SandBox Wars 3D

Socket Protocol

SandBox Wars 3D Socket Protocol

1. Packet specifications

1.1. Packet format

Lidgren packet data:

- TYPE (Reliable Ordered OR Unreliable)
- CHANNEL (0 ~ 31)

SandBox Wars 3D packet data (initialization packet):

- SERVICE ID (byte: 0 ~ 255)
- ACTUAL DATA (variable)

SandBox Wars 3D packed data (response packet):

- ERROR CODE (short: see "Error Codes" section)
- ACTUAL DATA (variable)

Legend:

Client ← Server	Only server to client, no reply
Client → Server	Only client to server, no reply
Client ↔ Server	Both can start, no reply
Client ↔ Server	Client starts, server replies
Client ↔ *Server*	Server starts, client replies
Client ↔ *Server*	Both can start, both can reply

TYPE	CH	DIRECTION	PROCEDURE
ReliableOrdered	0	*Client* ↔ Server	Service packets (login, register)
ReliableOrdered	1	Client ← Server	Special updates (spawn, detect, etc.)
Unreliable	0	Client ↔ Server	Frequent updates (position, etc.)

1.2. Derived types

1.2.1. Vector3f

Base types **3x** float

float(x coordinate)
float(y coordinate)
float(z coordinate)

1.2.2. Rigidbody

Base types **1x** bool + **2~4x** Vector3f

bool(static: true if rigidbody is not moving, false otherwise)
SKIP PADDING BITS
Vector3f(location)
Vector3f(rotation)
if not static
    Vector3f(inertia)
    Vector3f(momentum)

2. Socket Protocol

Notes:
RSA keys are 2048bits and will be regenerated at least once every 24 hours.
Every RSA-involving procedure cannot last more than 10 minutes.

2.1. Service

2.1.1. Registration

1. ***ReliableOrdered CH0 ID0***

```
Client → Server     byte(0)
                    string(username)
Client ← Server     if user already exists
                        short(2) [Username already taken]
                        INTERRUPT PROCEDURE
                    else
                        short(0) [OK]
                        int(public RSA key Exponent array's length)
                        int(public RSA key Modulus array's length)
                        byte[](public RSA key Exponent)
                        byte[](public RSA key Modulus)
Client → Server     int(data array's length)
                    byte[](RSA(password))
                    string(e-mail address)
Client ← Server     if RSA key timed out
                        short(7) [Procedure timed out]
                        INTERRUPT PROCEDURE
                    else if RSA can be decoded and is a valid password
                        if e-mail is valid and not already registered
                            short(0) [OK]
                        else
                            short(6) [E-Mail already registered]
                            INTERRUPT PROCEDURE
                    else
                        short(4) [Invalid password]
                        INTERRUPT PROCEDURE
```

Client invokes this procedure when is trying to make the server generate a new user, the user shouldn't be usable right after this procedure, it must be verified before proceeding with playing.
A client cannot invoke this procedure after logging in.

2.1.2. Log in

1. ***ReliableOrdered CH0 ID1***

```
Client → Server     byte(1)
                    string(username)
Client ← Server     if user exists
                        short(0) [OK]
                        string(random string)
                    else
                        short(3) [Invalid username]
                        INTERRUPT PROCEDURE
Client → Server     int(data array's length)
                    byte[](SHA1(password + random string))
```

Client ← Server if received data is valid for previously selected user
short(0) [OK]
else if same user is already logged in
short(5) [User already logged]
INTERRUPT PROCEDURE
else
short(4) [Invalid password]
INTERRUPT PROCEDURE

Client invokes this procedure before any other to allow server recognizing it as a valid user.
A client cannot invoke this procedure after logging in.

2.1.3. E-Mail verification

ReliableOrdered CH0 ID2

Client → Server byte(2)
string(e-mail address)
string(verification code)
Client ← Server if verification code is valid for specified e-mail address
short(0) [OK]
else if e-mail address is already verified
short(11) [E-Mail already verified]
else
short(10) [Invalid verification code]

Client invokes this procedure to confirm the validity of an e-mail address by posting a code that gets sent to the e-mail address specified during registration.

2.2. Special updates

2.2.1. Player or creature spawn

ReliableOrdered CH1 ID0

Client ← Server byte(0)
uint(ID of the spawned creature)

Server informs client of the presence of a creature.

2.2.2. Object spawn

ReliableOrdered CH1 ID1

Client ← Server byte(1)
uint(ID of the spawned item)
uint(item identifier) [see “Item Identifiers” section]

Server informs the client of the presence of an object.

2.2.3. Player or creature or object destruction

ReliableOrdered CH1 ID2

Client ← Server byte(2)
uint(ID of the destroyed thing)

Server informs the client that something destruction (got killed, got picked up, got destroyed or player’s avatar lost contact with).

2.2.4. Chat message (Client to Server)

ReliableOrdered CH1 ID3

Client → Server byte(3)
string(message)

Client sends a message to the server, the server will reply this message to all players in range of the sender.

2.2.5. Chat message (Server to Client)

ReliableOrdered CH1 ID3
Client ← Server byte(3)
string(sender's username)
string(message)

When a client sends a message to the server, this will reply the message to all players in range, including the sender itself.

2.3. Frequent updates

2.3.1. Player update (Client to Server)

Unreliable CH0 ID0
Client → Server byte(0)
Vector3f(location)
float(rotation y)
Vector3f(inertia)

Client informs server of an update of his position, should be sent every ROUNDTRIP_TIME/2 milliseconds, at least once every 500ms and maximum one time every 50ms.

2.3.2. Player or creature update (Server to Client)

Unreliable CH0 ID0
Client ← Server byte(0)
uint(ID of the creature to update)
Vector3f(location)
float(rotation y)
Vector3f(inertia)

This packet describes the new status of a player or a creature (ID 0 for the client's avatar).
This packet should be skipped if the client didn't receive a spawn notification for the creature ID.
This packet should be sent from the server every time a creature moves or rotates (max. one time every 10ms) and always every 1000ms since the last update.

2.3.3. Object update (Server to Client)

Unreliable CH0 ID1
Client ← Server byte(1)
uint(ID of the object to update)
Rigidbody(status)

This packet describes the new status of an object, it should be sent from the server every time an object moves or rotates (max. one time every 25ms) and always every 3000ms since the last update.

3. Identification Codes

3.1. Error Codes

0 OK
1 Invalid packet
2 Username already taken
3 Invalid username
4 Invalid password
5 User already logged
6 E-Mail already registered
7 Procedure timed out
8 Generic/Unknown error
9 Forbidden procedure call
10 Invalid verification code
11 E-Mail already verified

3.2. Item Identifiers

0 Debug box