

# **Riptide Networking**

## **Notes**

Version 2.1.0

February 6, 2024



# Contents

<b>I. Low Level Documentation</b>	<b>7</b>
<b>1. Messages</b>	<b>9</b>
1.1. Header . . . . .	9
1.1.1. Header Byte . . . . .	9
1.2. Value Encoding . . . . .	10
1.2.1. Byte . . . . .	10
1.2.2. Signed Byte . . . . .	10
1.2.3. Boolean . . . . .	11
1.2.4. Short . . . . .	11
1.2.5. UShort . . . . .	12
1.2.6. Int . . . . .	12
1.2.7. UInt . . . . .	13
1.2.8. Long . . . . .	13
1.2.9. ULong . . . . .	14
1.2.10. Float . . . . .	14
1.2.11. Double . . . . .	15
1.2.12. String . . . . .	16
1.2.13. VarLong . . . . .	16
1.2.14. VarULong . . . . .	16
1.2.15. Vector2 . . . . .	16
1.2.16. Vector3 . . . . .	17
1.2.17. Quaternion . . . . .	17
1.2.18. IMessageSerializable . . . . .	17
1.2.19. Array Length . . . . .	18
1.3. Message Types . . . . .	18
1.3.1. Unreliable . . . . .	18
1.3.2. Ack . . . . .	18
1.3.3. Connect . . . . .	19
1.3.4. Reject . . . . .	19
1.3.5. Heartbeat . . . . .	19
1.3.6. Disconnect . . . . .	19
1.3.7. Notify . . . . .	20
1.3.8. Reliable . . . . .	20
1.3.9. Welcome . . . . .	20
1.3.10. ClientConnected . . . . .	20

1.3.11. ClientDisconnected . . . . .	21
1.4. Enums . . . . .	21
1.4.1. Reject Reasons . . . . .	21
1.4.2. Disconnect Reasons . . . . .	21
<b>2. Protocol</b>	<b>23</b>
2.1. General Information . . . . .	23
2.2. Connection . . . . .	23
2.3. Heartbeat . . . . .	23
2.3.1. Client . . . . .	23
2.3.2. Server . . . . .	24
2.4. Reliable Messages . . . . .	24
2.4.1. Duplicate Detection . . . . .	24
2.4.2. Acknowledge Messages . . . . .	25
2.5. Transport Details . . . . .	26
2.5.1. UDP . . . . .	26
2.5.2. TCP . . . . .	26
<b>3. Events</b>	<b>27</b>
3.1. Transport . . . . .	27
3.1.1. IPeer . . . . .	27
3.1.2. IClient . . . . .	27
3.1.3. IServer . . . . .	27
3.2. Interface . . . . .	28
3.2.1. Client . . . . .	28
3.2.2. Server . . . . .	29
<b>4. Usage</b>	<b>31</b>
4.1. C# . . . . .	31
4.1.1. Initialize Logging . . . . .	31
4.1.2. Create Server . . . . .	31
4.1.3. Create Client . . . . .	31
4.1.4. Messages . . . . .	32
4.2. Python . . . . .	32
4.2.1. Create Server . . . . .	32
4.2.2. Create Client . . . . .	32
4.2.3. Messages . . . . .	32
<b>II. Changelog</b>	<b>33</b>
<b>5. Version 2.1.0</b>	<b>35</b>
5.1. Enums . . . . .	35

5.2. Message Types . . . . .	35
5.2.1. AckExtra . . . . .	35
5.3. Client.cs . . . . .	35
5.4. Peer.cs . . . . .	35
5.4.1. Methods . . . . .	35



## **Part I.**

# **Low Level Documentation**





# 1. Messages

## 1.1. Header

Name	Type	Comment
Header	1 Byte	Header byte
Sequence ID	2 Byte	Optional, only for reliable messages
Message ID	UShort	Only present in user defined messages

### 1.1.1. Header Byte

Name	Value	Description
Unreliable	0	An unreliable user message <b>Unreliable</b>
Ack	1	An internal unreliable ack message <b>Unreliable</b>
Connect	2	An internal unreliable connect message <b>Unreliable</b>
Reject	3	An internal unreliable connection rejection message. <b>Unreliable</b>
Heartbeat	4	An internal unreliable heartbeat message. <b>Unreliable</b>
Disconnect	5	An internal unreliable disconnect message. <b>Unreliable</b>
Notify	6	A notify message. <b>Notify-Type</b>
Reliable	7	A reliable user message. <b>Reliable</b>
Welcome	8	An internal reliable welcome message. <b>Reliable</b>
ClientConnected	9	An internal reliable client connected message. <b>Reliable</b>
ClientDisconnect	10	An internal reliable client disconnected message. <b>Reliable</b>

Runtime/Core/Transport/IPeer.cs

Maximum size of message Body defaults to **1225** Bytes (Runtime/Core/Message.cs:30)

## 1.2. Value Encoding

### 1.2.1. Byte

Unsigned 8 bit integer

#### Single

Name	Bit Count	Description
value	byte	Single byte, without any further processing

#### Array

Name	Bit Count	Description
Array Length	byte/ushort	<b>OPTIONAL</b> See Array Length for details
value	byte[]	Byte array with a maximum length of $2^{16}$ . Directly copied into message body.

### 1.2.2. Signed Byte

Signed 8 bit integer

#### Single

Name	Bit Count	Description
value	sbyte	Single signed byte, cast to two's complement encoded unsigned byte

#### Array

Name	Bit Count	Description
Array Length	byte/ushort	<b>OPTIONAL</b> See Array Length for details
value	sbyte[]	SByte array, no maximum length. Cast to byte and copied into array one by one

### 1.2.3. Boolean

#### Single

Name	Bit Count	Description
value	bool	Single boolean, encoded as single byte, with value 0x01 for <b>true</b> or value 0x00 for <b>false</b> . Values other than 0x01 will be interpreted as <b>false</b> when reading the message.

#### Array

Name	Bit Count	Description
Array Length	byte/ushort	<b>OPTIONAL</b> See Array Length for details. <b>Beware:</b> The Array length counts the number of boolean objects in the original Array. This is not equal to the number of bytes used for storing the Array
value	bool[]	bool array, no maximum length. Booleans are packet into bytes (8 booleans per byte). That means the first bool is represented as the lowest bit of the first byte, the second is the second lowest bit and so on

### 1.2.4. Short

16 Bit signed integer

#### Single

Name	Bit Count	Description
value	short	short, taking 2 bytes in the message. <b>Endianness is dependent on the host system's .net implementation. Assume Little Endian per default</b>

#### Array

Name	Bit Count	Description
Array Length	byte/ushort	<b>OPTIONAL</b> See Array Length for details

value	<code>short[]</code>	Shorts added sequentially using the method for adding single shorts. <b>Endianness is dependent on the host system's .net implementation. Assume Little Endian per default</b>
-------	----------------------	--

### 1.2.5. UShort

16 Bit unsigned integer

#### Single

Name	Bit Count	Description
value	<code>ushort</code>	ushort, taking 2 bytes in the message. <b>Endianness is dependent on the host system's .net implementation. Assume Little Endian per default</b>

#### Array

Name	Bit Count	Description
Array Length	<code>byte/ushort</code>	<b>OPTIONAL</b> See Array Length for details
value	<code>ushort[]</code>	UShorts added sequentially using the method for adding single ushorts. <b>Endianness is dependent on the host system's .net implementation. Assume Little Endian per default</b>

### 1.2.6. Int

32 Bit signed integer

#### Single

Name	Bit Count	Description
value	<code>int</code>	int, taking 4 bytes in the message. <b>Endianness is dependent on the host system's .net implementation. Assume Little Endian per default</b>

**Array**

Name	Bit Count	Description
Array Length	byte/ushort	OPTIONAL See Array Length for details
value	int []	Integers added sequentially using the method for adding single ints. <b>Endianness is dependent on the host system's .net implementation. Assume Little Endian per default</b>

**1.2.7. UInt**

32 Bit unsigned integer

**Single**

Name	Bit Count	Description
value	uint	uint, taking 4 bytes in the message. <b>Endianness is dependent on the host system's .net implementation. Assume Little Endian per default</b>

**Array**

Name	Bit Count	Description
Array Length	byte/ushort	OPTIONAL See Array Length for details
value	uint []	Integers added sequentially using the method for adding single uints. <b>Endianness is dependent on the host system's .net implementation. Assume Little Endian per default</b>

**1.2.8. Long**

64 Bit signed integer

**Single**

Name	Bit Count	Description
------	-----------	-------------

value	long	long, taking 8 bytes in the message. <b>Endianness is dependent on the host system's .net implementation. Assume Little Endian per default</b>
-------	------	--

**Array**

Name	Bit Count	Description
Array Length	byte/ushort	<b>OPTIONAL</b> See Array Length for details
value	long[]	Integers added sequentially using the method for adding single longs. <b>Endianness is dependent on the host system's .net implementation. Assume Little Endian per default</b>

**1.2.9. ULong**

64 Bit unsigned integer

**Single**

Name	Bit Count	Description
value	ulong	ulong, taking 8 bytes in the message. <b>Endianness is dependent on the host system's .net implementation. Assume Little Endian per default</b>

**Array**

Name	Bit Count	Description
Array Length	byte/ushort	<b>OPTIONAL</b> See Array Length for details
value	ulong[]	Integers added sequentially using the method for adding single ulongs. <b>Endianness is dependent on the host system's .net implementation. Assume Little Endian per default</b>

**1.2.10. Float**

32 Bit signed IEEE floating point

**Single**

Name	Bit Count	Description
value	float	float, taking 4 bytes in the message, Encoded in IEEE format, seperated into its single bytes. <b>Endianness is dependent on the host system's .net implementation. Assume Little Endian per default</b>

**Array**

Name	Bit Count	Description
Array Length	byte/ushort	<b>OPTIONAL</b> See Array Length for details
value	float[]	Floats added sequentially using the method for adding single IEEE floats. <b>Endianness is dependent on the host system's .net implementation. Assume Little Endian per default</b>

**1.2.11. Double**

64 Bit signed IEEE floating point

**Single**

Name	Bit Count	Description
value	double	double, taking 8 bytes in the message, Encoded in IEEE format, seperated into its single bytes. <b>Endianness is dependent on the host system's .net implementation. Assume Little Endian per default</b>

**Array**

Name	Bit Count	Description
Array Length	byte/ushort	<b>OPTIONAL</b> See Array Length for details
value	double[]	Floats added sequentially using the method for adding single IEEE doubles. <b>Endianness is dependent on the host system's .net implementation. Assume Little Endian per default</b>

**1.2.12. String**

UTF-8 Encoded String

**Single**

Name	Bit Count	Description
<b>length</b>	<i>byte/ushort</i>	Length of the encoded byte array
<b>value</b>	<i>byte[]</i>	UTF-8 encoded string

**Array**

Name	Bit Count	Description
<b>Array Length</b>	<i>byte/ushort</i>	<b>OPTIONAL</b> See Array Length for details
<b>value</b>	<i>string[]</i>	Strings added sequentially using the method for adding single Strings.

**1.2.13. VarLong**

Name	Bit Count	Description
<b>value</b>	<i>variable</i>	UTF-Like encoding: left most bit used in order to indicate continuation (1: not last byte, 0: is last byte). Next seven bits are then the actual value of that byte. Maximum 64 bit integers supported in Riptide. Encoded such that the <b>signed bit is the right most bit</b> , in order to minimize bits for negative numbers

**1.2.14. VarULong**

Name	Bit Count	Description
<b>value</b>	<i>variable</i>	UTF-Like encoding: left most bit used in order to indicate continuation (1: not last byte, 0: is last byte). Next seven bits are then the actual value of that byte. Maximum 64 bit integers supported in Riptide

**1.2.15. Vector2**



Name	Bit Count	Description
value.x	float	x component of Vector2
value.y	float	y component of Vector2

### 1.2.16. Vector3

Name	Bit Count	Description
value.x	float	x component of Vector3
value.y	float	y component of Vector3
value.z	float	z component of Vector3

### 1.2.17. Quaternion

Name	Bit Count	Description
value.x	float	x component of Quaternion
value.y	float	y component of Quaternion
value.z	float	z component of Quaternion
value.w	float	w component of Quaternion

### 1.2.18. IMessageSerializable

Custom Structures

#### Single

Name	Bit Count	Description
value	IMessage- Serializable	Serialized using Serialize() method of this Object, deserialized using Deserialize() method of the type. Expected type has to be declared by user, Type must have no-parameter constructor

#### Array

Name	Bit Count	Description
Array Length	byte/ushort	OPTIONAL See Array Length for details
value	IMessage- Serializable[]	IMessageSerializables added sequentially using the method for adding single IMessageSerializables.

### 1.2.19. Array Length

Supports array lengths up to 32767 elements. Larger arrays are not supported.

The actual serialisation of the array length depends on the value serialized. If the value is less than 127, a single byte is used. Otherwise, two bytes are used.

#### Values $\leq 127$

Name	Bit Count	Description
value	byte	Length serialized in a single byte with the highest bit set to 0

#### Values $> 127$ and $\leq 32767$

Name	Bit Count	Description
value	byte	High byte of the length, with the highest bit set to 1 in order to indicate 2 bytes used for the array length
value	byte	Lower byte of the length

#### Values $> 32767$

Throws argument out of range exception.

## 1.3. Message Types

### 1.3.1. Unreliable

An unreliable user message

Name	Bit Count	Description
Message Type	byte	Value set to 0
Message ID	ushort	Message ID
payload	byte[]	Payload defined by user, with data types serialized as described in Value Encoding

### 1.3.2. Ack

An internal unreliable ack message

Name	Bit Count	Description
Message Type	byte	Value set to 1

LastReceived-SeqId	ushort	Last remote sequence ID
AcksBitfield	ushort	Acks (binary flags)

### 1.3.3. Connect

An internal unreliable connect message

Name	Bit Count	Description
Message Type	byte	Value set to 2
connectBytes	byte[]	<b>OPTIONAL</b> Custom data to include when connecting. <b>Length of the Array is not included in the message</b>

### 1.3.4. Reject

An internal unreliable connection rejection message

Name	Bit Count	Description
Message Type	byte	Value set to 3
RejectReason	byte	Reason for the rejection of the connection. See also Reject Reasons
rejectMessage	byte[]	<b>OPTIONAL</b> custom byte[] containing additional data. See Value Encoding for value. Length of the array is not included. If this field is present, <b>RejectReason must be set to Custom</b>

### 1.3.5. Heartbeat

An internal unreliable heartbeat message

Name	Bit Count	Description
Message Type	byte	Value set to 4
Ping ID	byte	Ping ID of the message
RTT	short	Round trip time, -1 if not calculated yet

### 1.3.6. Disconnect

An internal unreliable disconnect message

Name	Bit Count	Description
Message Type	byte	Value set to 5

Reason	byte	Disconnect reason, see also Disconnect Reasons
Message	byte[]	<b>OPTIONAL</b> custom byte[] containing additional data. See Value Encoding for value. Length of the array is not included. If this field is present, <b>Disconnect Reason must be set to Kicked</b>

### 1.3.7. Notify

Name	Bit Count	Description
Message Type	4	Value set to 6
Message ID	ushort	Message ID

### 1.3.8. Reliable

A reliable user message

Name	Bit Count	Description
Message Type	byte	Value set to 7
Sequence ID	ushort	Sequence ID
Message ID	ushort	Message ID
payload	byte[]	Payload defined by user, with data types serialized as described in Value Encoding

### 1.3.9. Welcome

An internal reliable welcome message

Name	Bit Count	Description
Message Type	byte	Value set to 8
Sequence ID	ushort	Sequence ID
ID	ushort	Connection ID

### 1.3.10. ClientConnected

An internal reliable client connected message. Send to all clients when a new client connects.

Name	Bit Count	Description
Message Type	byte	Value set to 9
Sequence ID	ushort	Sequence ID

ID	ushort	Client ID
----	--------	-----------

### 1.3.11. ClientDisconnected

An internal reliable client disconnected message. Send to all still connected clients when a client disconnects.

Name	Bit Count	Description
Message Type	byte	Value set to 10
Sequence ID	ushort	Sequence ID
ID	ushort	Client ID

## 1.4. Enums

### 1.4.1. Reject Reasons

See `Core/Peer.cs`

Name	Value	Description
0	NoConnection	No response was received from the server (because the client has no internet connection, the server is offline, no server is listening on the target endpoint, etc.).
1	AlreadyConnected	The client is already connected.
2	Pending	A connection attempt is already pending.
3	ServerFull	The server is full.
4	Rejected	The connection attempt was rejected.
5	Custom	The connection attempt was rejected and custom data may have been included with the rejection message.

### 1.4.2. Disconnect Reasons

See `Core/Peer.cs`

Name	Value	Description
0	NeverConnected	No connection was ever established
1	ConnectionRejected	The connection attempt was rejected by the server
2	TransportError	The active transport detected a problem with the connection

3	TimedOut	The connection timed out or the real reason for the disconnect was lost / is unclear
4	Kicked	The client was forcibly disconnected by the server
5	ServerStopped	The server shut down
6	Disconnected	The disconnection was initiated by the client
7	PoorConnection	The connection's loss and/or resend rates exceeded the maximum acceptable thresholds, or a reliably sent message could not be delivered.

## 2. Protocol

### 2.1. General Information

- **HeartbeatInterval:** 1000 ms
- **Timeout:** 5000 ms
- **Resend Interval:**  $1.2 * \text{smoothRTT}$
- **Resend Attempts:** 15

### 2.2. Connection

1. Client initiates connection from Transport and Client starts heartbeat messages
2. The server either
  - accepts the connection (if no custom connection handler is specified)
  - adds the connection to the list of pending connections, if a custom connection handler is specified & calls the handler.
3. Once the connection is accepted, a welcome message is sent from the server to the client

### 2.3. Heartbeat

The heartbeat is used in order to check if connections are timed out, and to measure the round trip time for packets

#### 2.3.1. Client

The heartbeat is started once the connection is initiated (by calling the connect method). The behavior of the heartbeat depends on the current state of the client.

### **isConnecting**

If the maximum connect attempts are not reached, sends a connect message to the remote peer. If connectBytes is not Null, the connect bytes are appended. The connection attempt counter gets incremented. Otherwise, a local disconnect is called with reason "NeverConnected"

Finally, the next heartbeat event is scheduled.

### **isPending**

If the current connection attempt timed out, a local disconnect is called with reason "TimedOut"

Otherwise the next heartbeat event is scheduled.

### **isConnected**

If the current connection timed out, a local disconnect is called with reason "TimedOut". Otherwise a heartbeat is sent and the next heartbeat event is scheduled.

### **Other states**

The next heartbeat event is scheduled.

## **2.3.2. Server**

The heartbeat is started once the server starts.

## **2.4. Reliable Messages**

Reliable messages are messages that need to be acknowledged by the peer. If within 1.2 times the current smoothed round trip time (as determined by the heartbeat) no ack is received, the message is resent. If more than 15 attempts are tried, the message gets discarded, and a warning gets logged, if enabled.

### **2.4.1. Duplicate Detection**

See also `Core/Connection.cs:141`

First the gap between the received sequence ID and the previously received sequence ID is computed. The next steps depend on the sign of the gap:

#### **Positive Gap (larger sequence ID received)**

Once a reliable message with an newer sequence ID than the previous (= positive sequence Gap) is received, an 64 bit long bit field is shifted left by the gap since the last received sequence ID.



- if gap  $\leq 16$ :** the acks bitfield gets shifted by sequence bits. The new acks bitfield now consists of the two most minor bytes of the shifted value, the "Overflow" gets then or-ed into the duplicate filter bitfield  
The message is handled if the bit in the acks bit field at position sequenceGap is zero. When handling, this bit is flipped to one.  
The last received sequence ID is set to this message's sequence ID
- if gap  $\leq 80$ :** Shifts the acks bit field by sequenceGap-16. The shifted bits are or-ed into the duplicate filter bitfield, and the acks bit field is zeroed  
The message is handled if the bit in the duplicate bit field at position sequenceGap-16 is zero. When handling, this bit is flipped to one.  
The last received sequence ID is set to this message's sequence ID

#### Negative Gap (smaller sequence ID received)

- if gap  $\leq 16$ :** The message is handled if the bit in the acks bit field at position  $\text{abs}(\text{sequenceGap})$  is zero. When handling, this bit is flipped to one.
- if gap  $\leq 80$ :** The message is handled if the bit in the duplicate bit field at position  $\text{abs}(\text{sequenceGap})$  is zero. When handling, this bit is flipped to one.

#### Gap of 0

Is not handled.

Finally, an Ack message is sent for the sequence ID

### 2.4.2. Acknowledge Messages

See also `Core/Connection.cs:233`

First the gap between the received sequence ID and the previously received sequence ID is computed. The next steps depend on the sign of the gap:

#### Positive Gap (larger sequence ID received)

For each id in the gap, excluding the current message, first the acked messages bit field is shifted left by one. Then, the left most message is checked for ack status. If the message is already acked, the pending message gets cleared from the pending messages dict, if still present. If no ack for the message has been received yet, the message is resent if present in the pending messages dict.

Once all previous messages from the acked messages bit field are checked, the bit field gets shifted once more left to make space for the ack bit of the current sequenceID. The `ackedMessagesBitfield` is then or-ed with the remote acks bitfield from the ack message and the ack bit for the current sequence ID.

The `lastAckedSeqID` is then set to the remote last received SeqID

### Negative Gap (smaller sequence ID received)

According to comments in the source, this branch most likely never executes.

The bit corresponding to the sequence ID is set, and the local acked bitfield is or-ed with the remote acksBitField, ensuring that the bit corresponding to this ack is set to 1.

If there exists still a pending message for this ack, the pending message is cleared.

### Gap of 0

The remote and local bit fields are combined (using binary or), and the ack status of the oldest sequence ID is checked.

## 2.5. Transport Details

### 2.5.1. UDP

If not otherwise specified, UDP is used as transport.

- Default Socket buffer size: 1 MB (1024\*1024 Byte)
- Minimal Socket buffer size: 256Kb (256\*1024 Byte)
- Receive Polling Frequency: 2 Hz (Every 0.5 Seconds)

### 2.5.2. TCP

#### Package Format

Name	Bit Count	Description
packetLength	ushort	Length of the package
packet	Message	Content of the package (= the Message)

Please note that the byteOrder of packetLength is again most likely system dependent. Expect Little Endian as default. (if the Symbol "BIG\_ENDIAN" is not defined)

## 3. Events

### 3.1. Transport

#### 3.1.1. IPeer

##### DataReceived

Name	Bit Count	Description
dataBuffer	byte[]	An array containing the received data
amount	int	The number of bytes that were received
fromConnection	Connection	The connection which the data was received from

##### Disconnected

Name	Bit Count	Description
connection	Connection	The connection which was closed
reason	DisconnectReason	The reason for the disconnection

#### 3.1.2. IClient

Inherits events from IPeer

##### Connected

No Arguments

##### ConnectionFailed

No Arguments

#### 3.1.3. IServer

Inherits events from IPeer

**Connected**

Name	Bit Count	Description
connection	Connection	Connection that just got connected

**3.2. Interface****3.2.1. Client****Connected**

No Arguments

**ConnectionFailed**

Name	Bit Count	Description
message	Message	Additional data related to the failed connection attempt (if any)

**MessageReceived**

Name	Bit Count	Description
fromConnection	Connection	The connection from which the message was received
messageID	ushort	ID of the Message
message	Message	the received Message

**Disconnected**

Name	Bit Count	Description
reason	DisconnectReason	The reason for the disconnection
message	Message	additional data related to the disconnection (may be null)

**ClientConnected**

Name	Bit Count	Description
id	ushort	The numeric ID of the client that connected

**ClientDisconnected**

Name	Bit Count	Description
id	ushort	The numeric ID of the client that disconnected

**3.2.2. Server****ClientConnected**

Name	Bit Count	Description
client	Connection	The newly connected client

**MessageReceived**

Name	Bit Count	Description
fromConnection	Connection	The connection from which the message was received
messageID	ushort	ID of the Message
message	Message	the received Message

**ClientDisconnected**

Name	Bit Count	Description
client	Connection	The client that disconnected
reason	DisconnectReason	The reason for disconnection



## 4. Usage

### 4.1. C#

Based on <https://riptide.tomweiland.net/manual/overview/getting-started.html>

#### 4.1.1. Initialize Logging

```
1 RiptideLogger.Initialize(Debug.Log, Debug.Log, Debug.LogWarning, Debug.  
  |   LogError, false);
```

#### 4.1.2. Create Server

```
1 Server server = new Server();  
2 server.Start(7777, 10);
```

In order to process the messages:

```
1 private void FixedUpdate()  
2 {  
3     server.Update();  
4 }
```

#### 4.1.3. Create Client

```
1 Client client = new Client();  
2 client.Connect("127.0.0.1:7777");
```

In order to process the messages:

```
1 private void FixedUpdate()  
2 {  
3     client.Update();  
4 }
```

### 4.1.4. Messages

```
1 | Message message = Message.Create(MessageSendMode.Unreliable, 1);
```

## 4.2. Python

### 4.2.1. Create Server

```
1 | tcpTransport = TCPServer()  
2 | server: Server = Server(tcpTransport)  
3 | server.start(PORT, 10)
```

In order to process the messages:

```
1 | serverUpdater: FixedUpdateThread = FixedUpdateThread(server.update)  
2 | serverUpdater.start()
```

### 4.2.2. Create Client

```
1 | tcpTransport = TCPClient()  
2 | client: Client = Client(tcpTransport)  
3 | client.connect(("127.0.0.1", PORT))
```

In order to process the messages:

```
1 | clientUpdater: FixedUpdateThread = FixedUpdateThread(client.update)  
2 | clientUpdater.start()
```

### 4.2.3. Messages

```
1 | msg = message.create(MessageSendMode.Unreliable, MESSAGEID.HANDLED)  
2 | msg.putString("Hello World !")  
3 | client.send(msg)
```



## **Part II.**

# **Changelog**



## 5. Version 2.1.0

### 5.1. Enums

- DisconnectReason: Added PoorConnection item
- **MessageHeader: Removed AckExtra, added Notify. This decrements the values for Connect, Reject, Heartbeat and Disconnect by one**
- ConnectionState: Removed Rejected State

### 5.2. Message Types

#### 5.2.1. AckExtra

Removed

### 5.3. Client.cs

### 5.4. Peer.cs

#### 5.4.1. Methods

- **Added:** Disconnect(Connection, DisconnenctReason) (abstract, no implementation)