Framebuffer Protocol v1.4

Protocol::Vec3(float x, float y, float z)

A struct with 3 (32-bit) floating point values that represent a 3d vector.

- x: Distance on the x-axis
- y: Distance on the y-axis
- z: Distance on the z-axis

```
// Make a vector (xyz)
auto vec = Protocol::Vec3(1.0f, 2.0f, 3.0f);
```

Protocol::Vec4(float x, float y, float z, float w)

A struct with 4 (32-bit) floating point values that represent a 4d vector.

- x: Distance on the x-axis
- y: Distance on the y-axis
- z: Distance on the z-axis
- w: Distance on the w-axis

```
// Make a vector (xyzw)
auto vec = Protocol::Vec4(1.0f, 2.0f, 3.0f, 1.0f);
```

Protocol::Laser(unsigned int uuid, unsigned long long start_time, unsigned long long end_time, Protocol::Vec3 origin, Protocol::Vec4 direction)

A struct with information to represent a laser anywhere in the world.

- uuid: A unique (32-bit) universal identifier for each entity.
- start_time: UNIX epoch time in milliseconds when the laser was created serverside.
- end_time: UNIX epoch time in milliseconds serverside the laser should be destroyed.
- speed: Speed of the laser, 1.0f speed must be equal to 1 unit/sec.
- origin: Start position of the laser, where it spawned.
- direction: The quaternion (direction) of the laser.

```
// Increment every time we make a new entity for a unique id
static unsigned int uuid = 0; uuid++;

// Find the current UNIX epoch time
auto now = std::chrono::system_clock::now();
auto duration = now.time_since_epoch();
auto epoch_ms =
std::chrono::duration_cast<std::chrono::milliseconds>(duration
).count();

// TODO: Do NOT use these xyz(w) values, instead use actual
data...
auto pos = Protocol::Vec3(1.0f, 2.0f, 3.0f);
auto dir = Protocol::Vec4(1.0f, 1.0f, 1.0f, 1.0f);

// Laser in some direction from a point in the world.
auto laser = Protocol::Laser(uuid, epoch_ms, 10.0f, pos, dir);
```

Protocol::Player(unsigned int uuid, Protocol::Vec3 position, Protocol::Vec3 velocity, Protocol::Vec3 acceleration, Protocol::Vec4 direction)
A struct with information to represent physics information for a player.

- uuid: A unique (32-bit) universal identifier for each entity.
- position: The current position of the player.
- velocity: The current velocity of the player.
- acceleration: The current acceleration of the player.
- direction: The quaternion (direction) of the player.

```
// Increment every time we make a new entity for a unique id
static unsigned int uuid = 0; uuid++;

// TODO: Do NOT use these xyz(w) values, instead use actual data...
auto pos = Protocol::Vec3(1.0f, 2.0f, 3.0f);
auto vel = Protocol::Vec3(0.0f, 0.0f, 1.0f);
auto acc = Protocol::Vec3(0.0f, 0.0f, 0.0f);
auto dir = Protocol::Vec4(1.0f, 1.0f, 1.0f, 1.0f);

// Player with some position, velocity, acceleration and direction.
auto player = Protocol::Player(0, pos, vel, acc, dir);
```

Framebuffer Protocol v1.4 (Packets)

Protocol::PacketType()
A union and enum of all packet types, allows any packet to be used for the packet wrapper.

Protocol::PacketWrapper(flatbuffers::Offset<void> packet)
A wrapper for all our packages that we want to send over a network, packets you want to send must be wrapped with their packet type before they are built.
- packet: The packet that should be wrapped and sent.

```
flatbuffers::FlatBufferBuilder builder =
flatbuffers::FlatBufferBuilder(1024);

auto text = builder.CreateString("Lorem ipsum");
auto textPacket = Protocol::CreateTextC2S(builder, text);

auto packetWrapper = Protocol::CreatePacketWrapper(builder,
Protocol::PacketType_TextC2S, textPacket.Union());
```

Server To Client

Protocol::ClientConnectS2C(uint32_t uuid, unsigned long long time)
A packet used to notify the client about what ship it will be controlling when it receives the initial game state.
- uuid: Unique identifier of the player that will be controlled by the client.
- time: The server time when this packet was sent, add the packet transmit time to this to get accurate server time on the client (use this to find difference).

Protocol::GameStateS2C(flatbuffers::Offset<flatbuffers::Vector<const Protocol::Player*>> players, flatbuffers::Offset<flatbuffers::Vector<const Protocol::Laser*>> lasers)
A packet used to communicate the current game state to a client, used when a player connects to the server.
- players: List of all players on the server at the point of connection.
- lasers: List of all lasers that exist in the world at point of connection.

Protocol::SpawnPlayerS2C(const Protocol::Player* player)
A packet used to spawn a player, used when a new player enters the game or an existing respawns.
- player: The player that will be spawned, internal player information decides who and where.

Protocol::DespawnPlayerS2C(uint32_t uuid)
A packet used to despawn a player, used when a player disconnects or an existing dies.

- uuid: Unique identifier of the player that will be despawned.

Protocol::UpdatePlayerS2C(uint64_t time, const Protocol::Player* player)

A packet used to update player movement information, used when the server has decided on where the player is in the world every n-th tick.
- time: The UNIX epoch time when the player information was sent from the server.
- player: The player that will be updated, internal player information decides who and where.

Protocol::TeleportPlayerS2C(uint64_t time, const Protocol::Player* player)

A packet used to teleport the player, no dead reckoning should be performed on this information. Sent when server cannot smoothly lerp between points.
- time: The UNIX epoch time when the player information was sent from the server.
- player: The player that will be teleported, internal player information decides who and where.

Protocol::SpawnLaserS2C(const Protocol::Laser* laser)

A packet used to spawn a laser, used when a player shoots.
- laser: The laser that will be spawned, internal laser information decides how and where.

Protocol::DespawnLaserS2C(uint32_t uuid)

A packet used to despawn a laser, used when a laser dies.
- uuid: Unique identifier of the laser that will be despawned.

Protocol::CollisionS2C(uint32_t uuid_first, uint32_t uuid_second)

A packet used to notify clients of a server decision where two entities collided, could be used to keep track of lives.
- uuid_first: Unique identifier of the first entity involved in the collision.
- uuid_second: Unique identifier of the second entity involved in the collision.

Protocol::TextS2C(flatbuffers::Offset<flatbuffers::String> text)

A packet used to send plaintext from server to clients.
- text: The text that will be sent.

Client To Server

Protocol::InputC2S(unsigned long long time, unsigned short bitmap)

A packet used to send input from client to server, should be called every time there is a change of input buttons.

- time: The UNIX epoch time when the input was sent from the client.
- bitmap: A bitmap of all input keys (16-bit), in binary 1 means pressed and 0 is released.

```
// Pressed/released keys (true/false).
bool w = true;
bool a = false;
bool d = true;

// Same as (1 * 1) + (0 * 2) + (1 * 4) = 5
// The result will always be unique to the combination
// of pressed keys due to how bitmaps work.
unsigned short result = (w * 1) + (a * 2) + (d * 4);
```

KEY_TYPE	W	A	D	UP	DOWN	LEFT	RIGHT	SPACE	SHIFT
BIT_SHIFT	0	1	2	3	4	5	6	7	8
BASE_10	1	2	4	8	16	32	64	128	256

Protocol::TextC2S(flatbuffers::Offset<flatbuffers::String> text)

A packet used to send plaintext from client to server.

- text: The text that will be sent.