NailBuster GUI Widget Library v1.3

for GameMaker Studio v1.xx

Contents

Getting Started

Thanks for trying out my components. The best thing to start is to watch the youtube videos on getting started so you understand how they work and interact.

Remember the most important thing is that you never need to modify these objects directly. You simply drop the objects onto your room, set their size/position visually, and then modify the create code (within the room editor) to change the many options available to the widget.

Getting Started: https://www.youtube.com/watch?v=POTzMrKYx_4

Sample Menu: https://www.youtube.com/watch?v=B5k2sl_TKz8

To jump right in and sample, you can just import the sample project from the SampleProject Folder. Browse the create code on each object (especially the ui_defautls) to get an idea of the properties available to you. Then move into each room and browse the createcode for each instances.

Note that all configurable properties/settings start with the "ui" prefix. So if you see a variable in the create method called uiPulseSpeed, that means it's a property that you can change. If you see another variable called cur_pos then that is not a property and shouldn't be changed.

All objects descend from ui_defaults. An advantage of that is you can set the defaults for your project like uiTextColor, uiFont, uiBackColor by changing it within the create method of the ui_defaults object. That way you won't need to set it on all instances on each room, it is like a default schema for the current project. Of course, all the defaults can be overwritten at the individual level within the room editor.

Installing:

To add the Widget Library (first time) to an existing project do the following:

1> create a font resource call font_ui (case and exact spelling is important). Setup this font, it will be the default font for the components. I suggest Arial/14 or 16 size. You can set custom fonts on each component as well but it will need a default font always.

2> from this zip file go to the folder STEP1_AddFirst and drag the sprite file into your open project. Make sure you rename this sprite to "spr_ui" without quotes.

3> Drag the defaults object from the STEP1_AddFrist folder into your project. no need to rename

4> Go to the STEP2_AddSecond Folder and drag all object files into your open project...doesn't matter what order and no renaming required.

5> optional but nice is to create a group under objects in your project and drag the new objects into that group.

Updating your project to use a newer version of Widget Library

Updating an existing project is simply overwriting all object files from the update folder and overwriting them within your project objects folder.

This is a little more complicated if there are new ui_widgets to add as well as update, you need to determine which ui_XXXX.object.gmx files are new(from the update sub folder within zip) and drag/add those onto your existing project first. Once you've added new widget objects you need to copy all files from the update sub folder and overwrite the files inside your projects objects folder.

Make Backup of your project files first! Note the defaults object will get overwritten so you'll need to set the defaults again on that object.

open your project you want to update inside GameMaker Studio.

1> Using windows file explorer, copy all gmx files from within the update sub folder of your NailBuster widgets folder.
2> with your project open in GM:Studio go to help menu/open project in explorer.
3> Close your project and GameMaker Studio!
3> click the objects sub folder inside your project explorer window.
4> paste and overwrite all files from 1 (overwrite all).
5> ReOpen GameMaker Studio with your project and check that it updated.

Licensing

Using the widgets (objects)

The design of the GUI framework is to minimize the number of objects to manage your game menu GUI. To use the objects you simply add them to your room and modify the create code. You do NOT modify the widgets directly or assign sprites to them like you would your normal game objects. For example, with the button widget, you assign uiButtonSprite=spr_yoursprite and the button widget will use that for drawing.

To access the widgets around your code you simply use the instance name. So in your room editor you can add a label, (use rename instance name to something like "lbl_gamer") you can then change the text that gets displayed in code by using lbl_gamer.uiTextValue='John Smith'; So you can modify any widget in code by using the instance_name.property syntax.

Other examples would be: (set instance name in room for the listbox to : weapons) Then we use this outside of the create room:

```
weapons.items=0;    //clear the list array
weapons.items[0]='First weapon';
weapons.items[1]='Second weapon';
weapons.items[2]='Third weapon';
```

Interacting with Widgets

Widgets on their own will handle user interactions. However, we need to be able to respond to certain events in code when they happen. We may need to know when the OK button is pressed, or a checkbox is checked/unchecked. If you examine the sample project, look at the obj_MainUI object and look at its user_defined(0) code to see how it would work. It is hard to explain, but the youtube video of sample menu will show you how to do this.

Depending on the form, we may need to just respond to the OK/Cancel Buttons..or we may have to respond to each individual widget. The choice is yours depending on the current form.

All widgets will call user_event(0) when they are 'changed', you must create your own EventObject that will handle all these events from all the controls. You can decide to have a persistent object for your entire MenuGUI, or you can have a separate EventObject for each room. For each control you need monitor, you need to set the uiEventObject to your object, and set the uiControlName so that your object knows who was calling the event.

```
uiEventObject=obj_my_MainUI;
uiControlName='weapon_listbox';
```

All that you do now is on your EventObject, add a user_event(0) and handle all calls to it. Using the "other" is how you know who which widget called your object. Like this: If other.uiControlName='OKbutton' then xxxxx;

Now if you are only concerned with an Ok button response, then once you handle the OK button press event then you can grab all the properties of all the instances on the room. Like getting the play_sound.checked=true for checkboxes, or myarray[mylistbox.itemindex].

Widgets

ui_defaults

All objects descend from ui_defaults. An advantage of that is you can set the defaults for your project like uiTextColor, uiFont, uiBackColor by changing it within the create method of the ui_defaults object. That way you won't need to set it on all instances on each room, it is like a default color theme for the current project. Of course, all the defaults can be overwritten at the individual level within the room editor.

We'll discuss the base ui_defaults properties: Some are self explanatory; Using the sample project you can see/test some of the changes:

uiTextValue="";

This will set the TextValue of the control, like the text that appears on the button or label.

uiTextLabel="";

This is used for controls that need additional text. Like the heading of the listbox control, or the popup question on the label;

uiBackColor= c_white;

The background color of the control.

uiDrawBackColor = true;

If you want to make the control transparent (disable the uiBackColor) from drawing.

uiTextColor= c_black;

The color of the text that is drawn on the control

uiDrawOutline = false; //fill button when drawing.

Will draw an outline around a button, without flood-filling the background color;

uiBorderColor=c_white;

If widget has a border, like a listbox.

uiBorderBackColor=c_gray;

If widget has a border, like a listbox.

uiFont = font_ui;

This is the font used by the widget.

uiEnabled = true/false;

This will display the widget at 50% alpha, and will not allow any GUI interaction by the user. No events will be fired when disabled;

uiVisible= true/false;

This will hide/show widget.

uiSetWidth= 1000;

This will set the widgets width in pixels, don't have to worry about image_xscale. Normally you don't need to set this because its set by the room designer. But if you are creating the controls in code this will simplify it.

uiSetHeight= 1000;

This will set the widgets height in pixels, don't have to worry about image_yscale. Normally you don't need to set this because its set by the room designer. But if you are creating the controls in code this will simplify it.

ui_button_ani

Button widget have abilities to draw sprites at different locations within the instance size, you can have the a background drawn or be transparent. It supports text alignment, toggle option and button groups. To add a button to add the object to the room and change the create code, you do NOT assign a sprite to the object(or child object), to assign the sprite you use the uiButtonSprite property.

uiAlign = fa_left;

This is for horizontal text alignment, fa_left, fa_center,fa_right;

uiAlignV = fa_top;

This is for vertical text alignment, fa_middle, fa_top, fa_bottom

uiXscale = 1.0;

This will allow you to draw the button at a scale factor. this is the X scale to draw.

uiYscale = 1.0;

This will allow you to draw the button at a scale factor. this is the Y scale to draw.

uiButtonState = 0; //0 = up, 1 = down

The current state of the button in a button group. Can be used set/get the button state of button group (see button groups). Also to get the state when using the toggle mode.

*uiButtonSprite**= -1;***

The sprite to draw with the button.

*uiSpriteIndex**=0;***

Similar to the sprite index setting on normal objects in GMStudio;

uiSpriteAlign=fa_center;

Alignment of horizontal sprite in button.

uiSpriteAlignV=fa_middle;

alignment of vertical sprite in button

uiSpriteScale=1.0;

This will allow you to draw the sprite at a scale (same x/y scale). You normally may set this to 0 (zero) which means to fill the widget based on room design.

*uiButtonSpriteDown**= -1;***

The sprite to draw with the button is down. Is really useful when doing a toggle button. Like a volume control would show a mute icon instead of the speaker icon when up.

*uiSpriteIndexDown**=0;***

Sprite index when button is pressed. Works with uiButtonSpriteDown.

*uiDownColor=c_green**;***

The color of the background when button is pressed.

uiPressFlash=false;

Will cause the button to flash (uiDownColor) when pressed.

uiFlashCounter=0;

if uiFlashButton then how long to flash counter, it's a number (30 is equal to one second, 60 would equal 2 seconds)

uiBackColor2 = 0;

The background of the button is solid filled, but if you set this it will be gradient filled between uiBackColor and this color;

uiShadowColor =c_gray;

This is the color of the shadow that is draw when uiDrawBackColor=true;

uiIsToggle = false;

This will set the button to toggle, which means it will stay down once pressed and released when it is pressed again. You can check the state of the button by reading uiButtonState.

uiSpriteAngle=0-360;

This is the angle that the sprite will be drawn.

uiEnforceFocus=true; //check if button was in focus on press

When true then the button is considered 'pressed' when it is pressed and released. If false, then it allows user to move around and the button is considered 'pressed' when it is released on the widget. (for keyboards for example, mouse up pressed is preferred)

uiButtonGroup = 0; //button group

Button Groups

Button groups are when you group a set of buttons together so that when one is pressed the other buttons in the same group are released. To group buttons together you only need to set uiButtonGroup=x (where x is a number greater than 0) . So if you wanted multi button groups on the same screen, you would assign uiButtonGroup=1 on all the button widgets in the first group, and uiButtonGroup=2 for the other group.

To get which button is selected, check the ui_button_name.uiButtonState=1; or you can do it easier by setting it through the event object. See Interacting with Widgets...

ui_button

This button widget is a 'lite' version of the ui_button_ani. It doens't support animations and such, but if you need a very fast button without the overhead of the animated version then you can use this button. It is a legacy control, and as such, no new features will be added to this widget.

Button Animations

uiAnimated=false;

The best method to see it in action is through the sample project (button animation room). That way you can play with different speed and rotations.

To animate a widget you set the animation settings and then call uiAnimated=true; To stop the animation you set uiAnimated=false; To set animation settings you make the animation property NON-ZERO. So to enable uiTextWobble to 15 (which means wobble back and forth 15 degrees), to disable set uiTextWobble to 0 (remember that uiAnimated=true is needed). You can also compound animations, which means you can do all/some all at the same time.

The following properties are self explanatory...speed is between 0 and 1 (or more).

uiWobble=0; //set degree to wobble text 15= 15 degrees back and forth
uiWobbleSpeed=1;

uiPulse=0; // this is a percent: so setting is to 0.2 will pulse 20% (80-120%) of original size.
uiPulseSpeed=0.01; //should be a small number
uiPulseJello=false; //set to true to have a jello pulse effect.

uiFading=0; //0.5 would fade between 100% and 50%
uiFadingSpeed=0.03;

uiSparkle=0; //15 is a good number here, experiment
uiSparkleSpeed=1;

uiTextWobble=0; //set degree to wobble text 15= 15 degrees back and forth
uiTextWobbleSpeed=1;

uiGlowing=0; // 0.6 would be 60% glow, uses the uiFaddingSpeed for glow speed.

ui_checkbox

Checkbox widget that will display a string and a checkbox sprite. The user can select or deselected the checkbox on mouse press. Example,

```
uiTextValue="Don't ask again";
uiAlign=fa_right;
uiDrawBackColor=false;
uiTextColor=c_white;
```

To get the value of the checkbox in code check the checked variable. Like this or use the event object method.

```
if inst_box_name.checked then show_message('it is checked');
```

ui_checkgroup

Checkgroup widget is when you group a bunch of checkboxes into one widget. You can allow the user to select just one, or can select many. The checkgroup will automatically divide the number of items to evenly space out within the widgets room dimensions, so there's no need for you to concern yourself with each items height.

to assign items you use the items[] array:

```
items[0]='The first checkbox';      //string of checkbox
itemschecked[0]=0;                  //status of checkbox (one per each line!!);
items[1]='The second checkbox';     //string of checkbox
itemschecked[1]=0;                  //status of checkbox (one per each line!!);
```

```
items[2]='The last one checkbox';        //string of checkbox
itemschecked[2]=0;                       //status of checkbox (one per each line!!);
```

uiMultiSelect=false;

To configure multiselection, you set the uiMultiSelect=true/false variable.

itemindex = -1; (-1=no selection, or index of array of selection, 0 =first item)

To get the currently selected item from the checkbox group you use this variable. Doesn't make sense to use this variable if it is a multi-selection checkgroup.

ui_combobox

The combobox is a useful drop-down list widget. It will pop-up a choice window for the user to select a string. It does not support a scrolling drop-down list so it is best suited when there are a few options . It does automatically determine if the choice-list is shown upward or downward depending on the space left in the room.

to assign items you use the items[] array;

```
uiTextValue='Combo Box';
items[0]='Combo Line 1';
items[1]='Combo Line 2';
items[2]='Combo Line 3';
items[3]='Combo Line 4';
```

Once the user selects from the drop down list then the string is copied into the uiTextValue of the parent combobox. So you can grab the combobox.uiTextValue to see what was selected, or you can get the itemindex that was selected using combobox.itemindex (0= first item). You can use the modified property to see if it was modified since the create. See demo project to see examples.

itemindex = -1; (-1=no selection, or index of array of selection, 0=first item)

To get the currently selected item from the combobox you use this variable.

uiDrawLines=true;

This will determine if you want lines drawn between lines on drop-down list.

uiDrawLines=true;

This will determine if you want lines drawn between lines on drop-down list.

uiDrawArrowUp=false;

This will force the drop-down box to draw-upward.

ui_editbox

Editboxes are a powerful way of allowing keyboard entry of strings. Normally in GMStudio to get a string you would use get_async_str which is not always the 'nicest' way of getting a string. Using editboxes the app will appear more 'windows' like in getting strings from the keyboard. See demo for examples. It support many features like mouse cursor select, backspace, delete, arrow keys and even tab-order(shift-tab) support.

uiTextLimit=999;

Limits the length of the text that can be entered.

uiTextColor=c_black;

Color of the text when NOT in focus (selected).

uiFocusColor=c_gray;

Background of editbox when focused.

uiFocusTextColor=c_white;

Color of text when in focus.

uiCursorColor=c_white;

Cursor (flashing) color.

uiCursorWidth=6;

Line width of cursor;

uiPopupKeyboard=true;

This will determine if the ui_keyboard will be displayed when focused/pressed. If OS is brower or windows you may want to disable this by setting to false;

uiTabOrder=0 (set to 1,2,3,4 on individual widgets);

This will allow you to move between editboxes using the tab/shift-tab. Just set taborder to a non-zero integer. You must keep the order yourself, meaning you assign 1 to the first one, you assign 2 to the 2nd editbox...etc.

ui_gauge

Gauge allows you to display a nice, animated bar that can be used to show progress or health. See demo for different examples. It has ability to change colors depending on value of widget. To set the value of the gauge you use the uiSetValue=0-100 . It is percent based so you will have to calculate the 0-100. To read the current value you use uiValue, this is read only and you should NOT set uiValue.

uiValue=100; READ ONLY!

percent based 0-100....so 50 is 50 percent, use uiSetValue to set!!!

uiBarColor=c_green;

The color of 'full-bar', like all green if health if greater than uiBarLevel.

uiBarColorBlend=c_lime;

The color used with uiBarColor for blending.

uiBarLevel=35;

Percent based, 35 means that for 35-100% draw the uiBarColor, under 35% will draw uiBarColor2;

uiBarColor2=c_yellow;

The color that is used when value of gauge is less than uiBarLevel.

uiBarColorBlend2=c_white;

The color used with uiBarColor2 for blending.

uiBarLevel2=10;

Percent based, 15 means that for 10%-uiBarLevel% draw the uiBarColor2, under 10% will draw uiBarColor3;

uiBarColor3=c_red;

The color that is used when value of gauge is less than uiBarLevel2.

uiBarColorBlend3=c_maroon;

The color used with uiBarColor3 for blending.

uiBarFlash=15; //0=no flash, else speed to flash flash on level3 (dying…);

whether to flash when gauge is drawing BarColor3 (dying mode).

uiDrawSquare=false; //rounded or rectange;

whether to draw a rounded or rectangle box for widget.

uiAniDraw=true; //animate the movement….not jump to position

animate gauge movements between values, or jump/draw actual values instantly.

uiAniSpeed=1; //step added to value.

speed of movements when uiAniDraw is true.

uiSetValue=-1; //set only! do not read this value, must use uiValue to read.

To set the value of the gauge you use this. For example, my_gauge.uiSetValue=100, will set the gauge to 100%. To get the current value of the gauge you must use uiValue.

ui_keyboard

The keyboard widget is used with editbox widgets. When a user presses an editbox the keyboard will be displayed. All that is required is to add just ONE ui_keyboard object to a room that has the editbox widgets. The keyboard widget allows you to change the background colors, key colors and such. You can change which keys are displayed and assigned.

These are the colors you can assign on ui_keyboard

```
//key colors
uiKeyBackColor=c_silver;
uiKeyBackColor2=c_silver;
uiKeyTextColor=c_black;
uiKeyDownColor=c_yellow;

//special keys
uiSKeyBackColor=c_dkgray;
uiSKeyBackColor2=c_dkgray;
uiSKeyTextColor=c_yellow;
uiSKeyDownColor=c_teal;
```

The only other item that you may want to change is the keyboard panel height. This is a special number, as it is a ratio, so change the number +/- 20 at a time to see the effect.

```
uiKeyboardHeight=270;   //height of keyboard is a ratio to 600 pixels.
```

There are 3 rows to the keyboard, and 3 lines of keys. To change the keys this array[1,2,3] must be defined.

```
uiRowStartX[1]=10;
uiRowStartY[1]=1;
uiRowSpacingX[1]=10;
uiRow[1]='qwertyuiop';
uiRowShift[1]='QWERTYUIOP';
uiRowAlt[1]='1234567890';
```

You need to make sure the # of keys are identical for each index and row.

ui_label

Label widget that will display uiTextValue, by default it won't draw a background, but you can set uiDrawBackground=true if you'd like.

```
uiTextValue='NailBuster.com GUI Suite for GameMaker Studio';
uiAlign=fa_center;
uiTextColor=c_white;
```

ui_listbox_ext

The listbox is a powerful yet simple to implement widget. You just set the items[] array with the strings you want to display in the list. Set the colors and hilite colors are really all you need to do. See the sample project listbox room to see the different options.

to assign items you use the items[] array:

```
items[0]='The first Item';
items[1]='The second item';
items[2]='The last item';
```

itemindex = -1; (-1=no selection, or index of array of selection, 0=first item)

To get the currently selected item from the listbox you use this variable.

uiDrawLines=true;

This will determine if you want lines drawn between lines.

uiDrawHeader=true;

Whether to draw the header bars.

uiDelayEvent=1; //number of steps to delay

if you want to delay the press to call event (for highlighting). Normally the event object is fired when the mouse is pressed. If you want to delay that you would change this value.

uiMemo='';

You can display a listbox as a read-only memobox. Just disable the drawlines, and other items you don't want in your memobox and set uiMemo to a string. This string is automatically broken apart, so you can write a large string here. You can use # to cause a new line in your memo string.

uiDrawHilite=true;

Draws the hilite bar, you would want to disable this if you are displaying it as a memobox.

uiCredits=false;

Nice little animation, if you setup the listbox as a memobox you can set this to true. It will scroll the listbox from top to bottom, and repeat. It is even better if you start your uiMemo string to start and end with a few '#' to get some empty space and start and end. See sample project for example.

uiCustomDrawScript=-1;

The listbox supports custom draw scripts, this is the property to set to your custom script for drawing each line item. See Custom Draw Section.

*uiSurfaceHeight=0; //**override surface height (Default is textheight * numitems)***

Normally you won't need to manage this. For advance users only.

uiScrollArrows=true;

Display the scroll arrows on/off.

uiScrollColor=c_dkgray;

Color of the scroll arrows.

uiMoveToTop = false; ***//will reset the drawy position to top***

If you are resetting the items array, and want to start displaying the list from the top. Just set uiMoveToTop=true and the list will refresh from the top.

maxLineHeight=128;

The surface used by the listbox has clipping, this is the area before and after the list. This only should be changed if your max lineheight is going to be larger than 128 pixel.

ui_listbox

The ui_listbox widget is a 'lite' version of the ui_listbox_ext. It doesn't use the same resources and overhead of the listbox_ext widget. ui_listbox doesn't use surfaces for clipping and may be a more appropriate to use if you need just a simple listbox widget without all the overhead. It is a legacy widget so no new features will be added, but it is a very feature rich listbox as it stands. It does support custom draw scripts (however you cannot have variable heights when using this widget).

ui_listbox_horz

The listbox horizontal control is a useful list of items that are displayed to the user from left to right with scrolling and hilite ability. It is exactly like a listbox (vertical) but it is horizontal.

See ui_listbox_ext, as almost all the properties apply to the horizontal version. The major difference is that listbox_horz supports an array of sprites[] as well as strings items[]. Like the normal listbox, you assign the strings to items[x], but you would also set the matching sprites[x] index. Example:

```
items[0]='The first Item';
sprites[0]=spr_firstimg;
items[1]='The second item';
sprites[1]=spr_secondimg;
items[2]='The last item';
sprites[2]=spr_lastimg;
```

Another feature of the horizontal listbox is the ability to snap to center item. See demo project for example.

uiSnapLeft=true;

will snap the left item to the left of the widget (no left side clipping)....

uiLeftMargin=75;

when snapping this is snap margin line

uiAutoSelect=true;

will auto hilite the center item when using uiSnapLeft;

ui_panel

Panel widgets are a great way to group together widgets and save time in controlling multiple widgets. Think of a panel as a windows form, where each 'child' widget is linked to the ui_panel widget. Then by just moving the panel, all children will follow. Children will inherit positioning, alpha, zoom, enabled and visible properties.

Add a panel, assign it's colors or background sprite. Rename the panel instance name to something easy. Then you need to go to each child widget and assign the panel's instance name to its uiAnchor property.

So if you have a panel instance name called 'panel_config'. Then we would go to all the child widgets and on the create method we would set uiAnchor=panel_config. Now in code if we were to set panel_config.uiVisible=false, all the children(linked widgets) would be invisible. This is just one example of how to use the panels, there are many useful ways that they help make life easier.

These are properties that you may wish to change for panels.

uiBackColor = c_navy; //background color

uiBackColor2 = c_blue; //gradient with uiBackColor

uiPanelSprite = -1; //Panel for Sprite (optional)

uiSpriteIndex=0; //Spriteindex for uiPanelSprite

uiSpriteScale = 1.0; //0=auto-scale sprite to widget dimensions.

ui_slider

Slider widget is great for allowing entry by user using a slide control (horizontal only). You have ability to select min and max values, the step amount, and even have a custom sprite for the slider key. Just add a ui_slider object (horizontal) and size it in your room, it will auto-scale the slide to the widget dimensions in your room.

Here are properties for the slider widget:

uiMax=100;

This is the maxValue of your slider. If it is all the way to the right, this is the value of the slider.

uiMin=1;

This is the minimum value. If it all the way to the left this is the value of the slider.

uiInterval=5;

whether you want to keep the value within a interval. 5 would mean that values are divisible by 5. (5,10,15,20...)

uiValue=50; //integer

This is the current value of the slide. You can read/write to this property.

uiLineWidth=5;

This is the width of the line of the slider.

uiSelectSize=15; //radius of circle

This is the radius of the select circle, if you are not using a custom select sprite.

uiBorderColor=c_navy; //outline of line around slider and selector

uiBackColor = c_navy; //color of main slider line

uiBackColor2 = c_blue; //gradient of main slider line with uiBackColor

uiHiliteColor = c_white; //outline of selector

uiHiliteColor2 = c_teal; //background of selector

uiSlideSprite = -1; //if using a sprite for slider select

uiLineSprite = -1; //if using a sprite for line.

uiUpdateLabel = noone;

This is a useful feature, you can have the slider update a ui_label widget to the current value of the slider. Will set the textlabel of uiUpdateLabel to uiValue on change of slider.

ui_textani

This is a fun widget that allows you to animate, move, follow-path,type, simple text strings. It may change a lot as new types of animations are added. You need to see the text animation room in the sample project to see the available features. In general you add the ui_textani to the room and set the properties for animation and then set uiAnimated=true which will start the animation.

Here are some sample properties to get you started:

```
uiTextValue="this is Text in Motion";
uiTextColor=c_yellow;
```

```
uiPath=path1;   //you can set to an existing GMS path (see GMStudio paths)
uiPathSpeed=-10;  //play with the value but a negative value will move left.
uiAnimated=true;


uiTextValue="Game Over!";
uiTextColor=c_white;
uiFont=font_art;
uiPathSpeed=0;     //no path to move around
uiFading=0.5;      //50% fading alternating
uiPulse=0.2;       //40% pulse +20/-20 zoom
uiPulseSpeed= 0.05;
uiTextEffect=1;    //this will affect each letter in string
                   0= no character effect
                   1=alternate characters
                   2=alternate words (space sep.)
                   3=typewriter effect.
uiAnimated=true;
```

ui_textbox

The textbox widget is for use with the GameMaker Studio method in getting text from the user. In a mobile environment it will prompt the user using the mobile/native OS keyboard. See sample project under labels room for different examples;

uiTextValue='';

This is the value of the text within the textbox, you can set it yourself as a default.

uiTextLabel='Enter Something';

This is the prompt that will be displayed to user when they press the textbox widget.

uiGetNumber=false; //will prompt for number, instead of string

uiTextLimit = 0; //limits the number of characters user can input

ui_swipe

This is an experimental widget that helps within menu GUIs. It will fire an event when a swipe is performed on the screen (mobile screen swipe). You just have to add a ui_swipe object to the room and set it's uiEventObject. It will fire the user_event on the uiEventObject when it detects a swipe. In the user_event0 code you just need to see which property was found like this:

```
if other.SwipedLeft { do something}
```

```
else if other.SwipedRight {do something}

else if other.SwipedUp {do something}

else if other.SwipedDown {do something}
```

the only option is to set uiSwipeLength = 0.15; (15% of room width) You need to play around with this variable to see if it suits your needs.

Widget Moving Animation

Button moving is where you want to move the button to a different location, but animate that motion (10% per step). To move a button simply set the final positions (uiMoveX,uiMoveY) and then set uiAniMove=true to start the movement. You don't need to do anything else afterwards, the button will endup at the final position. If you look at the sample project menu room you can see a simple 'trick' so that it looks like the button is coming off screen into its final position.

```
uiMoveX=0;
uiMoveY=0;
uiAniMove=true;  //start moving
```

Button Hover

Button hover is what to do with Button Widgets when you mouse over them. You have the option of setting uiHoverMode to determine what to do.

uiHoverMode=1;

0 = None (no hover)
1 = Image blend uiHoverColor;
2 = Zoom, will Expand button
3 = Glow

uiHoverColor=c_dkgray;

Color of the blend when you uiHoverMode=1;

uiHoverInfo=0.8;

This depends on the uiHoverMode, for zoom it is the zoom factor (ex. =1.1), for image blend it is the alpha.

Custom Draw Scripts

This is an advanced section, that you should be comfortable with GML/scripts to accomplish. It is a powerful feature that allows you draw within a cell/line, and have the widget handle scrolling and GUI events.

For listbox widgets, both the horizontal and vertical you can have an advanced method of controlling how to draw the contents of each line. You just simply set uiCustomDrawScript=scr_yourscript; When the custom script is set, the listbox will call your script EACH TIME a line needs to be drawn. So if your listbox has 10 items, your custom script will be called 10 times (it is optimized that it only calls your script for visible lines, but you don't need to worry about that in your script). See the sample project's custom draw scripts to understand what it does. Here's a breakdown of the parameter that are sent to your custom script:

```
curListB    = argument0;        // self.id of calling listbox_ext;
curIndex    = argument1;        // index (0 based) of items of listbox
curX        = argument2;        //X  to draw
curY        = argument3;        //Y  to draw
curHeight   = argument4;        // default height of lineitem (usually textheight,  return must be the
                                        actual height for variable heights)
curWidth    = argument5;        // width of drawing area;
curLineSt   = argument6;        // cur string to display.
curSelected = argument7;        //if line is hilited  true/false
```

The important thing to remember is to draw within the rectangle (curX,curY,curX+curWidth,curY+curHeight). The listbox only contains items[] array so if you need more info to store by line then you would need to store that info into your own arrays. You ONLY DRAW the one line item, you do not draw more than one line in your custom drawscript, curIndex is the current item being drawn (0 = first item in list).

Make sure you set the curHeight as the return variable of your script. The listbox control allows for variable height lines so it needs to know how high the drawscript took. For listbox_horizontal you need to make sure you set curWidth instead of curHeight.

Also, you need to ensure that the items[] array has correct number of elements you expect. Meaning, if you have an external array/ds_list that you want to display, you must first ensure that the items[] array has the same # of elements (they could be dummy values). But the list needs to know how many items it needs to call/draw to your custom script.

Samples

You can use the sample project that will show you how many properties work. Here are just a few easy 'create code' sample to use to get the basics. You can copy and paste them depending on which widget you are using:

Button

```
uiTextValue='Home';
uiTextColor=c_navy;
uiYscale=2.0;
uiXscale=2.0;
uiAlign=fa_center;
uiAlignV=fa_middle;
uiDrawBackColor=false;
uiButtonSprite=spr_home;
uiSpriteAlign=fa_center;
uiSpriteAlignV=fa_middle;
uiSpriteScale=0;
uiDownColor=c_yellow;
```

ListBox

```
items[0]='Standard 1';
items[1]='Notice the';
items[2]='colors';
uiBorderColor=c_navy;
uiBorderBackColor=c_blue;
uiAlign=fa_left;
uiDrawLines=true;
uiDrawHeader=false;
uiDrawSquare=true;
```

Toggle Buttons

```
uiAlign=fa_center;
uiAlignV=fa_middle;
uiDrawBackColor=false;
uiSpriteAlign=fa_center;
uiSpriteAlignV=fa_middle;
uiSpriteScale=0;
uiBackColor=c_white;  //blend with white when down (use sprite)
uiIsToggle=true;
uiHoverMode=0;
uiButtonSprite=spr_volume;
uiSpriteIndex=0;
uiButtonSpriteDown=spr_volume;
uiSpriteIndexDown=1;
```

combo box

```
uiTextValue='Combo Box';
items[0]='Combo Line 1';
items[1]='Combo Line 2';
items[2]='Combo Line 3';
items[3]='Combo Line 4';
uiDrawArrowUp=true;
uiAlpha=1;
uiBackColor=c_orange;
uiTextColor=c_white;
```

Group Buttons

```
uiBackColor=c_gray;
uiDrawBackColor=false;
uiButtonSprite=spr_volume;
uiSpriteIndex=2;
uiSpriteAlign=fa_center;
uiSpriteAlignV=fa_middle;
uiSpriteScale=0;
uiDownColor=c_white;
uiButtonGroup=1;
uiHoverMode=0;
uiButtonState=1;  //default down this button
```

CheckBox

```
uiTextValue='Sound ON';
uiAlign=fa_right;
uiDrawBackColor=false;
uiTextColor=c_white;
```

Horizontal Listbox

```
uiItemWidth=150;  //set the width of each element...
for (fg=0;fg<10;fg+=1)
{   items[fg]='Wide List '+string(fg);
   sprites[fg]=choose(spr_home,spr_anib);
 }
uiTextLabel='Horz with center snap';
uiAlign=fa_center;
uiButtonList=true;
uiShowArrows=true;
uiDrawHeader=false;
//uiCustomDrawScript=scr_listbox_drawline;
uiDrawBackColor=false;

uiSnapLeft=true;    //will snap the left item to the left of the
widget (no left side clipping)....
uiLeftMargin=75;
//uiAutoSelect=true; //will auto hilite the center item
```

CheckGroup

```
items[0]='Knife';
items[1]='Sword';
items[2]='Club';
items[3]='Cannon';
items[4]='Gun';
uiTextLabel='Select Many';
uiMultiSelect=true;
itemschecked[0]=false;
itemschecked[1]=false;
itemschecked[2]=false;
itemschecked[3]=false;
itemschecked[4]=false;
```

Gauge

```
uiAniSpeed=3;  //fast animation
uiBarLevel=50;  //goes yellow (level2) at 50%
uiBarLevel2=25;  //goes red at 25%
uiDrawBackColor=false;  //transparent;
```

Gauge

```
uiDrawSquare=true;  //square
uiAniDraw=false;   //no animation
uiBackColor=c_black;
uiBorderColor=c_gray;
uiBarFlash=0;  //no flash on dying
uiBarColor=c_teal;
uiBarColor2=c_navy;
uiBarColor3=c_yellow;
```

TextAnimation

```
uiTextValue="Game Over!";
uiTextColor=c_white;
uiFont=font_art;
//uiPath=path1;
uiPathSpeed=0;
uiAlign=fa_center;
//uiXscale=1;
uiWobble=15;
//uiPulse=0.5;
//uiPulseSpeed=0.1;
uiTextEffect=1;
uiAnimated=true;
```

Custom Listbox

```
uiTextLabel='Custom Scroll List';
uiAlign=fa_left;
uiButtonList=true;
uiShowArrows=true;
uiDrawHeader=false;
uiCustomDrawScript=scr_listbox_drawline;
```

Label

```
uiTextValue='Right with Back';
uiAlign=fa_right;
uiBackColor=c_navy;
uiTextColor=c_white;
uiDrawBackColor=true;
```

Slider

```
uiMin=1;
uiMax=100;
uiInterval=1;
uiUpdateLabel=slide_inst;
```

TextBox

```
uiTextValue='Dogs are fun';
uiTextLabel='Edit it if you want';
uiAlign=fa_center;
```

Edit Box

```
uiTextColor=c_white;
uiBackColor=c_navy;
uiFocusColor=c_gray;
uiFocusTextColor=c_yellow;
uiCursorColor=c_red;
uiTabOrder=2;
```

End of Document