draw and redraw on the screen according to the events. This is the unique instruction in the `loop` which work as a background task in the examples.

```
static void handleTouchAndDisplay (void) ;
```

`handleTouchAndDisplay` handle the touch events, especially the press detection (`touchDown`), press move (`touchMove`), press release (`touchUp`), and the drawing of all views (see View section) on the screen. Generally, a touch event generate an action which is sent to a specific object in a view.

The *Context* have also a variety of properties and methods, such as :

- A screen rectangle : you can get the size on the screen rectangle

```
//--- Screen rect
static AWRect screenRect (void) ;
```

- A color (background color) : you can set or get the color of the screen and its opacity

```
//--- Color
static void setColor (const AWColor & inColor) ;
```

```
static AWColor color (void) ;  
static bool colorIsOpaque (void) ;
```

- A calibration method with the drawing of a calibration rectangle and specific points to touch:

```
static void calibrateTouch (void) ;
```

5.2 The Views

Every pieces of drawing on the screen are "Views".

A view is a rectangular section of the screen. It is responsible for handling all drawing and user-initiated events within its frame.

`ArduinoWidgets` provides the `View` class as an abstract view implementation that subclasses use as the basis for implementing custom display and user interaction. They are the most pervasive type of object in the `ArduinoWidgets` library; nearly every object you see on the screen is a view. Views are in the front line of both drawing and event handling, and hence are one of the more important types of objects to understand.

In a very real sense, a view draws itself. It also provides a surface that can be responsive to input from a touch event.

In addition to drawing content and responding to user events, `View` instances act as containers for other views. By nesting views within other views, an application creates a hierarchy of views. This view hierarchy provides a clearly defined structure for how views draw relative to each other and pass messages from one view to another, up to the enclosing window, and on to the application for processing.

`ArduinoWidgets` provides several type of views for containing graphics, texts and controls, with color attributes, which can send messages to its own view or to other views. A view can be opaque or transparent. In this last case, if the view is not validated, the drawing of views behind it is allowed.

The root `View` is the screen.

Each `View` is placed in a parent `View`, the `superView`, in which it is a frame. This frame can be moved and resized in the `superView` and the view's content moves with it.

The view is specified when a view instance is created programmatically using :

```
//--- Constructor
AWView (const AWRect & inRelativeFrame,
        const AIColor & inBackColor) ;
```

When it is necessary to know the frame rectangle of a view, the `absoluteFrame` method can return this frame rectangle.

To translate a frame you can use the method `translateBy`. To change the frame size you can use `setSize`.

To specify or to get the bgcolor of a view, you can use the methods `BackColor` or `setBackColor`.

```
//----- Frame
inline AWRect absoluteFrame (void) const { return
    mAbsoluteFrame ; }

//----- Frame change
void translateBy (const AWRInt inDx, const AWRInt inDy) ;
void setSize (const AWRSize & inNewSize) ;
```

```
//----- Background color
inline AWCOLOR backColor (void) const { return mBackColor ; }
void setBackgroundColor (const AWCOLOR & inBackColor) ;
```

To add a new View on the screen, just call `addView` or `addCenteredView`:

```
void addView (class AWCOLOR * inView) ;
void addCenteredView (class AWCOLOR * inView) ;
```

What is a View Hierarchy?

In addition to being responsible for drawing and handling user events, a view instance can act as a container, enclosing other view instances. Those views are linked together creating a view hierarchy. Unlike a class hierarchy, which defines the lineage of a class, the view hierarchy defines the layout of views relative to other views.

It permits a complex view to be constructed out of other views. For example, a graphical keypad might be a container view with a separate subview for each key.

The `context` instance maintains a reference to a single top-level view instance. The `context` view acts as the root of the visible view hierarchy in a window. The view instances enclosed within a view are called `subViews`. The parent view that encloses a view is referred to as its `superView`. Each view has another view as its `superView` and may be the `superView` for any number of `subViews`. While a view instance can have multiple `subViews`, it can have only one `superView`. In order for a view and its subviews to be visible to the user, the view must be inserted into a view hierarchy.

A view is added to a parent view via the method `addSubView` or `addCenteredSubView`, in a hierarchical order :

```
//----- Managing view hierarchy
void addSubView (AWCOLOR * inView) ; // if inView is non
NULL, view is added in front of other subviews
void addCenteredSubView (AWCOLOR * inView) ; // if inView is non
NULL, view is added in front of other subviews
```

To locate the `superView` use the method `superView`. You can also remove a `subView` from a `superView` with `removeFromSuperView`.

```
//----- Managing view hierarchy
void addSubView (AWCOLOR * inView) ; // if inView is non
NULL, view is added in front of other subviews
void addCenteredSubView (AWCOLOR * inView) ; // if inView is non
NULL, view is added in front of other subviews
void removeFromSuperView (void) ; // Does nothing if has no
super view
inline AWCOLOR * superView (void) { return mSuperView ; } //
Returns NULL if has no super view
```

View Tags

The View class defines methods that allow you to tag individual view objects with integer tags and to search the view hierarchy based on those tags. The receiver's subviews are searched depth-first, starting at the first subview returned by the receiver's subviews method.

The View method `tag` always returns -1. Subclasses can override this method to return a different value. It is common for a subclass to implement a `setTag` method that stores the tag value in an instance variable, allowing the tag to be set on an individual view basis.

```
//----- Tag
inline void setTag (const int inTag) { mTag = inTag ; }
inline int tag (void) const { return mTag ; }
```

User interactivity : sending and receiving Actions

A view or subview can be sensitive to touch actions of the user. In order to be responsive to a touch event, each view must redefine `touchDown`, `touchMove` and `touchUp`. The method `sendAction` call the function which is attached to the view by using `setAction`:

```
//----- Action
void sendAction (void) ;
inline AWACTION action (void) const { return mAction ; }
inline void setAction (AWACTION inAction) { mAction = inAction ; }

//----- Touch
virtual void touchDown (const AWPPoint & inPoint) ;
virtual void touchMove (const AWPPoint & inPoint) ;
virtual void touchUp (const AWPPoint & inPoint) ;
```

Drawing and display views

These methods force an immediate display of the screen, although the latter method iterates only through the list of invalidated views. The display method causes each view in the screen's view hierarchy to redraw itself, starting with the context view. This method is invoked on a one-shot window (that is, a window whose backing store is released when the window is offscreen) before the window is shown onscreen.

These methods mark views or regions of views for redrawing :

```
//----- Display view
void setNeedsDisplay (void) ;
void setNeedsDisplayInRect (const AWPRect & inRect) ;
```

The method `drawInRegion` must be redefined in each view : It is responsible of the drawing of the view.

```
//--- Draw method (to be overridden)
```

```
virtual void drawInRegion (const ARegion & inDrawRegion) const
;
```

A view can be opaque (visible) or transparent (invalid or invisible). An important aspect of the drawing of views is view opacity. A view does not have to draw every bit of its surface, but if it does it should declare itself to be opaque (by implementing `isOpaque` to return YES).

```
//--- Tell the view is opaque
virtual bool isOpaque (void) const ;

//----- Visibility
inline bool isVisible (void) const { return mIsVisible ; }
void setVisibility (const bool inIsVisible) ;

//----- Handle "on screen" state
inline bool isOnScreen (void) const { return mIsOnScreen ; }
```

AWView is a class that defines the basic drawing and event-handling. AWView itself does not draw content or respond to user events, so you typically don't interact with a direct instance of AWView. Instead you use an instance of a custom AWView subclass. A custom view class inherits from AWView and overrides many of its methods, which are invoked automatically by the `ArduinoWidgets` library.

There is a collection of ready to use subclass Views in the `ArduinoWidgets` library, which are :

- AWLine : display a line
- AWLabel : display of text on a line.
- AWPushButton : display a button
- AWRectView : display a rectangle with or without round corners
- AWSegmentedControl : display a radio button in a tab form
- AWSlider : display a linear potentiometer with one cursor
- AWDynamicSlider : display a linear potentiometer with one cursor
- AWTabview : display tabs which are usefull for switching from one View to another.
- AWSwitch : display a check box.
- AWAArrowPushButton : display a button with an arrow shape.
- AWKeyButton : display a key of keyboard
- AWKeyboardBackView : display an entire keyboard.

You will be able to create and add your custom Views !

5.3 The action

The action is the interactive part of the `ArduinoWidgets` library, which rely on the touch capabilities of the screen. An action is the message a control sends to the target or, from the perspective of the target, the method it implements to respond to the action.

The standard way to pass information between objects is message passing—one object invokes the method of another object. However, message passing requires that the object sending the message know who the receiver is and what messages it responds to. This is the reason why the action routines are located in the declaration part of the Arduino sketch, before the `setup`.

A control is a view object which implement touch capabilities. For exemple, `myControl` (a `PushButton` or a `Key` or a `Tab`, for example) is a control. In the `setup`, inside the initialization code of `myControl`, we add the following line :

```
myControl->setAction (myControlAction) ;
```

`myControlAction` is a routine which is declared before the `setup`. We suppose here that this control is a `PushButton` :

```
//--- myControl action
void myControlAction (AWView * inSender)
{
    AWPushButton * sendingButton = (AWPushButton *) inSender ;
    // Your Action code here
}
```

Your action code can be the display of a label's title, the control of an Arduino's output pin (a led, a PWM signal, the backlight of the screen, etc.).

You will find several example of how an action is implemented in the description of the `ArduinoWidgets` views.

5.4 The coordinate plane

A view is responsible for the drawing and event handling in a rectangular area of a window. In order to specify that rectangle of responsibility, you define its location as an origin point and size using a coordinate system.

This section describes the coordinate system used by views, how a view's location and size is specified, and how the size of a view interacts with its content.

All information about location is given to `ArduinoWidgets` in terms of coordinates on a plane. The coordinate plane is a two-dimensional grid, as illustrated in Figure 2.

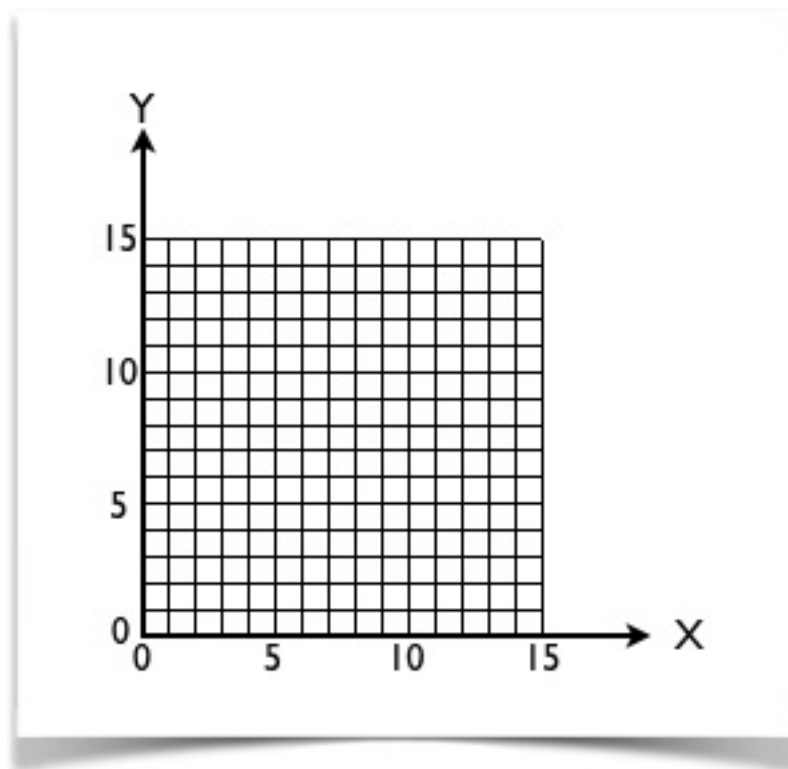


Figure 2: The coordinate plane

All grid coordinates are `AWInt` (in the range -32767 to 32767).

```
typedef int16_t AWInt ;
```

The origine of all coordinates is (0,0) and is located at the bottom left corner of the screen. Horizontal coordinates increase as you move from left to right, and vertical coordinates increase as you move from bottom to top.

Each view use the same coordinate system as its mother view.

5.5 Points and Pixels

Each point is at the intersection of a horizontal grid line and a vertical grid line. The coordinate origin (0,0) is in the lower-left corner of the screen.

You can store the coordinates of a point into an object `AWPoint` which contain 2 variables of type `AWInt`:

```
// AWPoint :  
AWInt x ;  
AWInt y ;
```

Figure 3 shows the relationship between points, grid lines, and pixels, the physical dots on the screen. A pixel is centered around a point. In other words, a point is at the center of a pixel.

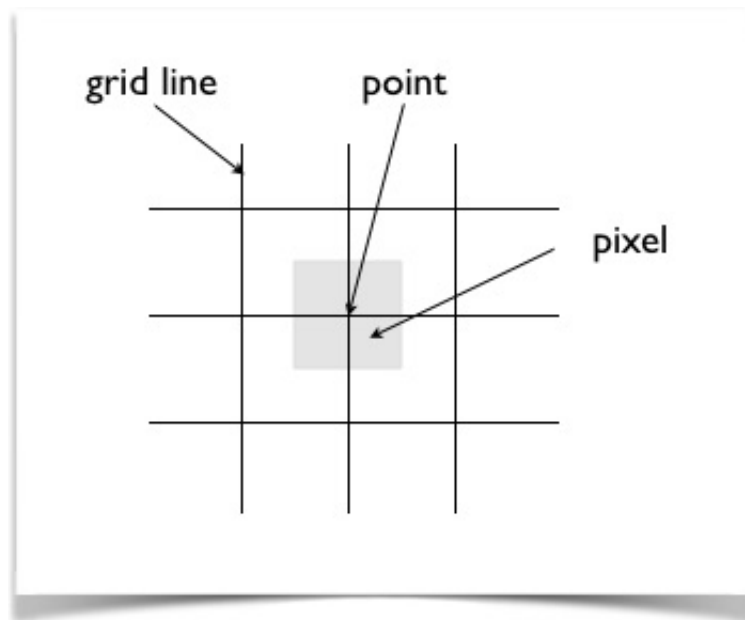


Figure 3: Points and Pixels

To create a `AWPoint` at coordinates X and Y, you call :

```
AWPoint myPoint ;  
myPoint.x = X;  
myPoint.y = Y;
```

The Point is not a View. The above `myPoint` is not drawn at this stage.

To compare two Points there is two special operators :

```
//--- Equatable  
inline bool operator == (const AWPoint & inP) const { return (x  
    == inP.x) && (y == inP.y) ; }
```

```
inline bool operator != (const AWPoint & inP) const { return
    !(*this == inP) ; }
```

The Point class has the following methods :

```
//--- Translation
void translateBy (const AWInt inDx, const AWInt inDy) { x +=
    inDx ; y += inDy ; }
void translateBy (AWPoint & inTranslation) { x += inTranslation
    .x ; y += inTranslation.y ; }

//--- Stroke line
void strokeLineInRegion (const AWPoint & inPoint, const
    AWRegion & inDrawRegion) const ;

static void strokeLineInRegion (const AWInt inP1X,
                                const AWInt inP1Y,
                                const AWInt inP2X,
                                const AWInt inP2Y,
                                const AWRegion &
                                    inDrawRegion) ;

//--- Draw Point
void drawInRegion (const AWRegion & inDrawRegion) const ;
static void drawPointInRegion (const AWInt inX, const AWInt inY
    , const AWRegion & inDrawRegion) ;
```

And a special drawing of circle (?? how to use it ??)

```
//--- Frame circle
void frameCircleInRegion (const AWInt inRadius,
    const AWRegion & inDrawRegion) const ;

static void frameCircleInRegion (const AWInt inCenterX,
    const AWInt inCenterY,
    const AWInt inRadius,
    const AWRegion & inDrawRegion) ;

//--- Fill circle
void fillCircleInRegion (const AWInt inRadius,
    const AWRegion & inDrawRegion) const ;

static void fillCircleInRegion (const AWInt inCenterX,
    const AWInt inCenterY,
    const AWInt inRadius,
    const AWRegion & inDrawRegion) ;
```

5.6 Rectangle

A rectangle is not a View. The class `AWRect` is :

```
class AWRect {
//--- Default constructor (invisible rectangle)
    inline AWRect (void) : origin (), size () {}

//--- Constructors
    inline AWRect (const AWPPoint inOrigin, const AWPSize inSize) :
        origin (inOrigin), size (inSize) {}
    inline AWRect (const AWPPoint inOrigin) : origin (inOrigin),
        size (AWPSize (1, 1)) {}
    inline AWRect (const AWPInt inX, const AWPInt inY, const AWPInt
        inWidth, const AWPInt inHeight) :
        origin (AWPPoint (inX, inY)),
        size (AWPSize (inWidth, inHeight)) {
    }
    AWRect (const AWPPoint & inP1, const AWPPoint & inP2) ;
    static AWRect horizontalLine (const AWPInt inX, const AWPInt inY,
        const AWPInt inWidth) ;
    static AWRect verticalLine (const AWPInt inX, const AWPInt inY,
        const AWPInt inHeight) ;

//--- Accessors
    inline bool isEmpty (void) const { return (size.width <= 0) ||
        (size.height <= 0) ; }
    bool containsPoint (const AWPPoint & inPoint) const ;
    AWPPoint topRight (void) const ;
    AWPPoint bottomRight (void) const ;
    AWPPoint topLeft (void) const ;
    inline AWPPoint bottomLeft (void) const { return origin ; }
    inline AWPInt minX (void) const { return origin.x ; }
    inline AWPInt maxX (void) const { return origin.x + size.width -
        1 ; }
    inline AWPInt minY (void) const { return origin.y ; }
    inline AWPInt maxY (void) const { return origin.y + size.height
        - 1 ; }

//--- Intersection
    AWRect operator & (const AWRect & inOtherRect) const ;
    bool intersects (const AWRect & inOtherRect) const ;

//--- Inclusion
    bool includesRect (const AWRect & inOtherRect) const ;

//--- Union (Returns the smallest rectangle that completely
    encloses both receiver rect and inOtherRect)
```

```

    AWRect operator + (const AWRect & inOtherRect) const ;
    void operator += (const AWRect inOtherRect) ;

//--- Difference (Returns 4 rects, possibly empty)
    void differenceFrom (const AWRect & inRect,
                        AWRect & outR1,
                        AWRect & outR2,
                        AWRect & outR3,
                        AWRect & outR4) const ;

//--- Transforming rectangle
    void inset (const AWInt inDx, const AWInt inDy) ;
    void translateBy (const AWInt inDx, const AWInt inDy) ;

//--- Equatable
    bool operator == (const AWRect & inRect) const { return (origin
        == inRect.origin) && (size == inRect.size) ; }
    inline bool operator != (const AWRect & inRect) const { return
        ! (*this == inRect) ; }

//--- Drawing
    void fillRectInRegion (const AWRRegion & inDrawRegion) const ;
    void frameRectInRegion (const AWRRegion & inDrawRegion) const ;
    void fillRoundRectInRegion (const AWInt inRadius, const
        AWRRegion & inDrawRegion) const ;
    void frameRoundRectInRegion (const AWInt inRadius, const
        AWRRegion & inDrawRegion) const ;
    void fillOvalInRegion (const AWRRegion & inDrawRegion) const ;
    void frameOvalInRegion (const AWRRegion & inDrawRegion) const ;

//--- Properties
    AWPPoint origin ;
    AWPSize size ;
} ;

```

A rectangle is the fondation of each AView.

A Rectangle is defined by an origine AWPPoint and an AWPsize. The origine Point is the bottom-left corner :

```

//--- Properties
    AWPPoint origin ;
    AWPSize size ;

// AWPsize :
    AWInt width ;
    AWInt height ;

```

```
// AWRect :
myRect = new AWRect (AWPoint, ASize);
myRect = new AWRect (X, Y, width, height);
```

The coordinates of the bottom left corner are X = left, Y = bottom.

For example, you can create a new rectangle myRect of left-bottom corner X=350, Y=50, width W=200 and height H=200 like this :

```
myRect = new AWRect (350, 50, 200, 200);
```

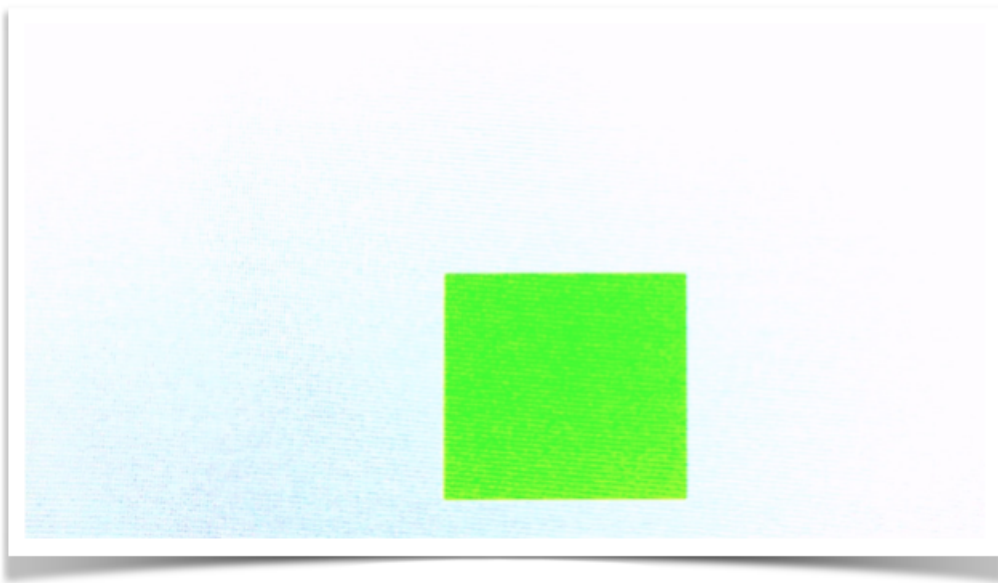


Figure 4: A Rectangle

This rect have a bottom left corner origin of coordinates (350, 50).

The width (200) and height (200) are the number in pixels in the horizontal and vertical directions.

Consequently, the coordinate of the top right corner of this rectangle is (14, 6) :

- left = X = 350
- bottom = Y = 50
- right = X + W - 1 = 350 + 200 - 1 = 549
- top = Y + H - 1 = 50 + 200 - 1 = 249

You can create rectangle from various combinations of AWInt, AWPoint and ASize, according to the 3 constructors.

You will see farther all the operations and transformations that you can do on AWRect such as :

- to be visible or invisible or empty (if W <= 0 or H <= 0)

- a pixel is a rectangle with $W = H = 1$
- a horizontal line is a rectangle with $W = 1$
- a vertical line is a rectangle with $H = 1$
- a 800x600 screen is a rectangle (0, 0, 800, 600)
- find corners points
- make intersection, inclusion or union of rectangles
- find differences (the opposite of intersection) of rectangles
- inset en translate rectangles
- draw and fill rectangles, round-rectangles, and ovals

The following example show how to display two rectangles : one without round corners and one with round corners :

First declare the new classes `RectangleView` and `RoundRectangleView`, and create an instance of each class :

```

//////////////////// RECTANGLE //////////////////////
class RectangleView : public AView
{
    RectangleView (const ARect & inViewFrame);
    virtual void drawInRegion (const ARegion & inRegion) const;
};
RectangleView::RectangleView (const ARect & inViewFrame) : //
    constructor
    AView(inViewFrame,
        AColor ()),
        RectColor (AColor::orange ())
    { }
void RectangleView::drawInRegion (const ARegion & inRegion)
    const
{
    ARect viewFrame = absoluteFrame () ;
    AContext::setColor (RectColor) ;
    viewFrame.fillRectInRegion (inRegion) ;
}
//--- Global variable RectView
RectangleView * RectView ;

//////////////////// ROUND CORNERS RECTANGLE //////////////////////
class RoundRectangleView : public AView
{
    RoundRectangleView (const ARect & inViewFrame);
    virtual void drawInRegion (const ARegion & inRegion) const;
};

```

```
};
RoundRectangleView::RoundRectangleView (const AWRect &
    inViewFrame) : // constructor
AWView(inViewFrame,
    AColor ()),
    RectColor (AColor::green ())
{ }
void RoundRectangleView::drawInRegion (const ARegion & inRegion)
    const
{
    ARect viewFrame = absoluteFrame () ;
    AContext::setColor (RectColor) ;
    viewFrame.fillRoundRectInRegion (AInt (50), inRegion) ;
}
//--- Global variable RoundRectView
RoundRectangleView * RoundRectView ;
```

Then draw the rectangles in the setup() :

```
//--- draw the Rectangle
RectView = new RectangleView (AWRect (350, 50, 200, 300)) ;
addView (RectView) ;

//--- draw the RoundRectangle
RoundRectView = new RoundRectangleView (AWRect (80, 200, 250,
    200)) ;
addView (RoundRectView) ;
```



Figure 5: Two Rectangles

5.7 Region

A region is not a View. The class `AWRegion` is :

```
class AWRegion {
//--- Default constructor (build an empty region)
    AWRegion (void) ;

//--- Region from a rectangle (empty if rectangle is empty)
    AWRegion (const AWRect & inRect) ;

//--- Destructor
    ~ AWRegion (void) ;

//--- release region, becomes empty
    void release (void) ;

//--- Accessors
    inline bool isEmpty (void) const { return NULL == mPtr ; }
    AWInt rectCount (void) const ;
    AWRect rectAtIndex (const AWInt inIndex) const ;

//--- Difference from a rectangle
    void operator -= (const AWRect & inRect) ;

//--- Adding a rectangle
    void operator += (const AWRect & inRect) ;

//--- Intersection with a rectangle
    AWRegion operator & (const AWRect & inRect) const ;
    bool intersects (const AWRect & inOtherRect) const ;

//--- Intersection of regions
    AWRegion operator & (const AWRegion & inRect) const ;

//--- Enclosing rectangle
    AWRect enclosingRect (void) const ;

//--- Testing a point
    bool containsPoint (const AWPoint & inPoint) const ;
    bool containsPoint (const AWInt inX, const AWInt inY) const ;

//--- Handle copy
    AWRegion (const AWRegion & inRegion) ;
    AWRegion & operator = (const AWRegion & inRegion) ;
} ;
```

The minimum Region is a Rectangle or can be empty.

A Region is made of a combination of one or more separate and non-empty rectangles.

The role of a Region is to manage more or less complex geometric entities which are not limited to a Rectangle.

A Region is not a View and do not allow any drawing by itself, but it allow several operations.

```
//--- Region from a rectangle (empty if rectangle is empty)
AWRegion (const AWRect & inRect) ;
AWRegion(); // build an empty region
```

The purpose of regions is to limit drawing within the region. For example, if you want to draw a half circle on the screen, you can set the region to half the square that would enclose the whole circle, and then draw the whole circle. Only the half within the region will actually be drawn.

First declare the new classes ClippingView :

```
////////// CLIPPING VIEW : A HALF CIRCLE //////////
class ClippingView : public AView
{
    ClippingView (const AWRect & inViewFrame);
    virtual void drawInRegion (const AWRegion & inRegion) const;
};
ClippingView::ClippingView (const AWRect & inViewFrame) :
    AView(inViewFrame,
        AColor ())          // outside the drawing region is opaque
    { }
void ClippingView::drawInRegion (const AWRegion & inRegion) const
{
    AWRegion drawingRegion = inRegion ;
    AWRect viewFrame = absoluteFrame () ;
    AWRect clipRectangle = viewFrame ;
    clipRectangle.size.width /= 1 ;
    clipRectangle.size.height /= 2 ;    // the clip rectangle hide
        the low half of the circle
    drawingRegion -= clipRectangle ;
    AContext::setColor (AColor::red ());
    viewFrame.fillOvalInRegion (drawingRegion) ;
}
//--- Global variable CLIPPING VIEW
ClippingView * crossView ;
```

Then draw the View in the setup() :

```
//--- create the clipping view (half-circle
crossView = new ClippingView(AWRect (300, 100, 250, 250)) ;
addView (crossView) ;
```

A Region is defined by a rectangle which extends from the bottom left corner (300, 100) with size (250,

250). The circle is centered in this rectangle.

Then a clipRectangle is defined as a the bottom half of the Region's rectangle.

Then this bottom half is removed from the region.

Then the circle is drawn but only the top half is visible !

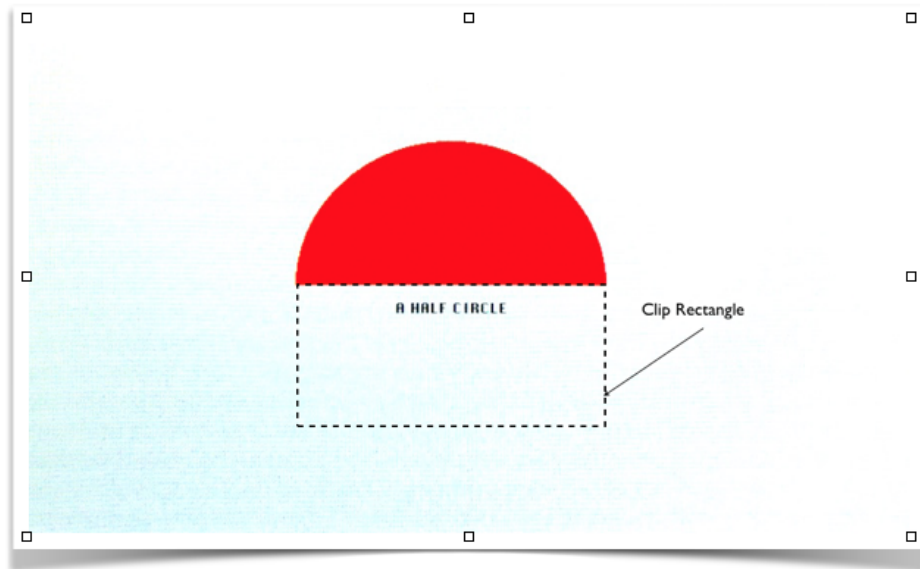


Figure 6: Region with clipping rectangle

•

There is various operations on regions, which are defined in the methods of the class `AWRegion`, for example :

- Release a region, becomes empty.
- Do operations such as addition or subtraction of a rectangle to/from the region.
- Intersection with a rectangle.
- Intersection of regions.
- Enclosing rectangle.
- Testing if a point is inside a region.
- Handle a copy.

5.8 Colors

A color is defined by 3 bytes (uint_8) and a boolean. The 3 bytes define respectively the red, green and blue component and the boolean "IsOpaque" define if the color is opaque or transparent.

A set of 17 colors is predefined to simplify your programming.

You can set colors, test colors or get colors.

A color is not a View. The class AWCOLOR is :

```
class AWCOLOR {
//--- Default constructor (clear color)
    inline AWCOLOR (void) : mRed (0), mGreen (0), mBlue (0),
        mIsOpaque (false) {}

//--- Constructor (custom color)
    inline AWCOLOR (const uint8_t inRed,
                    const uint8_t inGreen,
                    const uint8_t inBlue) :

        mRed (inRed),
        mGreen (inGreen),
        mBlue (inBlue),
        mIsOpaque (true) {
    }

//--- Colors
    inline static AWCOLOR black (void) {
        return AWCOLOR (0, 0, 0) ;
    }
    inline static AWCOLOR gray (void) {
        return AWCOLOR (128, 128, 128) ;
    }
    inline static AWCOLOR darkGray (void) {
        return AWCOLOR (64, 64, 64) ;
    }
    inline static AWCOLOR lightGray (void) {
        return AWCOLOR (192, 192, 192) ;
    }
    inline static AWCOLOR veryLightGray (void) {
        return AWCOLOR (224, 224, 224) ;
    }
    inline static AWCOLOR red (void) {
        return AWCOLOR (255, 0, 0) ;
    }
    inline static AWCOLOR green (void) {
        return AWCOLOR (0, 255, 0) ;
    }
    inline static AWCOLOR blue (void) {
        return AWCOLOR (0, 0, 255) ;
    }
}
```

```
inline static AWColor white (void) {
    return AWColor (255, 255, 255) ;
}
inline static AWColor yellow (void) {
    return AWColor (255, 255, 0) ;
}
inline static AWColor orange (void) {
    return AWColor (255, 127, 0) ;
}
inline static AWColor brown (void) {
    return AWColor (153, 102, 51) ;
}
inline static AWColor cyan (void) {
    return AWColor (0, 255, 255) ;
}
inline static AWColor magenta (void) {
    return AWColor (255, 0, 255) ;
}
inline static AWColor purple (void) {
    return AWColor (127, 0, 127) ;
}
inline static AWColor deepSkyBlue (void) {
    return AWColor (0, 0xBF, 255) ;
}
inline static AWColor lightSkyBlue (void) {
    return AWColor (0x87, 0xCE, 0xFA) ;
}

//--- Equatable
bool operator == (const AWColor & inOtherColor) const ;
inline bool operator != (const AWColor & inOtherColor) const {
    return !(*this == inOtherColor) ;
}

//--- Accessors
inline uint8_t redComponent (void) const { return mRed ; }
inline uint8_t greenComponent (void) const { return mGreen ; }
inline uint8_t blueComponent (void) const { return mBlue ; }
inline bool isOpaque (void) const { return mIsOpaque ; }
} ;
```

5.9 Fonts

The class `AWFont` contain font description and fonctions.

A font is not a View. The class `AWFont` is :

```
class AWFont {
//--- Default constructor: empty font
    AWFont (void) : mFont () {}

//--- Constructor from a font description
    AWFont (const AWFontInternalDefinition & definition) ;

    void drawStringInRegion (const AWInt inX,
                             const AWInt inY,
                             const char * inCString,
                             const AWRegion & inDrawRegion
                             ) const ;

    void drawStringInRegion (const AWInt inX,
                             const AWInt inY,
                             const String & inString,
                             const AWRegion & inDrawRegion
                             ) const ;

    AWInt ascent (void) const ;
    AWInt descent (void) const ;
    AWInt lineHeight (void) const ;

    AWInt advancement (const uint32_t inCodePoint) const ;

    AWRect stringRect (const AWInt inX,
                       const AWInt inY,
                       const char * inCString) const ;

    AWInt stringLength (const char * inCString) const ;
    AWInt stringLength (const String & inString) const ;

//--- Equatable
    inline bool operator == (const AWFont & inFont) const { return
        mFont == inFont.mFont ; }
    inline bool operator != (const AWFont & inFont) const { return
        !(*this == inFont) ; }
} ;
```

The definition of type `AWFontInternalDefinition` refer to one of these fonts which are already implemented in the `ArduinoWidget` library :

```
AWFont-ChicagoDigit-36.h
extern const AWFontInternalDefinition ChicagoDigit36 ;

AWFont-ChicagoFLF12.h
extern const AWFontInternalDefinition ChicagoFLF12 ;

AWFont-ChicagoFLF24.h
extern const AWFontInternalDefinition ChicagoFLF24 ;

AWFont-Geneva10.h
extern const AWFontInternalDefinition Geneva10 ;

AWFont-Geneva12.h
extern const AWFontInternalDefinition Geneva12 ;

AWFont-Geneva9.h
extern const AWFontInternalDefinition Geneva9 ;

AWFont-Lucida_Grande18.h
extern const AWFontInternalDefinition Lucida_Grande18 ;

AWFont-Lucida_Grande24.h
extern const AWFontInternalDefinition Lucida_Grande24 ;
```

These are examples of all fonts :

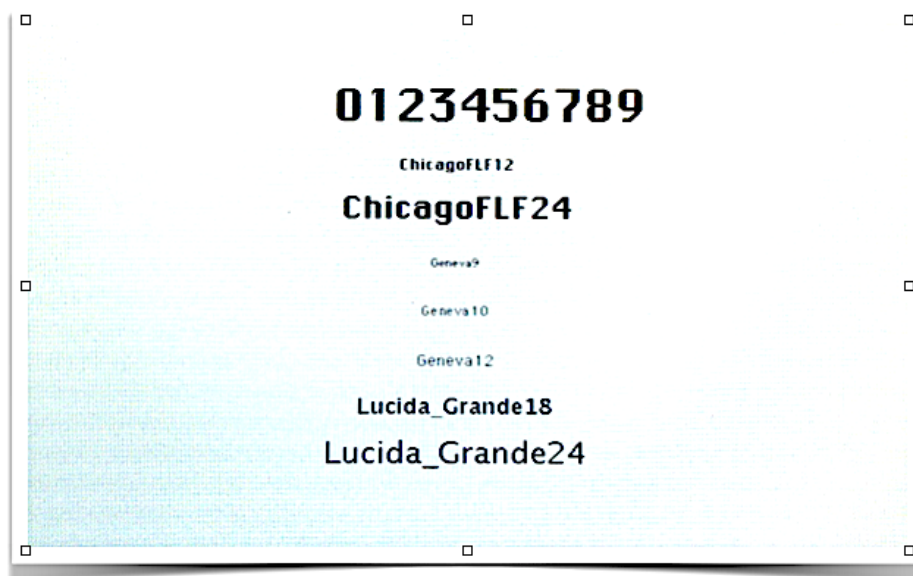


Figure 7: Fonts

6 The Views of ArduinoWidgets

This section describe the View subclasses which are already present in the `ArduinoWidgets` library, that you can use very simply.

For each View, the class declaration, which is in the `.h` file of the library, is reproduced without the private properties and methode.

As explained before, the `drawInRegion` method is "virtual" and must be defined in the instance of the view in your program.

6.1 Lines

The class `AWLine` is :

```
class AWLine : public AWView {
//--- Constructor
    AWLine (const AWPoint & inRelativePoint1,
            const AWPoint & inRelativePoint2) ;

//--- Draw
    virtual void drawInRegion (const AWRegion & inDrawRegion) const
        ;

//--- Tell the view is opaque
    virtual bool isOpaque (void) const { return false ; }
};
```

A line is defined by 2 `AWPoints`.

To create and draw a new line "myLine" from Point1 (50,50) and Point2 (400,400):

```
//--- Constructor
    AWLine (const AWPoint & inRelativePoint1,
            const AWPoint & inRelativePoint2) ;
```

This example display a line from 100,100 to 300,300

```
AWLine * myLine;
myLine = new AWLine (AWPoint(100,100), AWPoint(300,300));
addView (myLine) ;
```

Note that, in this case, you cannot use the origin and destination points for further operations because they are only arguments of the constructor of `myLine`.

This example display a line from `AWPoint lineOrigin(50,50)` to `AWPoint lineEnd(400,400)`

```
//--- create a black line
    AWPoint lineOrigin ;
    lineOrigin.x = 50 ;
```

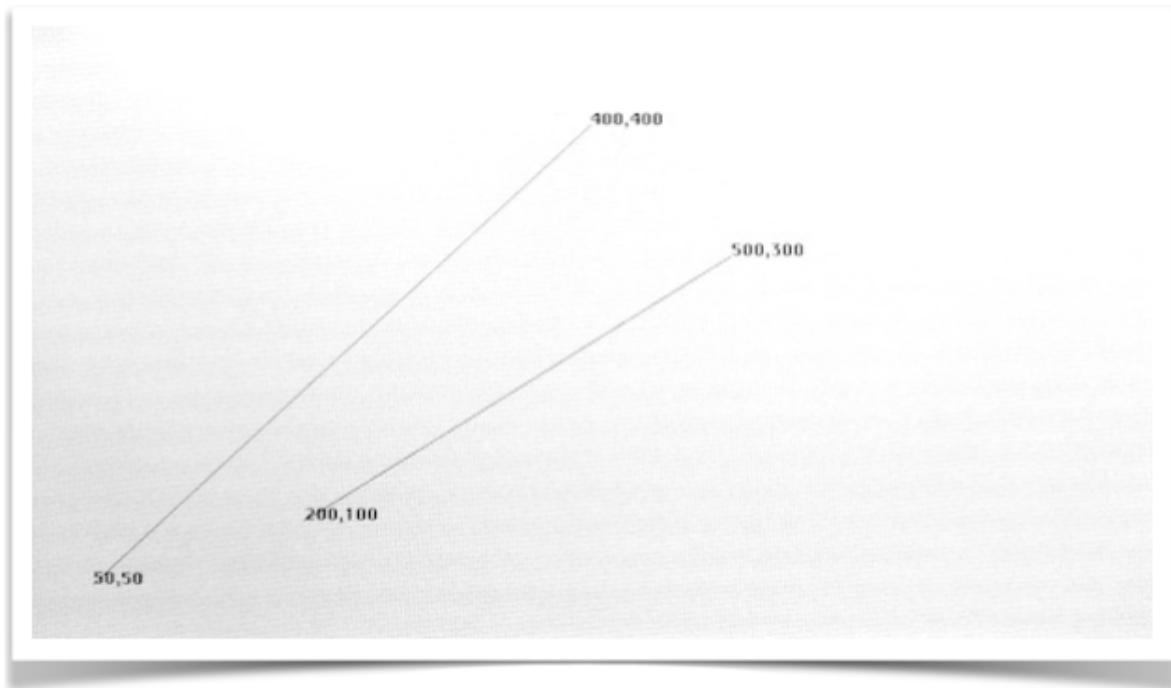



Figure 8: Points and Pixels

```
lineOrigin.y = 50 ;  
AWPoint lineEnd;  
lineEnd.x = 400 ;  
lineEnd.y = 400 ;  
AWLine * myLine1;  
myLine1 = new AWLine (lineOrigin, lineEnd);  
addView (myLine1) ;
```

Note that in this case you can use the lineOrigin and lineEnd points for further operations because they are declared outside the constructor of myLine1.

Note also that AWLine arguments are references (pointers) to AWPoints.

Note also that pixels are centered on points and, consequently, lines are centered on a grid line.

Figure 5 explain why the dimensions of a line is one pixel more than the width or height used in the drawing method.

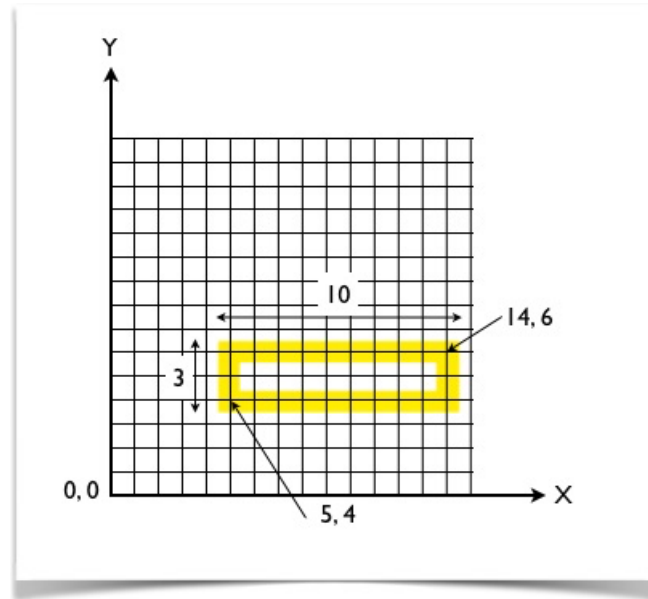


Figure 9: Lines and Pixels

6.2 Label and AutoLabel

A Label is a string of text, "Title", that can be displayed anywhere on the screen (Context) or on a View or subView, or Region

You can choose a Point and a width to define where the Label is displayed (if the size of the Label is greater than the size, the Label will be clipped). You can choose the alignment of the text with this enum :

```
typedef enum {
    kAWAlignmentLeft,
    kAWAlignmentCenter,
    kAWAlignmentRight
} AWAlignment ;
```

You can choose the style : the Font and size, the Color

The Font and size is chosen from this list :

- ChicagoDigit36
- ChicagoFLF12
- ChicagoFLF24
- Geneva10
- Geneva12
- Geneva9
- LucidaGrande18

- LucidaGrande24

You can choose the color from this list :

- black
- gray
- darkGray
- lightGray
- veryLightGray
- red
- green
- blue
- white
- yellow
- orange
- brown
- cyan
- magenta
- purple
- deepSkyBlue
- lightSkyBlue

or any other color according to the "Color" section 6.

Figure 9 show how it is simple to display a Label with these 2 lines of code in the setup():

```
addView (new ALabel(AWPoint ( 300, 250), AWInt (250),  
    AAlignment (kAAlignmentCenter), String ("ChicagoFLF24"),  
    AFont (ChicagoFLF24))) ;
```

or :

```
label = new ALabel(AWPoint ( 300, 340), AWInt (150),  
    AAlignment (kAAlignmentCenter), String ("Mylabel_:_"),  
    AFont (Lucida_Grande24)) ;  
addView (label) ;
```

The constructor and methods of ALabel are :

```

class AWLabel : public AWView {
//----- Constructor
    AWLabel (const AWPoint & inRelativeOrigin,
             const AWInt inWidth,
             const AWAlignment inAlignment,
             const String & inTitle,
             const AWFont & inFont = awkDefaultFont) ;

//----- Draw
    virtual void drawInRegion (const AWRegion & inDrawRegion) const
        ;

//----- Title
    void setTitle (const String & inTitle) ;
    String title (void) const { return mTitle; }

//----- Text color
    void setTextColor (const AWColor & inColor) ;

//----- Font
    inline AWFont font (void) const { return mFont ; }

//----- Alignment
    inline AWAlignment alignment (void) const { return mAlignment ; }
    void setAlignment (const AWAlignment inAlignment) ;
};

```

An AutoLabel is a simplified version of Label, without alignment and Width, which are calculated to match automatically the string :

```

class AWAutoLabel : public AWView {
//--- Constructor
    AWAutoLabel (const AWPoint & inRelativeOrigin,
                 const String & inTitle,
                 const AWFont & inFont = awkDefaultFont) ;

//--- Draw
    virtual void drawInRegion (const AWRegion & inDrawRegion) const
        ;

//----- Title
    void setTitle (const String & inTitle) ;

//----- Text color
    void setTextColor (const AWColor & inColor) ;

//----- Font

```



Figure 10: Labels and AutoLabel

```
inline AWFont font (void) const { return mTextFont ; }
} ;
```

The code in the setup() for the Figure 9 is :

```
addView (new AWLabel(AWPoint ( 300, 50), AWInt (250),
    AWAignment (kAWAlignmentCenter), String ("Lucida_Grande24")
    , AWFont (Lucida_Grande24))) ;
addView (new AWLabel(AWPoint ( 300, 100), AWInt (250),
    AWAignment (kAWAlignmentCenter), String ("Lucida_Grande18")
    , AWFont (Lucida_Grande18))) ;
addView (new AWLabel(AWPoint ( 300, 150), AWInt (250),
    AWAignment (kAWAlignmentCenter), String ("Geneva12"),
    AWFont (Geneva12))) ;
addView (new AWLabel(AWPoint ( 300, 200), AWInt (250),
    AWAignment (kAWAlignmentCenter), String ("Geneva10"),
    AWFont (Geneva10))) ;
addView (new AWLabel(AWPoint ( 300, 250), AWInt (250),
    AWAignment (kAWAlignmentCenter), String ("ChicagoFLF24"),
    AWFont (ChicagoFLF24))) ;
addView (new AWLabel(AWPoint ( 300, 300), AWInt (250),
    AWAignment (kAWAlignmentCenter), String ("ChicagoFLF12"),
    AWFont (ChicagoFLF12))) ;
addView (new AWLabel(AWPoint ( 300, 350), AWInt (300),
    AWAignment (kAWAlignmentCenter), String ("0123456789"),
    AWFont (ChicagoDigit36))) ;
```

```
addView (new AWLabel(AWPoint ( 300, 400), AWInt (250),  
    AWAAlignment (kAWAlignmentCenter), String ("Geneva9"), AWFont  
    (Geneva9))) ;  
  
addView (new AWAutoLabel(AWPoint ( 50, 200), String ("AutoLabel  
    "), AWFont (ChicagoFLF24))) ;
```

6.3 PushButton

A PushButton is a View with an Action. The View is a roundRectangle with a Title inside.

When the user click inside the Button, an action is sent to a receiver.

The example below show a unique PushButton centered in the screen with a number inside. Each time you click in the button, the number inside the button is incremented.

The code for this example include a declaration before the setup() :

```
//--- Current button value
int buttonValue = 0 ;

//--- button action
void bigButtonAction (AWView * inSender)
{
    AWPushButton * sendingButton = (AWPushButton *) inSender ;
    buttonValue ++ ;
    sendingButton->setTitle (String (buttonValue)) ;
}
```

and an initialization of the button in the setup() :

```
// create a big button centered on screen
AWPushButton * bigButton = new AWPushButton(AWRect (0, 0, 300,
    100), String (buttonValue), AWFont (ChicagoDigit36)) ;
bigButton->setAction (bigButtonAction) ;
addCenteredView (bigButton) ;
```



Figure 11: A PushButton

The receiver of the PushButton is the Title of the button, but, in general, it can be any receiver inside any other object View.

The `bigButtonAction` function is activated each time there is a click in this button. It execute what you want to do.

The `PushButton` class contain the following constructor and methods

```
class AWPushButton : public AWView {
    AWPushButton (const AWPoint & inRelativeBaselineOrigin,
                  const AWInt inWidth,
                  const String & inTitle,
                  const AWFont & inFont = awkDefaultFont)
        ;

    AWPushButton (const AWRect & inFrame,
                  const String & inTitle,
                  const AWFont & inFont = awkDefaultFont)
        ;

    //----- Title
    protected : String mTitle ;
    void setTitle (const String & inTitle) ;

    //----- Font
    inline AWFont font (void) const { return mFont ; }

    //--- Draw
    virtual void drawInRegion (const AWRegion & inDrawRegion) const
        ;

    //--- Properties
    inline AWCOLOR textColor (void) const { return mTextColor ; }
    void setTextColor (const AWCOLOR inTextColor) { mTextColor =
        inTextColor ; }

    protected : AWInt mStringDisplayLength ;
    inline AWInt verticalMargin (void) const { return
        mVerticalMargin ; }

    //--- Enabled state
    inline bool isEnabled (void) const { return mIsEnabled ; }
    void setEnabled (const bool inState) ;

    //----- Hilite state
    inline bool isHilited (void) const { return mHiliteState ; }

    //--- Tell the view is opaque or not
    virtual bool isOpaque (void) const ;

    //--- Touch
```



```
virtual void touchDown (const AWPPoint & inPoint) ;  
virtual void touchMove (const AWPPoint & inPoint) ;  
virtual void touchUp (const AWPPoint & inPoint) ;  
} ;
```

There will be further examples of PushButton together with examples of other views.

6.4 SegmentedControl

A SegmentedControl is

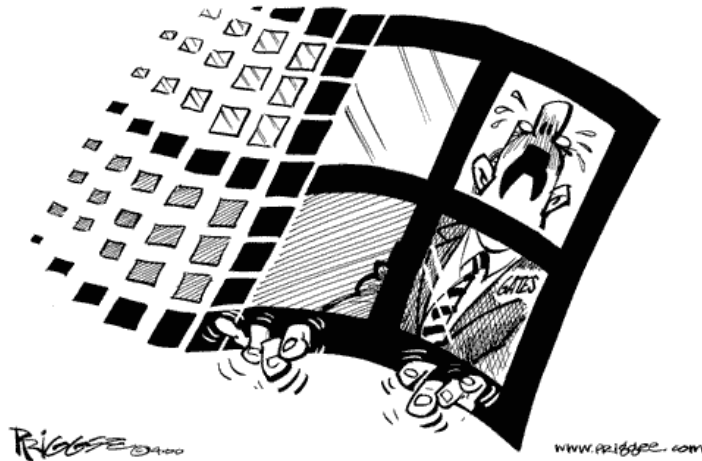


Figure 12: Aie !! missing figure !

The SegmentedControl class contain the following constructor and methods

```
typedef void (* AWSegmentedControlAction) (AWSegmentedControl *
    inSender, const AWInt inHilitedTabIndex) ;

class AWSegmentedControl : public AView {
//--- Constructor
    AWSegmentedControl (const APoint & inRelativeBaselineOrigin,
                        const AWInt inWidth,
                        const AFont & inFont =
                            awkDefaultFont) ;

//--- Draw
    virtual void drawInRegion (const ARegion & inDrawRegion) const
        ;

//--- Adding a Tab
    void addTab (const String & inTitle) ;

//--- Utilities
    ARect tabTitleRectForIndex (const AWInt inIndex) const ;
```

```
//----- Font
inline AFont font (void) const { return mFont ; }

//--- Properties
inline AInt selectedTabIndex (void) const { return
    mSelectedTabIndex ; }
void selectTabAtIndex (const AInt inIndex) ;

//----- Segmented control action
// If segmented control action is NULL (by default), touch up
// changes selection and send action (defined in AView)
// If not NULL, touch up does not change selection, and sends
// segmented control action
inline ASegmentedControlAction segmentedControlAction (void)
    const { return mSegmentedControlAction ; }
inline void setSegmentedControlAction (const
    ASegmentedControlAction inAction) {
    mSegmentedControlAction = inAction ;
}

//----- Touch
virtual void touchDown (const APoint & inPoint) ;
virtual void touchMove (const APoint & inPoint) ;
virtual void touchUp (const APoint & inPoint) ;

//----- Enabled state
inline bool isEnabled (void) const { return mIsEnabled ; }
void setEnabled (const bool inState) ;
} ;
```

6.5 Slider

A Slider is a complex View class which can control any entity by simply moving a cursor along a linear potentiometer

When the slider is moved (translated) an action is sent to a receiver with the value of the slider

In this example, one slider control the brightness of the backlight of the screen, and the 3 other sliders control the color of a colorView (a simple rectangle).

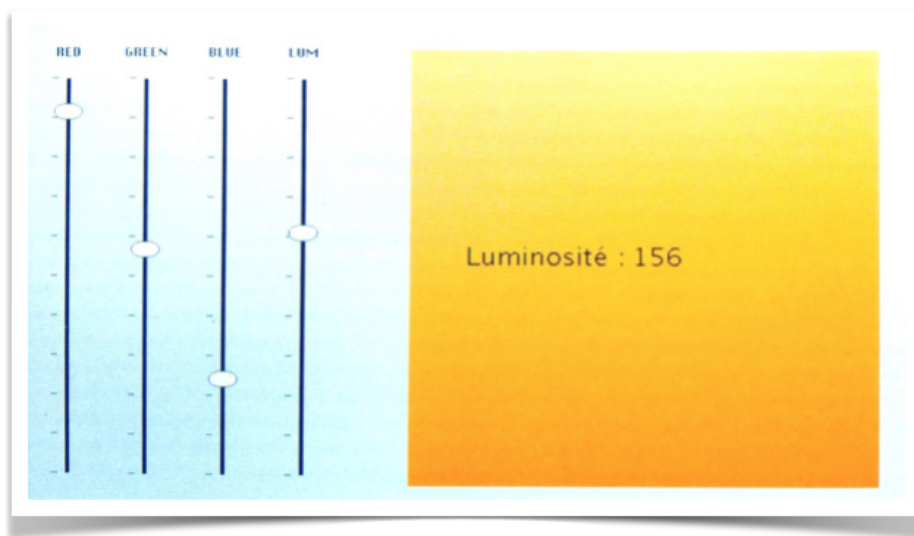


Figure 13: Four Sliders to control the brightness and the color of a Rectangle

This exemple include a declaration part before the setup() :

```
//--- Label
AWLabel * label1;
AWLabel * label2;

//--- Current color value
AWColor displayedColor = AWColor::white ();

AWSlider *redSlider;
AWSlider *greenSlider;
AWSlider *blueSlider;
AWSlider *backlightSlider;
AWView *colorView;

//--- Slider action
void sliderAction (AWView * inSender)
{
    AWSlider * sendingSlider = (AWSlider *) inSender ;
    AWInt pos = sendingSlider->knobPosition ();
    if (sendingSlider == backlightSlider) {
```

```

    analogWrite (BACKLIGHT, pos);
    label2->setTitle(pos);
}
else {
    AWColor newColor(redSlider->knobPosition (), greenSlider->
        knobPosition (), blueSlider->knobPosition ());
    colorView->setBackColor (newColor) ;
}

```

and an initialization of the sliders and receivers in the setup() :

```

redSlider = new AWSlider (AWPoint (30,30), 400, kVertical, true)
;
redSlider->setMaxKnobPosition (255);
redSlider->setKnobPosition (255);
redSlider->setAction (sliderAction);
addView (redSlider) ;
addView (new AWWLabel (AWPoint (25, 440), 40, kAWAlignmentCenter
, "RED")) ;

greenSlider = new AWSlider (AWPoint (100,30), 400, kVertical,
true) ;
greenSlider->setMaxKnobPosition (255);
greenSlider->setKnobPosition (255);
greenSlider->setAction (sliderAction);
addView (new AWWLabel (AWPoint (95, 440), 40, kAWAlignmentCenter
, "GREEN")) ;

blueSlider = new AWSlider (AWPoint (170,30), 400, kVertical,
true) ;
blueSlider->setMaxKnobPosition (255);
blueSlider->setKnobPosition (255);
blueSlider->setAction (sliderAction);
addView (new AWWLabel (AWPoint (165, 440), 40,
kAWAlignmentCenter, "BLUE")) ;

backlightSlider = new AWSlider (AWPoint (240,30), 400,
kVertical, true) ;
backlightSlider->setMaxKnobPosition (255);
backlightSlider->setKnobPosition (200);
backlightSlider->setAction (sliderAction);
addView (new AWWLabel (AWPoint (235, 440), 40,
kAWAlignmentCenter, "LUM")) ;

addView (greenSlider) ;
addView (blueSlider) ;
addView (backlightSlider) ;
colorView = new AWView (AWRect (350, 30, 420, 420), AWColor::

```

```

        white());
    addView (colorView) ;

    label1 = new AWidget(AWidget ( 400, 240), AWidget (150),
        AWidgetAlignment (kAWidgetAlignmentLeft), String ("Brightness:_:_"),
        AWidgetFont (Lucida\_Grande24)) ;
    addView (label1) ;
    label2 = new AWidget(AWidget ( 550, 240), AWidget (100),
        AWidgetAlignment (kAWidgetAlignmentLeft), String ("200"), AWidgetFont (
        Lucida_Grande24)) ;
    addView (label2) ;

```

The Slider class contain the following constructor and methods

```

const bool kHorizontal = false;
const bool kVertical = true;

class AWidgetSlider : public AWidgetView {
    AWidgetSlider (const AWidgetPoint & inOrigin,
                    const AWidgetInt inSize,
                    const bool inOrientation,
                    const bool inHasRuler = true) ;

    //--- Draw
    virtual void drawInRegion ( const AWidgetRegion & inDrawRegion )
        const ;

    //--- Orientation
    bool orientation() const { return mOrientation ; }

    //--- Knob color
    protected : AWidgetColor mKnobColor ;

    //--- Ruler display
    protected : void drawRulerInRegion ( const AWidgetRegion &
        inDrawRegion ) const ;
    bool hasRuler() const { return mHasRuler ; }
    //--- Set the number of scales on the slider. Any value < 1
        sets mHasRuler
    //--- to false so that no ruler is displayed
    void setHowManyScales ( const AWidgetInt inHowManyScales ) ;

    //--- Position
    inline AWidgetInt knobPosition (void) const { return mKnobPosition ;
        }
    void setKnobPosition ( AWidgetInt inKnobPosition, const bool
        inRefresh = false ) ;
    inline AWidgetInt maxKnobPosition (void) const { return

```

```
    mMaxKnobPosition ; }  
    void setMaxKnobPosition ( AWInt inMaxKnobPosition ) ;  
    protected : AWRect knobRect() const ;  
  
    //--- Enabled state  
    inline bool isEnabled (void) const { return mIsEnabled ; }  
    // void setEnabled (const bool inState) ;  
  
    //--- Tell the view is opaque or not  
    virtual bool isOpaque (void) const ;  
  
    //--- Touch  
    virtual void touchDown (const AWPPoint & inPoint) ;  
    virtual void touchMove (const AWPPoint & inPoint) ;  
    virtual void touchUp (const AWPPoint & inPoint) ;  
  
} ;
```

6.6 DynamicSlider

A DynamicSlider is

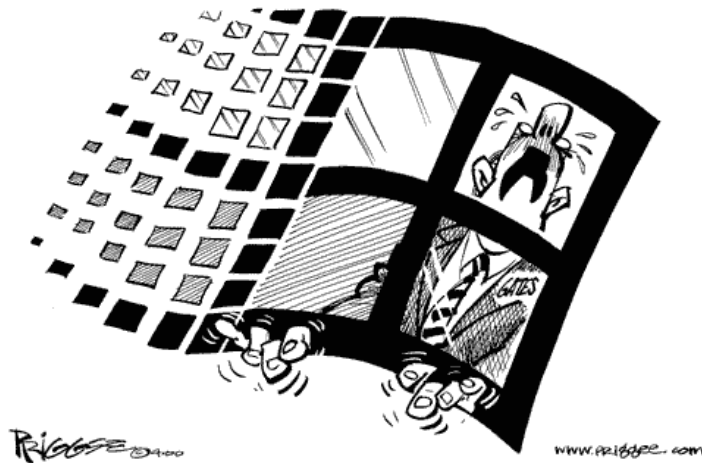


Figure 14: Aie !! missing figure !

The DynamicSlider class contain the following constructor and methods

```
class AWDynamicSlider : public AWSlider
{
    AWDynamicSlider ( const AWPPoint & inOrigin,
                      const AWInt inSize,
                      const bool inOrientation,
                      const bool inHasRuler = true ) ;

    //--- Draw
    virtual void drawInRegion ( const AWRRegion & inDrawRegion )
        const ;

    //--- Position
    inline AWInt dynamicKnobPosition (void) const { return
        mDynKnobPosition ; }
    void setDynamicKnobPosition ( AWInt inKnobPosition ) ;
};
```


6.7 TabView

A TabView is



Figure 15: Aie !! missing figure !

The TabView class contain the following constructor and methods

```
class AWTabView : public AView {
//--- Constructor
    AWTabView (const ARect & inRelativeFrame,
               const AFont & inFont = awkDefaultFont) ;

//--- Destructor
    virtual ~ AWTabView (void) ;

//--- Draw
    virtual void drawInRegion (const ARegion & inDrawRegion) const
        ;

//--- Adding a Tab
    void addTab (const String & inTitle, AView * inView) ;

//----- Font
    inline AFont font (void) const { return mFont ; }

//--- Utilities
    AInt titleHeight (void) const ;
}
```

```
AWRect horizontalSeparator (void) const ;
AWRect contentRectFromFrame (const AWRect & inFrame) const ;
AWRect titleRect (void) const ;
AWRect tabTitleRectForIndex (const AWInt inIndex) const ;

//--- Properties
inline AWInt selectedTabIndex (void) const { return
    mSelectedTabIndex ; }
void selectTabAtIndex (const AWInt inIndex) ;

//----- Badge
void setBadgeAtIndex (const AWInt inIndex, const bool
    inDisplayBadge) ; // Does nothing if index if out of mList
    bounds
bool hasBadgeAtIndex (const AWInt inIndex) const ; // return
    false if index if out of mList bounds
AWRect badgeRect (const AWInt inItemIndex) const ;

//----- Touch
virtual void touchDown (const AWPoint & inPoint) ;
virtual void touchMove (const AWPoint & inPoint) ;
virtual void touchUp (const AWPoint & inPoint) ;
} ;
```

6.8 Switch

A Switch is



Figure 16: Aie !! missing figure !

The Switch class contain the following constructor and methods

```
class AWSwitch : public AView {
//--- Constructor
    AWSwitch (const APoint & inBaseLineRelativeOrigin,
              const String & inTitle,
              const AFont & inFont = awkDefaultFont) ;

//--- Draw
    virtual void drawInRegion (const ARegion & inDrawRegion) const
        ;

//---
    void setTitle (const String & inTitle) ;

    ARect boxRect (void) const ;

//----- Font
    inline AFont font (void) const { return mFont ; }

//----- State
    inline bool checked (void) const { return mChecked ; }
```

```
void setChecked (const bool inChecked) ;

//----- Hilite state
protected : bool mHiliteState ; // false (by default): not
        hilited

//--- Enabled state
inline bool isEnabled (void) const { return mIsEnabled ; }
void setEnabled (const bool inState) ;

//----- Touch
virtual void touchDown (const AWPPoint & inPoint) ;
virtual void touchMove (const AWPPoint & inPoint) ;
virtual void touchUp (const AWPPoint & inPoint) ;
} ;
```

6.9 ArrowPushButton

An ArrowPushButton is



Figure 17: Aie !! missing figure !

The ArrowPushButton class contain the following constructor and methods

```
enum { kUpArrow, kDownArrow, kRightArrow, kLeftArrow } ;

class AArrowPushButton : public AView
{
    public: AArrowPushButton(const ARect & inRelativeFrame,
                           const uint32_t inArrowDirection,
                           const AColor & inArrowColor) ;

    //--- Arrow color
    private: uint32_t mArrowDirection ;
    //--- Arrow color
    private: AColor mArrowColor ;
    public: AColor arrowColor() const { return mArrowColor; } ;
    public: void setArrowColor(AColor & inArrowColor) {
        mArrowColor = inArrowColor; } ;

    //--- Enabled state
    inline bool isEnabled (void) const { return mIsEnabled ; }
    void setEnabled (const bool inState) ;
}
```

```
//--- On Off state management
inline bool isOn() const { return mIsOn ; }
inline void setIsOn (const bool inIsOn) { mIsOn = inIsOn ; }

inline bool onOffState() const { return mOnOffState ; }
inline void setOnOffState(const bool inOnOffState) {
    mOnOffState = inOnOffState ; }

//--- Draw
virtual void drawInRegion (const AWRRegion & inDrawRegion) const
    ;

//--- Tell the view is opaque or not
virtual bool isOpaque (void) const ;

//--- Touch
virtual void touchDown (const AWPPoint & inPoint) ;
virtual void touchMove (const AWPPoint & inPoint) ;
virtual void touchUp (const AWPPoint & inPoint) ;
};
```

6.10 KeyButton

A KeyButton is

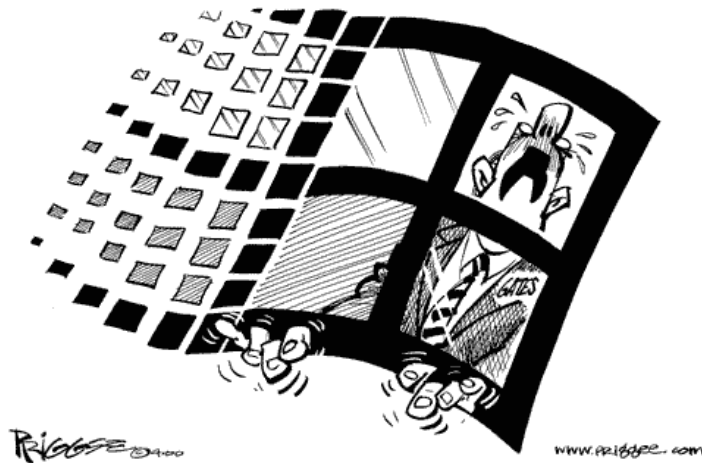


Figure 18: Aie !! missing figure !

The KeyButton class contain the following constructor and methods

```
class AWKeyButton : public AWView
{
    AWKeyButton (const ARect & inFrame,
                  const AColor & inBackColor) ;

    virtual void setShifted (const bool inShifted,
                              const AWView * const inSender
                              ) ;

    //----- Hilite state
    inline bool isHilited (void) const { return mHiliteState; }

    //----- Drawing
    protected : void drawFrameAndBackgroundInRegion (const ARegion
                                                         & inDrawRegion) const ;
    virtual bool isOpaque (void) const ;

    //--- Touch
    virtual void touchDown (const APoint & inPoint) ;
    virtual void touchMove (const APoint & inPoint) ;
}
```

```

    virtual void touchUp (const AWPPoint & inPoint) ;
};

class AWPNormalKeyButton : public AWKeyButton
{
    AWPNormalKeyButton (const AWPRect & inFrame,
                        const char inText,
                        const char inShiftText) ;

    virtual void setShifted (const bool inShifted,
                            const AWPView * const inSender
                            ) ;

    char keyChar () const { return mCurrentKey ; }

    //--- Draw
    virtual void drawInRegion (const AWPRegion & inDrawRegion) const
        ;
};

class AWPReturnKeyButton : public AWKeyButton
{
    AWPReturnKeyButton (const AWPRect & inFrame) ;

    //--- Draw
    virtual void drawInRegion (const AWPRegion & inDrawRegion) const
        ;
};

class AWPBackspaceKeyButton : public AWKeyButton
{
    AWPBackspaceKeyButton (const AWPRect & inFrame) ;

    //--- Draw
    virtual void drawInRegion (const AWPRegion & inDrawRegion) const
        ;
};

class AWPShiftKeyButton : public AWKeyButton
{
    AWPShiftKeyButton (const AWPRect & inFrame, const bool
                      inRightAlign) ;

    //---- Alignment of the arrow in the key
    virtual void setShifted (const bool inShifted,
                            const AWPView * const inSender
                            ) ;
};

```



```
//--- Draw
virtual void drawInRegion (const AWRRegion & inDrawRegion) const
    ;
};

class AWLeftArrowKeyButton : public AWKeyButton
{
    AWLeftArrowKeyButton (const AWRRect & inFrame) ;

    //--- Draw
    virtual void drawInRegion (const AWRRegion & inDrawRegion) const
        ;
};

class AWRightArrowKeyButton : public AWKeyButton
{
    AWRightArrowKeyButton (const AWRRect & inFrame) ;

    //--- Draw
    virtual void drawInRegion (const AWRRegion & inDrawRegion) const
        ;
};
```

6.11 Keyboard

A Keyboard is

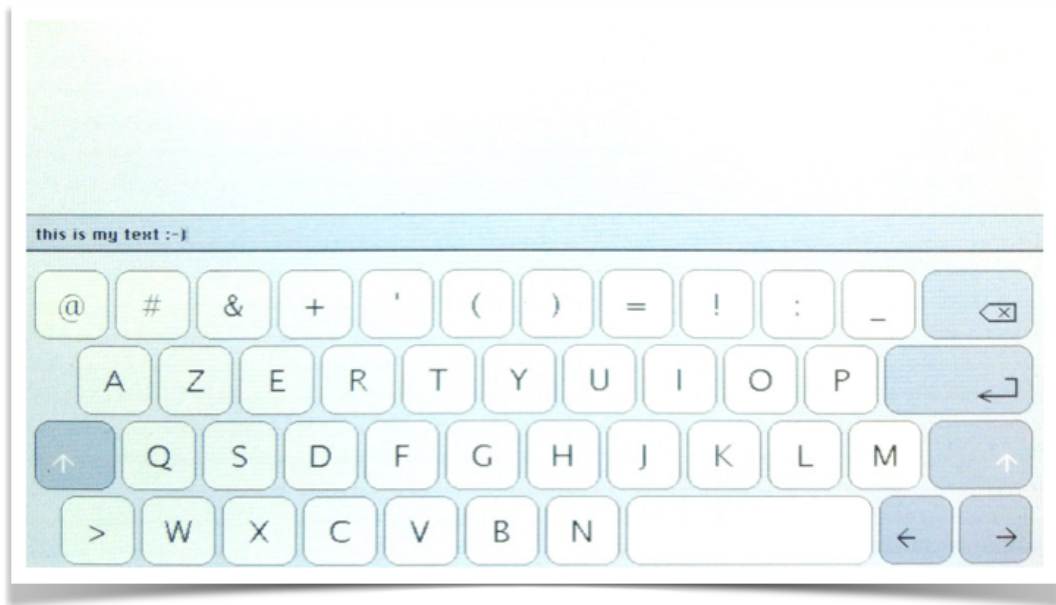


Figure 19: The keyboard view

The Keyboard class contain the following constructor and methods

```
typedef void (*AWKeyboardCallback) (const String &inText, const
    int inTag);

void launchKeyboard (const String &inText,
    const AWInt inMaxLength,
    AWKeyboardCallback inCallback,
    const int inTag = -1) ;
```

7 Custom View Examples

7.1 The Target custom view

This custom view is composed of :



Figure 20: "Target" custom view

7.2 The Keyboard and List

This custom view is composed of :

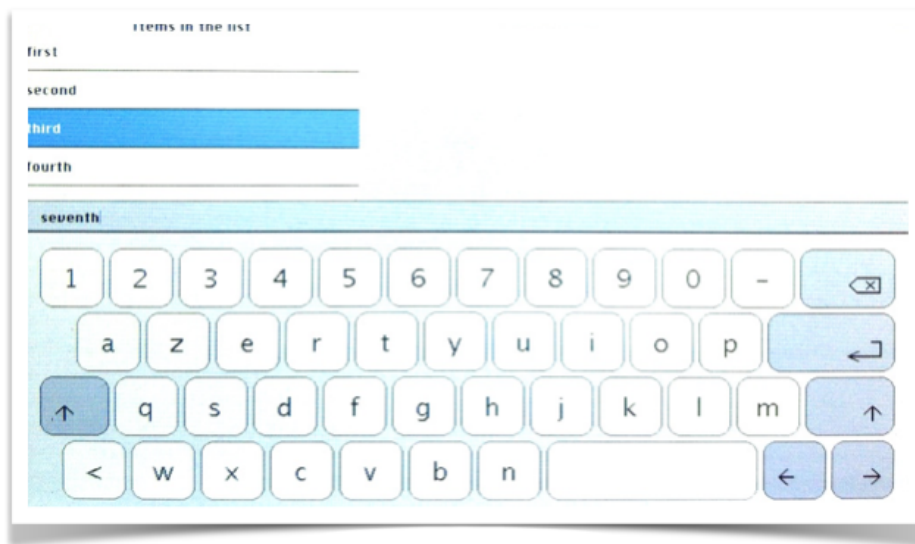


Figure 21: Keyboard and List

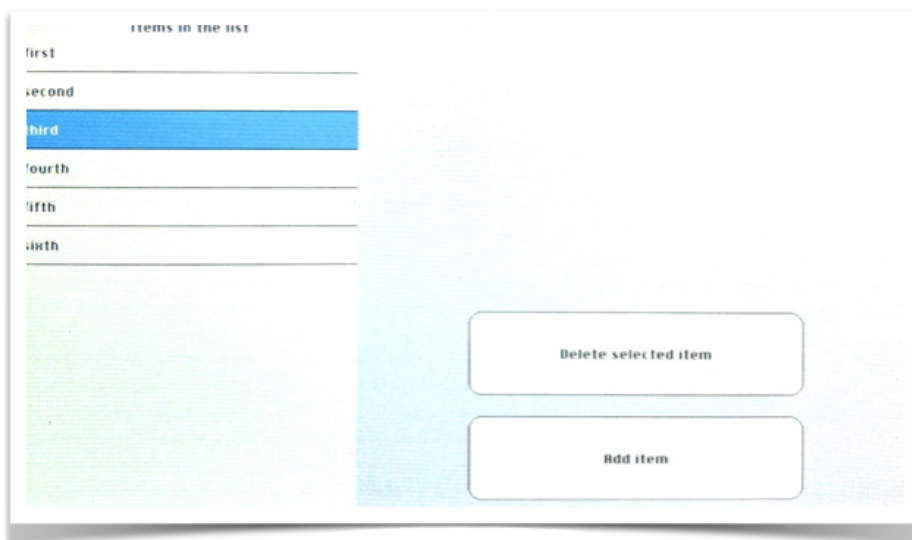


Figure 22: The List

8 The next steps of ArduinoWidgets

9 The calibration of the Touch screen