Overview

The GUI system in Frog displays a stack of layers, and each layer is a hierarchy of widgets. Widgets are things like buttons, labels, and sliders. In most cases, you only need to display one layer at a time. For example, a simple main menu would be able to show everything it needs in a single hierarchy of widgets. If you needed to show an error message above that though, you would likely want to make it a separate layer. By default, the topmost layer temporarily disables everything beneath it. This is useful in cases like warning messages, where you need to click "OK" before you can interact with the rest of the GUI. This behavior can be overridden though, so you can build complex interfaces with major components specified in separate layers.

Each GUI layer for a given project is specified in its own subfolder of "Graphics\GUI", and the name of the folder is used to refer to the layer. This folder must contain a Widgets.json file to specify the tree of widgets, and there is typically also a sprite resource file named Sprites.json. If you will be scripting the behavior of the layer in Lua rather than handling it in C++, there would additionally be a Scripts.lua file. Bitmaps needed for the sprites of a layer typically go in this folder as well.

Depending on the platform and settings, the GUI system can expect either a single pointing device (single mouse, single-touch, etc.) or multiple pointers (multi-touch, Wii remotes, multiple mice, etc.)

To add or remove GUI layers, use methods like GUIManager::Push, GUIManager::Pop, and GUIManager::Remove. If you would like to preload all the sprite animations for a given layer, use GUIManager::Preload. To allow preloaded data to be released, use GUIManager::UndoPreload.

To refer to an individual widget, you can use its "path", which is a '.'-delimited string that reflects the widget's position in the hierarchy. For example, if you had a container widget named "MyContainer" in a layer named "MyLayer", its path would be "MyLayer.MyContainer". If a button named "Example" was a child of that container, its path would be "MyLayer.MyContainer.Example". Not all widgets can be referenced by path. For example, there is no path for the scrollbar of a ScrollBox widget.

There are many features in the GUI system that involve scripting, but beware that all of them are disabled by default. For example, the TextExpression property of a Label widget does not work by default. Check whether your project supports these features before attempting to use them.

Event Handling

Different actions performed by the user generate events which can be handled in C++ or Lua. You can register for events in C++ using methods like PressButtonWidget::OnClickRegister. This involves providing a function pointer to the handler as well as an optional void* to provide context when the function is called. Only one function pointer may be registered at one time for a given event of a given widget. To handle events in Lua, define a function with the appropriate path and name in the associated layer's Scripts.lua file. For example, if you had a button named "Example" in a layer named "MyLayer", and you wanted a script for when it is clicked, you would need something along the lines of the following…

```
function GUI.MyLayer.Example.OnClick()
   -- Handle the click
end
```

Widget Specifications

The details of the widgets for a particular layer are given in that layer's Widgets.json file. The following example would simply show the string "Hello World!" in center of the window, assuming it is 1024×768.

```
{
   "Children":
   [
      {
         "Type": "Label",
         "Font": "Arial",
         "TextBounds": "0|0|1024|768",
         "TextAlignment": "Center",
         "Text": "Hello World!"
      }
   ]
}
```

Ordinarily, the text localization system would determine what string is displayed, but for purposes of simplicity, the "Text" property is used to show the given string as-is. This also assumes that the font "Arial" is loaded.

A layer is a type of container widget. In addition to other properties, a container widget can have an array of children. In this case, there is only one: the label. All widgets specified in an array of Children must provide a "Type". In other cases, such as the layer itself, the type is implicit. There is a set of widget types built into the GUI system, but individual games can register their own if needed.

Any widget definition can include a "Font", even if the widget cannot display text. If no font was explicitly specified for a given widget, it will ask its parent which one to use. In the "Hello World" example, the "Font" line could be moved out to the layer itself, and the label would default to using "Arial".

When a two-dimensional vector needs to be specified, express it as a string using the format "x|y". When a rectangle is needed, use "x|y|width|height". The format for a color is "red|green|blue|alpha" where each is an integer between 0 and 255 (inclusive).

Common

There are many types of widgets, but most share a common set of features that can be configured in their specifications.

- AncestorClippingIgnore : True if the clip areas of ancestors should not clip this widget or its descendants. (Default: false)
- Bounds : Region, relative to the widget's origin, associated with the widget for pointing device purposes. This is used for determining which widget the cursor is over.
- BoundsColor : Color to use when drawing an outline for the widget's bounds. (Default: "255|255|255|255" (opaque white))
- BoundsDraw : True if an outline should be drawn for the widget's bounds. This can be used in addition to BoundsFilledDraw. (Default: false)
- BoundsFilledColor : Color to use when drawing a filled-in rectangle for the widget's bounds. (Default: "58|58|58|255" (dark grey))
- BoundsFilledDraw : True if a filled-in rectangle should be drawn for the widget's bounds. This can be used in addition to BoundsDraw. (Default: false)
- CanChangeFocus : True if this widget can change the keyboard focus. This does not often need to be set. See Keyboard Focus. (Default: true)
- CanDrag : True if the widget can be dragged using a pointing device. (Default: false)
- CanHaveFocus : True if this widget can have the keyboard focus. This does not often need to be set. See Keyboard Focus. (Default: depends on the type of widget, but false for most types)
- ClickSound : Sound to play when a cursor clicks this widget while it is enabled and visible. See Sounds for more. (Default: defer to parent)
- ClickSoundEnabled : True if this widget should play a Click sound. See Sounds for more. (Default: depends on widget type)
- ClipArea : Drawing of this widget and its descendants is restricted to this region, relative to the parent. If there is no parent, it is relative to (0, 0). The final clip area used when drawing a given widget is the intersection of this clip area and those of all its ancestors. (Default: extra additional restrictions)
- Depth: The absolute stereoscopic depth position of the widget's origin. The absolute depth must be in the range [-1, 1], where positive depth goes into the screen and 0 is the depth of the screen itself. This is mutually exclusive with DepthOffset. (Default: same depth as parent. If there is no parent, use 0.)
- DepthOffset : The stereoscopic depth position of the widget's origin relative to its parent's origin. The absolute depth must be in the range [-1, 1], where positive depth goes into the screen and 0 is the depth of the screen itself. If it has no parent, this would be relative to 0. This is mutually exclusive with Depth. (Default: same depth as parent. If there is no parent, use 0.)
- DragArea : Region within which the widget can be dragged. This is relative to the parent widget. (Default: bounds of the parent. If there is no parent, use the bounds of the screen.)
- DragBufferArea : Region relative to the widget's origin that is kept within the drag area. For example, if this was "0|0|32|0" and the widget was being dragged to the right, it would stop moving farther to the right once its origin was within 32 pixels of the right side of the drag area. (Default: bounds of the widget)
- Enabled : Whether the user can initially interact with the widget. Some widgets have a different appearance when they are disabled. (Default: true)
- Font : Name of the font to use for this widget. If no font has been specified for a widget, it will ask its parent which to use. That allows you set a default font for a tree of widgets. (Default: defer to parent)
- HoldStartDelay : If the user presses and holds down the widget using a pointing device, the widget's OnHold event will be called repeatedly. This parameter gives the number of milliseconds after pressing before this cycle of OnHold calls begins. (Default: 500 ms)
- HoldIntervalDuration : If the user presses and holds down the widget using a pointing device, the widget's OnHold event will be called repeatedly. This parameter gives the number of milliseconds between calls to OnHold. (Default: 50 ms)
- InvalidPressSound : Sound to play when a cursor is pressed over this widget while it is disabled and visible. See Sounds for more. (Default: defer to parent)
- InvalidPressSoundEnabled : True if this widget should play an InvalidPress sound. See Sounds for more. (Default: depends on widget type)
- MouseOverSound : Sound to play when a cursor moves over this widget while it is enabled and visible. See Sounds for more. (Default: defer to parent)
- MouseOverSoundEnabled : True if this widget should play a MouseOver sound. See Sounds for more. (Default: depends on widget type)
- MouseScrollCoefficient : Coefficient used to modify the influence of mouse scrolling. (Default: 1.0)
- MouseScrollEnabled : True if mouse scrolling that happens over this widget should be processed. This can be used to

prevent mouse scrolling over a widget, like a “next” button, from affecting a parent, like a ScrollBox. (Default: true)

- Name : When a widget is specified in an array of children, it can be given a name to distinguish it from its siblings. If a name is not specified in this context, the default name is the widget's type. When a widget is created outside an array of children, the name depends on the context and cannot be specified. Names must not contain spaces.
- PointTestEnabled : False if this widget and its descendants should be ignored when testing whether a given point, like a mouse cursor, is over them. (Default: true)
- PointTestPerPixel : True if this widget should perform per-pixel tests when checking whether a given point, like a mouse cursor, is over it. In addition to the per-pixel test, it will still need to pass the Bounds point test. For now, this does not check text or vector drawing, like BoundsDraw or BoundsFilledDraw. It does not necessarily work with all underlying bitmap formats. On some platforms, like OpenGL, the underlying sprite animations may need to be kept in memory with 'KeepBitmapData'. (Default: false)
- Position : The absolute position of the widget's origin in screen coordinates. This is mutually exclusive with PositionOffset. (Default: same position as parent. If there is no parent, use (0, 0))
- PositionOffset : The position of the widget's origin relative to its parent's origin. If it has no parent, this would be relative to (0, 0). This is mutually exclusive with Position. (Default: same position as parent. If there is no parent, use (0, 0))
- PressSound : Sound to play when a cursor is pressed over this widget while it is enabled and visible. See Sounds for more. (Default: defer to parent)
- PressSoundEnabled : True if this widget should play a Press sound. See Sounds for more. (Default: depends on widget type)
- SpriteResourceFilename : Filename of the sprite resource file to use for this widget and its descendants. (Default: defer to parent. Layer widgets have a default associated Sprites.json.)
- Transitions : Specifies the transition effects for this widget. See Transitions. (Default: no transitions)
- Type : When a widget is specified in an array of children, this is required to specify what type of widget it is: (Container, Label, PressButton, etc.) When a widget is specified outside an array of children, the type is implicit and based on the context.
- Visible : Whether the widget is drawn. (Default: true)

Container

Container widgets can have an array of child widgets that are updated and normally drawn in-order from first to last.

- Children : JSON array that specifies the descendants of the container. (Default: no children)
- ChildrenDrawOrderReverse : True if the children should be drawn from last to first, rather than first to last. This does not affect the order in which descendants draw their children. (Default: false)

Layer

A Layer widget is a special type of Container, as explained in the Overview.

- DisableLayersBelow : Normally, a GUI layer temporarily disables all layers below it. If you don't want this particular layer to do that, set this property to false. (Default: true)

The following event handlers can be defined for this widget type in Lua.

- OnInit() : Called immediately after the layer has been created.
- OnDeinit() : Called immediately before the layer is removed.
- OnMouseUp() : Called when the main button of a pointing device is released.

Sprite

The Sprite widget is basically a wrapper for inserting a single child sprite into a GUI layer.

- Animation : The name of the animation to be displayed.
- ClippingAffectsDrawing : True if widget clipping should be applied when drawing. (Default: true)

Label

Label widgets display a single line of text. The contents are clipped and positioned based on the TextBounds.

Labels have an optional feature for making the text bounce from left to right if it does not fit horizontally within the text bounds. It can be made to move at a given speed and pause for a given amount of time at the ends. If the text fits within the text bounds, the bounce effect does not occur.

- Text : Display the given string. This is mutually exclusive with TextKey and TextExpression. (Default: no text)
- TextAlignment : How the text should be positioned relative to the text bounds. For example, if the TextAlignment is TopLeft, the top-left corner of the text will be in the top-left corner of the text bounds. The possible values are

TopLeft, Top, TopRight, Left, Center, Right, BottomLeft, Bottom, and BottomRight. (Default: TopLeft)

- TextAscentDescentMaxUse : Text can be positioned vertically based on either the height of the specific string to display or on the height of the font as a whole. In this case, the height of the font is the maximum ascent plus the maximum descent across all characters in the font. If you want the vertical position to be the same, regardless of the specific string, use true. Otherwise, use false. (Default: false)
- TextBounceX : True if the bounce effect should be used. This is mutually exclusive with TextScaleDownToFit, so be sure to turn that off if you use this. (Default: false)
- TextBounceXPauseDuration : When the bouncing effect is in use and the text has reached either the far left or far right, wait for this many milliseconds before moving again. (Default: 2000 ms)
- TextBounceXSpeed : When the bouncing effect is moving the text, move it at this many pixels per second. (Default: 50 pixels per second)
- TextBounds : Region relative to the widget's origin used for positioning and clipping of the text. This clipping is cumulative with normal widget clipping. If no TextBounds are specified, the regular bounds of the widget are used for this purpose.
- TextBoundsColor : Color to use when drawing the widget's text bounds. (Default: "255|0|255|255" (opaque magenta))
- TextBoundsDraw : True if the widget's text bounds should be drawn. (Default: false)
- TextColor : Color to apply when drawing the text. This behaves as a filter, so you can go from light to dark, but not from dark to light. (Default: "255|255|255|255" (opaque white))
- TextExpression : Evaluate the given string as a Lua expression and display the resulting text. This is mutually exclusive with Text and TextKey. (Default: no text)
- TextKey : Use the given string to get the text to display from the localized text system. This is mutually exclusive with Text and TextExpression. (Default: no text)
- TextScale : Scale used to adjust the size of the text. For example, to draw the text half as wide and twice as tall, use "0.5|2". (Default: "1|1")
- TextScaleDownToFit : Automatically scale down text to fit within the text bounds. This is mutually exclusive with TextBounceX. (Default: true)

NumberLabel

A NumberLabel is specialized Label for displaying numbers. Rather than setting Text, TextKey, or TextExpression, users of this widget should specify a floating-point value.

- TextFormatExpression : Lua expression for a function that returns a string to display for the current value. The current value is passed to the function as its only parameter. For example, if you had a Lua function named ScoreString for creating the string, you would set this to "ScoreString". (Default: value is displayed as a floating-point number)
- Value : The floating-point number to be displayed. (Default: 0)

IncrementalNumberLabel

An IncrementalNumberLabel is an extension of NumberLabel. It includes functionality to change its value over a set time to reach a target value. While doing so, the widget can scale itself over time to create a pulsing effect.

- DifferenceDecaySpeed : Controls the exponential side of the speed at which the displayed value approaches the target. This must be greater than or equal to 0 and less than or equal to 1. The distance to the target value decays exponentially based on this number, so that the decay slows to a stop. (Default: 0.0)
- IncrementRate : How much to change value per ms. (Default: 1.0)
- IncrementSound : The sound to play while the value is changing.
- Pulse : True or false if the label should pulse while a change is occurring. (Default: true)
- ScaleRate : How much to scale per ms. (Default: 1.0)
- ScaleOffsetMax : The max scale the pulse reaches before scaling back down (Default 1.3)

Buttons

The two main types of buttons in Frog are PressButton and ToggleButton. The most important difference is that a ToggleButton has a flag that is toggled on and off when the button is clicked. This is separate from the button being enabled or disabled. All of the parameters and defaults for Labels also apply to buttons. Note that a ToggleButton is `'not'` the same thing as a RadioButton.

A button can be made to look different depending on whether it is enabled, whether the cursor is over it, and whether it is being pressed. A ToggleButton can additionally be made to look different depending on whether it is currently toggled on or off. The names of the possible states depend on the type of button. The states for a PressButton are named…

- UpOutState : The cursor is not over the button.
- UpOverState : The cursor is over the button, but not pressed. Think of UpOver as `'hover'`.
- DownOverState : The cursor is over the button and pressed.

- DownOutState : The cursor was pressed over the button earlier, but it is not currently over it and it has not yet been released. This can typically be the same as the DownOverState.
- DisabledState : The button is not currently enabled.

The states for a ToggleButton are named…

- OnUpOutState : (Toggled on) The cursor is not over the button.
- OnUpOverState : (Toggled on) The cursor is over the button, but not pressed.
- OnDownOverState : (Toggled on) The cursor is over the button and pressed.
- OnDownOutState : (Toggled on) The cursor was pressed over the button earlier, but it is not currently over it and it has not yet been released.
- OnDisabledState : (Toggled on) The button is not currently enabled.
- OffUpOutState : (Toggled off) The cursor is not over the button.
- OffUpOverState : (Toggled off) The cursor is over the button, but not pressed.
- OffDownOverState : (Toggled off) The cursor is over the button and pressed.
- OffDownOutState : (Toggled off) The cursor was pressed over the button earlier, but it is not currently over it and it has not yet been released.
- OffDisabledState : (Toggled off) The button is not currently enabled.

None of these states are required if you don't want a distinct appearance for a given state. Depending on the current state of the button and what states are actually defined, the widget may fall back on settings given for a different state. UpOut doesn't fall back on anything. UpOver falls back on UpOut. DownOver falls back on UpOver, then UpOut. DownOut falls back on DownOver, then UpOut. Disabled falls back on UpOut. For example, if a sprite is given for an UpOver state, but not for an UpOut state, no sprite will be displayed when the button is in the UpOut state. However, in the opposite situation, where a sprite is given for an UpOut state and no sprite is given for an UpOver state, the same sprite will be displayed in both states.

A normal state definition is given as a JSON object. In addition to the usual parameters for widget child sprites, the following parameters may also be specified.

- BoundsFilledColor : Color to use when drawing a filled-in rectangle for the button's bounds during this state, rather than the BoundsFilledColor specified for the button. (Default: defer to parent)
- Font : Name of the font to use in this state, rather than the one specified by the button or its ancestors. (Default: defer to parent)
- TextColor : Color to apply when drawing the text during this state, rather than using the TextColor specified by the button. This behaves as a filter, so you can go from light to dark, but not from dark to light. (Default: "255|255|255|255" (opaque white))
- TextPositionOffset : Shift the position of the text bounds by this much while this state is active. This may also be defined at the button level, setting the TextPositionOffset for all child sprites associated with the button. (Default: no additional shift)
- TextScale : Scale used to adjust the size of the text during this state, rather than the TextScale specified by the button. For example, to draw the text half as wide and twice as tall, use "0.5|2". (Default: defer to parent)

There is a shortcut to specifying a state if the only item would be a sprite animation. In that case, rather than defining a JSON object for the state, just give the name of the animation.

In addition to the other parameters, a ToggleButton also accepts the following.

- On : True if the button should initially be toggled on. (Default: false)
- ToggleOnClick : True if the button should toggle on and off when clicked. (Default: true)

The following event handlers can be defined for a PressButton in Lua.

- OnClick() : Called when a pointing device is pressed and then released over this widget.
- OnPress() : Called when a pointing deviced presses down on this widget.

The following event handler can be defined for a ToggleButton in Lua.

- OnClick(toggledOn) : Called when a pointing device is pressed and then released over this widget. If this button is now toggled on, 'toggledOn' will be true.

RadioButton

A RadioButton is a type of ToggleButton that can be a member of a group where only one member is allowed to be toggled on. Unlike a normal ToggleButton, a RadioButton does not toggle off when you click on it. When one is toggled on, the others in the group toggle off automatically. There are different ways to establish a group. One option is to set the RadioButtonGroupName property to the same string for all the RadioButtons you would like to group. The scope of these names is limited to the GUI layer, so if you have two radio button in different layers with the same RadioButtonGroupName, they won't actually be part of the same group.

The other way to establish a group is by making the RadioButtons descendants of a common RadioButtonContainer widget.

RadioButtons that are descendants of that RadioButtonContainer that don't already have a RadioButtonGroupName will be mutually exclusive. However, this may not work as expected if the RadioButtons are also within another RadioButtonContainer, so don't let your RadioButtonContainers overlap.

- RadioButtonGroupName : All RadioButtons on a given layer that use the same string for this parameter are mutually exclusive. (Default: group not defined)

RadioButtonContainer

A RadioButtonContainer is one way of grouping radio buttons to make them mutually exclusive. See the RadioButton section for more.

Slider

A Slider widget is used to adjust a number. It can be a stand-alone widget, like a volume slider, or part of another widget, like a scrollbar of a ScrollBox widget.

- BackgroundSprite : Optional child sprite to be displayed behind the other parts of this widget. (Default: no background sprite)
- IncreaseButton : A button for increasing the value. Whenever its OnPress or OnHold events occur, the value will increase by the amount specified in SmallStep. (Default: no increase button)
- DecreaseButton : A button for decreasing the value. Whenever its OnPress or OnHold events occur, the value will decrease by the amount specified in SmallStep. (Default: no decrase button)
- Thumb : A button that can be dragged to adjust the value. Do not specify the Position, PositionOffset, CanDrag, or DragArea parameters of the Thumb, because those details are handled automatically based on other parameters. However, you will need to specify positions for representing the minimum value and the maximum value. You can set the position for the minimum value using either PositionMinimum or PositionMinimumOffset, and you can set the position for the maximum value using either PositionMaximum or PositionMaximumOffset. These four parameters work just like the usual Position and PositionOffset parameters for widgets. The slider should only be vertical or horizontal, so the positions should form either a horizontal line or a vertical line.
- Maximum : Maximum value that this slider can represent. (Default: 100)
- Minimum : Minimum value that this slider can represent. (Default: 0)
- StepMinimum : The value for this slider must be a multiple of this number, plus the minimum value. This allows you to control the granularity of the possible values. For example, if Minimum was 0, Maximum was 3, and StepMinimum was 1, then the only values this slider could have would be 0, 1, 2, and 3. If Minimum was 0.5 and StemMinimum was 1.0, then the slider could have the values 0.5 or 1.5, but not 1.0. (Default: 1)
- StepSmall : The amount by which the value is incremented or decremented when pressing or holding the increase or decrease buttons. (Default: 1)
- StepLarge : The amount by which the value is incremented or decremented when pressing or holding in the area between the maximum and minimum thumb positions. (Default: 10
- Value : The initial value for the slider. (Default: 0)

The following event handler can be defined for this type of widget in Lua.

- OnValueChange(newValue, oldValue, userMadeChange) : Called when the number associated with the slider changes. 'newValue' is the new number for the slider, and 'oldValue' was the number for the slider immediately before the event. If the change was triggered directly by the user interacting with the widget, 'userMadeChange' will be true. However, if the change was made from C++ or Lua, it will be false.

ScrollBox

A ScrollBox widget is a way to have a nested group of widgets that are clipped to a scrollable rectangle. Horizontal and vertical scrollbars may be added to adjust which part of the nested group is visible in the clipping area.

When referring to widgets within the ClipContainer of a ScrollBox, the path should treat those widgets as children of the ScrollBox itself. For example, if you had a PressButton named “ExampleButton” in the ClipContainer of a ScrollBox named “MyScrollBox”, the path to that button would be something along the lines of “MyLayer.MyScrollBox.ExampleButton”. There is no path for referring to the ClipContainer itself or the scrollbars since this should not be needed.

- ClipContainer : This is a type of Container, and its Children are the nested group of widgets for the ScrollBox. The ClipArea of the ClipContainer is the rectangle through which these descendants can be viewed. The Bounds of the ClipContainer specify the extent of the nested region that can be viewed by scrolling up, down, left, and right.
- BackgroundSprite : Optional child sprite that is drawn behind the other parts of the ScrollBox. (Default: no background sprite)
- VerticalScrollbar : Slider widget for scrolling up and down. The increase button scrolls down and the decrease button scrolls up. If the height of the ClipContainer's bounds are less than or equal to the height of the ClipContainer's ClipArea, this scrollbar will not be displayed. (Default: no scrollbar for scrolling up and down)

- HorizontalScrollbar : Slider widget for scrolling left and right. The increase button scrolls right and the decrease button scrolls left. If the width of the ClipContainer's bounds are less than or equal to the width of the ClipContainer's ClipArea, this scrollbar will not be displayed. (Default: no scrollbar for scrolling left and right)

DropDownList

A DropDownList widget is a way to have one item selected from a scrollable, collapsible list of options.

- ItemsOverOpenButton : True if items should be drawn above the OpenButton if they overlap. (Default: false)
- ItemTemplate : All items in the list are displayed using ToggleButtons, and ItemButton serves as a template for creating them. The button for a given item is toggled on when it is highlighted and off when it is not highlighted. Be aware that the keyboard can also be used to adjust the selection, so it is typically good to only specify the OnUpOutState and the OffUpOutState.
- OpenButton : ToggleButton that opens and closes the list. The text of the button is automatically set to that of the currently selected item.
- ScrollBox : ScrollBox that contains widgets that are created automatically to represent the list's options. Do not specify the Bounds or Children of the ScrollBox's ClipContainer, because this is determined automatically. Do not specify a HorizontalScrollbar either. The ClipArea of the ClipContainer must still be specified.

The following event handlers can be defined for this widget type in Lua.

- OnSelection(newSelectionName, oldSelectionName, userMadeSelection) : Called when the selection is set, regardless of whether it actually changed. 'newSelectionName' and 'oldSelectionName' specify the current selection as well as the selection immediately before the event. If the selection was triggered by the user directly, 'userMadeSelection' will be true. However, if the selection was set explicitly from C++ or Lua, it will be false.

- OnSelectionChange(newSelectionName, oldSelectionName, userMadeSelection) : OnSelectionChange is exactly the same as OnSelection, except OnSelectionChange is only called if the new selection is different from the old selection.

TextDocument

A TextDocument widget is a type of ScrollBox for displaying rich text. Content is automatically broken up into multiple lines based on newline characters and the width of the ClipContainer's ClipArea. Do not specify the Bounds of the ClipContainer, because this is determined automatically. Horizontal scrollbars are not necessary.

The text can include markup to insert sprites or modify the appearance of a section of text, and the syntax is similar to html. To use an alternate font, use "<Font=name>text to draw in that font</Font>". This can be used multiple times, but multiple Font tags cannot be nested. Any font referenced in this way must already be loaded. You can embed sprite animations from the GUI layer's sprite file with <Sprite=animationName>. Sprites will be displayed in-line with the sprite's origin on the baseline of the neighboring text. To change the scale of the text, use ”<TextScale=x|y>text to be drawn at a different scale</TextScale>”. This can be used multiple times, but multiple TextScale tags cannot be nested. To change the color of the text, use ”<TextColor=red|green|blue|alpha>text to be drawn in a different color</TextColor>”. This can be used multiple times, but multiple TextColor tags cannot be nested. All tags are case-sensitive.

- Text : Display the given string. This is mutually exclusive with TextKey, TextFilename, and TextExpression. (Default: no text)
- TextAlignment : How the text should be positioned within the ClipContainer. The vertical aspect only applies when the content does not vertically fill the clip area. The possible values are TopLeft, Top, TopRight, Left, Center, Right, BottomLeft, Bottom, and BottomRight. (Default: TopLeft)
- TextColor : Color to apply when drawing the text. This behaves as a filter, so you can go from light to dark, but not from dark to light. (Default: “255|255|255|255” (opaque white))
- TextExpression : Evaluate the given string as a Lua expression and display the resulting text. This is mutually exclusive with Text, TextKey, and TextFilename. (Default: no text)
- TextFilename : Display the contents of the given file. The filename should be relative to the “Text/[Language]” folder, and it should not include an extension. For example, if the current language is English, and the true filename is “Text\English\Help.txt”, you should simply specify “Help”. This is mutually exclusive with Text, TextKey, and TextExpression. (Default: no text)
- TextKey : Use the given string to get the text to display from the localized text system. This is mutually exclusive with Text, TextFilename, and TextExpression. (Default: no text)
- TextScale : Scale used to adjust the size of the text. For example, to draw the text twice as tall and half as wide, use “0.5|2”. (Default: “1|1”)

ProgressBar

A ProgressBar widget is non-mathmatical way to display a fraction. The fraction is determined by comparing the current value of the progress bar with the maximum and minimum values. The bar is filled from left to right by changing how much of the “ProgressSprite” is being clipped.

There is also an “indeterminate” mode for when you don't know what the fraction is. For example, if you're waiting for something, and you don't know how long it will take, use indeterminate mode. Rather than filling from left to right, the progress sprite is clipped so that the clipping rectangle appears to scroll from left to right. For example, you could make a graphic that looks like a row of lights where only one light is visible at a time. The PixelStep property still applies in this mode.

- BackgroundSprite : Optional child sprite that is drawn behind the other parts of this widget. (Default: no background sprite)
- ProgressSprite : Child sprite that is clipped in different ways to express both progress and the indeterminate effect.
- Maximum : Value that corresponds to full. (Default: 100)
- Minimum : Value that corresponds to empty. (Default: 0)
- StepMinimum : The value for this widget must be a multiple of this number, plus the minimum value. This allows you to control the granularity of the possible values. For example, if Minimum was 0, Maximum was 3, and StepMinimum was 1, then the only values this widget could have would be 0, 1, 2, and 3. If Minimum was 0.5 and StemMinimum was 1.0, the value could be 0.5 or 1.5, but not 1.0. (Default: 1)
- Value : Initial value for the widget. (Default: 0)
- PixelStep : The minimum increment by which the progress sprite is clipped. For example, if the graphic is broken up into sections that are 10 pixels wide, you would probably want to set this to 10. (Default: 1)
- Indeterminate : True if the progress bar should initially be in indeterminate mode. (Default: false)
- IndeterminateEffectBarWidth : Maximum horizontal amount of the progress bar sprite that may be visible at any one time while in indeterminate mode. (Default: 16)
- IndeterminateEffectDuration : Duration of the loop for the indeterminate mode effect in milliseconds. (Default: 1000)

StateContainer

A StateContainer is a type of Container where only one child is visible at a time. This could be used to create the appearance of a button that cycle through more than the two states offered by a ToggleButton. It could also be useful in creating a tabbed interface.

- State : Name of the child that should initially be visible. All other children will be hidden. (Default: first child in Children list)

The following event handler can be defined for this type of widget in Lua.

- OnStateChange(newStateName, oldStateName) : Called when the state of the widget changes. 'newStateName' is the name of the new current state, and 'oldStateName' is the name of the state that was current immediately before the event.

TextEntry

A TextEntry widget is a type of label that allows the user to enter text using their keyboard as long as the widget has the keyboard focus.

- AllCharactersAllowed : True if there are no restrictions on which characters can be entered. This is mutually exclusive with CharactersAllowed. (Default: false)
- CharactersAllowed : Whitelist of characters that can be entered. This is mutually exclusive with AllCharactersAllowed. (Default: ' ', 0-9, a-z, and A-Z)
- Password : True if the displayed characters should be replaced with '*' characters. (Default: false)
- TextLengthMax : Maximum allowed number of characters in the widget's text. This excludes the null-terminator. (Default: 64)

The following event handler can be defined for this type of widget in Lua.

- OnEnterPress() : Called when the widget receives a newline character from the keyboard while widget has the keyboard focus.

Selectors

Selectors are a type of widget that allow you to choose an item from a list. Much like buttons, you cannot directly create an item of type “Selector”, but you can create widgets like RingSelector, which is a specific type of Selector widget. Only one item can be selected at a time, and the selection can be shifted using a pair of buttons. If the items require explanations, they can be set as the detail text for each item and displayed using the optional DetailTextDocument.

- Items : JSON array containing a set of items to be added to the list automatically. See Selector Item for the format of these entries. (Default: empty)
- ItemTemplate : When an item is created, this JSON object is used to initialize it. It accepts the same properties as a Label. This should only actually include details common to all items in the selector.
- NextButton : PressButton that moves the selection to the next item when pressed or held.

- PreviousButton : PressButton that moves the selection to the previous item when pressed or held.
- DetailTextDocument : Optional TextDocument that will automatically display the detail text of the currently selected item. (Default: no detail TextDocument)
- WrapAround : True if the selection can wrap around to the other end of the list when attempting to go past either end. (Default: true)

The following event handler can be defined for this widget type in Lua.

- OnSelectionChange(newSelectionName, oldSelectionName, userMadeSelection) : Called when the selection is changed. 'newSelectionName' and 'oldSelectionName' specify the current selection as well as the selection immediately before the event. If the selection was triggered by the user directly, 'userMadeSelection' will be true. However, if the selection was set explicitly from C++ or Lua, it will be false.

Selector Item

Items for Selector widgets are based on Labels. The details of the individual items of a list can be provided as a JSON array of objects. Each object can have the following properties.

- Name : The name of the item.
- Animation : Name of the sprite animation to show for this item. (Default: no sprite)
- Text : Display the given string. This is mutually exclusive with TextKey and TextExpression. (Default: no text)
- TextExpression : Evaluate the given string as a Lua expression and display the resulting text. This is mutually exclusive with Text and TextKey. (Default: no text)
- TextKey : Use the given string to get the text to display from the localized text system. This is mutually exclusive with Text and TextExpression. (Default: no text)
- DetailText : Display the given string as the detail text when this item is selected. This is mutually exclusive with DetailTextKey, DetailTextFilename, and DetailTextExpression. (Default: no text)
- DetailTextExpression : Evaluate the given string as a Lua expression and display the resulting text as the detail text when this item is selected. This is mutually exclusive with DetailText, DetailTextKey, and DetailTextFilename. (Default: no text)
- DetailTextFilename : Display the contents of the given file as the detail text when this item is selected. The filename should be relative to the “Text/[Language]” folder, and it should not include an extension. For example, if the current language is English, and the true filename is “Text\English\Help.txt”, you should simply specify “Help”. This is mutually exclusive with DetailText, DetailTextKey, and DetailTextExpression. (Default: no text)
- DetailTextKey : Use the given string to get the text from the localized text system to display as the detail text when this item is selected. This is mutually exclusive with DetailText, DetailTextFilename, and DetailTextExpression. (Default: no text)

LinearSelector

A LinearSelector is a type of Selector that displays its items in a straight line with the selected item at a specified location. Items wrap around before and after the selected item. This way, the selected item is at the center, even when the selected item is at the beginning or end of a list.

- ItemSelectedPositionOffset : Position of the selected item relative to the selector widget. (Default: “0|0”)
- ItemNextPositionOffset : Difference in position between an item and the item before it. (Default: “0|0”)
- ItemCountHorizon : Maximum number of items to show in each direction. For example, if this was 2 and your list had 10 items, only 5 would be shown. There would be the selected item, 2 items before it, and 2 items after it. (Default: 0)

RingSelector

A RingSelector is a type of Selector that displays its items in a ring with the selected item at the bottom/front. The items are sorted so that the ones at the top of the ring are drawn behind those lower on the ring. They can also be scaled and colored based on this to create the illusion of perspective. When the selection is shifted, the newly selected item does not simply snap into position at the bottom of the ring. The items appear to move in a circle and slow to a stop as the selected item approaches the bottom. You can also optionally have each item drawn again to give the illusion of a reflection.

- RingBounds : This rectangle specifies the extents of the ring. For example, the left-most point on the ring will be at the center of the left edge of the RingBounds. The items will be positioned along this ring, but they will not be clipped by these bounds.
- ItemFrontColor : Color of items at the very front (bottom) of the ring. (Default: “255|255|255|255” (opaque white))
- ItemFrontScale : Scale of items at the very front (bottom) of the ring. (Default: “1|1”)
- ItemBackColor : Color of items at the very back (top) of the ring. (Default: “255|255|255|255” (opaque white))
- ItemBackScale : Scale of items at the very back (top) of the ring. (Default: “1|1”)
- ItemReflectionVisible : True if the item reflections should be shown. (Default: false)
- ItemReflectionScale : Scale multiplied by the scale of an item to get the reflection scale. (Default: “1|-1”)

- ItemReflectionColor : Color modulated with the color of an item to get the reflection color. (Default: “255|255|255|255” (opaque white))
- RingRotationSpeed : Controls the speed at which the selected item approaches the bottom of the ring. This must be greater than 0 and less than or equal to 1. The distance to the final rotation decays exponentially based on this number, so that the rotation slows to a stop. (Default 0.97)

ItemListSelector

An ItemListSelector is a type of simple Selector that displays its items either vertically or horizontally next to one another. The items can be spaced by a vector that allows the user to stagger items by any amount. An optional cursor is also available that will transition from item to item as the user interacts with the list. The offset for the cursor position from the given item is defined by the user. The cursor can also have a bobbing like transition and the details of the transition are defined by the user. Each item can also receive touch events that will update the cursor accordingly. The items are all placed in a scroll box widget that is detailed by the user in the definition.

- ItemPositioningType : The type of layout to position each item, “Vertical” or “Horizontal”. (Default: “Vertical”)
- ItemSpacing : The amount of spacing to use when positioning each item. (Default: 0|0)
- CursorSprite : The sprite definition to use for a cursor (Default: No cursor)
- CursorOffsetFromItem : The amount of spacing to put between the cursor and the selected item. (Default: 0|0)
- CursorPositionTransitionDuration : The duration of the transition to get from one item to another. (Default: 0)
- CursorBobbingTransitionOffset: The amount of spacing to put between the starting position of the bob and the peak. (Default: 0|0)
- CursorBobbingTransitionDuration : The duration of the transition to get from the starting position to the offset of the bob. (Default: 0)
- ScrollBox : The Scroll Box Widget to use for the items.

ParticleEffect

A ParticleEffect widget maintains and displays a particle effect that was specified in a separate JSON file. See the Frog 2D Particle System page for the details of the JSON file format.

- ParticleEffect : Path of the file with the details of the particle effect itself. The current graphics path will automatically be added to the beginning of the path. Do not include the file extension.

Table

A Table widget is a vertically scrollable table where at most one row can be selected. The cells of the table are stored by row. If the user clicks on the title of a column, the rows can be sorted by that column. The widths of the columns cannot be adjusted by the user.

Specifying a Table widget in JSON is complicated. Here is a general outline of what should be defined…

```
Table
    |____RowTemplate
    |         |____RowButton
    |         |____CellContainer
    |                   |____CellWidget1
    |                   |____CellWidget2
    |                   |____...
    |
    |____ScrollBox
              |____BackgroundSprite
              |____Columns (Container)
              |         |____Column 1 StateContainer
              |         |         |____ Unsorted Container
              |         |         |         |____ TableColumnTitleButton
              |         |         |         |____ Column Backdrop Sprite
              |         |         |____ Ascending Container
              |         |         |____ Descending Container
              |         |
              |         |____Column 2 StateContainer
              |         |____...
              |
              |____ClipContainer (instantiated rows are placed here automatically)
              |____VerticalScrollbar
```

The details given in “RowTemplate” are used as a template when adding rows to the table. Each row has two parts: RowButton and CellContainer. RowButton is a ToggleButton that goes behind the cells of the row. It is toggled on when the row is selected and toggled off when it is not. Clicking the RowButton selects the row, so its bounds should typically span the entire row. CellContainer is a Container where each child is the cell for a given column of the row. These children should be listed in order from left to right. To keep cell widgets from blocking the RowButtons, be sure to use empty bounds for cells unless they need to be clickable. For example, if the cell is a button, feel free to use non-empty bounds. However, if it is a regular Label, you should typically use TextBounds rather than Bounds.

The “ScrollBox” widget is a type of ScrollBox that contains everything visible in the TableWidget. The ScrollBox's ClipContainer holds all the rows, and you should not specify its Bounds or Children. No HorizontalScrollbar should be given for the ScrollBox.

The “Columns” widget is a Container where each child is a StateContainer that corresponds to a column. The columns should be listed in order from left to right. Each of these StateContainers has three states: Unsorted, Ascending, and Descending. This is because you may want a column or its title to appear differently based on whether you are sorting in ascending order, sorting in descending order, or not sorting primarily by that column. Each of these states is a Container. Unless this is a hidden column, the list of children for each state should include a button with the Type set to “TableColumnTitleButton”. This is a special type of PressButton that automatically causes the Table to be sorted by the associated column when clicked. The text for this button should be set to the title of the column. If you would like the column itself to have a distinct appearance for each of the three states, you can add other child widgets to each state. For example, you could add a Sprite widget with a unique animation to each state. When designing the graphics for a Table widget, be aware that the “Columns” container is drawn behind the rows.

The path for a row of a widget should treat the row as a child of the widget itself. For example, if you had a table named “MyTable” with a column named “TrackNumber”, the path for the track number on the 4th row would be something like “MyLayer.MyTable.Row4.TrackNumber”.

If you want row selection to go down by one when the enter key is pressed, set the table widget's “ShiftDownOnEnter” to true. The selection would wrap around to the top after the bottom is reached. This property defaults to false.

The following event handlers can be defined for this type of widget in Lua.

- OnEnterPress() : Called when the user presses the enter key while this widget has the keyboard focus.
- OnSelectionChange(newRowIndex, oldRowIndex, userMadeChange) : Called when the selection is changed to a different row. 'newRowIndex' is the index of the newly selected row, and 'oldRowIndex' is the index of the row that was selected immediately before the event. Both indices are 1-based, so the first row would be row 1. If the change was triggered directly by the user interacting with the widget, 'userMadeChange' will be true. However, if the change was made from C++ or Lua, it will be false.

The following is an abridged example showing the structurally important parts of a two-column Table widget. Less relevant parts were replaced with a ”…” comment. The first column is for a song title, and the second is the track number.

```
{
   "Name": "ExampleTable",
   "Type": "Table",

   "RowTemplate": {
      "Bounds": "0|0|512|16",  // each row is 512 pixels wide and 16 pixels tall
      "RowButton": {
           // ...
      },
      "Cells": {
         "Children": [
            {
               "Name": "SongTitle",
               "Type": "Label",
               // ...
            },
            {
               "Name": "TrackNumber",
               "Type": "NumberLabel",
               "PositionOffset": "256|0" // The first column was 256 pixels wide.
               // ...
            }
         ]
      }
   },

   "ScrollBox": {
      "PositionOffset": "0|32",  // Leave 32 pixels at the top for the title.
      "Bounds": "0|0|512|256",  // Leave room for 16 rows that are each 16 pixels tall.

      // Define the area that can be scrolled as well as how to clip that area.
      "ClipContainer": {
         "ClipArea": "0|0|512|256"  // Leave room for 16 rows that are each 16 pixels tall.
      },

      // Optional, non-scrolling sprite drawn below the rows and unused space.
      // This may be used as an alternative to specifying a separate sprite for the background
      // of each column's "Unsorted" state.
      //"BackgroundSprite": {
      //   "Animation": "Table_Background"
      //},

      "Columns": {
         "Children": [
            {
               // Column 1: Song Title
               // Position should be the same as the top-most cell for this column.
               // The column is 256 pixels wide.
               "Type": "StateContainer",
               "State": "Unsorted",
               "Children": [
                  {
                     "Name": "Unsorted",
                     "Type": "Container",
                     "Children": [
                        {
```

```
                    "Type": "TableColumnTitleButton",
                    "PositionOffset": "0|-32",
                    "Bounds": "0|0|256|32",
                    "TextKey": "SongTitle",
                    // ...
                 },
                 {
                    // Show the given sprite under the first column when not sorted primarily by this column.
                    "Type": "Sprite",
                    "Animation": "Column1_Highlight_Unsorted"
                 }
              ]
           },
           {
              "Name": "Ascending",
              "Type": "Container",
              "Children": [
                 {
                    "Type": "TableColumnTitleButton",
                    "PositionOffset": "0|-32",
                    "Bounds": "0|0|256|32",
                    "TextKey": "SongTitle",
                    // ...
                 },
                 {
                    // Show the given sprite under the first column when sorted primarily by this column in ascending order.
                    "Type": "Sprite",
                    "Animation": "Column1_Highlight_Ascending"
                 }
              ]
           },
           {
              "Name": "Descending",
              "Type": "Container",
              "Children": [
                 {
                    "Type": "TableColumnTitleButton",
                    "PositionOffset": "0|-32",
                    "Bounds": "0|0|256|32",
                    "TextKey": "SongTitle",
                    // ...
                 },
                 {
                    // Show the given sprite under the first column when sorted primarily by this column in descending order
                    "Type": "Sprite",
                    "Animation": "Column1_Highlight_Descending"
                 }
              ]
           }
        ]
     },
     {
        // Column 2: Track Number
        // Position should be the same as the top-most cell for this column.
        // The column is 256 pixels wide.
        "Type": "StateContainer",
        "PositionOffset": "256|0",
        "State": "Unsorted",
        "Children": [
           {
              "Name": "Unsorted",
              "Type": "Container",
              "Children": [
                 {
                    "Type": "TableColumnTitleButton",
                    "PositionOffset": "0|-32",
                    "Bounds": "0|0|256|32",
                    "TextKey": "TrackNumber",
                    // ...
                 },
                 {
                    // Show the given sprite under the first column when not sorted primarily by this column.
                    "Type": "Sprite",
                    "Animation": "Column1_Highlight_Unsorted"
                 }
              ]
           },
           {
              "Name": "Ascending",
              "Type": "Container",
              "Children": [
                 {
                    "Type": "TableColumnTitleButton",
                    "PositionOffset": "0|-32",
                    "Bounds": "0|0|256|32",
                    "TextKey": "TrackNumber",
                    // ...
                 },
                 {
                    // Show the given sprite under the first column when sorted primarily by this column in ascending order.
                    "Type": "Sprite",
                    "Animation": "Column1_Highlight_Ascending"
                 }
              ]
           },
           {
              "Name": "Descending",
              "Type": "Container",
              "Children": [
                        {
                           "Type": "TableColumnTitleButton",
                           "PositionOffset": "0|-32",
                           "Bounds": "0|0|256|32",
                           "TextKey": "TrackNumber",
                           // ...
                        },
                        {
                           // Show the given sprite under the first column when sorted primarily by this column in descending order
                           "Type": "Sprite",
                           "Animation": "Column1_Highlight_Descending"
                        }
                     ]
                  }
               ]
            }
         ]
      },

      "VerticalScrollbar": {
         "StepMinimum": 16, // Only allow scrolling at increments of one row.
         "StepSmall": 16,
         "StepLarge": 240,
         // ...
      }
   }
}
```

JSONEditor

The JSONEditor widget is an extension of the Table widget for editing JSON data in a structured way. Before editing data, a schema must be provided to detail the rules for the data to be edited. When the JSONValue tree is given to the widget for editing, you must also specify the associated type name from the schema. For example, if you wanted to edit the properties of a grid cell, you would provide the JSONValue for the root of that cell and specify something like "Cell", from the schema, as its type.

The widget will generate two columns based on widget templates, the schema, and the data. The first column is for the keys or indices and the second column is for the values. If you're editing a JSON object, the left column will be the keys of the children. If you're editing a JSON array, the left column defaults to the numerical indices of the children. However, there are cases where the left column for an array can show something more descriptive for the key. If the array consists of objects, the schema for those objects specifies a "PrimaryKey" member, and that member is a defined string, then that member will be used for the left column when the objects are listed in an array. This is useful for ordered collections of objects that contain a name.

The JSONEditor widget displays JSON objects and arrays which may contain other objects and arrays. When you click to edit a nested object or array, the rows of the table switch to showing the contents of that item. The editor widget can include a separate button called the "BackButton" which allows you to navigate back to the parent object or array. In this way, you can edit data hierarchically.

Much like the Table widget, the definition of a JSONEditor widget can be complicated. However, since this is only intended for tools, it can rely more on vector drawing, and the definition can likely be copied from one project to the next. When setting up the RowTemplate for the widget, leave the CellContainer empty. Its children will be added procedurally based on other templates, and not every row will have the same contents. Because of this inconsistency in the second column, you can only sort by the first column using the default TableWidget sort routine.

There are some additional items when specifying a JSONEditor widget, compared to a Table widget. In order to generate the grid cells for different types of data, templates must be provided at the root of the widget. There are also extra buttons for navigating and manipulating the data.

- BackButton : PressButton for going back to a parent JSONValue after having navigated to a child.
- ItemEditButtonTemplate : Template for the PressButton widgets for the second column which open nested JSON objects and arrays.
- ItemEnumerationTemplate : Template for the DropDownList widgets for choosing a value from a fixed set of possible values in the second column. This is used for booleans, fixed sets of strings, and fixed sets of numbers.
- ItemKeyReadOnlyTemplate : Template for the Label widgets for displaying keys in the left column which cannot be edited.
- ItemStringReadOnlyTemplate : Template for the Label widgets for displaying values in the second column which cannot be edited.
- ItemStringTemplate : Template for the TextEntry widgets for strings and numerical values that can be edited in the second column.

Schemas

The schema that specifies the rules for the data to be edited and displayed are provided as a JSON object. The schema contains a child object named "Types" which specifies the rules for individual named types. Each of these types must specify a base type, which is listed as "Type". The vaid base types are "Boolean", "Number", "String", "Array", and "Object".

When specifying a number or string type, you can specify an array of strings named "Values". If the base type is a string, items of this type can only have one of these values. If the base type is a number, it works more like a C enumeration. In this case, the strings in "Values" will be used for display purposes, but the underlying value will be the index number , starting at 0, of the associated string in the "Values" array.

Items with a base type of String, Number, or Boolean can have a "ReadOnly" property. If the property is true, the value will not be editable in the JSONEditor widget. Items of these types may also specify a "Default" value.

Items with a base type of Array must include an object named "Children" which specifies the type of each item in the array. Items with a base type of Object must also have a property named "Children", but in this case, the "Children" collection is an array. Each item in this array must be a JSON object to describe a member of the object, including a "Name" for the child.

See the JSONEditor description above for an explanation of the "PrimaryKey" property.

Keyboard

The Keyboard widget is a custom Container Widget in which the user can use on screen buttons to type into a Text Entry Widget. Certain keys must be defined to implement this widget while other keys can be left to the definition to define. The following is a sample definition of a typical Keyboard.

```
{
   "Name": "Keyboard",
   "Type": "Keyboard",
   "TextLengthMax": 8,
   "Position": "30|280",
   "Children":
   [
      {
         "Type": "Sprite",
         "Animation": "KeyboardBackground",
         "PositionOffset": "0|-100"
      },
      {
         "Type": "Sprite",
         "PositionOffset": "250|50",
         "Animation": "TextBox"
      },
      {
         "Name": "TextDisplay",
         "Type": "TextEntry",
         "TextAlignment": "Center",
         "TextAscentDescentMaxUse": true,
         "PositionOffset": "250|50",
         "Bounds": "0|0|300|50",
         "TextScale": "2|2"
      },

      {
         "Name": "KeySet",
         "Type": "StateContainer",
         "Children":
         [
            {
               "Name": "Uppercase",
               "Type": "Container",
               "Children":
               [
                  {
                     "Name": "Q",
                     "Type": "PressButton",
                     "Text": "Q",
                     "PositionOffset": "40|100",
                     "UpOutState": "KeyUp",
                     "DownOverState": "KeyDown"
                  },
                  {
                     "Name": "W",
                     "Type": "PressButton",
                     "Text": "W",
                     "PositionOffset": "120|100",
                     "UpOutState": "KeyUp",
                     "DownOverState": "KeyDown"
                  },
                  {
                     "Name": "E",
                     "Type": "PressButton",
                     "Text": "E",
                     "PositionOffset": "200|100",
                     "UpOutState": "KeyUp",
                     "DownOverState": "KeyDown"
                  }
               ]
            },
            {
               "Name": "Lowercase",
               "Type": "Container",
               "Children":
               [
                  {
                     "Name": "q",
                     "Type": "PressButton",
                     "Text": "q",
                     "PositionOffset": "50|100",
                     "UpOutState": "KeyUp",
                     "DownOverState": "KeyDown"
                  },
                  {
                     "Name": "w",
                     "Type": "PressButton",
                     "Text": "w",
                     "PositionOffset": "100|100",
                     "UpOutState": "KeyUp",
                     "DownOverState": "KeyDown"
                  },
                  {
                     "Name": "e",
                     "Type": "PressButton",
                     "Text": "e",
                     "PositionOffset": "150|100",
                     "UpOutState": "KeyUp",
                     "DownOverState": "KeyDown"
                  }
               ]
            }
         ]
      },

      {
         "Name": "Space",
         "Type": "PressButton",
         "TextKey": "SpaceKeyLabel",
         "PositionOffset": "200|340",
         "UpOutState": "SpaceKeyUp",
         "DownOverState": "SpaceKeyDown"
      },
      {
         "Name": "Clear",
         "Type": "PressButton",
         "TextKey": "ClearKeyLabel",
         "PositionOffset": "840|100",
         "UpOutState": "ClearKeyUp",
         "DownOverState": "ClearKeyDown"
      },
      {
         "Name": "Backspace",
         "Type": "PressButton",
         "TextKey": "BackspaceKeyLabel",
         "PositionOffset": "840|180",
         "UpOutState": "ClearKeyUp",
         "DownOverState": "ClearKeyDown"
      },
      {
         "Name": "CapsLock",
         "Type": "PressButton",
         "TextKey": "CapsLockKeyLabel",
         "PositionOffset": "840|260",
         "UpOutState": "ShiftKeyUp",
         "DownOverState": "ShiftKeyDown"
      },
      {
         "Name": "Enter",
         "Type": "PressButton",
         "TextKey": "EnterKeyLabel",
         "PositionOffset": "800|200",
         "UpOutState": "ShiftKeyUp",
         "DownOverState": "ShiftKeyDown"
      }
   ]
}
```

The buttons with names: Space, Clear, Backspace, CapsLock, and Enter must be defined separately as each will respond in a unique manner, Buttons listed in the KeySet State Container are handled differently. Theses keys are separated into two states, upper case and lower case. The CapsLock key defined will handle the switching of case. Any key in here will add the text that is found in the Name field to the string being created. Therefore the Keyboard is not limited to alphabet letters. You can define some keys that have a number in the lower case and a symbol in the upper. For example a “1” in lower case and a ”!” in upper.

The Enter key for the KeyboardWidget can register an OnPress callback by setting it directly with PressButtonWidget::OnClickRegister(const char* path, OnClickCallback callback, void* userData) and specifying the path the the enter button.

In some projects, you may want to show a KeyboardWidget only when a physical keyboard isn't available. For example, show the keyboard in iOS or 3DS, but not on PC and Mac. You can typically base the run-time decision on the return value of Keyboard::ConnectedCheck().

Joystick

On platforms that have no buttons or joysticks, like the iPhone, you can use a Joystick widget to simulate an analog stick.

- BackgroundSprite : Optional child sprite to be displayed behind the other parts of this widget. (Default: no background sprite)
- Nub : A PressButton that can be dragged to adjust the value for the joystick. The initial position of the nub will be its “rest position”. When it is not being dragged, it will return to this spot.

A Joystick's nub can also have the following property.

- ReturnSpeed : How fast the nub should return to the rest position when it isn't being dragged. The actual speed in pixels per second is proportionate to the dragging area of the nub, so the same "ReturnSpeed" should have a similar feel regardless of the size of the graphics in pixels. (Default: 20.0)

ScreenFillSprite

A ScreenFillSprite widget is a type of sprite widget that automatically positions and scales itself to fill the screen. This can be especially helpful in games that draw over the black bars for pillarbox or letterbox.

- FillMode: This controls the details of how the sprite is positioned and scaled. The following values are allowed. (Default: "FullKeepAspectRatio")
  - Full: Stretch the sprite to fill the part of the screen available to the game. This includes the extended area, if allowed. This does not necessarily preserve aspect ratio.
  - FullKeepAspectRatio: While preserving the aspect ratio, stretch the sprite to fill the part of the screen available to the game. This includes the extended area, if allowed. The sprite may be cropped and centered.

Widget Child Sprites

The Frog GUI system uses the Frog Sprite System to display things like buttons, backgrounds, and decals. Animations for these sprites are given in the sprite resource file of the relevant GUI layer. The "Sprite" widget is basically a wrapper for inserting a single sprite into a GUI layer, but it is not the only type of widget that can own a sprite. For example, a button can have separate sprites for any of its states. These child sprites are not actually widgets themselves, and they do not support all of the same parameters as widgets.

- Animation : The name of the animation to be displayed.
- Position : The absolute position of the sprite's origin in screen coordinates. This is mutually exclusive with PositionOffset. (Default: same position as parent widget)
- PositionOffset : The position of the sprite's origin relative to its parent widget's origin. This is mutually exclusive with Position. (Default: same position as parent widget)

Keyboard Focus

Certain widgets, like the TextEntry widget, receive text typed using a keyboard as long as the widget has the keyboard focus. Only one widget can have the focus at any one time, so you can only enter text into one widget at a time.

When the user presses a widget using a pointing device, it becomes a candidate to receive the keyboard focus. The widget is first asked if it can change the keyboard focus, and if it can't, the focus remains as it was before the user pressed with their pointing device. If the widget is allowed to change the focus, but it is not allowed to have the focus, this process is repeated recursively as though the parent had been pressed. If the widget is both allowed to change the focus and have the focus, it receives the focus. If no widget in the chain of ancestors can accept the focus, the focus does not change.

Gamepads

On platforms that have gamepads, but no pointing devices, some of the widgets can still be used. You can't simply point to the widget you want, so instead, widgets can be selected, and that selection can shift between widgets using the directional buttons or joystick on the gamepad. Here, that notion of selection is called "gamepad focus". Games can have more than one gamepad focus, and physical gamepads can be assigned to gamepad foci in different ways. For example, you could have all the gamepads control a single focus, which would be useful for a main menu. You could also assign each physical gamepad to a separate focus, which would be useful for setting player-specific preferences for local multiplayer.

Gamepad focus shifts between widgets along cardinal directions, so it is cleanest to position the widgets in grids. The links that connect the widgets are set in C++.

Widgets have separate events for interaction with gamepads. Instead of OnPress, OnRelease, OnClick, and OnMouseOver, the equivalent gamepad events are OnGamepadPress, OnGamepadRelease, OnGamepadClick, and OnGamepadFocusGain.

Buttons using gamepad focus can reuse the same sprite animations and states for interaction with cursors. For example, if the gamepad is pressing a button widget, the widget would display its DownOverState.

Slider

When using a gamepad to control a slider widget, the directions corresponding to increasing and decreasing the value are determined automatically. For example, if the thumb has its maximum position set 100 pixels to the right of its minimum position, the slider will use right for increasing and left for decreasing. If there is no thumb, the centers of the bounds of the increase and decrease buttons are used instead. The directions used for adjusting the value of a slider widget cannot also be used for links between widgets. If you try to use the same axis for both, the value of the slider should not change.

The appearance of the thumb and buttons can change to indicate that the slider has gamepad focus, similar to the way they change appearance when interacting with a cursor.

Sounds

There are certain widget events for which a sound may be specified. If a sound was not explicitly set for a given event for a given widget, it will defer to its parent to see which sound should be played. This allows you to set the sounds once for a group of widgets. It is even possible to set default sounds for all widgets in the game using the GUIManager.

If you don't want a widget to automatically play a sound inherited from its ancestors, you can disable the sound for that widget using a setting like "ClickSoundEnabled" or "PressSoundEnabled". The defaults for these settings depend on the widget type. For example, a PressButton would play a ClickSound by default, but a Sprite widget would not. If you want to override the default, set the setting explicitly for the widget in question.

Transitions

Widgets can be made to perform one-shot animations, such as having all the elements of a GUI layer fly onto the screen together. The details of these animations are given in Transitions section of a widget's specifications. Each transition for a given widget must have a unique name, but you can have an arbitrary number of them. The most common animations will likely be "Enter" and "Exit", although this is not a convention required by Frog itself. Frog comes with a set of transition effects that are available in all projects, but individual projects can register additional effect types through the GUIManager.

The following example specifies two transitions: one named "Enter" and the other named "Exit". When the Enter transition is played, this widget will begin offscreen below, wait for 200 milliseconds, and slide into place over the course of the following 500 ms. The Exit transition does the same thing, in reverse.

```
"Transitions":
{
   "Enter": {
      "Type": "FromBottom",
      "StartDelay": 200,
      "Duration": 500,
      "Progression": "Linear"
   },
   "Exit": {
      "Type": "ToBottom",
      "StartDelay": 200,
      "Duration": 500,
      "Progression": "Linear"
   }
}
```

If you only want to use the defaults for a given transition, you can simply give the name of the transition type rather than providing a JSON object. In the following example, the "Enter" animation is given in this way.

```
"Transitions":
{
   "Enter": "FromBottom",
   "Exit": {
      "Type": "ToBottom",
      "Duration": 500
   }
}
```

To begin a transition, use GUIManager::TransitionBegin. While it is in progress, all widgets should ignore input from the user.

The following transition effects are available in all projects.

FromBottom

Move from offscreen below to the current position according to the following parameters.

- StartDelay : Number of milliseconds after the transition is started before movement begins. (Default: 0 ms)
- Duration : Number of milliseconds between the beginning and end of movement. (Default: 500 ms)
- Progression : Pattern of the movement. See Progressions (Default: SmoothStop)

FromRight

Move from offscreen on the right to the current position. Parameters and defaults are the same as FromBottom.

FromLeft

Move from offscreen on the left to the current position. Parameters and defaults are the same as FromBottom.

FromTop

Move from offscreen above to the current position. Parameters and defaults are the same as FromBottom.

ToBottom

Move from the current position to offscreen below. Parameters and defaults are the same as FromBottom, except Progression defaults to SmoothStart.

ToRight

Move from the current position to offscreen on the right. Parameters and defaults are the same as FromBottom, except Progression defaults to SmoothStart.

ToLeft

Move from the current position to offscreen on the left. Parameters and defaults are the same as FromBottom, except Progression defaults to SmoothStart.

ToTop

Move from the current position to offscreen above. Parameters and defaults are the same as FromBottom, except Progression defaults to SmoothStart.

Linear

This follows the same rules as TimedTransitionLinear2D and accepts the same parameters, but it does not affect color. The following parameter is also accepted.

- RemainAfterFinished : True if this transition should remain in-effect, even after it has finished. (Default: true)

Automatic Reloading

When the Frog window receives the focus in Debug PC builds, a layer will be reloaded automatically if its Widgets.json file has been modified. For example, you could Alt-Tab from the game to PSPad, change the position of something in Widgets.json, Alt-Tab back to the game, and the widget would be in the new position.

Since it is being reloaded, certain things may need to be reinitialized in C++. For instance, any C++ button callbacks would need to be registered again. You can handle this using the LayerWidget's OnInit callback, which the replacement layer will inherit from the old layer. When the replacement layer is initialized, the same callback is called again. You can register your callback as part of your GUIManager::Push, InsertAbove, or InsertBelow calls.

Lua Functions

The GUI can be controlled from Lua using the following functions…

- GUIDropDownListItemAdd("path", "itemName", "itemText") : Add an item with the given name and literal string label to the end of the given DropDownList.

```
GUIDropDownListItemAdd("Test.AnimalDropDownTest", "Item_Fish", "Fish")
```

- GUIDropDownListItemExpressionAdd("path", "itemName", "itemTextExpression") : Add an item with the given name to the end of the given DropDownList. 'itemTextExpression' is a Lua expression that evaluates to the correct text for its label. This is useful for localized text.

```
GUIDropDownListItemAdd("Test.AnimalDropDownTest", "Item_Fish", "StaticText.Fish")
```

- GUIDropDownListItemRemove("path", "itemName") : Remove the first item with the given name from the specified DropDownList.
- GUIDropDownListSelectionClear("path") : Set the given DropDownList so that nothing is selected.
- GUIDropDownListSelectionGet("path") : Return the name of the currently selected item of the DropDownList or nil if nothing is selected.
- GUIDropDownListSelectionSet("path", "itemName") : Set the currently selected item for the given DropDownList. 'itemName' should be the name of the item to be selected.
- GUIFocusSet("path") : Attempt to give keyboard focus to the widget with the specified path. Sometimes, an ancestor of the specified widget will end up getting the focus if the specified widget doesn't accept it.
- GUILabelTextExpressionGet("path") : Return the Lua expression used for the text of the given label. Return nil if the label is not using a text expression.

- GUILabelTextExpressionSet("path", "textExpression") : Set the Lua expression used for the text of the given label.
- GUILabelTextGet("path") : Return the text displayed for the given label.
- GUILabelTextSet("path", "text") : Set the text displayed for the given label.
- GUINumberLabelTextFormatGet("path") : Return the Lua expression that evaluates to the function that will be called with the widget's value to get the string representation. Return nilif a format is not in use.
- GUINumberLabelTextFormatSet("path", "textFormat") : Set the Lua expression that evaluates to the function that will be called with the widget's value to get the string representation. Call with 'textFormat' set to nil to use the default format, which is the printf-style %f (1.0000002, etc.).

```
-- Use the StringFormatInteger(number) Lua function.
GUINumberLabelTextFormatSet('Test.TestNumberLabel', 'StringFormatInteger')
```

- GUINumberLabelValueGet("path") : Return the numerical value of the NumberLabel with the given path.
- GUINumberLabelValueSet("path", value) : Set the numerical value of the NumberLabel with the given path.
- GUIPop() : Remove the top GUI layer from the stack.
- GUIPreload("name") : Preload the sprite data of the GUI with the given name.
- GUIProgressBarIndeterminateGet("path") : Return true if the progress bar with the given path is in indeterminate mode.
- GUIProgressBarIndeterminateSet("path", indeterminate) : Set whether the progress bar with the given path is in indeterminate mode.
- GUIProgressBarValueGet("path") : Return the value of the progress bar with the given path.
- GUIProgressBarValueSet("path", value) : Set the value of the progress bar with the given path.
- GUIPush("name") : Add the GUI with the given name to the top of the stack.
- GUIRemove("name") : Remove the GUI layer with the given name.
- GUISliderValueGet("path") : Return the value of the slider with the given path.
- GUISliderValueSet("path", value) : Set the value of the slider with the given path.
- GUIStateContainerStateGet("path") : Return the name of the widget for the current state of the given StateContainer.
- GUIStateContainerStateSet("path", "name") : Set the current state of the given StateContainer by providing the state's widget name.
- GUITableEventRowIndexGet("path") : Return the index of the row of this data grid that was last involved in calling an event script. Return 0 if no row or cell has called an event script. This is useful when the cells of a row have event scripts and you need to know the row to which the event corresponds.

```
function GUI.TableTest.Table.RowButton.OnClick()
   local rowNumber = GUITableEventRowIndexGet("TableTest.Table")
   print("Button clicked on row " .. rowNumber .. "!")
end
```

- GUITableRowAdd("path") : Add a new row at the bottom of the given Table widget.
- GUITableRowCountGet("path") : Return the number of rows in the given Table widget.
- GUITableRowRemove("path", rowNumber) : Remove the specified row from the given Table width.
- GUITableSelectionGet("path") : Return the index of the currently selected row of the Table widget or 0 if nothing is selected.
- GUITableSelectionSet("path", rowNumber) : Select the given row of the specified Table widget.
- GUITextDocumentTextExpressionGet("path") : Return the Lua expression to be executed, interpreted, and displayed by the given TextDocument. Return nil if the TextDocument is not using a text expression.
- GUITextDocumentTextExpressionSet("path", "expression") : Set the Lua expression to be executed, interpreted, and displayed by the given TextDocument.
- GUITextDocumentTextGet("path") : Return the text interpreted and displayed by the given TextDocument.

- GUITextDocumentTextSet(“path”, “text”) : Set the text to be interpreted and displayed by the given TextDocument.
- GUIToggleButtonOnGet(“path”) : Return true if the given toggle button is on.
- GUIToggleButtonOnSet(“path”, on) : Set whether the the given toggle button is toggled to on. ('on' should be true or false.)
- GUIUndoPreload(“name”) : Undo the preloading of the sprite data of the GUI with the given name. This will not necessarily free anything, but it will decrement reference counts.
- GUIWidgetEnabledGet(“path”) : Return true if the widget with the given path is currently enabled.
- GUIWidgetEnabledSet(“path”, enabled) : Enable or disable the widget with the given path.

/share/Web/wiki/dokuwiki/data/pages/frog_gui_system.txt · Last modified: 2015/03/13 01:08 by mpellegrini