

# **ANKLANG Development Details**

The Anklang Project <[anklang.testbit.eu](https://anklang.testbit.eu)>

**June 2023**

## **Abstract**

API documentation and development internals of the Anklang project.

## Contents

<b>1</b>	<b>ANKLANG Development Details</b>	<b>4</b>
1.1	Jsonipc . . . . .	4
1.1.1	Callback Handling . . . . .	4
1.2	Serialization . . . . .	4
1.3	Ase Class Inheritance Tree . . . . .	5
<b>A</b>	<b>Appendix</b>	<b>7</b>
A.1	One-dimensional Cubic Interpolation . . . . .	7
A.2	Modifier Keys . . . . .	8

List of Tables

1    GDK drag-and-drop modifier keys . . . . . 8

# 1 ANKLANG 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 interfaces 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 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-generated 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 concise 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.

### 1.1.1 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    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
```

## 1.2 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.

### 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
```

## 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  $0 \leq x \leq 1$ , specifying the sample value of the curve segment between  $V_1$  and  $V_2$  to obtain.

To calculate the coefficients  $w_0, \dots, w_3$ , 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)  $\rightarrow$  (4) and (7)  $\rightarrow$  (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)  $\rightarrow$  (1) and (5)  $\rightarrow$  (10) to obtain  $w_2$  and  $w_3$ .

$$(11) (3) \rightarrow (1): w_3 + w_2 + \frac{V_2 - V_0}{2} + V_1 = V_2$$

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

With the resulting coefficients:

$$w_0 = V_1 \quad (\text{initial value})$$

$$w_1 = \frac{V_2 - V_0}{2} \quad (\text{initial slope})$$

$$w_2 = \frac{-V_3 + 4V_2 - 5V_1 + 2V_0}{2} \quad (\text{initial curvature})$$

$$w_3 = \frac{V_3 - 3V_2 + 3V_1 - V_0}{2} \quad (\text{rate change of curvature})$$

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  $V_0, \dots, V_3$ ,  $w_0, \dots, w_3$  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):

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

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

(14)

$$f(x) = V_3 (0.5x^3 - 0.5x^2) + V_2 (-1.5x^3 + 2x^2 + 0.5x) + V_1 (1.5x^3 - 2.5x^2 + 1) + V_0 (-0.5x^3 + x^2 - 0.5x)$$

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

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

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

$$m = \frac{i}{n}$$

$$W_3[i] = 0.5m^3 - 0.5m^2$$

$$W_2[i] = -1.5m^3 + 2m^2 + 0.5m$$

$$W_1[i] = 1.5m^3 - 2.5m^2 + 1$$

$$W_0[i] = -0.5m^3 + m^2 - 0.5m$$

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
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 dragged over. also, starting a new  
 selection without pressing any modifiers simply replaces  
 the first one. differences occur when holding a modifier  
 (shift or ctrl) when starting the second selection.

Gimp:

Shift upon button press:	the new selection 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 selection 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 button 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 benefit of the (new) users,  
 there should be more consistency regarding modifier association  
 with adding/subtracting/resizing/xoring to/from existing selections.

---  
ciaoTJ