The ANKLANG Manual

The Anklang Project <anklang.testbit.org>

November 2021

Abstract

Manual about usage, installation, development and implementation of the Anklang digital synthesizer and music creation software.

Contents

List of Tables

1 The ANKLANG Manual

This is the Anklang manual, Anklang is a music synthesis and composition program, released as Free Software under the GNU LGPL-2.1+ and the MPL-2.0.

The manual is structured into sections covering tutorial material, Howto descriptions, Man pages, file formats and conventions, development details with related considerations and an API reference.

It is written in Markdown and contributions are welcome, e.g. as pull requests or via the Anklang issue tracker.

2 Manual Pages

2.1 ANKLANG(1)

NAME

anklang - Music composition and modular synthesis application

SYNOPSIS

anklang [*OPTIONS*] [*FILES*...]

DESCRIPTION

Anklang is a digital audio synthesis application for live creation and composition of music and other audio material. It is released as free software under the MPL-2.0 and includes plugins under the GNU LGPL-2.1+.

Anklang comes with synthesis devices which can be arranged in tracks and controlled via MIDI input devices or pre-programmed clips which contain MIDI notes.

The Anklang sound engine is a dedicated process which is controlled by a user interface based on web technologies that can be run in a special process (like electron) or modern browsers like **firefox**(1) or **google-chrome**(1).

OPTIONS

Anklang supports short and long options which start with two dashes ('-').

-- Stop argument processing, treat remaining arguments as files.

-h, --help
: Show a help message about command line usage.

--fatal-warnings
: Abort on warnings and failing assertions, useful for test modes.

--check-integrity-tests
: Execute internal tests and benchmarks.

--disable-randomization
: Enable deterministic random numbers for test modes.

-v, --version
: Print information about the program version.

SEE ALSO

Anklang Website

3 Development Details

Technically, Anklang consists of a user interface front-end based on web technologies (HTML, SCSS, JS, Vue) and a synthesis engine backend written in C++. 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.

3.1 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_` 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.
- [_] Rename observe to handle or receive or onmsg.
- [_] Remove ”Jsonipc.initialize” predefinition

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

- [_] Turn Jsonhook into `shared_ptr<CbFunc>` with `using CbFunc = std::function<void (ValueS)>;`
- [_] Allow adding a Jsonhook Destroyer to e.g. disconnect Emittable::Connection
- [_] Add Jsonhook.destroy
- [_] Send hook ID to JS from Jsonhook impl with args for call
- [_] Send ”destroyed” to JS

3.2 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<SomeClass>`.

[_] 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 an persistent relative object link for project data storage.

4 Vue Component Reference

The Vue components used in Anklang are generally composed of a single file that provides:

a) Brief documentation;
b) CSS style information;
c) Html layout;
d) Assorted JS logic.

The Vue component object is implemented as part of (d). 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.

4.1 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.

4.1.1 B-ABOUTDIALOG

A modal b-dialog that displays version information about Anklang.

EVENTS:

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

4.1.2 B-BUTTON-BAR

A Vue container for tight packing of buttons.

SLOTS:

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

4.1.3 B-CHOICE

This 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.

PROPS:

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

EVENTS:

update:value (value)
Value change notification event, the first argument is the new value.

4.1.4 B-CLIPLIST

A Vue template to display a list of Ase.Clip instances.

PROPS:

project
The project containing playback tracks.
track
The Track containing the clips to display.

4.1.5 B-CLIPVIEW

A Vue template to display a small view of Ase.Clip.

PROPS:

clip The Ase.Clip to display.
tick The Ase.Track tick position.
***tickscale* (read-only)**
The pixel to PPQN ratio.

4.1.6 B-COLOR-PICKER

Vue template to display a color picker popup.

PROPS:

initialcolor
The initial color to display.

DATA:

color
The currently selected color.

4.1.7 B-CONTEXTMENU

A modal popup that displays contextmenu choices, see B-MENUITEM, B-MENUSEPARATOR. Using the `popup()` method, the menu can be popped up from the parent component, and setting up an `onclick` handler can be used to handle menuitem actions. Example:

```
<div @contextmenu.prevent="$refs.cmenu.popup">
  <b-contextmenu ref="cmenu" @click="menuactivation">...</b-contextmenu>
</div>
```

PROPS:

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.

EVENTS:

click (uri)
Event signaling activation of a submenu item, the `uri` of the submenu is provided as argument.
close Event signaling closing of a menu, regardless of whether menu item activation occoured or not.
startfocus
Auto focus a menu item upon popup, in the same way menu bar menus work.
keepmounted
Keep the menu and menu items mounted at all times, needed for map_kbd_hotkeys().

METHODS:

popup (event, { origin, data-contextmenu })
Popup the contextmenu, the `event` coordinates are used for positioning, 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.

4.1.8 B-DATABUBBLE

A mechanism to display data-bubble="" tooltip popups on mouse hover.

4.1.9 B-DEVICEEDITOR

Editor for audio signal devices.

PROPS:

device
Audio signal processing device.

4.1.10 B-DEVICEPANEL

Panel for editing of devices.

PROPS:

track
Container for the devices.

4.1.11 B-DIALOG

A dialog that captures focus and provides a modal shield that dims everything else. A *close* event is emitted on clicks outside of the dialog area, if *Escape* is pressed or the default *Close* button is actiavted.

PROPS:

shown
A boolean to control the visibility of the dialog, suitable for `v-model:shown` bindings.

EVENTS:

update:shown
An *update:shown* event with value `false` is emitted when the "Close" button activated.

SLOTS:

header
The *header* slot can be used to hold the modal dialog title.

default
The *default* slot holds the main content.

footer
By default, the *footer* slot holds a *Close* button which emits the *close* event.

CSS CLASSES:

b-dialog
The *b-dialog* CSS class contains styling for the actual dialog contents.

b-dialog-modalshield
This CSS class contains styling for masking content under the dialog.

B-EDITABLE - DISPLAY AN EDITABLE SPAN.

Methods:

- **activate()** - Show input field and start editing.

Props:

- **selectall** - Set to `true` to automatically select all editable text.
- **clicks** - Set to 1 or 2 to activate for single or double click.

Events:

- **change** - Emitted when an edit ends successfully.

4.1.12 B-FED-NUMBER

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.

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.

EVENTS:

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

4.1.13 B-FED-OBJECT

A field-editor for object input. A copy of the input value is edited, update notifications are provided via an `input` event.

PROPERTIES:

value
Object with properties to be edited.

readonly
Make this component non editable for the user.
debounce
Delay in milliseconds for `input` event notifications.

EVENTS:

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

4.1.14 B-FED-PICKLIST

A field-editor for picklist input.

PROPERTIES:

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

EVENTS:

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

4.1.15 B-FED-SWITCH

A field-editor switch to change between on and off.

PROPERTIES:

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

EVENTS:

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

4.1.16 B-FED-TEXT

A field-editor for text input.

PROPERTIES:

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

EVENTS:

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

4.1.17 B-FILEDIALOG

A modal b-dialog that allows file and directory selections.

PROPERTIES:

title The dialog title.
button
Title of the activation button.
cwd Initial path to start out with.
filters
List of file type constraints.

EVENTS:

select (path)
This event is emitted when a specific path is selected via clicks or focus activation.
close A *close* event is emitted once the ”Close” button activated.

4.1.18 B-FOLDERVIEW

A browser view for the selection of path components.

PROPERTIES:

entries
List of Ase.Entry objects.

EMITS

click (entry)
This event is emitted whenever an entry was activated via keyboard, mouse or touch.
select (entry)
This event is emitted whenever an entry gets focus.

4.1.19 B-HSCROLLBAR

Vue template to display a horizontal scrollbar.

PROPS:

value
The current scrollbar value.

4.1.20 B-ICON

This element displays icons from various icon fonts. Note, 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).

PROPS:

iconclass
A CSS class to apply to this icon.
fa The name of a ”Fork Awesome” icon (compatible with ”Font Awesome 4”), see the Fork Awesome Icons.
mi The name of a ”Material Icons” icon, see the Material Design Icons.
bc The name of an ”AnklangIcons Font” icon.
uc A unicode character literal, see the Unicode Lists and Symbols.
ic A prefixed variant of `fa`, `bc`, `mi`, `uc`.
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.

4.1.21 B-KNOB

This 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.

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.

EVENTS:

update:value (value)
Value change notification event, the first argument is the new value.
reset:value ()
Request to reset the value.

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`.

4.1.22 B-MENUBAR

The main menu bar at the top of the window.

4.1.23 B-MENUITEM

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

PROPS:

uri Descriptor for this menuitem that is passed on to its B-CONTEXTMENU `onclick`.
kbd Hotkey to display next to the menu item and to use for the context menu kbd map.
disabled
Boolean flag indicating disabled state.
fa, mi, bc, uc
Shorthands icon properties that are forwarded to a B-ICON used inside the menuitem.

EVENTS:

click Event emitted on keyboard/mouse activation, use `preventDefault()` to avoid closing the menu on clicks.

SLOTS:

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

4.1.24 B-MENUROW

Menuitems are packed horizontally inside a menurow.

PROPS:

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

SLOTS:

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

4.1.25 B-MENUSEPARATOR

A menu element that serves as a visual separator between other elements.

4.1.26 B-MENUTITLE

An element to be used as menu title.

SLOTS:

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

4.1.27 B-MORE

Indicator for adding or dropping new UI elements.

4.1.28 B-NOTEBOARD

Noteboard to post notifications for end users.

4.1.29 B-PARTLIST

A Vue template to display a list of Ase.Part instances.

PROPS:

project
The project containing playback tracks.
track
The Track containing the parts to display.

4.1.30 B-PARTTHUMB

A Vue template to display a thumbnail for a Ase.Part.

PROPS:

part The Ase.Part to display.
tick The Ase.Track tick position.

4.1.31 B-PIANO-ROLL

A Vue template to display notes from a MIDI event source as a piano roll.

PROPS:

msrc The MIDI event source for editing and display.

DATA:

hscrollbar
The horizontal scrollbar component, used to provide the virtual scrolling position for the notes canvas.

4.1.32 B-PLAYCONTROLS

A container holding the play and seek controls for a Ase.song.

PROPS:

project
Injected *Ase.Project*, using `b-shell.project`.

B-POSITIONVIEW - DISPLAY OF THE PROJECT TRANSPORT POSITION POINTER AND RELATED INFORMATION

Props:

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

4.1.33 B-PREFERENCESDIALOG

A modal b-dialog to edit preferences.

EVENTS:

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

4.1.34 B-PRO-GROUP

A property group is a collection of properties.

PROPERTIES:

name
Group name.
props
List of properties with cached information and layout rows.
readonly
Make this component non editable for the user.

4.1.35 B-PRO-INPUT

A property input element, usually for numbers, toggles or menus.

PROPERTIES:

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

4.1.36 B-SHELL

Shell for editing and display of a Ase.Project.

PROPS:

project
Implicit *Ase.Project*, using App.project().

4.1.37 B-STATUSBAR

Area for the display of status messages and UI configuration.

4.1.38 B-TOGGLE

This element provides a toggle button for boolean inputs. It supports the Vue v-model protocol by emitting an `input` event on value changes and accepting inputs via the `value` prop.

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.

EVENTS:

update:value (value)
Value change notification event, the first argument is the new value.

4.1.39 B-TRACKLIST

A container for vertical display of Ase.Track instances.

PROPS:

project
The *Ase.Project* containing playback tracks.

4.1.40 B-TRACKVIEW

A Vue template to display a project's Ase.Track.

PROPS:

project
The *Ase.project* containing playback tracks.
track
The *Ase.Track* to display.

4.1.41 B-TREESELECTOR-ITEM

An element representing an entry of a B-TREESELECTOR, which allows selections.

4.1.42 B-TREESELECTOR

A container that displays a tree and allows selections.

PROPS:

defaultcollapse
Set default for collapsible entries.

EVENTS:

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

4.2 Reference for envue.js

4.2.1 Component class

class **Envue.Component**

Component base class for wrapping Vue components.

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 (*args*)

Wrapper for Vue.$watch

vue_export (*vue_object*) [static]

Create a Vue options API object from *vue_object* for SFC exports.

4.2.2 Envue Functions

Envue.forward_access (*vm, classinstance, ignores*)

Forward all accesses to fields on `vm` to access fields on `classinstance`.

Envue.vue_export_from_class (*Class, vue_object*)

Create a Vue options API object that proxies access to a newly created `Class` instance.

4.3 Reference for util.js

4.3.1 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.

4.3.2 FocusGuard class

***class* FocusGuard**

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

4.3.3 Util Constants

`Util.ResizeObserver`

Work around FireFox 68 having ResizeObserver disabled

`Util.clone_descriptors`

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

`Util.KeyCode`

Symbolic names for key codes

4.3.4 Util Functions

`Util.now()`

Retrieve current time in milliseconds.

`Util.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.

`Util.capture_event (`*`eventname, callback`*`)`

Process all events of type `eventname` with a single callback exclusively

`Util.vue_component (`*`element`*`)`

Get Vue component handle from element or its ancestors

`Util.envue_object (`*`element`*`)`

Get Envue `$object` from element or its ancestors

`Util.drag_event (`*`event`*`)`

Start `drag_event (event)` handling on a Vue component's element, use `@pointerdown="Util.drag_event"`

`Util.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.

`Util.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.

`Util.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.

`Util.vm_scope_selector (`*`vm`*`)`

Retrieve CSS scope selector for vm_scope_style()

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

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

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

Util.array_remove (*array, item*)
Remove element `item` from `array`

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

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

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

Util.copy_recursively (*src*)
Create a new object that has the same properties and Array values as `src`

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

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

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

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

Util.weakoid (*object*)
Fetch a unique object id.

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

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

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

Util.extend_property (*prop, disconnector*)
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.

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

Util.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.

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

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

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

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

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

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

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

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

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

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

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

Util.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 while Firefox tends to increase the number of events with increasing wheel distance, Chromium tends to accumulate and send fewer events with higher values instead.

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

Util.list_focusables (*element*)
List all elements that can take focus and are descendants of `element` or the document.

Util.is_nav_input (*element*)
Check if `element` has inner input navigation

Util.is_button_input (*element*)
Check if `element` is button-like input

Util.element_midpoint (*element*)
Compute global midpoint position of element, e.g. for focus movement

Util.push_focus_root (*element, escapecb*)
Constrain focus to `element` and its descendants

Util.remove_focus_root (*element*)
Remove an `element` previously installed via push_focus_root()

Util.keydown_move_focus (*event*)
Move focus on UP/DOWN/HOME/END `keydown` events

Util.move_focus (*dir, subfocus*)
Move focus to prev or next focus widget

Util.forget_focus (*element*)
Forget the last focus elemtn inside `element`

Util.setup_shield_element (*shield, containee, closer*)
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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Util.dropdown (*menu, event, options*)
Popup `menu` using `event.currentTarget` as origin.

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

Util.is_navigation_key_code (*keycode*)
Check if a key code is used of rnavigaiton (and non alphanumeric).

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

Util.display_keyname (*keyname*)
Create display name from KeyEvent.code names.

Util.match_key_event (*event, keyname*)
Match an event's key code, considering modifiers.

Util.add_hotkey (*hotkey, callback, subtree_element*)
Add a global hotkey handler.

Util.remove_hotkey (*hotkey, callback*)
Remove a global hotkey handler.

Util.has_ancestor (*element, ancestor*)
Check if `ancestor` is an ancestor or `element`

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

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

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

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

Util.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.

Util.tmplstr (*a, e*)
Join template literal arguments into a String

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

`Util.hash53 (`*`key, seed`*`)`

Generate 53-Bit hash from `key`.

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 $x = 0 \rightarrow$ (1), i.e. (2), we obtain w_0 and from $x = 0 \rightarrow$ (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 $W_{0,\dots,3}[i]$, $i = [0, \dots, n]$, coefficients to speed up the resampling calculation, trading memory for computational performance. With $w_{0,\dots,3}$ 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 $V_0, \dots, V_4$, 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 $i = [0, 1, 2, \dots, n]$ by substituting $g(i) = f(\frac{i}{n})$ with

(15) $g(i) = V_3 W_3[i] + V_2 W_2[i] + V_1 W_1[i] + V_0 W_0[i]$

and using n precalculated coefficients $W_{0,\dots,3}$ 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 $W_{0,\dots,3}[0, \dots, n]$ 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 $V_0, \dots, V_3$.

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
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
```