**Description**

This library is designed to be an extension to Henning Karlsen's UTFT and UTouch libraries. Essentially it is quite similar to his newer UTFT\_Buttons library, because it itself has its own button types. Most of these button types have two ways of getting the touch data and that is either from the Raw touch points which are gotten from using **myTouch.TP\_X**  / **myTouch.TP\_Y**, or the calculated touch points from the **getX()** and **getY()** functions. The reason I implemented these two different ways is because under certain circumstances, one way is faster than the other.

These types are:

* Touch button (Rectangle/ Circle /Triangle) : Able to have centered text
* Latching button (Rectangle/ Circle/ Triangle) : Able to have centered text
* Touch Delay button (Rectangle/ Circle/ Triangle)
* Radio buttons (Rectangle and Circle) : Able to have centered text
* Touchable sliders (Horizontal and Vertical)
* Keyboard (Standard and Mobile)

Using the raw *myTouch.TP\_X/Y* is technically faster because it does not require the user to calibrate the screen or worry about touch precision, but it does require the user to manually map the boundaries of the screen and the use of the map function. However the *getX/Y* functions do not require such hassling work, but they do require the screen to be calibrated and certain variables in the .cpp file to be changed. The getX/Y functions also need to be set to a precision setting, as either Low, Medium, Hi or Extreme. Low is the fastest of the four settings, because it requires minimal calculations to be done per touch, but it is not as precise; whereas Extreme has the highest touch precision but it takes a lot more time to calculate per touch thereby slowing down the processor from other tasks.

* **TouchButton / TouchCircle / TouchTriangle:**
  + The Touch button is your normal box shaped button. It needs two sets of X and Y coordinates to work, top left and bottom right. It can also be any size or orientation and accept either the touch data from myTouch.TP\_X/Y or getX/Y.
  + TouchCircle is nearly identical, but it only requires one X/Y coordinate and a radius. The X/Y coordinates you enter will become the center of the circle, and the radius will determine how big the circle is.
  + TouchTriangle is just like TouchCircle, but instead of a radius, you have a base. The triangle you make is an equilateral, no other triangle shapes are implemented yet. The TouchTriangle does require two other variables to be entered and those are direction and degree. Direction is either "up, down, left or right" all lower case, and the degree is 0 - 359.

\*\* These buttons can have centered text by entering the text into the SetTouch(Button/Circle/Triangle) Text functions

* **Latchbutton / LatchCircle / LatchTriangle:**
  + Just like the touch button and touch circle functions, they too can be any size and orientation, but this type button will need its own identification number. The current maximum allowed is 10 but this can be changed in the TFT\_Extension.h file,  *Num\_Of\_Buttons 10.*

\*\* These buttons can have centered text by entering the text into the SetLatch(Button/Circle/Triangle) Text functions

* **TouchDelayButton / TouchDelayCircle / TouchDelayTriangle:**
  + What this function does is once you press the button it will look at the X/Y pairs and start a timer. If the button is held down for a set time, the button will either return true if that time has passed, or false if it has not.
* **TextButton:** (Replaced by both SetTouchButtonText and SetLatchButtonText)
  + ~~This allows a button to have text in the center of it at two different font sizes, Big and Small. Right now this will center the text even if the text is large than the button, so this function is still a work in progress. I also plan on making the button be able to rotate with the text and still work as a normal button. This will also be implemented to the other buttons as well.~~
* **RadioButton / RadioCircleButton:**
  + If you are familiar with the old style car radios, then you will know how these particular buttons work. Only one button can be on at a time and if another is pressed, it goes on and turns off the previous button. To get the radio buttons to work, you first need to tell the compiler how many groups you want and the number of buttons per group. This can be done with the functions *TotalRadioButtons* and *TotalRadioCircleButtons.* An array is needed to set the different number of buttons per group.
  + You can have a total of 5 groups with 10 buttons in each group and both button types work independently from one another, so both can have 5 groups and 10 buttons.
  + There are quite few functions dedicated to the radio buttons, in which they can change the appearance of the buttons either individually, as a group or all as one. *See the radio button example sketch*.In the event you want to use menus with the radio buttons, the functions *ResetRadioButton* and *ResetRadioCircleButton* can clean/reset the buttons so the screen can be changed. NOTE: This does not reset the states of the buttons. Currently it only does one group per function. (5 groups, 5 functions.)

\*\*Both types of radio buttons now are able to have text in the center of them by using the RB\_Text or RCB\_Text functions. **Please note** the text will not be cut off, nor will the buttons size be altered if the text is larger than the button.

* **TouchButton \_Draw/ TouchCircle\_Draw / TouchTriangle\_Draw:**
  + All three are nearly identical to their counterparts TouchButton, TouchCircle and TouchTriangle, but these require an ID number to work.

*These type of buttons also need to be colored, told whether they should be rounded or not, and filled or not. This is done with the corresponding functions, SetTouchButtonColors, SetTouchCircleColors and SetTouchTriangleColors.*

**Note: Circles do not have the option to be rounded, because they are already round XD**

* **LatchButton \_Draw/ LatchCircle\_Draw / LatchTriangle\_Draw:**
  + See "Latchbutton / LatchCircle / latchTriangle:" for description

*Again, these type of buttons also need to be colored, told whether they should be rounded or not, and filled or not. This is done with the corresponding functions, SetLatchButtonColors, SetLatchCircleColors and SetLatchTriangleColors.*

**Resetting The Buttons**

All the \_Draw functions are able to be reset when switching screens or using a menu system. They can either be reset individually or by all of the same type.

* ResetTouchButton(byte ID);
* ResetTouchCircle(byte ID);
* ResetTouchTriangle(byte ID);
* ResetLatchButton(byte ID);
* ResetLatchCircle(byte ID);
* ResetLatchTriangle(byte ID);
* ResetAllTouchButton();
* ResetAllTouchCircle();
* ResetAllTouchTriangle();
* ResetAllLatchButton();
* ResetAllLatchCircle();
* ResetAllLatchTriangle();

**Sliders:**

* VertSlider
  + Like the buttons, this needs two sets of coordinates, top left and bottom right. These slider need ID numbers in order to have multiple sliders on the screen to work. A color is also need to indicate progression. Please note, the background color is set to black.
* HorSlider
  + -Same as VertSlider

**BarGraphs:**

* VertBarGraph
  + as the counterpart of the sliders, these also need two sets of coordinates TL and BR. The bargraph needs a value and a set range as well as an ID and color. The value inputted will be mapped to the range and shown accordingly.
* HorBarGraph
  + -Same as VertBarGraph
* ResetV\_barGraph
  + Like all the other reset functions, these all the bargraphs to be repainted on the screen when switching screens or using a menu system.
* ResetH\_barGraph
  + -Same as above

**Other Functions:**

* Triangle
  + Standard wireframe for making a triangle, takes three set of coordinates.
* drawTriangle
  + This will make the triangle in a circular plane, meaning it can be rotated to any given degree. It requires a X/Y coordinate, a base size, and the degree, default is 0.
* fillTriangle
  + Just like the drawTriangle function it requires the same input. I am trying to find a more efficient way to fill the triangle at any given degree.
* Polygon
  + This will create a closed polygon based on the sides entered (max 360). It needs an X/Y coordinate, the number of sides you want to make, a diameter of how big you want it to be, a RGB color, Filled or Not Filled, and a degree(optional).
* fillPoly
  + Contrary to its name, it fills in a triangle based on the three sets of coordinates provided.
* drawStar
  + Creates a perfect star. Takes a X/Y coordinate, diameter, RGB color, Filled/Not filled, and a factor(optional, default is 2.0). The factor determine the thickness of the triangular points
* drawArc
  + Takes an X/Y coordinate, a radius, thickness, a start and stop point, a RGB color and boolean T/F(default is False) to determine if you want to close the oval like a pie slice.
* drawOvalArc
  + Like drawArc, but this requires two radius values and two thicknesses.
* SetupStandard\_KB
  + needed to setup the keyboard's font and orientation, but only requires user to enter the font type. Bigfont not supported.
* SetupMobile\_KB
  + needed to setup the keyboard's font and orientation, Bigfont is the default font type.
* Standard\_KeyBoard
  + Creates a keyboard on screen that mimics a standard computer keyboard but without the F buttons(F1 - F12) and window/option button.
* Moblie\_KeyBoard
  + Creates a keyboard on screen that mimics the IPhone/Android keyboards.
* ReceiveMsg
  + Need to receive incoming message.

**Special functions:**

* drawMoon
  + Needs a X/Y coordinate, a radius and a phase WANING\_GIBBOUS 7, THIRD\_QUARTER 6, WANING\_CRESCENT 5, NEW 4, WAXING\_CRESCENT 3, FIRST\_QUARTER 2, WAXING\_GIBBOUS 1, FULL 0. Can be entered by name or value.
* rounded\_Square
  + Takes a X/Y coordinate, a height and width value, a radius, RGB color and Filled/Not filled. The radius determines how round the corners will be.
* smiley\_Face
  + Create the standard yellow smiley face. Takes a X/Y coordinate, a radius, and a value for the eyes and mouth. (Eyes max is 9, mouth max is 7) Presets are can be found in the .h file
* HourGlass
* SpeechBubble
* drawGauge
* ResetGauge