ANKLANG Development Details

The Anklang Project <anklang.testbit.eu>

August 2023

Abstract

API documentation and development internals of the Anklang project.

Contents

List of Tables

1 ANKLANG Development Details

Technically, Anklang consists of a user interface front-end based on web technologies (HTML, DOM, CSS, JavaScript, Lit) and a synthesis engine backend written in C++.

1.1 ASE - Anklang synthesis engine

The `ase/` subdirectory contains the C++ implementation of the `AnklangSynthEngine` executable which contains the core component for audio data processing and audio plugin handling. It interfaces with the HTML DOM based user interface via an IPC layer with JSON messages that reflect the C++ API.

The synthesis engine can load various audio rendering plugins which are executed in audio rendering worker threads. The main synthesis engine thread coordinates synchronization and interafces between the engine and the UI via an IPC interface over a web-socket that uses remote method calls and event delivery marshalled as JSON messages.

1.1.1 Serialization

Building on Jsonipc, a small serializaiton framework provided by ase/serialize.hh is used to marshal values, structs, enums and classes to/from JSON. This is used to store preferences and project data. The intended usage is as follows:

```
std::string jsontext = Ase::json_stringify (somevalue);
bool success = Ase::json_parse (jsontext, somevalue);
// The JSON root will be of type 'object' if somevalue is a class instance
std::string s;                                          // s contains:
s = json_stringify (true);                              // true
s = json_stringify (-0.17);                             // -0.17
s = json_stringify (32768);                             // 32768
s = json_stringify (Ase::Error::IO);                    // "Ase.Error.IO"
s = json_stringify (String ("STRing"));                 // "STRing"
s = json_stringify (ValueS ({ true, 5, "HI" }));        // [true,5,"HI"]
s = json_stringify (ValueR ({ {"a", 1}, {"b", "B"} })); // {"a":1,"b":"B"}
```

In the above examples, `Ase::Error::IO` can be serialized because it is registered as `Jsonipc::Enum<Ase::Error>` with its enum values. The same works for serializable classes registered through `Jsonipc::Serializable<SomeCl`

[_] Serialization of class instances will have to depend on the Scope/InstanceMap, so instance pointers in copyable classes registered as `Jsonipc::Serializable<>` can be marshalled into a JsonValue (as `{$id,$class}` pair), then be resolved into an InstanceP stored in an Ase::Value and from there be marshalled into a persistent relative object link for project data storage.

1.2 Jsonipc

Jsonipc is a header-only IPC layer that marshals C++ calls to JSON messages defined in jsonipc/jsonipc.hh. The needed registration code is very straight forward to write manually, but can also be auto-genrated by using jsonipc/cxxjip.py which parses the exported API using CastXML.

The Anklang API for remote method calls is defined in api.hh. Each class with its methods, struct with its fields and enum with its values is registered as a Jsonipc interface using conscise C++ code that utilizes templates to derive the needed type information.

The corresponding Javascript code to use `api.hh` via async remote method calls is generated via `Jsonipc::ClassPrinter::to_string()` by `AnklangSynthEngine --js-api`.

- [√] `shared_ptr<Class> from_json()` - lookup by id in InstanceMap or use `Scope::make_shared` for Serializable.

- [√] `to_json (const shared_ptr<Class> &p)` - marshal Serializable or {id} from InstanceMap.
- [√] `Class* from_json()` - return `&*shared_ptr<Class>`
- [√] `to_json (Class *r)` - supports Serializable or `Class->shared_from_this()` wrapping.
- [√] `Class& from_json()` - return `*shared_ptr<Class>`, throws on `nullptr`. **!!!**
- [√] `to_json (const Class &v)` - return `to_json<Class*>()`
- [√] No uses are made of copy-ctor implementations.
- [√] Need virtual ID serialization API on InstanceMap.
- [√] Add `jsonvalue_as_string()` for debugging purposes.

1.2.1 Callback Handling

Javascript can register/unregister remote Callbacks with *create* and *remove*. C++ sends events to inform about a remote Callback being *called* or unregistered *killed*.

```
void    Jsonapi/Trigger/create (id);     // JS->C++
void    Jsonapi/Trigger/remove (id);     // JS->C++
void    Jsonapi/Trigger/_<id>  ([...]);  // C++->JS
void    Jsonapi/Trigger/killed (id);     // C++->JS
```

1.3 Ase Class Inheritance Tree

```
Ase::SharedBase
   |
   +Ase::Emittable
      |
      +Ase::Property
      |  |
      |  +Ase::Properties::LambdaPropertyImpl
      |
      +Ase::Object
         |
         +Ase::Gadget
         |  |
         |  +Ase::Device
         |  |  |
         |  |  +Ase::NativeDevice
         |  |  |  |
         |  |  |  +Ase::NativeDeviceImpl
         |  |  |
         |  |  +Ase::Track
         |  |  |  |
         |  |  |  +Ase::TrackImpl
         |  |  |
         |  |  +Ase::Project
         |  |  |  |
         |  |  |  +Ase::ProjectImpl
         |  |  |
         |  |  +Ase::ClapDeviceImpl
         |  |
         |  +Ase::Clip
         |  |  |
         |  |  +Ase::ClipImpl
         |  |
         |  +Ase::Monitor
```

```
|  |  |
|  |  +Ase::MonitorImpl
|  |
|  +Ase::Server
|  |  |
|  |  +Ase::ServerImpl
|  |
|  +Ase::GadgetImpl
|
+Ase::ResourceCrawler
   |
   +Ase::FileCrawler
```

2 Web Component Implementations

The user interface components used in Anklang are implemented as custom HTML elements and are generally composed of a single file that provides:

a) A brief documentation block;
b) CSS style information, that is extracted at build time via JsExtract;
c) An HTML layout specified with lit-html expressions;
d) An assorted JavaScript class that defines a new custom HTML element, possibly via Lit.

Simple components that have no or at most one child element and do not require complex HTML layouts with lit-html can be implemented directly via customElements.define().

Components with complex layouts that need lit-html or that act as containers with several HTMLSlotElements (for multiple types of children) which require a ShadowRoot, should be implemented as LitElements by extending LitComponent (our convenience wrapper around LitElement.

Note that a Lit component is an HTML element, it extends ReactiveElement which always extends HTMLElement and none of the other HTML element interfaces.

2.1 Legacy Vue Components

Some components are still implemented via Vue and are slowly phased out. We often use `<canvas>` elements for Anklang specific displays, and Vue canvas handling comes with certain caveats:

1) Use of the `Util.vue_mixins.dom_updates` mixin (now default) allows to trigger the `dom_update()` component method for `$forceUpdate()` invocations and related events.
2) A `methods: { dom_update() {}, }` component entry should be provided that triggers the actual canvas rendering logic.
3) Using a `document.fonts.ready` promise, Anklang re-renders all Vue components via `$forceUpdate()` once all webfonts have been loaded, `<canvas>` elements containing text usually need to re-render themselves in this case.

Envue components:

Envue components are created to simplify some of the oddities of dealing with Vue-3 components. The function `Envue.Component.vue_export` creates a Vue component definition, so that the Vue component instance (`$vm`) is tied to an `Envue.Component` instance (`$object`). Notes:

- The Vue lifetime component can be accessed as `$object.$vm`.
- The Envue component can be accessed as `$vm.$object`.
- Accesses to `$vm.*` fields e.g. from within a `<template/>` definition are forwarded to access `$object.*` fields.
- Vue3 components are Proxy objects, *but* assignments to these Proxy objects is *not* reactive.
- To construct reactive instance data with async functions, use `observable_from_getters()`.

Vue uses a template compiler to construct a render() function from HTML `<template/>` strings. The Javascript expressions in templates are sandboxed and limited in scope, but may refer to Vue component properties that are exposed through `hasOwnProperty()`. In order to support Envue instance methods and fields in template expressions, all members present after Envue construction are forwarded into the Vue component.

3 UI Component Reference

3.1 BAboutDialog

The <b-aboutdialog> element is a modal [b-dialog] that displays version information about Anklang.

3.1.1 Events:

close
A *close* event is emitted once the "Close" button activated.

3.2 BPushButton

The <push-button> element is a wrapper for an ordinary HTMLElement. It is styled like a <button> and can behave like it, but cannot become a focus element.

3.3 BHFlex

The <h-flex> element is a horizontal flex container element. See also the Flex visual cheatsheet.

3.4 BVFlex

The <v-flex> element is a vertical flex container element. See also the Flex visual cheatsheet.

3.5 BGrid

The <c-grid> element is a simple grid container element. See also Grid Container and the Grid visual cheatsheet.

3.6 BButtonBar

The <b-buttonbar> element is a container for tight packing of buttons.

3.6.1 Slots:

slot All contents passed into this element will be packed tightly and styled as buttons.

3.7 BChoiceInput

The <b-choiceinput> element provides a choice popup to choose from a set of options. It supports the Vue v-model protocol by emitting an `input` event on value changes and accepting inputs via the `value` prop.

3.7.1 Props:

value
Integer, the index of the choice value to be displayed.
choices
List of choices: `[ { icon, label, blurb }... ]`

3.7.2 Events:

valuechange
Event emitted whenever the value changes, which is provided as `event.target.value`.

3.7.3 BClipList class

***class* BClipList**
The <b-cliplist> element container holds BClipView elements.

3.8 BClipView

The <b-clipview> element displays a small view of a MIDI clip.

3.9 BContextMenu

The <b-contextmenu> element implements a modal popup that displays contextmenu choices, see BMenuItem, BMenuRow, BMenuTitle and BMenuSeparator. Using the `popup()` method, the menu can be popped up from the parent component, and setting up a `.activate` handler can be used to handle menuitem actions. Example:

```
<div @contextmenu="e => querySelector('b-contextmenu').popup">
  <b-contextmenu .activate="menuactivation">...</b-contextmenu>
</div>
```

Note that keyboard presses, mouse clicks, drag selections and event bubbling can all cause menu item clicks and contextmenu activation. In order to deduplicate multiple events that arise from the same user interaction, *one* popup request and *one* click activation is processed per animation frame.

3.9.1 Properties:

.activate(uri)
Property callback which is called once a menu item is activated.
***.isactive (uri)* -> bool**
Property callback used to check if a particular menu item should stay active or be disabled.

3.9.2 Attributes:

xscale
Consider a wider area than the context menu width for popup positioning.
yscale
Consider a taller area than the context menu height for popup positioning.

3.9.3 Events:

click (event)
Event signaling activation of a menu item, the `uri` can be found via `event.target.uri`.
close (event)
Event signaling closing of the menu, regardless of whether menu item activation occoured or not.

3.9.4 Methods:

popup (event, { origin, data-contextmenu })
Popup the contextmenu, propagation of `event` is halted and the event coordinates or target is used for positioning unless `origin` is given.
The `origin` is a reference DOM element to use for drop-down positioning.
The `data-contextmenu` element (or `origin`) has the `data-contextmenu=true` attribute assigned during popup.
close()
Hide the contextmenu.

map_kbd_hotkeys (active)
Activate (deactivate) the kbd=... hotkeys specified in menu items.

3.9.5 BContextMenu class

***class* BContextMenu**

…

map_kbd_hotkeys (*active*)
Activate or disable the kbd=... hotkeys in menu items.
find_menuitem (*uri*)
Find a menuitem via its uri.

3.10 DataBubbleImpl

The **DataBubbleImpl** and **DataBubbleIface** classes implement the logic required to extract and display data-bubble="" tooltip popups on mouse hover.

3.10.1 DataBubbleIface class

***class* DataBubbleIface**

…

update (*element, text*)
Set the data-bubble attribute of element to text or force its callback
callback (*element, callback*)
Assign a callback function to fetch the data-bubble attribute of element
force (*element*)
Force data-bubble to be shown for element
unforce (*element*)
Cancel forced data-bubble for element
clear (*element*)
Reset the data-bubble attribute, its callback and cancel a forced bubble

3.10.2 Component class

***class* Component**
Component base class for wrapping Vue components.Let this.$vm point to the Vue component, and $vm.$object point to this.

***new* Component (*vm*)**
Let this.$vm point to the Vue component, and $vm.$object point to this.
update ()
Force a Vue component update.
observable_from_getters (*tmpl*)
Wrapper for Util.observable_from_getters().
[$watch]{#_watch data-4search="ui/b/envue.js:$watch;func"} (*args*)
Wrapper for Vue.$watch
vue_export (*vue_object*) [static]
Create a Vue options API object from *vue_object* for SFC exports.

3.10.3 Functions

forward_access (*vm, classinstance, ignores*)
Forward all accesses to fields on vm to access fields on classinstance.

vue_export_from_class (*Class, vue_object*)
Create a Vue options API object that proxies access to a newly created `Class` instance.

3.11 BIcon

The <b-icon> element displays icons from various icon fonts. In order to style the color of icon font symbols, simply apply the `color` CSS property to this element (styling `fill` as for SVG elements is not needed).

3.11.1 Props:

iconclass
A CSS class to apply to this icon.
ic A prefixed variant of `fa`, `bc`, `mi`, `uc`.
Either a prefixed icon font symbol or a unicode character literal, see the Unicode Lists and Symbols.
The 'bc-' prefix indicates an icon from the "AnklangIcons Font" symbols.
The 'fa-' prefix indicates an icon from the "Fork Awesome" collection (compatible with "Font Awesome 4"), see the Fork Awesome Icons.
The 'mi-' prefix indicates an icon from the "Material Icons" collection, see the Material Design Icons.
nosize
Prevent the element from applying default size constraints.
fw Apply fixed-width sizing.
lg Make the icon 33% larger than its container.
hflip
Flip the icon horizontally.
vflip
Flip the icon vertically.

3.12 BKnob

The <b-knob> element provides a knob for scalar inputs. It supports the Vue v-model protocol by emitting an `input` event on value changes and accepting inputs via the `value` prop.

3.12.1 Props:

bidir
Boolean, flag indicating bidirectional inputs with value range `-1…+1`.
value
Float, the knob value to be displayed, the value range is `0…+1` if *bidir* is `false`.
format
String, format specification for popup bubbles, containinig a number for the peak amplitude.
label
String, text string for popup bubbles.
hscroll
Boolean, adjust value with horizontal scrolling (without dragging).
vscroll
Boolean, adjust value with vertical scrolling (without dragging).
width4height
Automatically determine width from externally specified height (default), otherwise determines height.

3.12.2 Implementation Notes

The knob is rendered based on an SVG drawing, which is arranged in such a way that adding rotational transforms to the SVG elements is sufficient to display varying knob levels. Chrome cannot render individual SVG nodes into seperate layers (GPU textures) so utilizing GPU acceleration requires splitting the original SVG into several SVG layers, each of which can be utilized as a seperate GPU texture with the CSS setting `will-change: transform`.

3.12.3 Functions

`spin_drag_start (`*element, event, value_callback*`)`
Setup drag handlers for numeric spin button behavior.

`spin_drag_stop (`*event_or_element*`)`
Stop sping drag event handlers and pointer grab.

`spin_drag_pointermove (`*event*`)`
Handle sping drag pointer motion.

`spin_drag_change ()`
Turn accumulated spin drag motions into actual value changes.

`spin_drag_granularity (`*event*`)`
Calculate spin drag acceleration (slowdown) from event type and modifiers.

3.13 BMenuBar

The <b-menubar> element contains main menus at the top of the window.

3.14 BMenuItem

The <b-menuitem> element can be used as a descendant of a BContextMenu. The menuitem can be activated via keyboard focus or mouse click and will notify its BContextMenu about the click and its `uri`. If no `uri` is specified, the BContextMenu will still be notified to be closed, unless `$event.preventDefault()` is called.

3.14.1 Properties:

***.isactive*: bool (uri)**
Property callback used to check if a particular menu item should stay active or be disabled.

3.14.2 Attributes:

uri Unique identifier for this menu item.
kbd Hotkey to acivate the menu item, displayed next to the menu item label.
disabled
Boolean flag indicating disabled state.
ic Shorthands icon properties that are forwarded to a B-ICON used inside the menuitem.

3.14.3 Events:

click
Event emitted on keyboard or mouse activation, use `event.stopPropagation()` to prevent bubbling to the contextmenu.

3.14.4 Slots:

default
All contents passed into this slot will be rendered as contents of this element.

3.15 BMenuRow

The <b-menurow> element can contain BMenuItem elements, that are packed horizontally inside a menurow.

3.15.1 Props:

noturn
Avoid turning the icon-label direction in menu items to be upside down.

3.15.2 Slots:

default
All contents passed into this slot will be rendered as contents of this element.

3.16 BMenuSeparator

The <b-menuseparator> element is a menu element that serves as a visual separator between other elements.

3.17 BMenuTitle

The <b-menutitle> element can be used as menu title inside a BContextMenu.

3.17.1 Slots:

default
All contents passed into this slot will be rendered as contents of this element.

3.18 BMore

The <b-more> element is an indicator for adding or dropping new UI elements.

3.19 BNumberInput

The <b-numberinput> element is a field-editor for integer or floating point number ranges. The input `value` will be constrained to take on an amount between `min` and `max` inclusively.

3.19.1 Properties:

value
Contains the number being edited.
min The minimum amount that `value` can take on.
max The maximum amount that `value` can take on.
step A useful amount for stepwise increments.
allowfloat
Unless this setting is `true`, numbers are constrained to integer values.
readonly
Make this component non editable for the user.

3.19.2 Events:

valuechange
Event emitted whenever the value changes, which is provided as `event.target.value`.

3.20 BObjectEditor

The <b-objecteditor> element is a field-editor for object input. A copy of the input value is edited, update notifications are provided via an `input` event.

3.20.1 Properties:

value
: Object with properties to be edited.

readonly
: Make this component non editable for the user.

3.20.2 Events:

input
: This event is emitted whenever the value changes through user input or needs to be constrained.

3.21 BPartList

The <b-partlist> element allows to arrange Clip objects for playback.

3.21.1 Constants

`list_actions`
: List menu actions for PianoRoll.

3.21.2 Functions

`ntool (`***toolmode, drag_event, cursor, predicate***`)`
: Add/register piano-roll canvas tool

`notes_canvas_drag_select (`***event, MODE***`)`

3.21.3 Select Tool

Crosshair The Select Tool allows selection of single notes or groups of notes. Modifier keys can be used to modify the selection behavior.

3.21.4 Horizontal Select

`notes_canvas_drag_paint (`***event, MODE***`)`

3.21.5 Paint Tool

Pen With the note Paint Tool, notes can be placed everywhere in the grid by clicking mouse button 1 and possibly keeping it held during drags.

`notes_canvas_drag_move (`***event, MODE***`)`

3.21.6 Move Tool

Pen With the note Move Tool, selected notes can be moved clicking mouse button 1 and keeping it held during drags. A copy will be made instead of moving the selected notes if the ctrl key is pressed during drag.

notes_canvas_drag_resize (*event, MODE*)

3.21.6.1 Resizing Notes

H-Resize When the Paint Tool is selected, the right edge of a note can be draged to make notes shorter or longer in duration.

notes_canvas_drag_erase (*event, MODE*)

3.21.7 Erase Tool

Eraser The Erase Tool allows deletion of all notes selected during a mouse button 1 drag. The deletion can be aborted by the Escape key.

note_hover_body (*coords, tick, key, notes*)
Detect note if hovering over its body

note_hover_tail (*coords, tick, key, notes*)
Detect note if hovering over its tail

note_hover_head (*coords, tick, key, notes*)
Detect note if hovering over its head

notes_canvas_tool_from_hover (*piano_roll, pointerevent*)
Get drag tool and cursor from hover position

target_coords (*event, target*)
Translate event offsetX,offsetY into taret element

3.22 BPianoRoll

The <b-piano-roll> element allows note editing.

3.22.1 Functions

piano_layout ()
Determine layout in pixels.
set_canvas_font (*ctx, size*)
Assign canvas font to drawing context
paint_piano ()
Paint piano key canvas
paint_notes ()
Paint piano roll notes
paint_timeline ()
Paint timeline digits and indicators
paint_timegrid (*canvas, with_labels*)
Paint timegrid into any canvas

3.23 BPlayControls

The <b-playcontrols> element is a container holding the play and seek controls for a Ase.song.

3.24 BPositionView

The <b-positionview> element displays the project transport position pointer and related information.

3.24.1 Props:

- **project** - The object providing playback API.

3.25 BStatusBar

The <b-statusbar> element is an area for the display of status messages and UI configuration.

3.26 BSwitchInput

The <b-switchinput> element is a field-editor switch to change between on and off.

3.26.1 Properties:

value
Contains a boolean indicating whether the switch is on or off.
readonly
Make this component non editable for the user.

3.26.2 Events:

valuechange
Event emitted whenever the value changes, which is provided as `event.target.value`.

3.27 BTextInput

The <b-textinput> element is a field-editor for text input.

3.27.1 Properties:

value
Contains the text string being edited.
readonly
Make this component non editable for the user.

3.27.2 Events:

valuechange
Event emitted whenever the value changes, which is provided as `event.target.value`.

3.28 BToggle

The <b-toggle> element implements a simple toggle button for boolean audio processor input properties. Its `value` can be accessed as a property and `valuechange` is emitted on changes.

3.28.1 Props:

value
Boolean, the toggle value to be displayed, the values are `true` or `false`.
label
String, label to be displayed inside the toggle button.

3.28.2 Events:

valuechange
Event emitted whenever the value changes, which is provided as `event.target.value`.

3.28.3 HueSaturation class

***class* HueSaturation**
Class with logic and spline approximations to calculate ZHSV.

3.28.4 Functions

`srgb_from_hsv` (*hue, saturation, value*)
Calculate sRGB from `hue, saturation, value`.
`hsv_from_srgb` (*srgb*)
Calculate `{ hue, saturation, value }` from `srgb`.
`zhsl_from` (*srgb, gamut*)
Calculate `{ hue, saturation, lightness }` from `srgb`.
`srgb_from_zhsv` (*hue, saturation, value, gamut*)
Calculate sRGB from `{ hue, saturation, value }`.
`hex_from_zhsv` (*hue, saturation, value, gamut*)
Calculate hexadecimal color from `{ hue, saturation, value }`.
`zhsv_from` (*srgb, gamut*)
Calculate `{ hue, saturation, value }` from `srgb`.
`srgb_from_zhsl` (*hue, saturation, lightness, gamut*)
Calculate sRGB from `{ hue, saturation, value }`.
`hex_from_zhsl` (*hue, saturation, lightness, gamut*)
Calculate hexadecimal color from `{ hue, saturation, value }`.
`color2rgba` (*color*)
Yield [R, G, B, A] from `color`.
`zmod_assignop` (*col, prop, op, num, perc*)
Re-assign a specific color property.
`zmod` (*colorlike, mods*)
Apply a variety of modifications to an input color.
`zmod4` (*colorlike*)
Find zmod() for a color.
`zlerp` (*c1, c2, t*)
Interpolate between 2 colors.
`lgrey` (*lightness*)
Yield a grey tone with CIELAB lightness.

3.29 WorkerClip

Ase::Clip proxy

3.30 WorkerHost

Global `host` instance for scripts.

3.30.1 WorkerHost class

***class* WorkerHost**
…

`piano_roll_clip` ()
Create handle for the currently selected clip.
`script_name` ()
Retrieve the file name of the current user script.
`script_uuid` ()
Retrieve the UUID set via use_api().

`api_level ()`
: Retrieve numeric API level supported by this runtime.

`use_api (`*`api_level, script_uuid`*`)`
: Request use of API level `api_level` and provide UUID for controllers.

`register (`*`category, label, fun, blurb, params`*`)`
: Register a script function.

4 NAME

`jsextract.js` - Script to extract snippets marked with JsExtract

5 SYNOPSIS

node **jsextract.js** [*OPTIONS*] [*file.js*...]

6 DESCRIPTION

The **jsextract.js** processes its input files by looking for markers such as `JsExtract.css`...`` and extracting the template string contents into a corresponding `*.jscss` file. This can be preprocessed or copied into a corresponding `*.css` file, to be loaded via `JsExtract.fetch_css()`.

7 OPTIONS

-O <*dir*>
Specify the directory for output files.

7.1 FocusGuard

Install a FocusGuard to allow only a restricted set of elements to get focus.

7.1.1 Constants

`kbd_modifiers`
Keyboard modifiers used for hotkeys.
`KeyCode`
Symbolic names for key codes

7.1.2 Functions

`display_keyname (`***keyname***`)`
Create display name from KeyEvent.code names.
`hotkey_name_from_event (`***event***`)`
Create hotkey name from KeyboardEvent.
`match_key_event (`***event, keyname***`)`
Match an event's key code, considering modifiers.
`is_navigation_key_code (`***keycode***`)`
Check if a key code is used of rnavigaiton (and non alphanumeric).
`activeElement ()`
Get the currently focussed Element, also inspecting open shadowRoot hierarchies.
`list_focusables (`***element***`)`
List elements that can take focus including shadow DOMs and are descendants of `element` or the document
`push_focus_root (`***element, escapecb***`)`
Constrain focus to `element` and its descendants
`remove_focus_root (`***element***`)`
Remove an `element` previously installed via push_focus_root()
`element_midpoint (`***element***`)`
Compute global midpoint position of element, e.g. for focus movement
`keydown_move_focus (`***event***`)`
Move focus on UP/DOWN/HOME/END `keydown` events
`move_focus (`***dir, subfocus***`)`
Move focus to prev or next focus widget
`forget_focus (`***element***`)`
Forget the last focus element inside `element`
`add_hotkey (`***hotkey, callback, subtree_element***`)`
Add a global hotkey handler.
`remove_hotkey (`***hotkey, callback***`)`
Remove a global hotkey handler.
`add_key_filter (`***keycode, callback***`)`
Add a global keymap.
`remove_key_filter (`***keycode***`)`
Remove a global keymap.
`add_keymap (`***keymap***`)`
Add a global keymap.

remove_keymap (*keymap*)
Remove a global keymap.
is_button_input (*element*)
Check if element is button-like input
is_nav_input (*element*)
Check if element has inner input navigation
shortcut_lookup (*mapname, label, shortcut*)
Lookup shortcut for label in mapname, default to shortcut
shortcut_dialog (*mapname, label, shortcut*)
Display shortcut editing dialog

7.1.3 LitComponent class

***class* LitComponent**
A LitElement with reactive render() and updated() methods.

7.1.4 Constants

JsExtract
API to mark template strings for extraction and fetch extracted assets.

- JsExtract.css`body { font-weight:bold; }`
  Mark CSS text inside a Javascript file for extration by jsextract.js.
- node jsextract.js inputfile.js
  Extract JsExtract.css`` strings into inputfile.jscss.
- await JsExtract.fetch_css ('/path/to/inputfile.js');
  Use the browser fetch() API to retrieve the extracted string as text/css from '/path/to/inputfile.css'.
- JsExtract.fetch_css (import.meta);
  Variant of the above that utilizes import.meta.url.
- JsExtract.css_url (import.meta);
  Variant of the above that just provides the stylesheet URL.

7.1.5 Functions

lit_update_all (*root*)
Call requestUpdate() on all LitElements
css_url (*base_url*)
Construct the stylesheet URL from a base URL (enforcing .css extension).
fetch_css (*base_url*)
Fetch (extracted) asset from a base URL (enforcing .css extension) as "text/css"

7.1.6 Functions

tofloat (*value, fallback, min, max*)
Yield value as number in [min..max], converts percentages.
load_import (*id, cwd, opts*)
Load import files in nodejs to implement @import.
find_import (*id, cwd, opts*)
Provide canned CSS files or fetch css for use with @import.
add_import (*filename, csstext*)
Provide filename as @import "filename"; source with contents csstext.

7.2 ScriptHost

A ScriptHost object represents a Worker script in the Main thread.

A ScriptHost object (defined in `script.js`) runs in the Main Javascript thread. It starts a Worker via `host.js` which creates a WorkerHost object in the Worker thread. The ScriptHost<->WorkerHost objects bridge needed API from the Main thread to the Worker thread. The WorkerHost then loads and calls into a user provided ES Module, the actual user script which communicates via the WorkerHost global variable `host`. See also #WorkerHost.

7.2.1 Functions

`count_newlines` (***str***)
: Count newlines

`find_tags` (***string***)
: List tag names matching `^</tag>` in `string`.

`process_file` (***filename, config***)
: Extract and process sections of an SFC file.

`write_style` (***filename, ofile, config, stylestring***)
: Process and write style information

7.2.2 Functions

`jsonapi_finalization_registration` (***object***)
: Jsonipc handler for object creation

`jsonapi_finalization_gc` (***gcinfo***)
: Jsonipc handler for IDs of GC-ed objects.

7.2.3 PointerDrag class

class **PointerDrag**
: Meld all pointer drag handling functions into a single `drag_event(event,MODE)` method.

7.2.4 vue_mixins class

class **vue_mixins**
: …

  `vuechildren () [static]`
  : Provide `$children` (and `$vue_parent`) on every component.

  `autodataattrs () [static]`
  : Automatically add `$attrs['data-*']` to `$el`.

  `dom_updates () [static]`
  : Vue mixin to provide DOM handling hooks. This mixin adds instance method callbacks to handle dynamic DOM changes such as drawing into a `<canvas/>`. Reactive callback methods have their data dependencies tracked, so future changes to data dependencies of reactive methods will queue future updates. However reactive dependency tracking only works for non-async methods.

    - dom_create() - Called after `this.$el` has been created
    - dom_change() - Called after `this.$el` has been reassigned or changed. Note, may also be called for `v-if="false"` cases.
    - dom_update() - Reactive callback method, called with a valid `this.$el` and after Vue component updates. Dependency changes result in `this.$forceUpdate()`.

- dom_draw() - Reactive callback method, called during an animation frame, requested via `dom_queue_draw()`. Dependency changes result in `this.dom_queue_draw()`.
- dom_queue_draw() - Cause `this.dom_draw()` to be called during the next animation frame.
- dom_destroy() - Callback method, called once `this.$el` is removed.

7.2.5 Constants

`ResizeObserver`
Work around FireFox 68 having ResizeObserver disabled

`clone_descriptors`
Copy PropertyDescriptors from source to target, optionally binding handlers against closure.

7.2.6 Functions

`now ()`
Retrieve current time in milliseconds.

`frame_stamp ()`
Retrieve a timestamp that is unique per (animation) frame.

`debounce (`*`callback, options`*`)`
Yield a wrapper function for `callback` that throttles invocations. Regardless of the frequency of calls to the returned wrapper, `callback` will only be called once per `requestAnimationFrame()` cycle, or after `milliseconds`. The return value of the wrapper functions is the value of the last callback invocation. A `cancel()` method can be called on the returned wrapper to cancel the next pending `callback` invocation. Options:

- `wait` - number of milliseconds to pass until `callback` may be called.
- `restart` - always restart the timer once the wrapper is called.
- `immediate` - immediately invoke `callback` and then start the timeout period.

`capture_event (`*`eventname, callback`*`)`
Process all events of type `eventname` with a single callback exclusively

`coalesced_events (`*`event`*`)`
Expand pointer events into a list of possibly coalesced events.

`vue_component (`*`element`*`)`
Get Vue component handle from element or its ancestors

`envue_object (`*`element`*`)`
Get Envue `$object` from element or its ancestors

`drag_event (`*`event`*`)`
Start `drag_event (event)` handling on a Vue component's element, use `@pointer-down="drag_event"`

`unrequest_pointer_lock (`*`element`*`)`
Clear (pending) pointer locks Clear an existing pointer lock on `element` if any and ensure it does not get a pointer lock granted unless request_pointer_lock() is called on it again.

`has_pointer_lock (`*`element`*`)`
Check if `element` has a (pending) pointer lock Return:

- 2- if `element` has the pointer lock;
- 1- if the pointer lock is pending;
- 0- otherwise.

request_pointer_lock (*element*)
Request a pointer lock on element and track its state Use this function to maintain pointer locks to avoid stuck locks that can get granted *after* exitPointerLock() has been called.

adopt_style (*element, csstext, stylesheet_name*)
Ensure the root node of element contains a csstext (replaceable via stylesheet_name)

add_style_sheet (*element, url*)
Ensure the root node of element has a url stylesheet link.

vm_scope_selector (*vm*)
Retrieve CSS scope selector for vm_scope_style()

vm_scope_style (*vm, css*)
Attach css to Vue instance vm, use vm_scope_selector() for the vm CSS scope

assign_forin (*target, source*)
Loop over all properties in source and assign to ‘target

assign_forof (*target, source*)
Loop over all elements of source and assign to ‘target

array_remove (*array, item*)
Remove element item from array if present via indexOf

array_index_equals (*array, item*)
Find array index of element that equals item

map_from_kvpairs (*kvarray*)
Generate map by splitting the key=value pairs in kvarray

range (*bound, end, step*)
Generate integers [0..bound[if one arg is given or [bound..end[by incrementing step.

freeze_deep (*object*)
Freeze object and its properties

copy_deep (*src*)
Create a new object that has the same properties and Array values as src

equals_recursively (*a, b*)
Check if a == b, recursively if the arguments are of type Array or Object

clamp (*x, min, max*)
Return 1 x clamped into 1 min and 1 max.

fwdprovide (*injectname, keys*)
Create a Vue component provide() function that forwards selected properties.

hyphenate (*string*)
Generate a kebab-case (’two-words’) identifier from a camelCase (’twoWords’) identifier

weakid (*object*)
Fetch a unique id for any object.

weakid_lookup (*id*)
Find an object from its unique id.

join_classes (*args*)
Join strings and arrays of class lists from args.

object_zip (*obj, keys, values*)
Assign obj[k] = v for all k of keys, v of values.

object_await_values (*obj*)
Await and reassign all object fields.

extend_property (*prop, disconnector, augment*)
Extend Ase.Property objects with cached attributs. Note, for automatic `.value_` updates, a `disconnector` function must be provided as second argument, to handle disconnection of property change notifications once the property is not needed anymore.

promise_state (*p*)
Extract the promise `p` state as one of: 'pending', 'fulfilled', 'rejected'

compile_expression (*expression, context*)
Turn a JS `$event` handler expression into a function. This yields a factory function that binds the scope to create an expression handler.

VueifyObject (*object, vue_options*)
VueifyObject - turn a regular object into a Vue instance. The *object* passed in is used as the Vue `data` object. Properties with a getter (and possibly setter) are turned into Vue `computed` properties, methods are carried over as `methods` on the Vue() instance.

fnv1a_hash (*str*)
Produce hash code from a String, using an FNV-1a variant.

split_comma (*str*)
Split a string when encountering a comma, while preserving quoted or parenthesized segments.

escape_html (*unsafe*)
Properly escape test into `&` and related sequences.

parse_hex_color (*colorstr*)
Parse hexadecimal CSS color with 3 or 6 digits into [R, G, B].

parse_hex_luminosity (*colorstr*)
Parse hexadecimal CSS color into luminosity.

parse_hex_brightness (*colorstr*)
Parse hexadecimal CSS color into brightness.

parse_hex_pgrey (*colorstr*)
Parse hexadecimal CSS color into perception corrected grey.

parse_hex_average (*colorstr*)
Parse hexadecimal CSS color into average grey.

parse_colors (*colorstr*)
Parse CSS colors (via invisible DOM element) and yield an array of rgba tuples.

compute_style_properties (*el, obj*)
Retrieve a new object with the properties of `obj` resolved against the style of `el`

inside_display_none (*element*)
Check if `element` or any parentElement has `display:none`

is_displayed (*element*)
Check if `element` is displayed (has width/height assigned)

wheel_delta (*ev*)
Retrieve normalized scroll wheel event delta in CSS pixels (across Browsers) This returns an object `{x,y}` with negative values pointing LEFT/UP and positive values RIGHT/DOWN respectively. For zoom step interpretation, the x/y pixel values should be reduced via `Math.sign()`. For scales the pixel values might feel more natural, because browsers sometimes increase the

number of events with increasing wheel distance, in other cases values are accumulated so fewer events with larger deltas are sent instead.

wheel2scrollbars (*event, refs, scrollbars*)
Use deltas from `event` to call scrollBy() on `refs[scrollbars...]`.

setup_shield_element (*shield, containee, closer, capture_escape*)
Setup Element shield for a modal containee. Capture focus movements inside `containee`, call `closer(event)` for pointer clicks on `shield` or when `ESCAPE` is pressed.

fullstop (*event*)
Stop (immediate) propagation and prevent default handler for an event.

swallow_event (*type, timeout*)
Use capturing to swallow any `type` events until `timeout` has passed

prevent_event (*event_or_null*)
Prevent default or any propagation for a possible event.

dialog_backdrop_mousedown (*ev*)
Close dialog on backdrop clicks via hiding at mousedown

dialog_backdrop_mouseup (*ev*)
Close dialog on backdrop clicks via actual closing at mouseup

dialog_backdrop_autoclose (*dialog, install_or_remove*)
Install handlers to close a dialog on backdrop clicks.

popup_position (*element, opts*)
Determine position for a popup

resize_canvas (*canvas, csswidth, cssheight, fill_style*)
Resize canvas display size (CSS size) and resize backing store to match hardware pixels

dash_xto (*ctx, x, y, w, d*)
Draw a horizontal line from (x,y) of width `w` with dashes `d`

hstippleRect (*ctx, x, y, width, height, stipple*)
Draw a horizontal rect `(x,y,width,height)` with pixel gaps of width `stipple`

roundRect (*ctx, x, y, width, height, radius, fill, stroke*)
Fill and stroke a canvas rectangle with rounded corners.

gradient_apply_stops (*grad, stoparray*)
Add color stops from `stoparray` to `grad`, `stoparray` is an array: [(offset,color)...]

linear_gradient_from (*ctx, stoparray, x1, y1, x2, y2*)
Create a new linear gradient at (x1,y1,x2,y2) with color stops `stoparray`

canvas_ink_vspan (*font_style, textish*)
Measure ink span of a canvas text string or an array

midi_label (*numish*)
Retrieve the 'C-1' .. 'G8' label for midi note numbers

align8 (*int*)
Align integer value to 8.

telemetry_subscribe (*fun, telemetryfields*)
Call `fun` for telemtry updates, returns unsubscribe handler.

telemetry_unsubscribe (*telemetryobject*)
Call `fun` for telemtry updates, returns unsubscribe handler.

in_keyboard_click ()
Check if the current click event originates from keyboard activation.

keyboard_click_event (*fallback*)
Retrieve event that triggers keyboard_click().

keyboard_click (*element, event, callclick*)
Trigger element click via keyboard.

in_array (*element, array*)
Check whether `element` is contained in `array`

matches_forof (*element, iteratable*)
Check whether `element` is found during `for (... of iteratable)`

element_text (*element, filter*)
Extract filtered text nodes from Element.

clone_menu_icon (*menu, uri, title*)
Clone a menuitem icon via its `uri`.

keyboard_map_name (*keyname*)
Retrieve user-printable name for a keyboard button, useful to describe KeyboardEvent.code.

has_ancestor (*node, ancestor, escape_shadowdom*)
Check if `ancestor` is an ancestor of `node`, maybe including shadowRoot elements.

closest (*element, selector*)
Find the closest element or parent matching `selector`, traversing shadow DOMs.

root_ancestor (*element*)
Retrieve root ancestor of `element`

find_element_from_point (*root, x, y, predicate, visited*)
Find an element at `(x,y)` for which `predicate (element)` is true.

create_note (*text, timeout*)
Show a notification popup, with adequate default timeout

markdown_to_html (*element, markdown_text*)
Generate `element.innerHTML` from `markdown_text`

assign_async_cleanup (*map, key, cleaner*)
Assign `map[key] = cleaner`, while awaiting and calling any previously existing cleanup function

observable_force_update ()
Method to be added to a `observable_from_getters()` result to force updates.

observable_from_getters (*tmpl, predicate*)
Create a reactive dict from the fields in `tmpl` with async callbacks.

Once the resolved result from `predicate()` changes and becomes true-ish, the `getter()` of each field in `tmpl` is called, resolved and assigned to the corresponding field in the observable binding returned from this function. Optionally, fields may provide a `notify` setup handler to install a notification callback that re-invokes the `getter`. A destructor can be returned from `notify()` once resolved, that is executed during cleanup phases. The `default` of each field in `tmpl` may provide an initial value before `getter` is called the first time and in case `predicate()` becomes false-ish. The first argument to `getter()` is a function that can be used to register cleanup code for the getter result.

```
const data = {
  val: { getter: c => async_fetch(), notify: n => add_listener (n), },
};
dict = this.observable_from_getters (data, () => this.predicate());
// use dict.val
```

When the `n()` callback is called, a new *getter* call is scheduled. A handler can be registered with `c (cleanup);` to cleanup resources left over from an `async_fetch()` call.

`tmplstr (a, e)`
Join template literal arguments into a String

`strpad (string, len, fill)`
Pad `string` with `fill` until its length is `len`

`lrstrip (str)`
Strip whitespace from start and end of string.

`collect_text_content (node_or_array)`
Gather text content from `node_or_array`.

`hash53 (key, seed)`
Generate 53-Bit hash from `key`.

`add_destroy_callback (callback)`
Add a `callback` to `this` to be called from `call_destroy_callbacks()`.

`del_destroy_callback (callback)`
Remove a `callback` from `this`, previously added via `add_destroy_callback()`.

`call_destroy_callbacks ()`
Call destroy callbacks of `this`, clears the callback list.

7.3 AseCachingWrapper

Caching wrapper for ASE classes

7.3.1 AseCachingWrapper class

***class* AseCachingWrapper**
…

`[__add__]{#add data-4search="ui/wrapper.js:add;func"} (prop, defaultvalue, callback)`
Add property to cache
`[__del__]{#del data-4search="ui/wrapper.js:del;func"} (prop, callback)`
Remove property caching request
`[__cleanup__]{#cleanup data-4search="ui/wrapper.js:cleanup;func"} ()`
Remove all references

7.3.2 Constants

`finalization_cleanup_registry`
FinalizationRegistry to call cleanup callback upon object destruction.

7.3.3 Functions

`wrap_ase_object (aseobj, fields, callback)`
Wrap AseObject to cache properties and support **add**, **cleanup** and auto cleanup.

define_reactive (*object, properties_object*)
Define reactive properties on `object`, to be used with reactive_wrapper(). See also Object.defineProperties.

reactive_wrapper (*effect, notifier, keepwatching*)
Make `effect()` wrapper to watch reactive properties, on changes run `notifier()`.

7.3.4 Functions

lrstrip (*str*)
Strip whitespace from start and end of string.

fix_indent (*txt*)
Remove C-comment style `\n\s\*\s`

extract_md (*txt*)
Look for a markdown documentation block

process_file (*filename, config*)
Extract and process block comments of a file

8 Releasing

Releases of the Anklang project are hosted on GitHub under Anklang Releases. A release encompasses a distribution tarball that has the release version number baked into the misc/version.sh script.

8.1 Versioning

The Anklang project uses **`MAJOR.MINOR.MICRO[.DEVEL][-SUFFIX]`** version numbers with the folloing uses:

- **`MAJOR`** - The major number is currently 0, so all bets are off. It is planned to signify major changes to users.
- **`MINOR`** - The minor number indicates significant changes, often these are user visible improvements.
- **`MICRO`** - The micro number increases with every release.
- **`DEVEL`** - The devel part is optional and increases with every new commit, it numbers builds between official releases. The presence of the `[.DEVEL]` part indicates a version ordered *after* its corresponding `MAJOR.MINOR.MICRO` release.
- **`SUFFIX`** - An optional suffix is sometimes used for e.g. release candidates. The presence of the `[-SUFFIX]` part indicates a version ordered *before* its corresponding `MAJOR.MINOR.MICRO` release.

Git tags are used to store release versions, development versions are derived from those tags similar to how `git describe` works. The current version can always be obtained by invoking `misc/version.sh`.

8.2 Release Assets

The script `misc/mkassets.sh` can be used to create and clean up a release build directory and it triggers the necessary rules to create a distribution tarball and to build the release assets. All assets are built from the distribution tarball without any Git dependency. Producing a distribution tarball depends on Git however.

A Appendix

A.1 One-dimensional Cubic Interpolation

With four sample values V_0, V_1, V_2 and V_3, cubic interpolation approximates the curve segment connecting V_1 and V_2, by using the beginning and ending slope, the curvature and the rate of curvature change to construct a cubic polynomial.

The cubic polynomial starts out as:

(1) $f(x) = w_3x^3 + w_2x^2 + w_1x + w_0$

Where $0 <= x <= 1$, specifying the sample value of the curve segment between V_1 and V_2 to obtain.

To calculate the coefficients $w_0, ..., w_3$, we set out the following conditions:

(2) $f(0) = V_1$
(3) $f(1) = V_2$
(4) $f'(0) = V_1'$
(5) $f'(1) = V_2'$

We obtain V_1' and V_2' from the respecting slope triangles:

(6) $V_1' = \frac{V_2 - V_0}{2}$
(7) $V_2' = \frac{V_3 - V_1}{2}$

With (6) → (4) and (7) → (5) we get:

(8) $f'(0) = \frac{V_2 - V_0}{2}$
(9) $f'(1) = \frac{V_3 - V_1}{2}$

The derivation of $f(x)$ is:

(10) $f'(x) = 3w_3x^2 + 2w_2x + w_1$

From $x = 0$ → (1), i.e. (2), we obtain w_0 and from $x = 0$ → (10), i.e. (8), we obtain w_1. With w_0 and w_1 we can solve the linear equation system formed by (3) → (1) and (5) → (10) to obtain w_2 and w_3.

(11) (3) → (1): $w_3 + w_2 + \frac{V_2 - V_0}{2} + V_1 = V_2$
(12) (5) → (10): $3w_3 + 2w_2 + \frac{V_2 - V_0}{2} = \frac{V_3 - V_1}{2}$

With the resulting coefficients:

$$
\begin{aligned}
w_0 &= V_1 && (\textit{initial value}) \\
w_1 &= \frac{V_2 - V_0}{2} && (\textit{initial slope}) \\
w_2 &= \frac{-V_3 + 4V_2 - 5V_1 + 2V_0}{2} && (\textit{initial curvature}) \\
w_3 &= \frac{V_3 - 3V_2 + 3V_1 - V_0}{2} && (\textit{rate change of curvature})
\end{aligned}
$$

Reformulating (1) to involve just multiplications and additions (eliminating power), we get:

(13) $f(x) = ((w_3x + w_2)x + w_1)x + w_0$

Based on $V_0, ..., V_3$, $w_0, ..., w_3$ and (13), we can now approximate all values of the curve segment between V_1 and V_2.

However, for practical resampling applications where only a specific precision is required, the number of points we need out of the curve segment can be reduced to a finite amount. Lets assume we require n equally spread values of the curve segment, then we can precalculate n sets of $W_{0,...,3}[i]$, $i = [0, ..., n]$, coefficients to speed up the resampling calculation, trading memory for computational performance. With $w_{0,...,3}$ in (1):

$$\begin{aligned} f(x) = & \frac{V_3 - 3V_2 + 3V_1 - V_0}{2}x^3 + \\ & \frac{-V_3 + 4V_2 - 5V_1 + 2V_0}{2}x^2 + \\ & \frac{V_2 - V_0}{2}x + \\ & V1 \end{aligned}$$

sorted for $V_0, ..., V_4$, we have:

(14)

$$\begin{aligned} f(x) = & V_3\,(0.5x^3 - 0.5x^2) + \\ & V_2\,(-1.5x^3 + 2x^2 + 0.5x) + \\ & V_1\,(1.5x^3 - 2.5x^2 + 1) + \\ & V_0\,(-0.5x^3 + x^2 - 0.5x) \end{aligned}$$

With (14) we can solve $f(x)$ for all $x = \frac{i}{n}$, where $i = [0, 1, 2, ..., n]$ by substituting $g(i) = f(\frac{i}{n})$ with

(15) $g(i) = V_3W_3[i] + V_2W_2[i] + V_1W_1[i] + V_0W_0[i]$

and using n precalculated coefficients $W_{0,...,3}$ according to:

$$\begin{aligned} m &= \frac{i}{n} \\ W_3[i] &= 0.5m^3 - 0.5m^2 \\ W_2[i] &= -1.5m^3 + 2m^2 + 0.5m \\ W_1[i] &= 1.5m^3 - 2.5m^2 + 1 \\ W_0[i] &= -0.5m^3 + m^2 - 0.5m \end{aligned}$$

We now need to setup $W_{0,...,3}[0, ..., n]$ only once, and are then able to obtain up to n approximation values of the curve segment between V_1 and V_2 with four multiplications and three additions using (15), given $V_0, ..., V_3$.

A.2 Modifier Keys

There seems to be a lot of inconsistency in the behaviour of modifiers (shift and/or control) with regards to GUI operations like selections and drag and drop behaviour.

According to the Gtk+ implementation, modifiers relate to DND operations according to the following list:

Table 1: GDK drag-and-drop modifier keys

Modifier		Operation	Note / X-Cursor
none	→	copy	(else move (else link))
SHIFT	→	move	GDK_FLEUR
CTRL	→	copy	GDK_PLUS, GDK_CROSS

Modifier		Operation	Note / X-Cursor
SHIFT+CTRL	→	link	GDK_UL_ANGLE

Regarding selections, the following email provides a short summary:

```
From: Tim Janik <timj@gtk.org>
To: Hacking Gnomes <Gnome-Hackers@gnome.org>
Subject: modifiers for the second selection
Message-ID: <Pine.LNX.4.21.0207111747190.12292-100000@rabbit.birnet.private>
Date: Thu, 11 Jul 2002 18:10:52 +0200 (CEST)

hi all,

in the course of reimplementing drag-selection for a widget,
i did a small survey of modifier behaviour in other (gnome/
gtk) programs and had to figure that there's no current
standard behaviour to adhere to:

for all applications, the first selection works as
expected, i.e. press-drag-release selects the region
(box) the mouse was draged over. also, starting a new
selection without pressing any modifiers simply replaces
the first one. differences occour when holding a modifier
(shift or ctrl) when starting the second selection.

Gimp:
Shift upon button press:        the new seleciton is added to the existing one
Ctrl upon button press:         the new selection is subtracted from the
                                existing one
Shift during drag:              the selection area (box or circle) has fixed
                                aspect ratio
Ctrl during drag:               the position of the initial button press
                                serves as center of the selected box/circle,
                                rather than the upper left corner

Gnumeric:
Shift upon button press:        the first selection is resized
Ctrl upon button press:         the new seleciton is added to the existing one

Abiword (selecting text regions):
Shift upon button press:        the first selection is resized
Ctrl upon button press:         triggers a compound (word) selection that
                                replaces the first selection

Mozilla (selecting text regions):
Shift upon button press:        the first selection is resized

Nautilus:
Shift or Ctrl upon buttn press: the new selection is added to or subtracted
                                from the first selection, depending on whether
                                the newly selected region was selected before.
                                i.e. implementing XOR integration of the newly
                                selected area into the first.
```

i'm not pointing this out to start a flame war over what selection style
is good or bad and i do realize that different applications have
different needs (i.e. abiword does need compound selection, and
the aspect-ratio/centering style for gimp wouldn't make too much
sense for text), but i think for the benfit of the (new) users,
there should me more consistency regarding modifier association
with adding/subtracting/resizing/xoring to/from existing selections.

---
ciaoTJ