Zerobuf

zero-hassle protocol buffers

Why Zerobuf?

- Flatbuffers promises zero copy, but does not deliver
- C++ code usable as data storage for application
- Random read/write access to serialised data

Features

- C++ class from flatbuffers .fbs:
  - Zero-copy raw pointer get/set
  - Single-copy STL vector get/set
  - Copyable, assignable
- ZeroBuf base class:
  - Universally unique type
  - Update notification
  - Zero-copy serializable buffer
  - To and from JSON

[camera.fbs]

```
namespace zerobuf.render;

table Camera
{
  origin: [float:3];
  lookAt: [float:3];
  up: [float:3];
}

root_type Camera;
```

```
> bin/zerobufCxx.py camera.fbs
```

[camera.h]

```
namespace zerobuf { namespace render {
class Camera : public zerobuf::ZeroBuf
{
public:
    CameraBase();
    CameraBase( const CameraBase& from );

    float* getOrigin();
    const float* getOrigin() const;
    std::vector< float > getOriginVector() const;
    void setOrigin( float value[ 3 ] );
    void setOrigin( const std::vector< float >& value );
    void setOrigin( const std::string& value );
…
}; }}
```

[ZeroBuf.h]

```
namespace zerobuf {
class Zerobuf
{
public:
    virtual servus::uint128_t getZerobufType() const = 0;
    virtual void notifyUpdated() {}

    ZEROBUF_API const void* getZerobufData() const;
    ZEROBUF_API size_t getZerobufSize() const;
    ZEROBUF_API void setZerobufData( const void* data, size_t si

    ZEROBUF_API std::string toJSON() const;
    ZEROBUF_API void fromJSON( const std::string& json );
}; }
```

How?

- One memory buffer for all class data
- Based on a python code generator
- Using Allocator trait
- Native support in ZeroEQ
  - zeq::Publisher::publish(const zerobuf::Zerobuf&);
  - zeq::Subscriber::subscribe(zerobuf::Zerobuf&);
  - zerobuf::Zerobuf::notifyUpdated()
- Self-contained classes
  - getZerobufType(), read/writeJSON()

Binary Allocation

```
[header][dynamic headers][static storage][dynamic storage]
header: version (==endianness)
dynamic storage headers: 8b offset, 8b size
  for all dynamic arrays in order of spec
  WIP dynamic nested classes
static storage: 1,2,4,8,16b (* static array size)
  for all static arrays and variables in order of spec
  later static nested classes
dynamic storage layout: detail of the Allocator
```

Done

- FBS python parser and C++ code generator
  - Members, enums, static and dynamic arrays
- Zerobuf base class
- Allocator interface, NonMovingAllocator
- Typesafe event UUID
- WIP
  - Strings (==dynamic nested members)

Todo

- Endian swap on receive
- Eliminate copies in applications
- Zerocopy in ZeroEq/ZeroMQ
  - Locked zerobuffer until async send completes?
- Profiling
  - Allocator behaviour in real scenarios
  - CompactAllocator or Allocator::compact
  - Alignment of members