BLUE BRAIN PROJECT

ZeroBuf

Zero-copy, zero-serialize, zero-hassle protocol buffers

Why Zerobuf?

- Zero-copy: object data is directly distributable
- Zero-serialize: one portable data storage buffer per class
- Zero-hassle: random read/write access to serialised data
- An alternative to protobuf, flatbuffers and Cap'n Proto

Example: Static sized ZeroBuf

BLUE BRAIN PROJECT

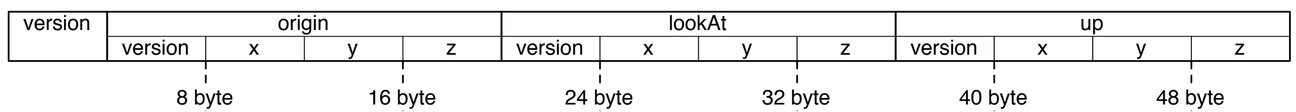
FBS Schema

```
namespace zerobuf.render;

table Vector3f
{
  x: float;
  y: float;
  z: float;
}

table Camera
{
  origin: Vector3f = 0, 0, 1;
  lookAt: Vector3f;
  up: Vector3f = 0, 1, 0;
}
```

C++ Code

```
namespace render = zerobuf::render;

render::Camera camera;
camera.getOrigin() = render::Vector3f(0, 0, 1);
```

Memory Layout



Example: Dynamic sized ZeroBuf

BLUE BRAIN PROJECT

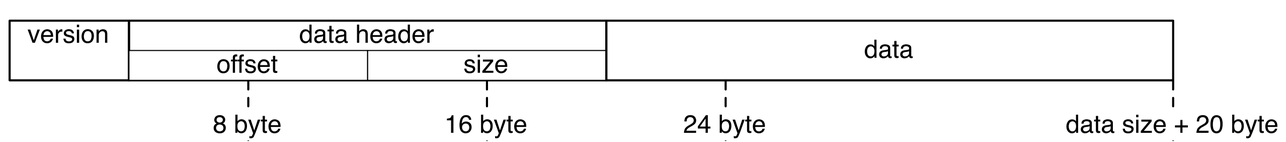
FBS Schema

```
namespace zerobuf.render;

table ImageJPEG
{
  data:[ubyte];
}
```

C++ Code

```
namespace render = zerobuf::render;

render::ImageJPEG imageJPEG;
glReadPixels( … );
tjCompress2( …, ptr, size );
imageJPEG.setData( ptr, size );
```

Memory Layout



Features

BLUE BRAIN PROJECT

- Python .fbs to C++ code generator
- Setters and getters for all members
- Static arrays and dynamic vectors
- Nested structures
  - Static and dynamic sized members
  - Arrays and vectors of static elements (builtin and static Zerobuf)
- Copyable, assignable
- toJSON, fromJSON

Features

BLUE BRAIN PROJECT

- Zero-copy read access to all members
- Seamless integration with ZeroEQ
  - zeq::Publisher::publish(zerobufObject);
  - zeq::Subscriber::subscribe(zerobufObject);
  - zeq::http::Server::add(zerobufObject);
- Universally unique type identifier
- toBinary, fromBinary: directly forwarded to memory buffer
- Update notification using C++ function or Qt signals

To Do

- Endian swap on receive
- Zerocopy in ZeroEq
  - Lock or COW zerobuf until async zmq_send completes?
- Profiling and Optimisation
  - Allocator behaviour in real scenarios
  - Alignment of members
  - No offset iff one dynamic member
  - Automatic compact?