SimpleUI

Simon Schofield November 2019. Version 2.0

A Simple User-Interface code library for Processing Users.

This SimpleUI user-interface library is very simple and straight-forward way to create basic user-interfaces for your application. The simplicity of use is somewhat at the expense of greater flexibility, but if you just want to add buttons, sliders, menus and text-boxes with very little code overhead or complexity, then this might be for you. See *limitations* section.

This document is intended to be used in conjunction with the 5 code example files listed below:

SimpleUI_Example1_GettingStarted

SimpleUI_Example1_AddingWidgets

SimpleUI_Example1_CatchingEvents

SimpleUI_Example1_FileDialogues

SimpleUI_Example1_DeferredEvents

All these examples contain an identical copy of the SimpleUI.pde file.

General Principles

User-interface components, such as buttons, menus and sliders are generically called “widgets”.

The SimpleUI library can add the following widget types

Plain Button – a button that you simply click to make something happen

Toggle Button – a button that acts like a “check box”, and toggles between either on or off.

Radio Button – A button in a group, where only one button in that group can be selected a any one time. An example would be tool-selection in Photoshop

Drop Down Menu – A drop-down list of options, each option acts like a button

Slider – A horizontal slider for setting values

Text Input Box – A text input box for entering text or numbers

Getting Started

Copy the SimpleUI.pde file from any of the above listed examples into your project. This should then appear as a tab in your project. You should not really have to look at the SimpleUI tab in order to use it. However, you are free to look through it to see how it works, and you are free to adapt it if you are feeling brave.

See *SimpleUI_Example1_GettingStarted* to see a working example of how to begin using the SimpleUI library. The SimpleUI class/object should be instantiated as a **global** variable, like so

```
// top of project in first tab
SimpleUI myUI;

void setup(){
        myUI = new SimpleUI() ;
       }
```

Once created, it can be used to add widgets to the user interface, like so

```
                    …..
                    myUI = new SimpleUI() ;
                    myUI.addPlainButton("hello", 100,100) ;
                    ….
```

You must also include simpleUI's call to *update()* in the project's draw method like so

```
          void draw(){
                    myUI.update();
                    }
```

Any user-interaction with a widget creates a "user-interaction event" which is in the form of an instance of the class *UIEventData. This is sent* to the function *handleUIEvent()*. You **must** write or copy this function yourself into your own code …

```
          void handleUIEvent(UIEventData uied){

          }
```

… probably in the first tab of the project somewhere near to the setup and draw methods. This function is called by the simpleUI system every time an event is generated from the user-interface so if you do not include this function, the program will not even compile.

Adding Widgets

See *SimpleUI_Example1_AddingWidgets* for how to add all the different widgets.

The system relies somewhat on each widget having a unique "label". This is the name you give the widget, and the name you see on the widget when it appears in the window. So be careful when naming widgets to ensure uniqueness of name. This name is called the "uiLabel".

To keep things simple, when a widget detects a user-event, (for example a click on a particular button), that widget generates a "user-interaction event" and sends it to the function *handleUIEvent,* which you should have already included in the project.

The user-interaction event data (e.g. which slider was moved, and what the value of slider was set to) is bundled up into a class called *UIEventData.* This class contains all the data you would ever wish to know about the event. Some events are very simple, such as just the name of the button clicked, and some events are more complex, such as slider movement, and the new value of the slider. All this data is put into the *UIEventData* class for you to use when you need to.

The example *SimpleUI_Example1_AddingWidgets* shows how you might "catch" events from the different widgets to make different things happen in your application as a consequence.

Limitations

As of Version 2 the system does not cater for application-resizing so will work best with a fixed-size application.

Widgets only show their name, and do not support user-defined image-icons.

The menus do support fly-out sub menus

Text input boxes are very limited. They only work when you “hover” the mouse over them. They do not support cursor insertion, cut and paste etc. They do not support modifier keys, so shift-key may produce nonsense.

Class SimpleUI

Constructor

public SimpleUI();

Creates an instance of the SimpleUI library.

void update()

You must call this in the Processing project’s draw() method.

Adding Widgets

All widget-adding methods place widgets in the window at x,y. There is nothing to stop you placing two widgets at the same location. Most widgets are 30 pixels high, so need to be spaced accordingly.

All widget adding methods return the actual widget created. This is so you can further modify the widget at the moment of creation. See the toggle button method example.

ButtonBaseClass addPlainButton(String label, int x, int y)

Adds a simple button with text label at position x,y

ButtonBaseClass addToggleButton(String label, int x, int y)

Adds a toggle button with text label at position x,y. The following code snipped shows how to set a toggle button to “checked” or “true” at its moment of creation.

```
BaseButtonClass tog = myUI.addToggleButton(“milk?”,200,330);

tog.setSelected(true);
```

ButtonBaseClass addToggleButton(String label, int x, int y, boolean initialState)

Adds a toggle button with text label at position x,y. The parameter *initialState* sets the toggle button to “checked” or “true” at its moment of creation.

ButtonBaseClass addRadioButton(String label, int x, int y, String groupID)

Adds a radio button with text label at position x,y. The radio-group is defined by the forth parameter. Once the groups has been created, set any one to “checked” or “on” using the same approach as for toggle buttons.

SimpleLabel addLabel(String label, int x, int y, String txt)

Adds a non-interactive text label at position x,y. This might be useful for labelling up sections of your user-interface. Note, the string shown is that stored in *label*, but can be appended with the string in *txt*. Leave txt as “” (an empty string) if you don’t want to use this feature.

Menu addMenu(String label, int x, int y, String[] menuItems)

Adds a drop-down menu at position x,y. The top of the menu shows the *label* string. The items are defined by the list. The following code snipped shows how to add a menu with four items.

```
String[] catMenuItems =  { "tabby", "long hair", "siamese",  "tom"};
 myUI.addMenu("cats",200, 5, catMenuItems);
```

Slider addSlider(String label, int x, int y)

Adds a slider at position x,y.

TextInputBox addTextInputBox(String label, int x, int y)

Creates an empty text input box at x,y .

TextInputBox addTextInputBox(String label, int x, int y, String content)

Creates a text input box at x,y filled with the string *content*.

void addCanvas(int x, int y, int w, int h)

Adds a canvas area, which is a rectangle defined by top left hand corner at x,y of width w and height h.

Once created, the canvas will generate user-interaction events for any mouse interactions (including mouse moves) in the canvas area. You can only add one canvas to the system. When it generates an event, the uiLabel field in the event is set to “canvas”.

Removing widgets

You can add and remove widgets at any time during the life-time of a session. This enables you to alter the user-interface as you go along. For example you may wish to add widgets to support a particular “tool selection”. You may wish to also remove them later.

void removeWidget(String uilabel)

Removes the widget named *uiLabel*.

void clearAll()

Will clear all the added widgets.

Getting information

You can get and set the “state” of a particular widget, if it contains any state, at any time using the following methods.

Widget getWidget(String uilabel)

Returns the widget named *uilabel*. You can then call the widget’s methods, so long as you cast to the correct widget-type. The following code snippet shows how to do this with a slider called “volume”.

```
Widget w = myUI.getWidget(“volume”);

float val =  ((Slider)w).getSliderValue();
```

If there is no widget called “volume” the method returns a *new Widget()* type.

boolean getToggleButtonState(String uilabel)

Returns the true/false state of a toggle button named *uilabel*. If there is no matching widget the method returns false.

String getRadioGroupSelected(String groupName)

Returns the uilabel name string of the currently selected widget within a radio group *groupName.*

float getSliderValue(String uilabel)

Returns the slider value of a slider named uilabel. The value of a slider is always between 0.0 and 1.0

void setSliderValue(String uilabel, float v)

Sets the slider value of a slider named uilabel. The value of a slider is always between 0.0 and 1.0

String getText(String uilabel)

Returns the text content of a text box called uilabel. If there is no matching widget the method returns and empty string. The following code snippet shows how to extract an integer number from a text box called “Red Value”.

```
int i = int( myUI.getText(“Red Value”) );
```

void setText(String uilabel, String content)

Sets the text content of a text box called uilabel.

File Dialogue Methods

See the example *SimpleUI_Example_4_FileDialogues* for a fully working example of using the file-load and file-save dialogues.

public void openFileLoadDialog(String prompt)

Pops up the file-load dialogue. When a file is selected and “Open” is clicked, an event is sent to the *handleUIEvent* function containing the *fileSelection* string field. If you have more than one circumstance in which the file load dialogue is deployed, you can use the String *prompt* to identify the particular circumstance, this is contained in the event as the *fileDialogPrompt* string field.

public void openFileSaveDialog(String prompt)

Pops up the file-save dialogue. When a location is selected, the filename defined and “save” is clicked, an event is sent to the *handleUIEvent* function containing the *fileSelection* string field. If you have more than one circumstance in which the file save dialogue is deployed, you can use the String *prompt* to identify the particular circumstance, this is contained in the event as the *fileDialogPrompt* string field.

You may have to explicitly set the file extension in the file name field of the save file dialogue.

The UIEventData Class

This is the class generated every time there is a user-interaction with a widget. The widget creats an instance of UIEventData, fills it with the relevant information, and then sends it to the *simpleUIEvent(..)* function that needs to be defined within the project. Once the event is received by the *simpleUIEvent(..)* function, that data within is used to identify the source of the event (which widget created the event in the first place), and any other associated data (such as slider value, toggle value, mouse position etc.).

The UIEventdata contains data fields for every conceivable widget's needs, as such it is really a list of publicly accessible variables, and only has two methods.

Methods

boolean eventIsFromWidget(String lab)

Used as a shorthand to identify the source of the event. If the string lab matches the uiLable of the source widget it returns true, otherwise returns false

void print(int verbosity)

You can see the contents of the event by printing them to the console. The verbosity parameter determines how much info is printed.

0 = don't print anything

1= just print the source-widget's name

2 = Source widget plus all data fields (the public variables)

3 = same as 2, but with added mouse-moves in the canvas area, which can really spam the system as one is generated every pixel move, hence only use verbosity 3 if you really need to.

Public Variables

String callingUIManager;

You will only need to use this if you have more than one SimpleUI instance in operation. For instance you may have a main UI instance for most things but may also have a sub-panel of widgets for particular circumstances (tool settings) which may be controlled by another UI instance.

String uiComponentType;

This is the type of widget e.g. ButtonBaseClass, ToggleButton, Slider - it is identical to the class name

String uiLabel;

this is the unique shown label for each widget, and is used to identify the calling widget.

String mouseEventType;

The type of mouse event that generated the event. Particularly useful if you are handling canvas events. Values are *mousePressed, mouseReleased, mouseDragged, mouseMoved*

int mousex; int mousey;

The location of the mouse when the event was generated

Other public variable fields

These are context specific, and relate to the particular type of widget that generated the event. For instance, a slider will set the *sliderValue* field, and a menu will set the *menuItem* field etc.

boolean toggleSelectState;

Set to true of false if the source widget is a toggleButton

String radioGroupName;

The name of the radio-button-group if the source widget is a radioButton

String menuItem;

The name of the menu-item if the source widget is a menu

float sliderValue;

The value of the slider if the source widget is a slider (value is 0…1)

String fileDialogPrompt;

The prompt that was set when *openFileLoadDialog() or openFileSaveDialog()* method was called. This can be helpful in identifying the correct file-load or file-save process to follow.

String fileSelection;

The path and filename that was generated by the *openFileLoadDialog or openFileSaveDialog.*

public char keyPress;

The character pressed if the source widget is a text box

public String textContent;

The full content of the text box if the source widget is a text box