ZeroBuf

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

Why Zerobuf?

- Zero-copy: data distributed is directly usable
- 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

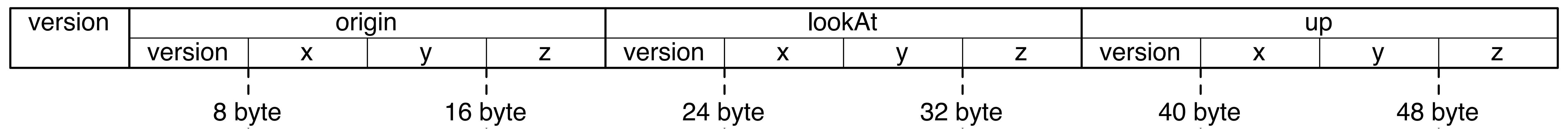
FBS Schema

```
namespace zerobuf.render;

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

table Camera
{
  origin: Vector3f;
  lookAt: Vector3f;
  up: Vector3f;
}
```

C++ Code

```
namespace render = zerobuf::render;

render::Camera camera;
camera.getLookAt() = render::Vector3f(0.,0.,1.);
assert( camera.toBinary().size == 52 );
```

Memory Layout



Example: Dynamic sized ZeroBuf

FBS Schema

```
TBD once we have a real world
example
```

C++ Code

```
namespace render = zerobuf::render;

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

Memory Layout

version	origin				lookAt				up			
	version	x	y	z	version	x	y	z	version	x	y	z

Features

- Python fbs to C++ code generator
- Setters and getters for all members
- Static arrays and dynamic vectors of elements
- Nested structures
    - Static and dynamic sized members
    - Arrays and vectors of static elements
- 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

To Do

- Endian swap on receive
- Zerocopy in ZeroEq/ZeroMQ
  - Lock zerobuf until async send completes?
- Profiling
  - Allocator behaviour in real scenarios
  - Alignment of members