Simutrans

Theme and GUI specification

Rev. 50

Table of Contents

Preface

This is the draft for GUI them and, to some extent, remake of the existing GUI implementation in Simutrans. This document is still live and may change without notice. In this version some thoughts and visions are documented, but not yet realised in code. This theme system will replace the current skin system.

Goals

The them will allow the user to design basic elements of the GUI as images. These images will be used to assemble the part of GUI they represent. The user can select a theme through a new theme selector dialogue in Simutrans. Beside the graphical representation that must be provided as images, the user can tweak a loaded theme in respect to various element sizes, such as buttons, edit fields, margins etc... This is done through the property tab in the theme selector.

Another goal is to have the GUI to work in portable devices where larger buttons are required for touch screens. In this scenario it is more important to preserve the size of interactive elements than pure visual. This can be done by a collapse scheme where less important details are reduced in size and then eventually removed to give space for the more important interactive elements.

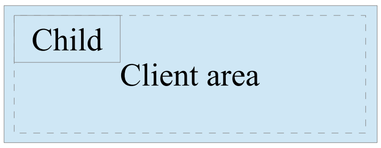
Introductaion

Themes

All visual controls can be given a personal look through a theme. Some visual components are similar and can share the theme images as fall back when an image is missing. This simplifies the process of creating a theme because not all visual components need to be present.

GUI Components

These components are used to build the GUI. Some components are containers that can hold a collection of other components. This collection becomes the container's child components. All visual components have a client area where the child components are drawn. When the client area changes each child control may be notified so it can take actions to the new client size.



The client area is always inside the bounding box and always of the same size or smaller than the bounding box. However, it doesn't need to be equal centred inside the bounding box. The client origin property positions the client rectangle relative to the bounding box origin (position). All child components are positioned relative to its parent client origin.

A component, to build the GUI, can be classified by one or more properties.

Non visible

These usually serve as a container to organize other visible components. They can for example make the decision if their children should start to collapse or not, or make sure they align right when the client area change in size (layout controls). Non visible controls can't be interactive or seen.

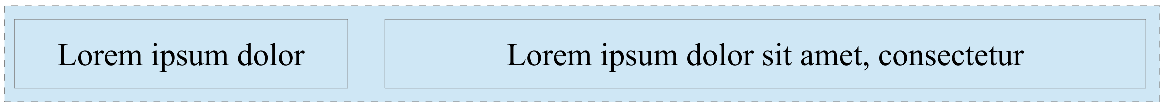
Container

The container adds the possibility to have child controls. The client area is the same as the bounding box. A container object's children automatically becomes subscribers to notification messages within the container.

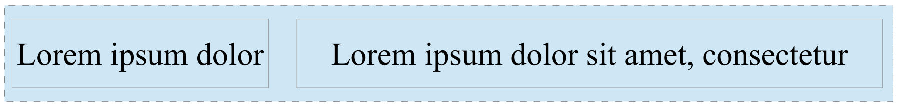
Layout

This is a descendent of the container control. It will position the child controls according to a set of predefined rules. For example it can space them out evenly in a grid, horizontal proportional, flow, stacked. It might even be possible to have different layouts for the collapse stages.

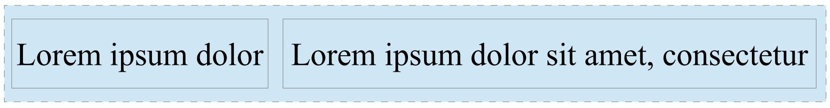
Example of proportional layout with 2 buttons.



The space needed by the two buttons represent 100% When the client area shrinks the buttons will shrink in proportion to the client area.



The collapse stages may try to keep as much of the button text as possible by capping the size and reduce the spacing before next collapse stage.

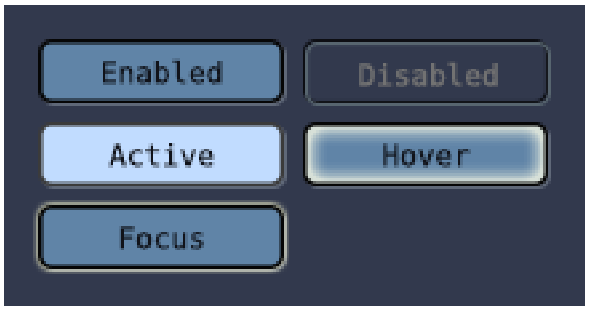


Visible

These components have a paint routine, drawing the component and giving it a visual style. Some examples are label, bevel, divider etc... Pure visible components can be seen but not interact with the user and makes them more light than an interactive component. For example, adding a border to a container object and move in the client area will give you a group box.

Interactive

These components are drawn (seen) and can interact with the user. Their state can be reflected in the theme/GUI. Examples are buttons, text edit, windows, toolbars etc... Some common states could be enabled, disabled, active, focused, hover etc...



Alignment

All container and visual controls can be aligned against each other both horizontal and/or vertical, meaning a control can be aligned either in vertical, horizontal or both at the same time.

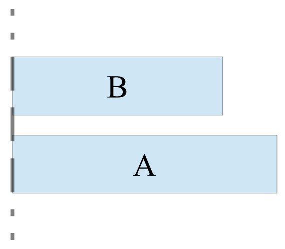
Interior alignment aligns a control's bounding box to another control's bounding box interior (inside), while exterior alignment aligns to another control's exterior bounding box (outside).

An offset can be applied to the final alignment position. This is handy when controls are lined up with a fixed space between them. The offset can be either a positive or negative value.

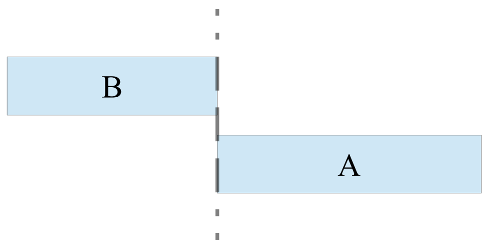
If one of the alignment directions (horizontal or vertical) is set to none, but have an offset not set to 0. The offset becomes an absolute position for that direction.

Horizontal alignment of control A against control B with offset 0.

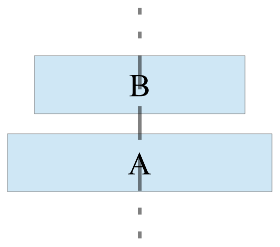
Interior Left
`A.align_to(&B, ALIGN_LEFT)`



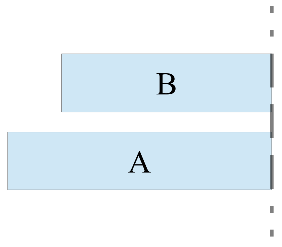
Exterior Left
`A.align_to(&B, ALIGN_EXTERIOR_H | ALIGN_LEFT)`



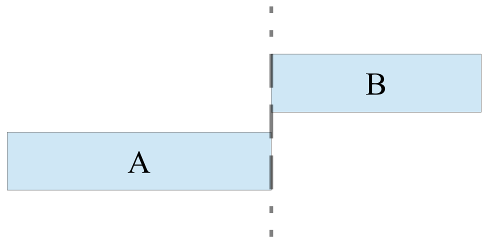
Interior Centre H and Exterior Centre H
`A.align_to(&B, ALIGN_CENTER_H)` or `A.align_to(&B, ALIGN_EXTERIOR_H | ALIGN_CENTER_H)`



Interior Right
`A.align_to(&B, ALIGN_RIGHT)`



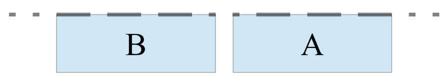
Exterior Right
`A.align_to(&B, ALIGN_EXTERIOR_H | ALIGN_RIGHT)`



Vertical alignment of control A against control B with offset 0.

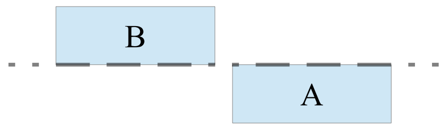
Interior Top

`A.align_to(&B, ALIGN_TOP)`



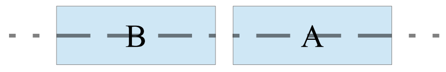
Exterior Top

`A.align_to(&B, ALIGN_EXTERIOR_V | ALIGN_TOP)`



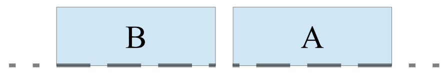
Interior Centre V and Exterior Centre V

`A.align_to(&B, ALIGN_CENTER_V)` or `A.align_to(&B, ALIGN_EXTERIOR_V | ALIGN_CENTER_V)`



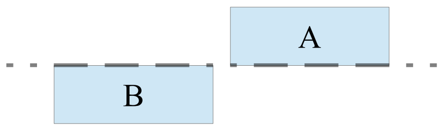
Interior Bottom

`A.align_to(&B, ALIGN_BOTTOM)`



Exterior Bottom

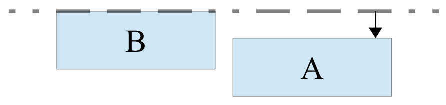
`A.align_to(&B, ALIGN_EXTERIOR_V | ALIGN_BOTTOM)`



Offset alignment examples.

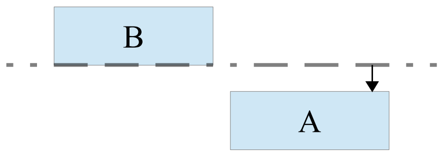
Interior Top with +10 offset

`A.align_to(&B, ALIGN_TOP, koord(0,10))`



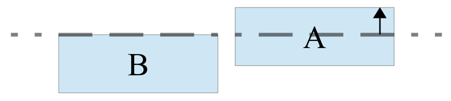
Exterior Top with +10 offset

`A.align_to(&B, ALIGN_EXTERIOR_H | ALIGN_LEFT, koord(0,10))`



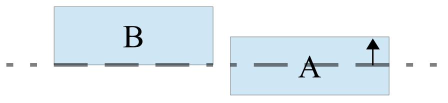
Interior Top with -10 offset

`A.align_to(&B, ALIGN_TOP, koord(0,-10))`



Exterior Top with -10 offset

`A.align_to(&B, ALIGN_EXTERIOR_H | ALIGN_LEFT, koord(0,-10))`



Collapsing GUI

When a client area becomes to small to fit all its children, it will star to collapse the children to reclaim more space for the more important GUI controls. This is an important process to target mobile devices with small touch screens. All interactive GUI elements must be large enough to use, even when a window needs to be small due to the screen size.

The collapse process takes part in multiple step where the least significant GUI elements start to shrink and as a last resort, disappear. This will free up more client area to keep the important GUI controls unchanged.

Each GUI control will have a built in collapse scheme, activated by its parent client change notification. The theme may carry information of controls minimum size or collapse stage. In this way a theme can be tailored for mobile devices.

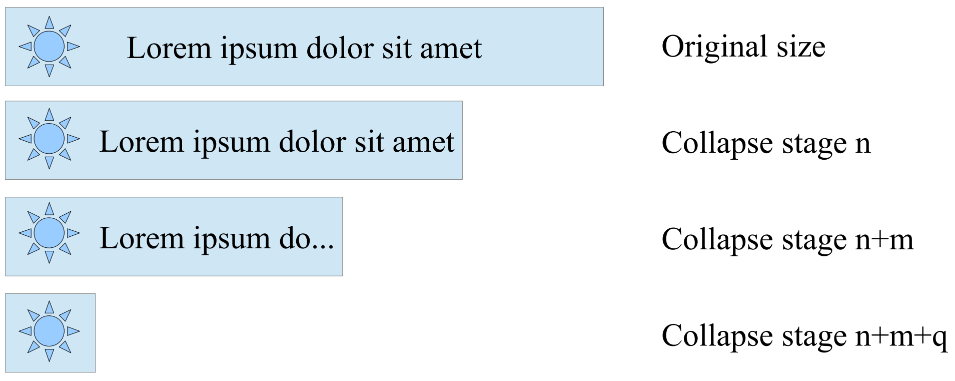
Examples of actions for pure visible GUI components.

- Padding shrinks.
- Distance between controls shrinks.
- Dividers are removed.
- Labels width shrinks and incomplete text ends in …
- Labels may be removed.

Examples of actions for interactive GUI components.

- Button width may shrink and have incomplete text ending in …
- Buttons with pictures have their text completely removed.
- List view areas shrinks to predefined minimum.
- Group boxes collapse (can be expanded and collapsed by user)
- Scroll box appear in window.

Example of a collapsing button with text and icon.



The collapse stages are configurable in the them and doesn't need to be in an interval. Collapse stage 0 is when all controls are visible in their original state. When the client area shrinks and more space is needed, the parent increments the collapse stage by one and notifies all its children.

By setting the first collapse stage at a higher starting point, a certain control type can be delayed from collapsing. This is very handy when you want, for example, buttons to be the last control to collapse.

On the contrary, if the client area grows the children are again notified and the collapse level will decrease. When the collapse level is back at 0 all controls are again in their original state. Should the client area grow further, the controls may follow a border or another control. In this case the alignment is linked to another control and will follow its position and not be resized.

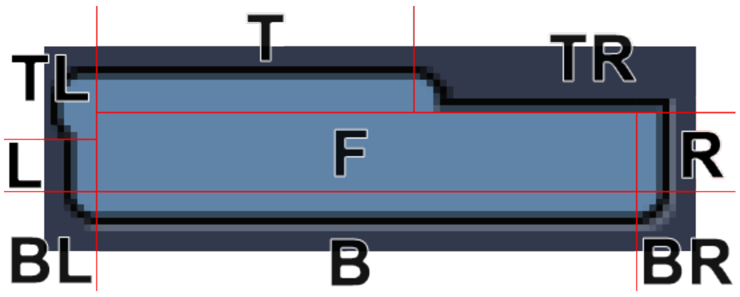
Some control may increase their client area (resize) the window's client gets larger. A good example of this would be map previews or graphs that would benefit from a larger client area.

Theme drawing

Each themed GUI element is assembled from a set of predefined *rectangular* theme elements, separated by a *guard colour*. The finished GUI elements doesn't need to be of the same size as the original image. In the theme configuration dialogue the user can tweak some GUI parameters as height and width. The theme elements are repeated to extend the size.

Here are all button elements in their original theme size.

TL	top left
T	top
TR	top right
L	left
F	face
R	right
BL	bottom right
B	bottom
BR	bottom right

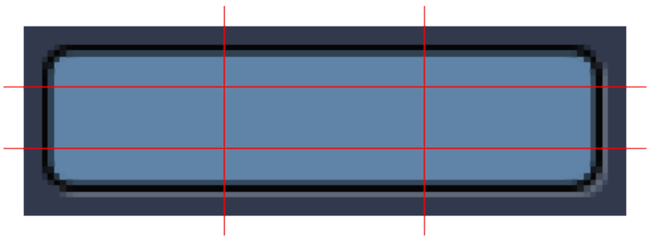


Each rectangle can be of different size to create a non uniformed button shape, as this example. As long the button's side images; L, T, R and B can be tiled (repeated), the button can be resized in all directions.

The buttons minimum size is when a corner image; TL, TR, BL or BR are touching each other.

If the element's background has a pattern, the tile operation can't be done one pixel at a time. For this we need to tell the theme the repeatable pattern size. The user can then only resize the theme in pattern size intervals.

A more simple approach to define an element is to not include guard lines at all. If no guard lines are found, the element can still be resized. The theme will divide the image in a 3 by 3 grid to create the 9 needed theme rectangles.



When using this simplified approach, bare in mind that the smallest size is still when the corner images are touching. With this approach it is more beneficial to create quadratic buttons that can be sized into the desired rectangular shape.

Theme override

If the theme needs to be adjusted by the end user, it can be overridden by a theme.tab file located in the theme folder. The reason to do this might be that the size of various GUI elements needs to be enlarged on a hires-display or portable device. Or made smaller to fitt more information on the screen.

Beside the GUI element sizes, text colours can also be overridden. This might be necessarily if a dark theme is used to change the text to a brighter colour.

At this moment, the theme.tab file is experimental and the syntax has not yet been settled down in stone. The variables for now are as follow:

```
gui_titlebar_height
gui_frame_left
gui_frame_top
gui_frame_bottom
gui_hspace
gui_vspace
gui_button_width
gui_button_height
button_color_text
button_color_disabled_text
gui_text_color
gui_static_text_color
gui_disabled_text_color
gui_highlight_color
gui_shadow_color
gui_face_color
gui_button_text_color
gui_indicator_width
gui_indicator_height
gui_tab_header_vsize
front_window_bar_color
front_window_text_color
bottom_window_bar_color
bottom_window_text_color
tooltip_background_color
tooltip_text_color
toolbar_max_width
toolbar_max_height
cursor_overlay_color
```

Example:

```
gui_titlebar_height   = 24
gui_button_width      = 100
gui_button_height     = 24
gui_static_text_color = #AABBCC
gui_disable_text      = #777777
```

The unit for size and distance are pixels while a colour is defined by the colour index as an integer or as an RGB Hex value #RRGGBB that will be converted to RGB555 and mapped tot he nearest indexed colour.

This might change to the true RGB555 colour further down the road.

GUI controls

ExampleA GUI control must be owned by a component capable of having child controls. The lowest level for this is the container control class. All controls capable of having children must be derived from this class or its descendants.

All GUI controls, including non visual, have a pointer back to its parent so it can send notification messages to all its siblings. An example of this might be mouse enter and mouse leave events.

Buttons

There a many types of buttons. The common for all of them are that they can be disabled, enabled, active, hover and focused. If no theme is defined for them, they have a default fall back look.

Round button