Rtosc
Realtime Open Sound Control

Mark McCurry

2018

What is OSC?

- Path + argument types + argument data
- Types include: **i**:*int32*, **s**:*string*, **b**:*binary-blob*, **f**:*float32*, **h**:*int64*, **t**:*timetag*, **d**:*float64*, **S**:*symbol*, **r**:*rgb*, **m**:*4-byte-MIDI*, **c**:*int8*, **T**:*true*, **F**:*false*, **N**:*nil*, and **I**:*Inf*.

What is OSC?

- Message serialization
- Semi-complex inter-process communication

What Doesn't OSC Normally Do?

What Does Rtosc Add?

- Low level Serialization (C)
- High Level Dispatch/Metadata (C++)
- Restricted problem domain

OSC Serialization

```
char buffer[256];
```

OSC Serialization

```
char buffer[256];
int ret = rtosc_message(buffer, sizeof(buffer),
                        "/path", "si",
                        "some arguments", 123);
```

OSC Serialization

```
char buffer[256];
int ret = rtosc_message(NULL, 0,
                        "/path", "si",
                        "some arguments", 123);
```

OSC Serialization

```
char buffer[256];
int ret = rtosc_message(buffer, sizeof(buffer),
                        "/path", "si",
                        "some arguments", 123);
const char *args = rtosc_argument_string(buffer);
```

OSC Serialization

```
char buffer[256];
int ret = rtosc_message(buffer, sizeof(buffer),
                        "/path", "si",
                        "some arguments", 123);
const char *args = rtosc_argument_string(buffer);

const char *first_arg = rtosc_argument(buffer, 0).s;
int second_arg        = rtosc_argument(buffer, 1).i;
```

Working With Messages

- Messages need to be dispatched to handle them

Working With Messages

- Messages need to be dispatched to handle them
- ZynAddSubFX has a lot of parameters

Working With Messages

- Messages need to be dispatched to handle them
- ZynAddSubFX has a lot of parameters
- Dispatch needs to be **fast**

Dispatch Tree

```
Message: /foo/bar

Tree:-| abc
      | xxx
      | yyy
      | foo(*)
      +-|qwerty
        |path
        |bar(*)
```

Dispatch Example

```
struct Envelope {
  float attack, decay, release;
};
```

Dispatch Example

```
struct Envelope {
  float attack, decay, release;
};

rtosc::Ports ports = {
  {"attack", NULL, NULL,
        [](const char *msg, rtosc::RtData &rt) {
        }},
};
```

Dispatch Example

```
struct Envelope {
  float attack, decay, release;
};

rtosc::Ports ports = {
  {"attack:f", NULL, NULL,
        [](const char *msg, rtosc::RtData &rt) {
        Envelope &obj = *(Envelope*)rt.obj;
        obj.attack = rtosc_argument(msg, 0).f;
        }},
};
```

Dispatch Example

```
struct Envelope {
  float attack, decay, release;
};

rtosc::Ports ports = {
  {"attack:f", ":max\0=15\0", NULL,
        [](const char *msg, rtosc::RtData &rt) {
        Envelope &obj = *(Envelope*)rt.obj;
        obj.attack = rtosc_argument(msg, 0).f;
        }},
};
```

Dispatch Example

```
struct Envelope {
  float attack, decay, release;
};

#define rObject Envelope
rtosc::Ports ports = {
  rParamF(attack, rLinear(0, 15), rMap(unit, sec),
          "Attack Time"),
};
```

Dispatch Example

```
struct Envelope {
  float attack, decay, release;
};

#define rObject Envelope
rtosc::Ports ports = {
  rParamF(attack,  rLinear(0, 15), rMap(unit, sec),
          "Attack Time"),
  rParamF(decay,   rLinear(0, 15), rMap(unit, sec),
          "Decay Time"),
  rParamF(release, rLinear(0, 15), rMap(unit, sec),
          "Release Time"),
};
```

Metadata

- Minimum/Maximum
- Linear/Log scaling
- Documentation strings
- Units
- Option symbol $\rightharpoonup$ value mappings

Metadata

- Reflection
- Avoids repetition
- Keeps information near code use

Metadata Improvements

- osc-doc API reference
- Learning MIDI/Plugin-host bindings
- Reuse of metadata in generating the GUI

Rtosc Performance

- Rtosc Is fast

Rtosc Performance

- Rtosc Is fast
- No really

Sonic Pi - Integration

Impl	per op	ops per second	speedup
Encoding an average message			
fast_osc	1.2 us	800,000	9.6x
samsosc	3.8 us	260,000	3.1x
osc-ruby	12 us	83,000	–
Decoding an average message			
fast_osc	0.6 us	1,700,000	50x
samsosc	4.7 us	230,000	7.4x
osc-ruby	29 us	34,000	–

Liblo point of comparison

Impl.	per op	ops per second	speedup
Decoding an average message			
liblo	218 ns	4,600,000	-
rtosc	53 ns	19,000,000	4.1x
Encoding an average message			
liblo	383 ns	2,600,000	-
rtosc	125 ns	8,000,000	3.1x
Dispatch message on single layer			
liblo	530 ns	1,900,000	-
rtosc	54 ns	19,000,000	10x

Liblo algorithm scaling

ZynAddSubFX has:

- 3,805,225 unique OSC paths
- e.g. /part1/kit5/adpars/VoicePar7/AmpLfo/Pfreq
- An average depth of 6.11 subpaths
- With minimal hashing an average of 6.11 matches are needed for rtosc, and 3,805,225 for liblo

Liblo algorithm scaling

- Liblo match time: 18.3 ms
- Rtosc match time: 380 ns

Liblo algorithm scaling

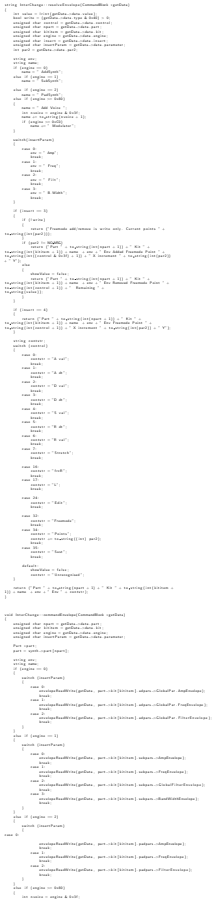
- Liblo match time: 18.3 ms
- Rtosc match time: 380 ns
- Liblo: ≈55 messages per second
- Rtosc: ≈2,600,000 messages per second

Discussion of Trade offs

- Fast, but maintainable

A comparison

```
#define rObject LFOParams
#undef rChangeCb
#define rChangeCb if (obj->time) { obj->last_update_timestamp = obj->time->time(); }
static const rtosc::Ports _ports = {
    rSelf(LFOParams),
    rPaste,
    rOption(loc, rProp(internal),
            rOptions(ad_global_amp, ad_global_freq, ad_global_filter,
                     ad_voice_amp, ad_voice_freq, ad_voice_filter, unspecified),
            "location of the filter"),
    rParamF(Pfreq, rShort("freq"), rLinear(0.0,1.0),
            rDefaultDepends(loc),
            rPreset(ad_global_amp, 0x1.42850ap-1), // 80
            rPreset(ad_global_freq, 0x1.1a3468p-1), // 70
            rPreset(ad_global_filter, 0x1.42850ap-1),
            rPreset(ad_voice_amp, 0x1.6ad5acp-1), // 90
            rPreset(ad_voice_freq, 0x1.93264cp-2), // 50
            rPreset(ad_voice_filter, 0x1.93264cp-2),
            "frequency of LFO\n"
            "lfo frequency = (2^(10*Pfreq)-1)/12 * stretch\n"
            "true frequency is [0,85.33] Hz"),
    rParamZyn(Pintensity, rShort("depth"),
              rDefaultDepends(loc),
              rDefault(0), rPreset(ad_voice_amp, 32),
              rPreset(ad_voice_freq, 40), rPreset(ad_voice_filter, 20),
              "Intensity of LFO"),
    rParamZyn(Pstartphase, rShort("start"), rSpecial(random),
              rDefaultDepends(loc), rDefault(64), rPreset(ad_voice_freq, 0),
              "Starting Phase"),
    rOption(PLFOtype, rShort("type"), rOptions(sine, triangle, square, up, down,
                exp1, exp2), rDefault(sine), "Shape of LFO"),
    rParamZyn(Prandomness, rShort("a.r."), rSpecial(disable), rDefault(0),
            "Amplitude Randomness (calculated uniformly at each cycle)"),
    rParamZyn(Pfreqrand, rShort("f.r."), rSpecial(disable), rDefault(0),
            "Frequency Randomness (calculated uniformly at each cycle)"),
    rParamZyn(Pdelay, rShort("delay"), rSpecial(disable),
              rDefaultDepends(loc), rDefault(0), rPreset(ad_voice_amp, 30),
              "Delay before LFO start\n0..4 second delay"),
    rToggle(Pcontinous, rShort("c"), rDefault(false),
            "Enable for global operation"),
    rParamZyn(Pstretch, rShort("str"), rCentered, rDefault(64),
        "Note frequency stretch"),
};
#undef rChangeCb
```



Current/Future Work

- Automations
- Automated Analysis of Trees
- Faster message encode/decode
- More standardized metadata

Conclusions

Rtosc is:

- A way to handle OSC **Inside** RT apps
- A library designed to retrofit existing apps
- Powerful thanks to low level API and metadata powered high level
- Maintainable
- Fast

Questions?

- https://github.com/fundamental/rtosc