

# **Proposal for an XML format representing Time, Positions and Parts of Audio Waveforms**

(Linux Audio Conference, 23-Mar-07)

Jens Gulden (FLP @ TU Berlin) - [jgulden@cs.tu-berlin.de](mailto:jgulden@cs.tu-berlin.de)

Hanns Holger Rutz (SeaM @ HfM Weimar) - [hanns.rutz@hfm-weimar.de](mailto:hanns.rutz@hfm-weimar.de)

## What?

- document format for **positions in** and **parts of time** regarding media files (audio files, other time-based media)

## Why?

Organization of Time is the most fundamental point in time-based arts (hence the name!!),

particularly when dealing with "containers" such as parts of concrète sound recordings

Present formats and systems have a rather poor concept of time positions

(e.g. no hierarchical nesting of time containers, no extensibility, no meta-data)

No agreed upon standard for time positions, hence bad interchangeability

Useful to decouple time-description and actual media files

## **Design Goals**

easy-to-generate

easy-to-parse

extensible, flexible

## **Approach**

XML (Extensible Markup Language)

Prototype implementation using SuperCollider 3

## **Design Goals: Easy-to-Generate**

XML is text-based and essentially "human-writable"

XML libraries available for many languages

Different ways of expressing a time value

(at 4'32", for 272 seconds, at sample frame 11995200 etc.)

## **Design Goals: Easy-to-Parse**

XML is text-based and essentially "human-readable"

XML libraries available for many languages

## **Design Goals: Extensible, Flexible**

New elements and attributes can be added to the basic XML format

Properly written parsers can skip or omit unused or unknown data

Easy transcoders can be setup to filter out or transform certain elements in the file

## The Model

[<http://swiki.hfbk-hamburg.de:8888/MusicTechnology/786>]

Positions in time shall be call **Anchors**

Parts of time shall be call **Slices**

A Slice is delimited by a start and end anchor

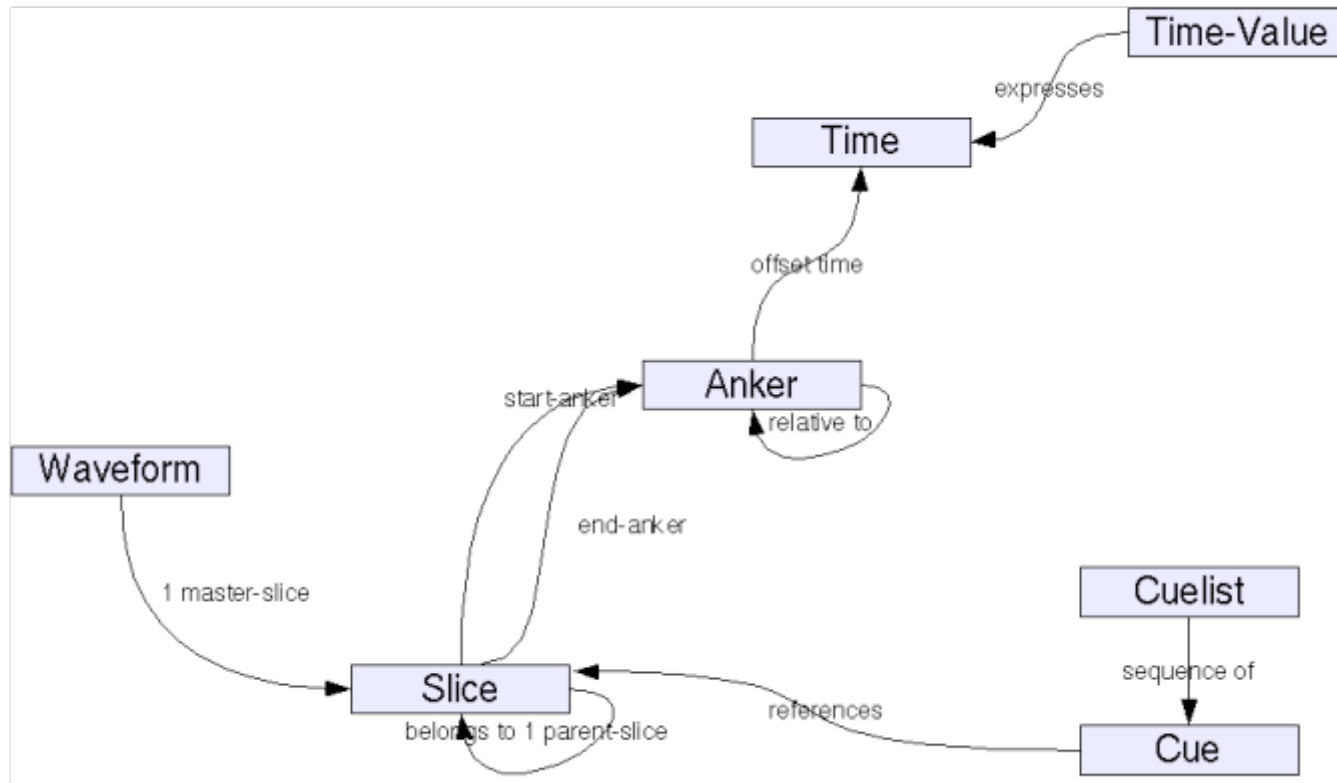
Time represented by an Anchor is expressed as a **Time-Value**

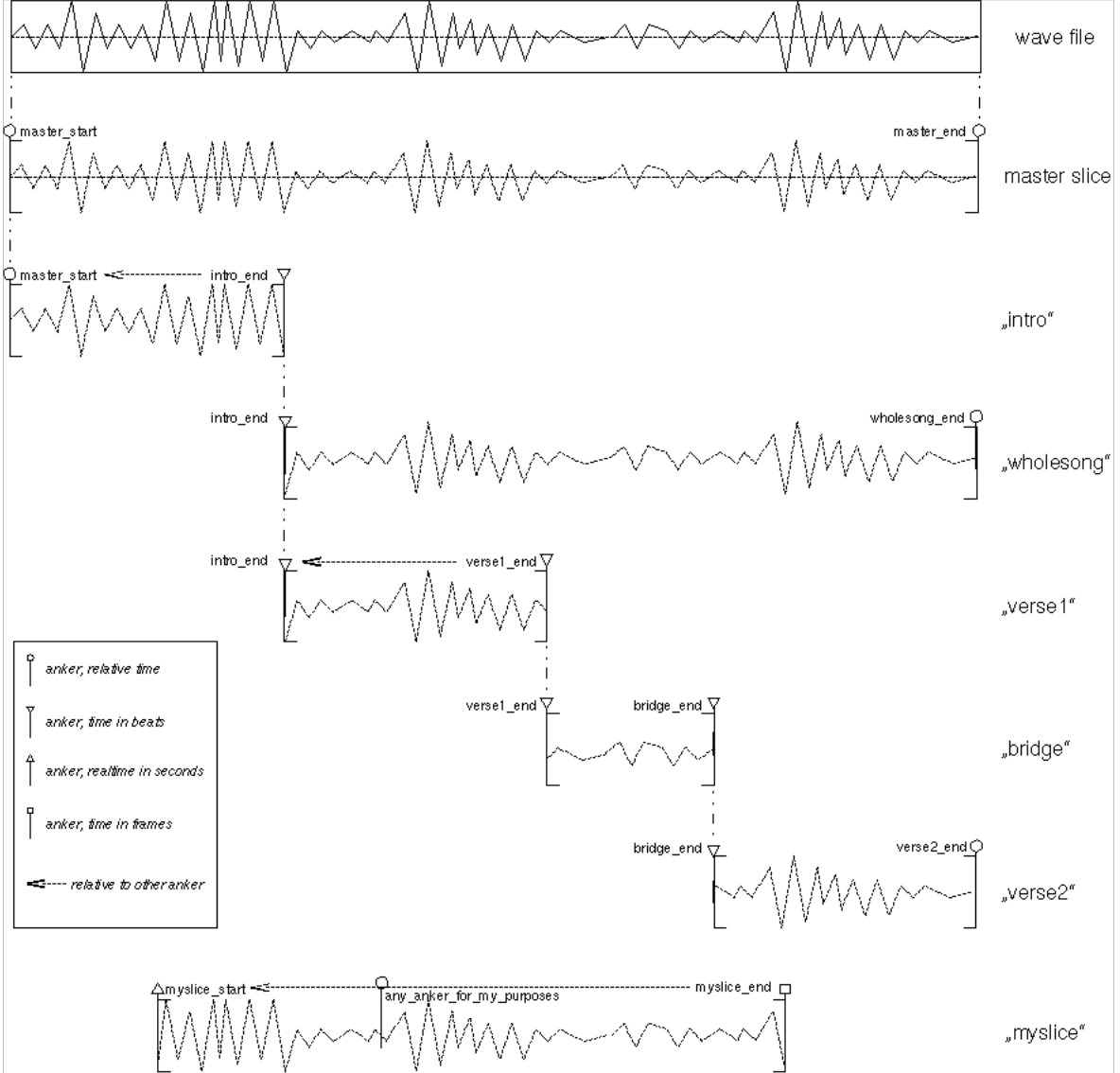
(for now, a number and a unit, but could also be a symbol...)

Slices are always relative to a parent slice up to an (often implicit) master-slice (the whole media span).

Anchors are relative to other anchors, often the start anchor of a parent slice

As an extension, slices can be seen as **Cues** which may be assembled to **Cue-Lists**





## An Example

- As a first approach, we use a root element `<instrument>`. Sub elements can be `<wave>` for an audio waveform to be partitioned, and `<cueList>` for a list of `<cue>` elements that refer to slices in the audio waveform (as defined in the `<wave>` section).
- Time units: `s` for seconds, `#` for sample frames, blank (none) for percentages, `ms` for milliseconds, `~` for bars/beats
- shortcuts: implicit anchors using the `from` and `to` attributes of `<slice>`. `<pad>` elements to advance in time
- these files can be read into SuperCollider 3 using helper classes, and played back using Nick Collins' BBCut library  
(example follows later)

```
<instrument>
  <wave src="examples/audio/NosNodOrr.wav">
    <!-- wavefile is implicitly used as 'master-slice' -->
    <slice name="start" to="20.55839s"/>

    <!-- empty slice, starts at end of previous slice: -->
    <pad to="27.69878s"/>

    <!-- next slice, starts at end of previous slice: -->
    <slice name="zwischen" to="34.83637s"/>

    <!-- relative time-values: -->
    <slice name="wummer" from="0.33" to="0.5"/>

    <!-- full syntax variant: -->
    <slice name="basis">
      <start>
        <anchor time="34.9s"/> <!--shortcut-anchor-->
      </start>
    <end>
      <anchor> <!-- full anchor syntax -->
        <time>
          <seconds>49.1638</seconds>
        </time>

        <!--<time value="49.1638s"/> (shortcut)-->
      </anchor>
    </end>
  </slice>
```

```
<!-- specifying length instead of end-anchor: -->  
<slice name="funny" from="49.19s" length="7.14s"/>
```

```
<!-- length only, starts at end of previous slice: -->  
<slice name="melody1" length="10.26s"/>
```

```
<!-- time-value via frame-position: -->  
<slice name="melody2" to="42256#"/>
```

```
<!-- relative time-value: -->  
<slice name="end" to="1.0"/>
```

```
</wave>
```

```
<!-- a cuelist for playing slices in a different order -->
```

```
<cuelist name="song">  
  <cue slice-ref="start"/>  
  <cue slice-ref="zwischen"/>  
  <cue slice-ref="wummer" repeat="4"/>  
  <cue slice-ref="basis"/>  
  <cue slice-ref="funny"/>  
  <cue slice-ref="melody1" repeat="2"/>  
  <cue slice-ref="melody2"/>  
  <cue slice-ref="basis"/>  
  <cue slice-ref="melody1" repeat="2"/>  
  <cue slice-ref="melody3"/>  
  <cue slice-ref="wummer" repeat="8"/>  
  <cue slice-ref="funny"/>  
  <cue slice-ref="melody1"/>
```

```
<cue slice-ref="melody2"/>
<cue slice-ref="melody3"/>
<cue slice-ref="end"/>
<cue slice-ref="melody2"/>
<cue slice-ref="melody1"/>
<cue slice-ref="melody2"/>
<cue slice-ref="melody3"/>
<cue slice-ref="end"/>
</cuelist>
</instrument>
```

## **Real (haha) World Application: Eisenkraut**

[<http://sciss.de/eisenkraut>]

Eisenkraut: cross-platform audio file editor written in Java, using SuperCollider 3 as realtime sound engine

Time positions: only Markers (i.e. Anchors) at the moment, no Regions (or Slides or Cues)

### Problems:

- marker information varying across different audio file formats
- need to write out whole audio file when updating markers
- not possible to switch

### Possible circumvention:

- Upcoming version\*: OpenSoundControl (OSC) server, making it controllable from SC3

- Long-term: separate marker files as proposed in this paper

## Eisenkraut: creating + reading Markers from an OSC client

```
e = Eisenkraut.default; e.connect;
e.sendMsg( '/doc', \open, "~/scwork/kalligraphie/mauerseegler.aif".standardizePath );
```

```
// create a geometric series of 20 markers across the file
```

```
(
fork { var len, scale, pos, names, marks;
  len  = e.query( '/doc/active/timeline', \length ).first;
  pos  = Array.geom( 20, 0.1, 1.2 );
  scale = len / (pos.last * 1.2);
  pos  = pos.collect({ arg t; (t * scale).asInteger });
  names = Array.fill( 20, { arg i; "Mark #" ++ (i+1) });
  marks = (pos ++ names).unlace( 20 ).flatten;
  e.listSendMsg([ '/doc/active/markers', \add ] ++ marks );
} )
```

```
// queries all markers of the active document
```

```
(
fork { var msg, rate, num;
  rate  = e.query( '/doc/active/timeline', \rate ).first;
  num   = e.query( '/doc/active/markers', \count ).first;
  msg   = e.get( '/doc/active/markers', [ \range, 0, num ] );
  msg.pairsDo({ arg pos, name;
    ("Marker '" ++ name ++ "' at frame " ++ pos ++ " = "
     ++ (pos/rate).asTimeString( 0.001 )).postln;
  });
} )
```

## Eisenkraut: interfacing with custom editors

- don't put more in the server (Eisenkraut) than is necessary
- custom things and details should be added by a client (SC3 here)
- example from Kalligraphie (sound installation january-march 2007):

```
(
s.waitForBoot({
  GUI.swing;
  ~kallig = Kalligraphie( s );
  ~rhizom = KalligRhizomEisK( ~kallig );
  ~rhizom.registerWithEisenkraut;
});
)

(
e = Eisenkraut.default;
~line = ~rhizom.lines.first;
~line.words;
~sentence = ~line.versions.first;
~line.words.do({ arg word, idx; e.sendMessage( '/doc/active/markers', \add,
~sentence.positions[ idx ], word )});
)
)
```

- we store the structure in a custom XML file. looks like this:  
(times are in sample frames, sr = 44.1 kHz!)

```
<rhizom>
  <lines>
    <line text="wir haben alles verwendet was uns begegnet ist" line="0">
      <versions>
        <sentence line="0">
          <words>
            <word len="19153" fadein="0.02" fadeout="0.02" pos="4520495"/>
            <word len="18831" fadein="0.02" fadeout="0.02" pos="4539648"/>
            <word len="20110" fadein="0.02" fadeout="0.02" pos="4558479"/>
            <word len="29591" fadein="0.02" fadeout="0.02" pos="4578589"/>
            <word len="10716" fadein="0.02" fadeout="0.02" pos="4608180"/>
            <word len="14156" fadein="0.02" fadeout="0.02" pos="4618896"/>
            <word len="29459" fadein="0.02" fadeout="0.02" pos="4633052"/>
            <word len="21653" fadein="0.02" fadeout="0.02" pos="4662511"/>
          </words>
        </sentence>
        <sentence gain="-2" line="0">
          <words>
            <word len="19865" fadein="0.02" fadeout="0.06" pos="0"/>
            <word len="26352" fadein="0.06" fadeout="0.02" pos="19865"/>
            <word len="19845" fadein="0.02" fadeout="0.02" pos="46217"/>
            <word len="34971" fadein="0.02" fadeout="0.02" pos="66062"/>
            <word len="13495" fadein="0.02" fadeout="0.02" pos="101033"/>
            <word len="15347" fadein="0.02" fadeout="0.02" pos="114528"/>
            <word len="30076" fadein="0.02" fadeout="0.02" pos="129875"/>
            <word len="25269" fadein="0.02" fadeout="0.12" pos="159951"/>
          </words>
        </sentence>
      [... etc ...]
```

- can we make a small XML example that conforms to our initially proposed model?

*Sure!*

```
(
~doc = DOMDocument.new;
~root = ~doc.createElement( "instrument" );
~doc.appendChild( ~root );
// now write the first version of the first sentence
~wave = ~doc.createElement( "wave" );
~wave.setAttribute( "src", Kalligraphie.workDir ++ "rhizomwhisper2.aif" );
~line.words.do({ arg word, idx; var slice, pos, len;
  slice = ~doc.createElement( "slice" );
  slice.setAttribute( "name", word.asString );
  pos = ~sentence.positions[ idx ];
  len = ~sentence.lengths[ idx ];
  slice.setAttribute( "from", pos.asString ++ "#" );
  slice.setAttribute( "to", (pos + len).asString ++ "#" );
  ~wave.appendChild( slice );
});
~root.appendChild( ~wave );
// now make a cue list of those words
~cueList = ~doc.createElement( "cuelist" );
~cueList.setAttribute( "name", "First sentence" );
~line.words.do({ arg word; var cue;
  cue = ~doc.createElement( "cue" );
  cue.setAttribute( "slice-ref", word.asString );
  ~cueList.appendChild( cue );
});
~root.appendChild( ~cueList );
~file = File( "~/Desktop/test.xml".standardizePath, "w" );
~doc.write( ~file ); ~file.close;
)
```

XML-Output from above (note: we are only using some basic elements of the model, we could as well create parent slices for each line, and parent slices for each version of the sentence!)

```
<instrument>
  <wave src="/Users/rutz/scwork/kalligraphie/rhizomwhisper2.aif">
    <slice name="wir" to="4539648#" from="4520495#" />
    <slice name="haben" to="4558479#" from="4539648#" />
    <slice name="alles" to="4578589#" from="4558479#" />
    <slice name="verwendet" to="4608180#" from="4578589#" />
    <slice name="was" to="4618896#" from="4608180#" />
    <slice name="uns" to="4633052#" from="4618896#" />
    <slice name="begegnet" to="4662511#" from="4633052#" />
    <slice name="ist" to="4684164#" from="4662511#" />
  </wave>
  <cue list name="First sentence" loop="true">
    <cue slice-ref="wir" />
    <cue slice-ref="haben" />
    <cue slice-ref="alles" />
    <cue slice-ref="verwendet" />
    <cue slice-ref="was" />
    <cue slice-ref="uns" />
    <cue slice-ref="begegnet" />
    <cue slice-ref="ist" />
  </cue list>
</instrument>
```

- play it back using the `SFSliceInstrument` helper class (utilizing BBCut):

```
~instr = SFSliceInstrument.new;  
// don't worry, BBCut will scan the whole 40 minute file,  
// takes a _lot_ of time!!  
~instr.loadXML( "~/Desktop/test.xml".standardizePath );  
~sfCueList = ~instr.getCueList( "First sentence" );  
~sfCueList.play( 1.0, true ); // <speed>, <verbosity>  
~sfCueList.cues = ~sfCueList.cues.scramble;  
~sfCueList.play( 1.0, true );
```

**Questions? Discussion?**

-----  
\*current online version of Eisenkraut is 0.63. the next version 0.70 will include the demonstrated OSC facilities