Protocol

Revision as of 16:48, 8 August 2018 by Fayaru (talk | contribs) (→Join Game)
(diff) ← Older revision | Latest revision (diff) | Newer revision → (diff)

Heads up!

This article is about the protocol for the latest **stable** release of Minecraft **computer edition** (1.12.2, protocol 340). For the computer edition pre-releases, see Pre-release protocol. For Pocket Edition, see Pocket Edition Protocol Documentation.

This page presents a dissection of the current **Minecraft (https://minecraft.net/) protocol**.

If you're having trouble, check out the FAQ or ask for help in the IRC channel #mcdevs on chat.freenode.net (irc://irc.freenode.net/mcdevs) (More Information (http://wiki.vg/MCDevs)).

Note: While you may use the contents of this page without restriction to create servers, clients, bots, etc... you still need to provide attribution to #mcdevs if you copy any of the contents of this page for publication elsewhere.

The changes between versions may be viewed at Protocol History.

Contents

Spawn Mob
Spawn Painting
Spawn Player
Animation (clientbound)
Statistics
Block Break Animation
Update Block Entity
Block Action
Block Change
Boss Bar
Server Difficulty
Tab-Complete (clientbound)
Chat Message (clientbound)
Multi Block Change
Confirm Transaction (clientbound)
Close Window (clientbound)
Open Window
Window Items
Window Property
Set Slot
Set Cooldown
Plugin Message (clientbound)
Named Sound Effect
Disconnect (play)
Entity Status
Explosion
Unload Chunk
Change Game State
Keep Alive (clientbound)
Chunk Data
Effect
Particle
Join Game
Map
Entity
Entity Relative Move
Entity Look And Relative Move
Entity Look
Vehicle Move (clientbound)
Open Sign Editor
Craft Recipe Response
Player Abilities (clientbound)
Combat Event
Player List Item
Player Position And Look (clientbound)
Use Bed
Unlock Recipes
Destroy Entities
Remove Entity Effect
Resource Pack Send
Respawn

- Entity Head Look
- Select Advancement Tab
- World Border
- Camera
- Held Item Change (clientbound)
- Display Scoreboard
- Entity Metadata
- Attach Entity
- Entity Velocity
- Entity Equipment
- Set Experience
- Update Health
- Scoreboard Objective
- Set Passengers
- Teams
- Update Score
- Spawn Position
- Time Update
- Title
- Sound Effect
- Player List Header And Footer
- Collect Item
- Entity Teleport
- Advancements
- Entity Properties
- Entity Effect

Serverbound

- Teleport Confirm
- Tab-Complete (serverbound)
- Chat Message (serverbound)
- Client Status
- Client Settings
- Confirm Transaction (serverbound)
- Enchant Item
- Click Window
- Close Window (serverbound)
- Plugin Message (serverbound)
- Use Entity
- Keep Alive (serverbound)
- Player
- Player Position
- Player Position And Look (serverbound)
- Player Look
- Vehicle Move (serverbound)
- Steer Boat
- Craft Recipe Request
- Player Abilities (serverbound)
- Player Digging
- Entity Action
- Steer Vehicle
- Crafting Book Data

Definitions

The Minecraft server accepts connections from TCP clients and communicates with them using *packets*. A packet is a sequence of bytes sent over the TCP connection. The meaning of a packet depends both on its packet ID and the current state of the connection. The initial state of each connection is Handshaking, and state is switched using the packets Handshake and Login Success.

Data types

All data sent over the network (except for VarInt and VarLong) is big-endian, that is the bytes are sent from most significant byte to least significant byte. The majority of everyday computers are little-endian, therefore it may be necessary to change the endianness before sending data over the network.

Name	Size (bytes)	Encodes	Notes
Boolean	1	Either false or true	True is encoded as `0x01`, false as `0x00`.
Byte	1	An integer between -128 and 127	Signed 8-bit integer, two's complement
Unsigned Byte	1	An integer between 0 and 255	Unsigned 8-bit integer
Short	2	An integer between -32768 and 32767	Signed 16-bit integer, two's complement
Unsigned Short	2	An integer between 0 and 65535	Unsigned 16-bit integer
Int	4	An integer between -2147483648 and 2147483647	Signed 32-bit integer, two's complement
Long	8	An integer between -9223372036854775808 and 9223372036854775807	Signed 64-bit integer, two's complement
Float	4	A single-precision 32-bit IEEE 754 floating point number	
Double	8	A double-precision 64-bit IEEE 754 floating point number	
String (n)	≥ 1 ≤ (n×3) + 3	A sequence of Unicode scalar values (http://unicode.org/glossary/#unicode_scalar_value)	UTF-8 string prefixed with its size in bytes as a VarInt. Maximum length of `n` characters, which varies by context. The encoding used on the wire is regular UTF-8, *not* Java's "slight modification" (https://docs.oracle.com/en/java/javase/18/docs/api/java.base/java/io/DataInput.html#modified-utf-8). However, the length of the string for purposes of the length limit is its number of UTF-16 code units, that is, scalar values > U+FFFF are counted as two. Up to `n × 3` bytes can be used to encode a UTF-8 string comprising `n` code units when converted to UTF-16, and both of those limits are checked. Maximum `n` value is 32767. The + 3 is due to the max size of a valid length VarInt.
Text Component	Varies	See Text formatting#Text components	Encoded as a NBT Tag, with the type of tag used depending on the case: ▪ As a String Tag: For components only containing text (no styling, no events etc.). ▪ As a Compound Tag: Every other case.
JSON Text Component	≥ 1 ≤ (262144×3) + 3	See Text formatting#Text components	Encoded as a String with max length of 262144.
Identifier	≥ 1 ≤ (32767×3) + 3	See Identifier below	Encoded as a String with max length of 32767.
VarInt	≥ 1 ≤ 5	An integer between -2147483648 and 2147483647	Variable-length data encoding a two's complement signed 32-bit integer; more info in their section

VarLong	≥ 1 ≤ 10	An integer between -9223372036854775808 and 9223372036854775807	Variable-length data encoding a two's complement signed 64-bit integer; more info in their section
Entity Metadata	Varies	Miscellaneous information about an entity	See Entity_metadata#Entity Metadata Format
Slot	Varies	An item stack in an inventory or container	See Slot Data
NBT	Varies	Depends on context	See NBT
Position	8	An integer/block position: x (-33554432 to 33554431), z (-33554432 to 33554431), y (-2048 to 2047)	x as a 26-bit integer, followed by z as a 26-bit integer, followed by y as a 12-bit integer (all signed, two's complement). See also the section below.
Angle	1	A rotation angle in steps of 1/256 of a full turn	Whether or not this is signed does not matter, since the resulting angles are the same.
UUID	16	A UUID	Encoded as an unsigned 128-bit integer (or two unsigned 64-bit integers: the most significant 64 bits and then the least significant 64 bits)
BitSet	Varies	See #BitSet below	A length-prefixed bit set.
Fixed BitSet (n)	n	See #Fixed BitSet below	A bit set with a fixed length of *n* bits.
Optional X	0 or size of X	A field of type X, or nothing	Whether or not the field is present must be known from the context.
Array of X	count times size of X	Zero or more fields of type X	The count must be known from the context.
X Enum	size of X	A specific value from a given list	The list of possible values and how each is encoded as an X must be known from the context. An invalid value sent by either side will usually result in the client being disconnected with an error or even crashing.
Byte Array	Varies	Depends on context	This is just a sequence of zero or more bytes, its meaning should be explained somewhere else, e.g. in the packet description. The length must also be known from the context.

Identifier

Identifiers are a namespaced location, in the form of `minecraft:thing`. If the namespace is not provided, it defaults to `minecraft` (i.e. `thing` is `minecraft:thing`). Custom content should always be in its own namespace, not the default one. Both the namespace and value can use all lowercase alphanumeric characters (a-z and 0-9), dot (`.`), dash (`-`), and underscore (`_`). In addition, values can use slash (`/`). The naming convention is `lower_case_with_underscores`. More information (https://minecraft.net/en-us/article/minecraft-snapshot-17w43a). For ease of determining whether a namespace or value is valid, here are regular expressions for each:

- Namespace: `[a-z0-9.-_]`
- Value: `[a-z0-9.-_/]`

VarInt and VarLong

Variable-length format such that smaller numbers use fewer bytes. These are very similar to Protocol Buffer Varints (http://developers.google.com/protocol-buffers/docs/encoding#varints): the 7 least significant bits are used to encode the value and the most significant bit indicates whether there's another byte after it for the next part of the number. The least significant group is written first, followed by each of the more significant groups; thus, VarInts are effectively little endian (however, groups are 7 bits, not 8).

VarInts are never longer than 5 bytes, and VarLongs are never longer than 10 bytes. Within these limits, unnecessarily long encodings (e.g. `81 00` to encode 1) are allowed.

Pseudocode to read and write VarInts and VarLongs:

```
private static final int SEGMENT_BITS = 0x7F;
private static final int CONTINUE_BIT = 0x80;
```

```
public int readVarInt() {
    int value = 0;
    int position = 0;
    byte currentByte;

    while (true) {
        currentByte = readByte();
        value |= (currentByte & SEGMENT_BITS) << position;

        if ((currentByte & CONTINUE_BIT) == 0) break;

        position += 7;

        if (position >= 32) throw new RuntimeException("VarInt is too big");
    }

    return value;
}
```

```
public long readVarLong() {
    long value = 0;
    int position = 0;
    byte currentByte;

    while (true) {
        currentByte = readByte();
        value |= (long) (currentByte & SEGMENT_BITS) << position;

        if ((currentByte & CONTINUE_BIT) == 0) break;

        position += 7;

        if (position >= 64) throw new RuntimeException("VarLong is too big");
    }

    return value;
}
```

```
public void writeVarInt(int value) {
    while (true) {
        if ((value & ~SEGMENT_BITS) == 0) {
            writeByte(value);
            return;
        }

        writeByte((value & SEGMENT_BITS) | CONTINUE_BIT);

        // Note: >>> means that the sign bit is shifted with the rest of the number rather than being left alone
        value >>>= 7;
```

```
    }
}
```

```
public void writeVarLong(long value) {
    while (true) {
        if ((value & ~((long) SEGMENT_BITS)) == 0) {
            writeByte(value);
            return;
        }

        writeByte((value & SEGMENT_BITS) | CONTINUE_BIT);

        // Note: >>> means that the sign bit is shifted with the rest of the number rather than being left alone
        value >>>= 7;
    }
}
```

⚠️ Note Minecraft's VarInts are identical to LEB128 (https://en.wikipedia.org/wiki/LEB128) with the slight change of throwing a exception if it goes over a set amount of bytes.

⚠️ Note that Minecraft's VarInts are not encoded using Protocol Buffers; it's just similar. If you try to use Protocol Buffers Varints with Minecraft's VarInts, you'll get incorrect results in some cases. The major differences:

- Minecraft's VarInts are all signed, but do not use the ZigZag encoding. Protocol buffers have 3 types of Varints: `uint32` (normal encoding, unsigned), `sint32` (ZigZag encoding, signed), and `int32` (normal encoding, signed). Minecraft's are the `int32` variety. Because Minecraft uses the normal encoding instead of ZigZag encoding, negative values always use the maximum number of bytes.
- Minecraft's VarInts are never longer than 5 bytes and its VarLongs will never be longer than 10 bytes, while Protocol Buffer Varints will always use 10 bytes when encoding negative numbers, even if it's an `int32`.

Sample VarInts:

Value	Hex bytes	Decimal bytes
0	0x00	0
1	0x01	1
2	0x02	2
127	0x7f	127
128	0x80 0x01	128 1
255	0xff 0x01	255 1
25565	0xdd 0xc7 0x01	221 199 1
2097151	0xff 0xff 0x7f	255 255 127
2147483647	0xff 0xff 0xff 0xff 0x07	255 255 255 255 7
-1	0xff 0xff 0xff 0xff 0x0f	255 255 255 255 15
-2147483648	0x80 0x80 0x80 0x80 0x08	128 128 128 128 8

Sample VarLongs:

Value	Hex bytes	Decimal bytes
0	0x00	0
1	0x01	1
2	0x02	2
127	0x7f	127
128	0x80 0x01	128 1
255	0xff 0x01	255 1
2147483647	0xff 0xff 0xff 0xff 0x07	255 255 255 255 7
9223372036854775807	0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0x7f	255 255 255 255 255 255 255 255 127
-1	0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0x01	255 255 255 255 255 255 255 255 255 1
-2147483648	0x80 0x80 0x80 0x80 0xf8 0xff 0xff 0xff 0xff 0x01	128 128 128 128 248 255 255 255 255 1
-9223372036854775808	0x80 0x80 0x80 0x80 0x80 0x80 0x80 0x80 0x80 0x01	128 128 128 128 128 128 128 128 128 1

Position

Note: What you are seeing here is the latest version of the Data types article, but the position type was different before 1.14 (https://wiki.vg/index.php?title=Data_types&oldid=14345#Position).

64-bit value split into three **signed** integer parts:

- x: 26 MSBs
- z: 26 middle bits
- y: 12 LSBs

For example, a 64-bit position can be broken down as follows:

Example value (big endian): `01000110000001110110001100` `10110000010101101101001000` `001100111111`

- The red value is the X coordinate, which is `18357644` in this example.
- The blue value is the Z coordinate, which is `-20882616` in this example.
- The green value is the Y coordinate, which is `831` in this example.

Encoded as follows:

```
((x & 0x3FFFFFF) << 38) | ((z & 0x3FFFFFF) << 12) | (y & 0xFFF)
```

And decoded as:

```
val = read_long();
x = val >> 38;
y = val << 52 >> 52;
z = val << 26 >> 38;
```

Note: The above assumes that the right shift operator sign extends the value (this is called an arithmetic shift (https://en.wikipedia.org/wiki/Arithmetic_shift)), so that the signedness of the coordinates is preserved. In many languages, this requires the integer type of `val` to be signed. In the absence of such an operator, the following may be useful:

```
if x >= 1 << 25 { x -= 1 << 26 }
if y >= 1 << 11 { y -= 1 << 12 }
if z >= 1 << 25 { z -= 1 << 26 }
```

Fixed-point numbers

Some fields may be stored as fixed-point numbers (https://en.wikipedia.org/wiki/Fixed-point_arithmetic), where a certain number of bits represents the signed integer part (number to the left of the decimal point) and the rest represents the fractional part (to the right). Floating points (float and double), in contrast, keep the number itself (mantissa) in one chunk, while the location of the decimal point (exponent) is stored beside it.

Essentially, while fixed-point numbers have lower range than floating points, their fractional precision is greater for higher values. This makes them ideal for representing global coordinates of an entity in Minecraft, as it's more important to store the integer part accurately than position them more precisely within a single block (or meter).

Coordinates are often represented as a 32-bit integer, where 5 of the least-significant bits are dedicated to the fractional part, and the rest store the integer part.

Java lacks support for fractional integers directly, but you can represent them as integers. To convert from a double to this integer representation, use the following formulas:

```
abs_int = (int) (double * 32.0D);
```

And back again:

```
double = (double) (abs_int / 32.0D);
```

Bit sets

The types BitSet and Fixed BitSet represent packed lists of bits. The Notchian implementation uses Java's `BitSet` (https://docs.oracle.com/javase/8/docs/api/java/util/BitSet.html) class.

BitSet

Bit sets of type BitSet are prefixed by their length in longs.

Field Name	Field Type	Meaning
Length	VarInt	Number of longs in the following array. May be 0 (if no bits are set).
Data	Array of Long	A packed representation of the bit set as created by `BitSet.toLongArray` (https://docs.oracle.com/javase/8/docs/api/java/util/BitSet.html#toLongArray--).

The *i*th bit is set when `(Data[i / 64] & (1 << (i % 64))) != 0`, where *i* starts at 0.

Fixed BitSet

Bit sets of type Fixed BitSet (n) have a fixed length of *n* bits, encoded as `ceil(n / 8)` bytes. Note that this is different from BitSet, which uses longs.

Field Name	Field Type	Meaning
Data	Byte Array (n)	A packed representation of the bit set as created by `BitSet.toByteArray` (https://docs.oracle.com/javase/8/docs/api/java/util/BitSet.html#toByteArray--).

The *i*th bit is set when `(Data[i / 8] & (1 << (i % 8))) != 0`, where *i* starts at 0. This encoding is *not* equivalent to the long array in BitSet.

Other definitions

Term	Definition
Player	When used in the singular, Player always refers to the client connected to the server.
Entity	Entity refers to any item, player, mob, minecart or boat etc. See the Minecraft Wiki article (https://minecraft.wiki/w/Entity) for a full list.
EID	An EID — or Entity ID — is a 4-byte sequence used to identify a specific entity. An entity's EID is unique on the entire server.
XYZ	In this document, the axis names are the same as those shown in the debug screen (F3). Y points upwards, X points east, and Z points south.
Meter	The meter is Minecraft's base unit of length, equal to the length of a vertex of a solid block. The term “block” may be used to mean “meter” or “cubic meter”.
Global palette	A table/dictionary/palette mapping nonnegative integers to block states. The block state IDs can be constructed from this table (https://minecraft.wiki/w/Data_values) by multiplying what the Minecraft Wiki calls “block IDs” by 16 and adding the metadata/damage value (or in most programming languages `block_id << 4 | metadata`).
Notchian	The official implementation of vanilla Minecraft as developed and released by Mojang.

Packet format

Without compression

Field Name	Field Type	Notes
Length	VarInt	Length of packet data + length of the packet ID
Packet ID	VarInt	
Data	Byte Array	Depends on the connection state and packet ID, see the sections below

With compression

Once a Set Compression packet (with a non-negative threshold) is sent, zlib compression is enabled for all following packets. The format of a packet changes slighty to include the size of the uncompressed packet.

Compressed?	Field Name	Field Type	Notes
No	Packet Length	VarInt	Length of Data Length + compressed length of (Packet ID + Data)
No	Data Length	VarInt	Length of uncompressed (Packet ID + Data) or 0
Yes	Packet ID	VarInt	zlib compressed packet ID (see the sections below)
	Data	Byte Array	zlib compressed packet data (see the sections below)

The length given by the Packet Length field is the number of bytes that remain in that packet, including the Data Length field.

If Data Length is set to zero, then the packet is uncompressed; otherwise it is the size of the uncompressed packet.

If compressed, the uncompressed length of (Packet ID + Data) must be equal to or over the threshold set in the packet Set Compression, otherwise the receiving party will disconnect.

Compression can be disabled by sending the packet Set Compression with a negative Threshold, or not sending the Set Compression packet at all.

Handshaking

Clientbound

There are no clientbound packets in the Handshaking state, since the protocol immediately switches to a different state after the client sends the first packet.

Serverbound

Handshake

This causes the server to switch into the target state.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x00	Handshaking	Server	Protocol Version	VarInt	See protocol version numbers (currently 340 in Minecraft 1.12.2)
			Server Address	String (255)	Hostname or IP, e.g. localhost or 127.0.0.1, that was used to connect. The Notchian server does not use this information.
			Server Port	Unsigned Short	Default is 25565. The Notchian server does not use this information.
			Next State	VarInt Enum	1 for status, 2 for login

Legacy Server List Ping

⚠️ This packet uses a nonstandard format. It is never length-prefixed, and the packet ID is an Unsigned Byte instead of a VarInt.

While not technically part of the current protocol, legacy clients may send this packet to initiate Server List Ping, and modern servers should handle it correctly.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0xFE	Handshaking	Server	Payload	Unsigned Byte	always 1 (`0x01`)

See Server List Ping#1.6 for the details of the protocol that follows this packet.

Play

Clientbound

Spawn Object

Sent by the server when a vehicle or other object is created.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x00	Play	Client	Entity ID	VarInt	EID of the object
			Object UUID	UUID	
			Type	Byte	The type of object (see Entities#Objects)
			X	Double	
			Y	Double	
			Z	Double	
			Pitch	Angle	
			Yaw	Angle	
			Data	Int	Meaning dependent on the value of the Type field, see Object Data for details.
			Velocity X	Short	Same units as Entity Velocity. Always sent, but only used when Data is greater than 0 (except for some entities which always ignore it; see Object Data for details).
			Velocity Y	Short	
			Velocity Z	Short	

Spawn Experience Orb

Spawns one or more experience orbs.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x01	Play	Client	Entity ID	VarInt	
			X	Double	
			Y	Double	
			Z	Double	
			Count	Short	The amount of experience this orb will reward once collected

Spawn Global Entity

With this packet, the server notifies the client of thunderbolts striking within a 512 block radius around the player. The coordinates specify where exactly the thunderbolt strikes.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x02	Play	Client	Entity ID	VarInt	The EID of the thunderbolt
			Type	Byte Enum	The global entity type, currently always 1 for thunderbolt
			X	Double	
			Y	Double	
			Z	Double	

Spawn Mob

Sent by the server when a mob entity is spawned.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x03	Play	Client	Entity ID	VarInt	
			Entity UUID	UUID	
			Type	VarInt	The type of mob. See Entities#Mobs
			X	Double	
			Y	Double	
			Z	Double	
			Yaw	Angle	
			Pitch	Angle	
			Head Pitch	Angle	
			Velocity X	Short	Same units as Entity Velocity
			Velocity Y	Short	Same units as Entity Velocity
			Velocity Z	Short	Same units as Entity Velocity
			Metadata	Entity Metadata	

Spawn Painting

This packet shows location, name, and type of painting.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x04	Play	Client	Entity ID	VarInt	
			Entity UUID	UUID	
			Title	String (13)	Name of the painting. Max length 13
			Location	Position	Center coordinates (see below)
			Direction	Byte Enum	Direction the painting faces (North = 2, South = 0, West = 1, East = 3)

Calculating the center of an image: given a (width × height) grid of cells, with `(0, 0)` being the top left corner, the center is `(max(0, width / 2 - 1), height / 2)`. E.g. `(1, 0)` for a 2×1 painting, or `(1, 2)` for a 4×4 painting.

List of paintings by coordinates in `paintings_kristoffer_zetterstrand.png` (where x and y are in pixels from the top left and width and height are in pixels or 16ths of a block):

Name	x	y	width	height
Kebab	0	0	16	16
Aztec	16	0	16	16
Alban	32	0	16	16
Aztec2	48	0	16	16
Bomb	64	0	16	16
Plant	80	0	16	16
Wasteland	96	0	16	16
Pool	0	32	32	16
Courbet	32	32	32	16
Sea	64	32	32	16
Sunset	96	32	32	16
Creebet	128	32	32	16
Wanderer	0	64	16	32
Graham	16	64	16	32
Match	0	128	32	32
Bust	32	128	32	32
Stage	64	128	32	32
Void	96	128	32	32
SkullAndRoses	128	128	32	32
Wither	160	128	32	32
Fighters	0	96	64	32
Pointer	0	192	64	64
Pigscene	64	192	64	64
BurningSkull	128	192	64	64
Skeleton	192	64	64	48
DonkeyKong	192	112	64	48

The Minecraft Wiki article on paintings (https://minecraft.wiki/w/Painting%23Canvases) also provides a list of painting names to the actual images.

Spawn Player

This packet is sent by the server when a player comes into visible range, *not* when a player joins.

This packet must be sent after the Player List Item packet that adds the player data for the client to use when spawning a player. If the Player List Item for the player spawned by this packet is not present when this packet arrives, Notchian clients will not spawn the player entity. The Player List Item packet includes skin/cape data.

Servers can, however, safely spawn player entities for players not in visible range. The client appears to handle it correctly.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x05	Play	Client	Entity ID	VarInt	Player's EID
			Player UUID	UUID	See below for notes on offline mode (https://minecraft.wiki/w/Server.properties%23online-mode) and NPCs
			X	Double	
			Y	Double	
			Z	Double	
			Yaw	Angle	
			Pitch	Angle	
			Metadata	Entity Metadata	

When in online mode (https://minecraft.wiki/w/Server.properties%23online-mode), the UUIDs must be valid and have valid skin blobs.

In offline mode, UUID v3 is used with the String `OfflinePlayer:<player name>`, encoded in UTF-8 (and case-sensitive).

For NPCs UUID v2 should be used. Note:

```
<+Grum> i will never confirm this as a feature you know that :)
```

In an example UUID, `xxxxxxxx-xxxx-Yxxx-xxxx-xxxxxxxxxxxx`, the UUID version is specified by `Y`. So, for UUID v3, `Y` will always be `3`, and for UUID v2, `Y` will always be `2`.

Animation (clientbound)

Sent whenever an entity should change animation.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x06	Play	Client	Entity ID	VarInt	Player ID
			Animation	Unsigned Byte	Animation ID (see below)

Animation can be one of the following values:

ID	Animation
0	Swing main arm
1	Take damage
2	Leave bed
3	Swing offhand
4	Critical effect
5	Magic critical effect

Statistics

Sent as a response to Client Status 0x04 (id 1).

Packet ID	State	Bound To	Field Name		Field Type		Notes
0x07	Play	Client	Count		VarInt		Number of elements in the following array
			Statistic	Name	Array	String (32767)	https://gist.github.com/Alvin-LB/8d0d13db00b3c00fd0e822a562025eff (https://gist.github.com/Alvin-LB/8d0d13db00b3c00fd0e822a562025eff)
				Value		VarInt	The amount to set it to

Block Break Animation

0–9 are the displayable destroy stages and each other number means that there is no animation on this coordinate.

Block break animations can still be applied on air; the animation will remain visible although there is no block being broken. However, if this is applied to a transparent block, odd graphical effects may happen, including water losing its transparency. (An effect similar to this can be seen in normal gameplay when breaking ice blocks)

If you need to display several break animations at the same time you have to give each of them a unique Entity ID. The entity ID does not need to correspond to an actual entity on the client. It is valid to use a randomly generated number.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x08	Play	Client	Entity ID	VarInt	Entity ID of the entity breaking the block
			Location	Position	Block Position
			Destroy Stage	Byte	0–9 to set it, any other value to remove it

Update Block Entity

Sets tile entity associated with the block at the given location.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x09	Play	Client	Location	Position	
			Action	Unsigned Byte	The type of update to perform, see below
			NBT Data	NBT Tag	Data to set. May be a TAG_END (0), in which case the block entity at the given location is removed (though this is not required since the client will remove the block entity automatically on chunk unload or block removal)

Action field:

- **1**: Set data of a mob spawner (everything except for SpawnPotentials: current delay, min/max delay, mob to be spawned, spawn count, spawn range, etc.)
- **2**: Set command block text (command and last execution status)
- **3**: Set the level, primary, and secondary powers of a beacon
- **4**: Set rotation and skin of mob head
- **5**: Set type of flower in flower pot
- **6**: Set base color and patterns on a banner
- **7**: Set the data for a Structure tile entity
- **8**: Set the destination for a end gateway
- **9**: Set the text on a sign
- **10**: Declare a shulker box, no data appears to be sent and the client seems to do fine without this packet. Perhaps it is a leftover from earlier versions?
- **11**: Set the color of a bed

Block Action

This packet is used for a number of actions and animations performed by blocks, usually non-persistent.

See Block Actions for a list of values.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x0A	Play	Client	Location	Position	Block coordinates
			Action ID (Byte 1)	Unsigned Byte	Varies depending on block — see Block Actions
			Action Param (Byte 2)	Unsigned Byte	Varies depending on block — see Block Actions
			Block Type	VarInt	The block type ID for the block, not including metadata/damage value. This must match the block at the given coordinates.

Block Change

Fired whenever a block is changed within the render distance.

⚠️ Changing a block in a chunk that is not loaded is not a stable action. The Notchian client currently uses a *shared* empty chunk which is modified for all block changes in unloaded chunks; while in 1.9 this chunk never renders in older versions the changed block will appear in all copies of the empty chunk. Servers should avoid sending block changes in

unloaded chunks and clients should ignore such packets.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x0B	Play	Client	Location	Position	Block Coordinates
			Block ID	VarInt	The new block state ID for the block as given in the global palette (https://minecraft.wiki/w/Data_values%23Block_IDs) (When reading data: `type = id >> 4`, `meta = id & 15`, when writing data: `id = type << 4 | (meta & 15)`)

Boss Bar

Packet ID	State	Bound To	Field Name		Field Type	Notes
0x0C	Play	Client	UUID		UUID	Unique ID for this bar
			Action		VarInt Enum	Determines the layout of the remaining packet
			Action	**Field Name**		
			0: add	Title	Chat	
				Health	Float	From 0 to 1. Values greater than 1 do not crash a Notchian client, and start rendering part of a second health bar (https://i.johni0702.de/nA.png) at around 1.5.
				Color	VarInt Enum	Color ID (see below)
				Division	VarInt Enum	Type of division (see below)
				Flags	Unsigned Byte	Bit mask. 0x1: should darken sky, 0x2: is dragon bar (used to play end music)
			1: remove	*no fields*	*no fields*	Removes this boss bar
			2: update health	Health	Float	as above
			3: update title	Title	Chat	
			4: update style	Color	VarInt Enum	Color ID (see below)
				Dividers	VarInt Enum	as above
			5: update flags	Flags	Unsigned Byte	as above

ID	Color
0	Pink
1	Blue
2	Red
3	Green
4	Yellow
5	Purple
6	White

ID	Type of division
0	No division
1	6 notches
2	10 notches
3	12 notches
4	20 notches

Server Difficulty

Changes the difficulty setting in the client's option menu

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x0D	Play	Client	Difficulty	Unsigned Byte	0: peaceful, 1: easy, 2: normal, 3: hard

Tab-Complete (clientbound)

The server responds with a list of auto-completions of the last word sent to it. In the case of regular chat, this is a player username. Command names and parameters are also supported. The client sorts these alphabetically before listing them.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x0E	Play	Client	Count	VarInt	Number of elements in the following array
			Matches	Array of String (32767)	One eligible command, note that each command is sent separately instead of in a single string, hence the need for Count

Chat Message (clientbound)

Identifying the difference between Chat/System Message is important as it helps respect the user's chat visibility options. See processing chat for more info about these positions.

⚠️ Game info accepts json formatting but does not display it, although the deprecated §-based formatting works. This is not an issue when using the Title packet, so prefer that packet for displaying information in that slot. See MC-119145 (https://bugs.mojang.com/browse/MC-119145) for more information.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x0F	Play	Client	JSON Data	Chat	Limited to 32767 bytes
			Position	Byte	0: chat (chat box), 1: system message (chat box), 2: game info (above hotbar).

Multi Block Change

Fired whenever 2 or more blocks are changed within the same chunk on the same tick.

⚠️ Changing blocks in chunks not loaded by the client is unsafe (see note on Block Change).

Packet ID	State	Bound To	Field Name		Field Type		Notes
0x10	Play	Client	Chunk X		Int		Chunk X coordinate
			Chunk Z		Int		Chunk Z coordinate
			Record Count		VarInt		Number of elements in the following array, i.e. the number of blocks affected
			Record	Horizontal Position	Array	Unsigned Byte	The 4 most significant bits (`0xF0`) encode the X coordinate, relative to the chunk. The 4 least significant bits (`0x0F`) encode the Z coordinate, relative to the chunk.
				Y Coordinate		Unsigned Byte	Y coordinate of the block
				Block ID		VarInt	The new block state ID for the block as given in the global palette (https://minecraft.wiki/w/Data_values%23Block_IDs) (When reading data: `type = id >> 4`, `meta = id & 15`, when writing data: `id = type << 4 | (meta & 15)`)

To decode the position into a world position:

```
worldX = (horizPos >> 4 & 15) + (chunkX * 16);
worldY = vertPos;
worldZ = (horizPos & 15) + (chunkZ * 16);
```

Confirm Transaction (clientbound)

A packet from the server indicating whether a request from the client was accepted, or whether there was a conflict (due to lag). If the packet was not accepted, the client must respond with a serverbound confirm transaction packet.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x11	Play	Client	Window ID	Byte	The ID of the window that the action occurred in
			Action Number	Short	Every action that is to be accepted has a unique number. This number is an incrementing integer (starting at 0) with separate counts for each window ID.
			Accepted	Boolean	Whether the action was accepted

Close Window (clientbound)

This packet is sent from the server to the client when a window is forcibly closed, such as when a chest is destroyed while it's open.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x12	Play	Client	Window ID	Unsigned Byte	This is the ID of the window that was closed. 0 for inventory.

Open Window

This is sent to the client when it should open an inventory, such as a chest, workbench, or furnace. This message is not sent anywhere for clients opening their own inventory.

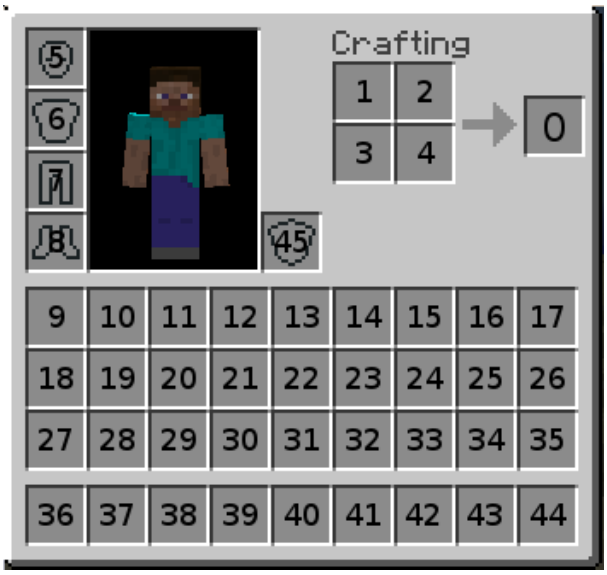
Packet ID	State	Bound To	Field Name	Field Type	Notes
0x13	Play	Client	Window ID	Unsigned Byte	A unique id number for the window to be displayed. Notchian server implementation is a counter, starting at 1.
			Window Type	String (32)	The window type to use for display. See Inventory for a list.
			Window Title	Chat	The title of the window
			Number Of Slots	Unsigned Byte	Number of slots in the window (excluding the number of slots in the player inventory). Always 0 for non-storage windows (e.g. Workbench, Anvil).
			Entity ID	Optional Int	EntityHorse's EID. Only sent when Window Type is “EntityHorse”

See Inventory for further information.

Window Items

Sent by the server when items in multiple slots (in a window) are added/removed. This includes the main inventory, equipped armour and crafting slots.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x14	Play	Client	Window ID	Unsigned Byte	The ID of window which items are being sent for. 0 for player inventory.
			Count	Short	Number of elements in the following array
			Slot Data	Array of Slot	



The inventory slots

See inventory windows for further information about how slots are indexed.

Window Property

This packet is used to inform the client that part of a GUI window should be updated.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x15	Play	Client	Window ID	Unsigned Byte	
			Property	Short	The property to be updated, see below
			Value	Short	The new value for the property, see below

The meaning of the Property field depends on the type of the window. The following table shows the known combinations of window type and property, and how the value is to be interpreted.

Window type	Property	Value
Furnace	0: Fire icon (fuel left)	counting from fuel burn time down to 0 (in-game ticks)
	1: Maximum fuel burn time	fuel burn time or 0 (in-game ticks)
	2: Progress arrow	counting from 0 to maximum progress (in-game ticks)
	3: Maximum progress	always 200 on the notchian server
Enchantment Table	0: Level requirement for top enchantment slot	The enchantment's xp level requirement
	1: Level requirement for middle enchantment slot	
	2: Level requirement for bottom enchantment slot	
	3: The enchantment seed	Used for drawing the enchantment names (in SGA) clientside. The same seed *is* used to calculate enchantments, but some of the data isn't sent to the client to prevent easily guessing the entire list (the seed value here is the regular seed bitwise and `0xFFFFFFF0`).
	4: Enchantment ID shown on mouse hover over top enchantment slot	The enchantment id (set to -1 to hide it)
	5: Enchantment ID shown on mouse hover over middle enchantment slot	
	6: Enchantment ID shown on mouse hover over bottom enchantment slot	
	7: Enchantment level shown on mouse hover over the top slot	The enchantment level (1 = I, 2 = II, 6 = VI, etc.), or -1 if no enchant
	8: Enchantment level shown on mouse hover over the middle slot	
	9: Enchantment level shown on mouse hover over the bottom slot	
Beacon	0: Power level	0-4, controls what effect buttons are enabled
	1: First potion effect	Potion effect ID (https://minecraft.wiki/w/Data_values%23Status_effects) for the first effect, or -1 if no effect
	2: Second potion effect	Potion effect ID (https://minecraft.wiki/w/Data_values%23Status_effects) for the second effect, or -1 if no effect
Anvil	0: Repair cost	The repair's cost in xp levels
Brewing Stand	0: Brew time	0 – 400, with 400 making the arrow empty, and 0 making the arrow full
	1: Fuel time	0 - 20, with 0 making the arrow empty, and 20 making the arrow full

Set Slot

Sent by the server when an item in a slot (in a window) is added/removed.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x16	Play	Client	Window ID	Byte	The window which is being updated. 0 for player inventory. Note that all known window types include the player inventory. This packet will only be sent for the currently opened window while the player is performing actions, even if it affects the player inventory. After the window is closed, a number of these packets are sent to update the player's inventory window (0).
			Slot	Short	The slot that should be updated
			Slot Data	Slot	

To set the cursor (the item currently dragged with the mouse), use -1 as Window ID and as Slot.

This packet can only be used to edit the hotbar of the player's inventory if window ID is set to 0 (slots 36 through 44). If the window ID is set to -2, then any slot in the inventory can be used but no add item animation will be played.

Set Cooldown

Applies a cooldown period to all items with the given type. Used by the Notchian server with enderpearls. This packet should be sent when the cooldown starts and also when the cooldown ends (to compensate for lag), although the client will end the cooldown automatically. Can be applied to any item, note that interactions still get sent to the server with the item but the client does not play the animation nor attempt to predict results (i.e block placing).

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x17	Play	Client	Item ID	VarInt	Numeric ID of the item to apply a cooldown to.
			Cooldown Ticks	VarInt	Number of ticks to apply a cooldown for, or 0 to clear the cooldown.

Plugin Message (clientbound)

> *Main article: Plugin channels*

Mods and plugins can use this to send their data. Minecraft itself uses a number of plugin channels. These internal channels are prefixed with `MC|`.

More documentation on this: http://dinnerbone.com/blog/2012/01/13/minecraft-plugin-channels-messaging/ (http://dinnerbone.com/blog/2012/01/13/minecraft-plugin-channels-messaging/)

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x18	Play	Client	Channel	String (20)	Name of the plugin channel used to send the data
			Data	Byte Array	Any data, depending on the channel. `MC|` channels are documented here. The length of this array must be inferred from the packet length.

Named Sound Effect

> *See also: #Sound Effect*

Used to play a sound effect on the client. Custom sounds may be added by resource packs.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x19	Play	Client	Sound Name	String (256)	All sound effect names as of 1.12.2 can be seen here (http://pokechu22.github.io/Burger/1.12.2.html#sounds).
			Sound Category	VarInt Enum	The category that this sound will be played from (current categories (https://gist.github.com/konwboj/7c0c380d3923443e9d55))
			Effect Position X	Int	Effect X multiplied by 8 (fixed-point number with only 3 bits dedicated to the fractional part)
			Effect Position Y	Int	Effect Y multiplied by 8 (fixed-point number with only 3 bits dedicated to the fractional part)
			Effect Position Z	Int	Effect Z multiplied by 8 (fixed-point number with only 3 bits dedicated to the fractional part)
			Volume	Float	1 is 100%, can be more
			Pitch	Float	Float between 0.5 and 2.0 by Notchian clients

Disconnect (play)

Sent by the server before it disconnects a client. The client assumes that the server has already closed the connection by the time the packet arrives.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x1A	Play	Client	Reason	Chat	Displayed to the client when the connection terminates.

Entity Status

Entity statuses generally trigger an animation for an entity. The available statuses vary by the entity's type (and are available to subclasses of that type as well).

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x1B	Play	Client	Entity ID	Int	
			Entity Status	Byte Enum	See below

See Entity statuses for a list of which statuses are valid for each type of entity.

Explosion

Sent when an explosion occurs (creepers, TNT, and ghast fireballs).

Each block in Records is set to air. Coordinates for each axis in record is int(X) + record.x

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x1C	Play	Client	X	Float	
			Y	Float	
			Z	Float	
			Radius	Float	Currently unused in the client
			Record Count	Int	Number of elements in the following array
			Records	Array of (Byte, Byte, Byte)	Each record is 3 signed bytes long, each bytes are the XYZ (respectively) offsets of affected blocks.
			Player Motion X	Float	X velocity of the player being pushed by the explosion
			Player Motion Y	Float	Y velocity of the player being pushed by the explosion
			Player Motion Z	Float	Z velocity of the player being pushed by the explosion

Unload Chunk

Tells the client to unload a chunk column.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x1D	Play	Client	Chunk X	Int	Block coordinate divided by 16, rounded down
			Chunk Z	Int	Block coordinate divided by 16, rounded down

It is legal to send this packet even if the given chunk is not currently loaded.

Change Game State

Used for a wide variety of game state things, from whether to bed use to gamemode to demo messages.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x1E	Play	Client	Reason	Unsigned Byte	See below
			Value	Float	Depends on Reason

Reason codes:

Reason	Effect	Value
0	Invalid Bed	Would be used to switch between messages, but the only used message is 0 for invalid bed
1	End raining	
2	Begin raining	
3	Change gamemode	0: Survival, 1: Creative, 2: Adventure, 3: Spectator
4	Exit end	0: Immediately send Client Status of respawn without showing end credits; 1: Show end credits and respawn at the end (or when esc is pressed). 1 is sent if the player has not yet received the "The end?" advancement, while if they do have it 0 is used.
5	Demo message	0: Show welcome to demo screen, 101: Tell movement controls, 102: Tell jump control, 103: Tell inventory control
6	Arrow hitting player	Appears to be played when an arrow strikes another player in Multiplayer
7	Fade value	The current darkness value. 1 = Dark, 0 = Bright, Setting the value higher causes the game to change color and freeze
8	Fade time	Time in ticks for the sky to fade
10	Play elder guardian mob appearance (effect and sound)	

Keep Alive (clientbound)

The server will frequently send out a keep-alive, each containing a random ID. The client must respond with the same packet. If the client does not respond to them for over 30 seconds, the server kicks the client. Vice versa, if the server does not send any keep-alives for 20 seconds, the client will disconnect and yields a "Timed out" exception.

The Notchian server uses a system-dependent time in milliseconds to generate the keep alive ID value.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x1F	Play	Client	Keep Alive ID	Long	

Chunk Data

> *Main article: Chunk Format*
> *See also: #Unload Chunk*

The server only sends skylight information for chunk pillars in the Overworld (https://minecraft.wiki/w/Overworld), it's up to the client to know in which dimenison the player is currently located. You can also infer this information from the primary bitmask and the amount of uncompressed bytes sent. This packet also sends all block entities in the chunk (though sending them is not required; it is still legal to send them with Update Block Entity later).

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x20	Play	Client	Chunk X	Int	Chunk coordinate (block coordinate divided by 16, rounded down)
			Chunk Z	Int	Chunk coordinate (block coordinate divided by 16, rounded down)
			Ground-Up Continuous	Boolean	See Chunk Format
			Primary Bit Mask	VarInt	Bitmask with bits set to 1 for every 16×16×16 chunk section whose data is included in Data. The least significant bit represents the chunk section at the bottom of the chunk column (from y=0 to y=15).
			Size	VarInt	Size of Data in bytes
			Data	Byte array	See data structure in Chunk Format
			Number of block entities	VarInt	Number of elements in the following array
			Block entities	Array of NBT Tag	All block entities in the chunk. Use the x, y, and z tags in the NBT to determine their positions.

Effect

Sent when a client is to play a sound or particle effect.

By default, the Minecraft client adjusts the volume of sound effects based on distance. The final boolean field is used to disable this, and instead the effect is played from 2 blocks away in the correct direction. Currently this is only used for effect 1023 (wither spawn) and effect 1028 (enderdragon death); it is ignored on other effects.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x21	Play	Client	Effect ID	Int	The ID of the effect, see below
			Location	Position	The location of the effect
			Data	Int	Extra data for certain effects, see below
			Disable Relative Volume	Boolean	See above

Effect IDs:

ID	Name	Data
Sound		
1000	Dispenser dispenses	
1001	Dispenser fails to dispense	
1002	Dispenser shoots	
1003	Ender eye launched	
1004	Firework shot	
1005	Iron door opened	
1006	Wooden door opened	
1007	Wooden trapdoor opened	
1008	Fence gate opened	
1009	Fire extinguished	
1010	Play record	Special case, see below for more info
1011	Iron door closed	
1012	Wooden door closed	
1013	Wooden trapdoor closed	
1014	Fence gate closed	
1015	Ghast warns	
1016	Ghast shoots	
1017	Enderdragon shoots	
1018	Blaze shoots	
1019	Zombie attacks wood door	
1020	Zombie attacks iron door	
1021	Zombie breaks wood door	
1022	Wither breaks block	
1023	Wither spawned	
1024	Wither shoots	
1025	Bat takes off	
1026	Zombie infects	
1027	Zombie villager converted	
1028	Ender dragon death	
1029	Anvil destroyed	
1030	Anvil used	
1031	Anvil landed	
1032	Portal travel	
1033	Chorus flower grown	

1034	Chorus flower died	
1035	Brewing stand brewed	
1036	Iron trapdoor opened	
1037	Iron trapdoor closed	
Particle		
2000	Spawns 10 smoke particles, e.g. from a fire	Direction, see below
2001	Block break + block break sound	block id (this differs from normal global palette use)
2002	Splash potion. Particle effect + glass break sound.	Potion ID (http://minecraft.gamepedia.com/Data_values#Potions)
2003	Eye of Ender entity break animation — particles and sound	
2004	Mob spawn particle effect: smoke + flames	
2005	Bonemeal particles	How many particles to spawn (if set to 0, 15 are spawned)
2006	Dragon breath	
2007	Instant splash potion	Potion ID (http://minecraft.gamepedia.com/Data_values#Potions)
3000	End gateway spawn	
3001	Enderdragon growl	

Smoke directions:

ID	Direction
0	South-East
1	South
2	South-West
3	East
4	(Up or middle ?)
5	West
6	North-East
7	North
8	North-West

Play record: This is actually a special case within this packet. You can start/stop a record at a specific location. Use a valid Record ID (https://minecraft.wiki/w/Music_Discs) to start a record (or overwrite a currently playing one), any other value will stop the record.

Particle

Displays the named particle

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x22	Play	Client	Particle ID	Int	See below
			Long Distance	Boolean	If true, particle distance increases from 256 to 65536
			X	Float	X position of the particle
			Y	Float	Y position of the particle
			Z	Float	Z position of the particle
			Offset X	Float	This is added to the X position after being multiplied by random.nextGaussian()
			Offset Y	Float	This is added to the Y position after being multiplied by random.nextGaussian()
			Offset Z	Float	This is added to the Z position after being multiplied by random.nextGaussian()
			Particle Data	Float	The data of each particle
			Particle Count	Int	The number of particles to create
			Data	Array of VarInt	Length depends on particle. "iconcrack" has length of 2, "blockcrack", "blockdust", and "fallingdust" have lengths of 1, the rest have 0.

Particle IDs:

Particle Name	Particle ID
explode	0
largeexplode	1
hugeexplosion	2
fireworksSpark	3
bubble	4
splash	5
wake	6
suspended	7
depthsuspend	8
crit	9
magicCrit	10
smoke	11
largesmoke	12
spell	13
instantSpell	14
mobSpell	15
mobSpellAmbient	16
witchMagic	17
dripWater	18
dripLava	19
angryVillager	20
happyVillager	21
townaura	22
note	23
portal	24
enchantmenttable	25
flame	26
lava	27
footstep	28
cloud	29
reddust	30
snowballpoof	31
snowshovel	32
slime	33
heart	34

barrier	35
iconcrack_(id)_(data)	36
blockcrack_(id+(data<<12))	37
blockdust_(id)	38
droplet	39
take	40
mobappearance	41
dragonbreath	42
endrod	43
damageindicator	44
sweepattack	45
fallingdust	46
totem	47
spit	48

Join Game

See Protocol Encryption for information on logging in.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x23	Play	Client	Entity ID	Int	The player's Entity ID (EID)
			Gamemode	Unsigned Byte	0: Survival, 1: Creative, 2: Adventure, 3: Spectator. Bit 3 (0x8) is the hardcore flag.
			Dimension	Int Enum	-1: Nether, 0: Overworld, 1: End; also, note that this is not a VarInt but instead a regular int.
			Difficulty	Unsigned Byte	0: peaceful, 1: easy, 2: normal, 3: hard
			Max Players	Unsigned Byte	Was once used by the client to draw the player list, but now is ignored
			Level Type	String Enum (16)	default, flat, largeBiomes, amplified, default_1_1
			Reduced Debug Info	Boolean	If true, a Notchian client shows reduced information on the debug screen (https://minecraft.wiki/w/Debug_screen). For servers in development, this should almost always be false.

Map

Updates a rectangular area on a map (https://minecraft.wiki/w/Map) item.

Packet ID	State	Bound To	Field Name		Field Type		Notes
0x24	Play	Client	Item Damage		VarInt		The damage value (map ID) of the map being modified
			Scale		Byte		From 0 for a fully zoomed-in map (1 block per pixel) to 4 for a fully zoomed-out map (16 blocks per pixel)
			Tracking Position		Boolean		Specifies whether the icons are shown
			Icon Count		VarInt		Number of elements in the following array
			Icon	Direction And Type	Array	Byte	0xF0 = Type, 0x0F = Direction
				X		Byte	
				Z		Byte	
			Columns		Byte		Number of columns updated
			Rows		Optional Byte		Only if Columns is more than 0; number of rows updated
			X		Optional Byte		Only if Columns is more than 0; x offset of the westernmost column
			Z		Optional Byte		Only if Columns is more than 0; z offset of the northernmost row
			Length		Optional VarInt		Only if Columns is more than 0; length of the following array
			Data		Optional Array of Unsigned Byte		Only if Columns is more than 0; see Map item format (https://minecraft.wiki/w/Map_item_format)

For icons, a direction of 0 is a vertical icon and increments by 22.5° (360/16).

Types are based off of rows and columns in `map_icons.png`:

Icon type	Result
0	White arrow (players)
1	Green arrow (item frames)
2	Red arrow
3	Blue arrow
4	White cross
5	Red pointer
6	White circle (off-map players)
7	Small white circle (far-off-map players)
8	Mansion
9	Temple
10-15	Unused (blue square)

Entity

This packet may be used to initialize an entity.

For player entities, either this packet or any move/look packet is sent every game tick. So the meaning of this packet is basically that the entity did not move/look since the last such packet.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x25	Play	Client	Entity ID	VarInt	

Entity Relative Move

This packet is sent by the server when an entity moves less then 8 blocks; if an entity moves more than 8 blocks Entity Teleport should be sent instead.

This packet allows at most 8 blocks movement in any direction, because short range is from -32768 to 32767. And `32768 / (128 * 32)` = 8.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x26	Play	Client	Entity ID	VarInt	
			Delta X	Short	Change in X position as `(currentX * 32 - prevX * 32) * 128`
			Delta Y	Short	Change in Y position as `(currentY * 32 - prevY * 32) * 128`
			Delta Z	Short	Change in Z position as `(currentZ * 32 - prevZ * 32) * 128`
			On Ground	Boolean	

Entity Look And Relative Move

This packet is sent by the server when an entity rotates and moves. Since a short range is limited from -32768 to 32767, and movement is offset of fixed-point numbers, this packet allows at most 8 blocks movement in any direction. (`-32768 / (32 * 128) == -8`)

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x27	Play	Client	Entity ID	VarInt	
			Delta X	Short	Change in X position as `(currentX * 32 - prevX * 32) * 128`
			Delta Y	Short	Change in Y position as `(currentY * 32 - prevY * 32) * 128`
			Delta Z	Short	Change in Z position as `(currentZ * 32 - prevZ * 32) * 128`
			Yaw	Angle	New angle, not a delta
			Pitch	Angle	New angle, not a delta
			On Ground	Boolean	

Entity Look

This packet is sent by the server when an entity rotates.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x28	Play	Client	Entity ID	VarInt	
			Yaw	Angle	New angle, not a delta
			Pitch	Angle	New angle, not a delta
			On Ground	Boolean	

Vehicle Move (clientbound)

Note that all fields use absolute positioning and do not allow for relative positioning.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x29	Play	Client	X	Double	Absolute position (X coordinate)
			Y	Double	Absolute position (Y coordinate)
			Z	Double	Absolute position (Z coordinate)
			Yaw	Float	Absolute rotation on the vertical axis, in degrees
			Pitch	Float	Absolute rotation on the horizontal axis, in degrees

Open Sign Editor

Sent when the client has placed a sign and is allowed to send Update Sign. There must already be a sign at the given location (which the client does not do automatically) - send a Block Change first.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x2A	Play	Client	Location	Position	

Craft Recipe Response

Response to the serverbound packet (Craft Recipe Request), with the same recipe ID. Appears to be used to notify the UI.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x2B	Play	Client	Window ID	Byte	
			Recipe	VarInt	A recipe ID

Player Abilities (clientbound)

The latter 2 floats are used to indicate the field of view and flying speed respectively, while the first byte is used to determine the value of 4 booleans.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x2C	Play	Client	Flags	Byte	Bit field, see below
			Flying Speed	Float	
			Field of View Modifier	Float	Modifies the field of view, like a speed potion. A Notchian server will use the same value as the movement speed (send in the Entity Properties packet).

About the flags:

Field	Bit
Invulnerable	0x01
Flying	0x02
Allow Flying	0x04
Creative Mode (Instant Break)	0x08

Combat Event

Packet ID	State	Bound To	Field Name		Field Type	Notes
0x2D	Play	Client	Event		VarInt Enum	Determines the layout of the remaining packet
			Event	**Field Name**		
			0: enter combat	*no fields*	*no fields*	
			1: end combat	Duration	VarInt	
				Entity ID	Int	
			2: entity dead	Player ID	VarInt	
				Entity ID	Int	
				Message	Chat	

Player List Item

Sent by the server to update the user list (<tab> in the client).

Packet ID	State	Bound To	Field Name			Field Type			Notes
0x2E	Play	Client	Action			VarInt			Determines the rest of the Player format after the UUID
			Number Of Players			VarInt			Number of elements in the following array
			Player	UUID		Array	UUID		
				Action	**Field Name**				
				0: add player	Name		String (16)		
					Number Of Properties		VarInt		Number of elements in the following array
					Property	Name	Array	String (32767)	
						Value		String (32767)	
						Is Signed		Boolean	
						Signature		Optional String (32767)	Only if Is Signed is true
					Gamemode			VarInt	
					Ping			VarInt	Measured in milliseconds
					Has Display Name			Boolean	
					Display Name			Optional Chat	Only if Has Display Name is true
				1: update gamemode	Gamemode			VarInt	
				2: update latency	Ping			VarInt	Measured in milliseconds
				3: update display name	Has Display Name			Boolean	
					Display Name			Optional Chat	Only send if Has Display Name is true
				4: remove player	*no fields*			*no fields*	

The Property field looks as in the response of Mojang API#UUID -> Profile + Skin/Cape, except of course using the protocol format instead of JSON. That is, each player will usually have one property with Name “textures” and Value being a base64-encoded JSON string as documented at Mojang API#UUID -> Profile + Skin/Cape. An empty properties array is also acceptable, and will cause clients to display the player with one of the two default skins depending on UUID.

Ping values correspond with icons in the following way:

- A ping that negative (i.e. not known to the server yet) will result in the no connection icon.
- A ping under 150 milliseconds will result in 5 bars
- A ping under 300 milliseconds will result in 4 bars
- A ping under 600 milliseconds will result in 3 bars
- A ping under 1000 milliseconds (1 second) will result in 2 bars
- A ping greater than or equal to 1 second will result in 1 bar.

Player Position And Look (clientbound)

Updates the player's position on the server. This packet will also close the “Downloading Terrain” screen when joining/respawning.

If the distance between the last known position of the player on the server and the new position set by this packet is greater than 100 meters, the client will be kicked for “You moved too quickly :((Hacking?)”.

Also if the fixed-point number of X or Z is set greater than `3.2E7D` the client will be kicked for “Illegal position”.

Yaw is measured in degrees, and does not follow classical trigonometry rules. The unit circle of yaw on the XZ-plane starts at (0, 1) and turns counterclockwise, with 90 at (-1, 0), 180 at (0, -1) and 270 at (1, 0). Additionally, yaw is not clamped to between 0 and 360 degrees; any number is valid, including negative numbers and numbers greater than 360.

Pitch is measured in degrees, where 0 is looking straight ahead, -90 is looking straight up, and 90 is looking straight down.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x2F	Play	Client	X	Double	Absolute or relative position, depending on Flags
			Y	Double	Absolute or relative position, depending on Flags
			Z	Double	Absolute or relative position, depending on Flags
			Yaw	Float	Absolute or relative rotation on the X axis, in degrees
			Pitch	Float	Absolute or relative rotation on the Y axis, in degrees
			Flags	Byte	Bit field, see below
			Teleport ID	VarInt	Client should confirm this packet with Teleport Confirm containing the same Teleport ID

About the Flags field:

```
<Dinnerbone> It's a bitfield, X/Y/Z/Y_ROT/X_ROT. If X is set, the x value is relative and not absolute.
```

Field	Bit
X	0x01
Y	0x02
Z	0x04
Y_ROT	0x08
X_ROT	0x10

Use Bed

This packet tells that a player goes to bed.

The client with the matching Entity ID will go into bed mode.

This Packet is sent to all nearby players including the one sent to bed.

Any packets sent with a location not currently occupied by a bed will be ignored by clients.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x30	Play	Client	Entity ID	VarInt	Sleeping player's EID
			Location	Position	Block location of the head part of the bed

Unlock Recipes

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x31	Play	Client	Action	VarInt	0: init, 1: add, 2: remove
			Crafting Book Open	Boolean	If true, then the crafting book will be open when the player opens its inventory.
			Filtering Craftable	Boolean	If true, then the filtering option is active when the players opens its inventory.
			Array size 1	VarInt	Number of elements in the following array
			Recipe IDs	Array of VarInt	
			Array size 2	Optional VarInt	Number of elements in the following array, only present if mode is 0 (init)
			Recipe IDs	Optional Array of VarInt, only present if mode is 0 (init)	

Action:

- 0 (init) = All the recipes in the list 2 will added to the recipe book. All the recipes in list 1 will be tagged as displayed, recipes that aren't tagged will be shown in the notification. VERIFY LIST ORDER?
- 1 (add) = All the recipes in the list are added and their icon will be shown in the notification.
- 2 (remove) = Remove all the recipes in the list. This allows them to re-displayed when they are readded.

Recipe ID: These are hardcoded values in the client and server, all the recipe json files will be loaded in a specific order (alphabetical, like sounds) and internal ids will be assigned in that order. There are also inbuilt recipes like fireworks, banners, etc., these are the first recipes to have their id assigned. Due the fact that the recipes are loaded in a specific order will the ids very likely change when recipes get added. Custom recipes are scheduled for Minecraft 1.13 (https://twitter.com/dinnerbone/status/856505341479145472), so most likely will things change a bit in that version.

Destroy Entities

Sent by the server when a list of entities is to be destroyed on the client.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x32	Play	Client	Count	VarInt	Number of elements in the following array
			Entity IDs	Array of VarInt	The list of entities of destroy

Remove Entity Effect

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x33	Play	Client	Entity ID	VarInt	
			Effect ID	Byte	See this table (https://minecraft.wiki/w/Status_effect%23List_of_effects)

Resource Pack Send

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x34	Play	Client	URL	String (32767)	The URL to the resource pack.
			Hash	String (40)	A 40 character hexadecimal and lowercase SHA-1 hash of the resource pack file. (must be lower case in order to work) If it's not a 40 character hexadecimal string, the client will not use it for hash verification and likely waste bandwidth — but it will still treat it as a unique id

Respawn

To change the player's dimension (overworld/nether/end), send them a respawn packet with the appropriate dimension, followed by prechunks/chunks for the new dimension, and finally a position and look packet. You do not need to unload chunks, the client will do it automatically.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x35	Play	Client	Dimension	Int Enum	-1: The Nether, 0: The Overworld, 1: The End
			Difficulty	Unsigned Byte	0: Peaceful, 1: Easy, 2: Normal, 3: Hard
			Gamemode	Unsigned Byte	0: survival, 1: creative, 2: adventure, 3: spectator. The hardcore flag is not included
			Level Type	String (16)	Same as Join Game

⚠️ Avoid changing player's dimension to same dimension they were already in unless they are dead. If you change the dimension to one they are already in, weird bugs can occur, such as the player being unable to attack other players in new world (until they die and respawn).

If you must respawn a player in the same dimension without killing them, send two respawn packets, one to a different world and then another to the world you want. You do not need to complete the first respawn; it only matters that you send two packets.

Entity Head Look

Changes the direction an entity's head is facing.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x36	Play	Client	Entity ID	VarInt	
			Head Yaw	Angle	New angle, not a delta

Select Advancement Tab

Sent by the server to indicate that the client should switch advancement tab. Sent either when the client switches tab in the GUI or when an advancement in another tab is made.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x37	Play	Client	Has id	Boolean	Indicates if the next field is present
			Optional Identifier	String (32767)	See below

The Identifier can be one of the following:

Optional Identifier
minecraft:story/root
minecraft:nether/root
minecraft:end/root
minecraft:adventure/root
minecraft:husbandry/root

If no or an invalid identifier is sent, the client will switch to the first tab in the GUI.

World Border

Packet ID	State	Bound To	Field Name		Field Type	Notes
0x38	Play	Client	Action		VarInt Enum	Determines the format of the rest of the packet
			Action	**Field Name**		
			0: set size	Diameter	Double	Length of a single side of the world border, in meters
			1: lerp size	Old Diameter	Double	Current length of a single side of the world border, in meters
				New Diameter	Double	Target length of a single side of the world border, in meters
				Speed	VarLong	Number of real-time *milli*seconds until New Diameter is reached. It appears that Notchian server does not sync world border speed to game ticks, so it gets out of sync with server lag. If the world border is not moving, this is set to 0.
			2: set center	X	Double	
				Z	Double	
			3: initialize	X	Double	
				Z	Double	
				Old Diameter	Double	Current length of a single side of the world border, in meters
				New Diameter	Double	Target length of a single side of the world border, in meters
				Speed	VarLong	Number of real-time *milli*seconds until New Diameter is reached. It appears that Notchian server does not sync world border speed to game ticks, so it gets out of sync with server lag. If the world border is not moving, this is set to 0.
				Portal Teleport Boundary	VarInt	Resulting coordinates from a portal teleport are limited to ±value. Usually 29999984.
				Warning Time	VarInt	In seconds as set by `/worldborder warning time`
				Warning Blocks	VarInt	In meters
			4: set warning time	Warning Time	VarInt	In seconds as set by `/worldborder warning time`
			5: set warning blocks	Warning Blocks	VarInt	In meters

The Notchian client determines how solid to display the warning by comparing to whichever is higher, the warning distance or whichever is lower, the distance from the current diameter to the target diameter or the place the border will be after warningTime seconds. In pseudocode:

```
distance = max(min(resizeSpeed * 1000 * warningTime, abs(targetDiameter - currentDiameter)), warningDistance);
if (playerDistance < distance) {
    warning = 1.0 - playerDistance / distance;
} else {
    warning = 0.0;
}
```

Camera

Sets the entity that the player renders from. This is normally used when the player left-clicks an entity while in spectator mode.

The player's camera will move with the entity and look where it is looking. The entity is often another player, but can be any type of entity. The player is unable to move this entity (move packets will act as if they are coming from the other entity).

If the given entity is not loaded by the player, this packet is ignored. To return control to the player, send this packet with their entity ID.

The Notchian server resets this (sends it back to the default entity) whenever the spectated entity is killed or the player sneaks, but only if they were spectating an entity. It also sends this packet whenever the player switches out of spectator mode (even if they weren't spectating an entity).

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x39	Play	Client	Camera ID	VarInt	ID of the entity to set the client's camera to

The notchian also loads certain shaders for given entities:

- Creeper → `shaders/post/creeper.json`
- Spider (and cave spider) → `shaders/post/spider.json`
- Enderman → `shaders/post/invert.json`
- Anything else → the current shader is unloaded

Held Item Change (clientbound)

Sent to change the player's slot selection.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x3A	Play	Client	Slot	Byte	The slot which the player has selected (0–8)

Display Scoreboard

This is sent to the client when it should display a scoreboard.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x3B	Play	Client	Position	Byte	The position of the scoreboard. 0: list, 1: sidebar, 2: below name, 3 - 18: team specific sidebar, indexed as 3 + team color.
			Score Name	String (16)	The unique name for the scoreboard to be displayed.

Entity Metadata

Updates one or more metadata properties for an existing entity. Any properties not included in the Metadata field are left unchanged.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x3C	Play	Client	Entity ID	VarInt	
			Metadata	Entity Metadata	

Attach Entity

This packet is sent when an entity has been leashed (https://minecraft.wiki/w/Lead) to another entity.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x3D	Play	Client	Attached Entity ID	Int	Attached entity's EID
			Holding Entity ID	Int	ID of the entity holding the lead. Set to -1 to detach.

Entity Velocity

Velocity is believed to be in units of 1/8000 of a block per server tick (50ms); for example, -1343 would move (-1343 / 8000) = −0.167875 blocks per tick (or −3,3575 blocks per second).

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x3E	Play	Client	Entity ID	VarInt	
			Velocity X	Short	Velocity on the X axis
			Velocity Y	Short	Velocity on the Y axis
			Velocity Z	Short	Velocity on the Z axis

Entity Equipment

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x3F	Play	Client	Entity ID	VarInt	Entity's EID
			Slot	VarInt Enum	Equipment slot. 0: main hand, 1: off hand, 2–5: armor slot (2: boots, 3: leggings, 4: chestplate, 5: helmet)
			Item	Slot	

Set Experience

Sent by the server when the client should change experience levels.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x40	Play	Client	Experience bar	Float	Between 0 and 1
			Level	VarInt	
			Total Experience	VarInt	See Experience#Leveling up (https://minecraft.wiki/w/Experience%23Leveling_up) on the Minecraft Wiki for Total Experience to Level conversion

Update Health

Sent by the server to update/set the health of the player it is sent to.

Food saturation (https://minecraft.wiki/w/Food%23Hunger_vs._Saturation) acts as a food "overcharge". Food values will not decrease while the saturation is over zero. Players logging in automatically get a saturation of 5.0. Eating food increases the saturation as well as the food bar.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x41	Play	Client	Health	Float	0 or less = dead, 20 = full HP
			Food	VarInt	0–20
			Food Saturation	Float	Seems to vary from 0.0 to 5.0 in integer increments

Scoreboard Objective

This is sent to the client when it should create a new scoreboard (https://minecraft.wiki/w/Scoreboard) objective or remove one.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x42	Play	Client	Objective Name	String (16)	An unique name for the objective
			Mode	Byte	0 to create the scoreboard. 1 to remove the scoreboard. 2 to update the display text.
			Objective Value	Optional String (32)	Only if mode is 0 or 2. The text to be displayed for the score
			Type	Optional String (16)	Only if mode is 0 or 2. “integer” or “hearts”

Set Passengers

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x43	Play	Client	Entity ID	VarInt	Vehicle's EID
			Passenger Count	VarInt	Number of elements in the following array
			Passengers	Array of VarInt	EIDs of entity's passengers

Teams

Creates and updates teams.

Packet ID	State	Bound To	Field Name		Field Type	Notes
0x44	Play	Client	Team Name		String (16)	A unique name for the team. (Shared with scoreboard).
			Mode		Byte	Determines the layout of the remaining packet
			0: create team	Team Display Name	String (32)	
				Team Prefix	String (16)	Displayed before the names of players that are part of this team
				Team Suffix	String (16)	Displayed after the names of players that are part of this team
				Friendly Flags	Byte	Bit mask. 0x01: Allow friendly fire, 0x02: can see invisible players on same team
				Name Tag Visibility	String Enum (32)	`always`, `hideForOtherTeams`, `hideForOwnTeam`, `never`
				Collision Rule	String Enum (32)	`always`, `pushOtherTeams`, `pushOwnTeam`, `never`
				Color	Byte	For colors, the same Chat colors (0-15). -1 indicates RESET/no color.
				Entity Count	VarInt	Number of elements in the following array
				Entities	Array of String (40)	Identifiers for the entities in this team. For players, this is their username; for other entities, it is their UUID.
			1: remove team	*no fields*	*no fields*	
			2: update team info	Team Display Name	String (32)	
				Team Prefix	String (16)	Displayed before the names of entities that are part of this team
				Team Suffix	String (16)	Displayed after the names of entities that are part of this team
				Friendly Flags	Byte	Bit mask. 0x01: Allow friendly fire, 0x02: can see invisible entities on same team
				Name Tag Visibility	String Enum (32)	`always`, `hideForOtherTeams`, `hideForOwnTeam`, `never`
				Collision Rule	String Enum (32)	`always`, `pushOtherTeams`, `pushOwnTeam`, `never`
				Color	Byte	For colors, the same Chat colors (0-15). -1 indicates RESET/no color.
			3: add players to	Entity Count	VarInt	Number of elements in the following array

			team	Entities	Array of String (40)	Identifiers for the entities added. For players, this is their username; for other entities, it is their UUID.
			4: remove players from team	Entity Count	VarInt	Number of elements in the following array
				Entities	Array of String (40)	Identifiers for the entities removed. For players, this is their username; for other entities, it is their UUID.

Update Score

This is sent to the client when it should update a scoreboard item.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x45	Play	Client	Entity Name	String (40)	The entity whose score this is. For players, this is their username; for other entities, it is their UUID.
			Action	Byte	0 to create/update an item. 1 to remove an item.
			Objective Name	String (16)	The name of the objective the score belongs to
			Value	Optional VarInt	The score to be displayed next to the entry. Only sent when Action does not equal 1.

Spawn Position

Sent by the server after login to specify the coordinates of the spawn point (the point at which players spawn at, and which the compass points to). It can be sent at any time to update the point compasses point at.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x46	Play	Client	Location	Position	Spawn location

Time Update

Time is based on ticks, where 20 ticks happen every second. There are 24000 ticks in a day, making Minecraft days exactly 20 minutes long.

The time of day is based on the timestamp modulo 24000. 0 is sunrise, 6000 is noon, 12000 is sunset, and 18000 is midnight.

The default SMP server increments the time by `20` every second.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x47	Play	Client	World Age	Long	In ticks; not changed by server commands
			Time of day	Long	The world (or region) time, in ticks. If negative the sun will stop moving at the Math.abs of the time

Title

Packet ID	State	Bound To	Field Name		Field Type	Notes
0x48	Play	Client	Action		VarInt Enum	
			Action	**Field Name**		
			0: set title	Title Text	Chat	
			1: set subtitle	Subtitle Text	Chat	
			2: set action bar	Action bar text	Chat	Displays a message above the hotbar (the same as position 2 in Chat Message (clientbound), except that it correctly renders formatted chat. See MC-119145 (https://bugs.mojang.com/browse/MC-119145) for more information.)
			3: set times and display	Fade In	Int	Ticks to spend fading in
				Stay	Int	Ticks to keep the title displayed
				Fade Out	Int	Ticks to spend out, not when to start fading out
			4: hide	*no fields*	*no fields*	
			5: reset	*no fields*	*no fields*	

"Hide" makes the title disappear, but if you run times again the same title will appear. "Reset" erases the text.

The title is visible on screen for Fade In + Stay + Fade Out ticks.

Sound Effect

This packet is used to play a number of hardcoded sound events. For custom sounds, use Named Sound Effect.

⚠️ Numeric sound effect IDs are liable to change between versions

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x49	Play	Client	Sound ID	VarInt	ID of hardcoded sound event (events (http://pokechu22.github.io/Burger/1.12.2.html#sounds) as of 1.12.2)
			Sound Category	VarInt Enum	The category that this sound will be played from (current categories (https://gist.github.com/konwboj/7c0c380d3923443e9d55))
			Effect Position X	Int	Effect X multiplied by 8 (fixed-point number with only 3 bits dedicated to the fractional part)
			Effect Position Y	Int	Effect Y multiplied by 8 (fixed-point number with only 3 bits dedicated to the fractional part)
			Effect Position Z	Int	Effect Z multiplied by 8 (fixed-point number with only 3 bits dedicated to the fractional part)
			Volume	Float	1.0 is 100%, capped between 0.0 and 1.0 by Notchian clients
			Pitch	Float	Float between 0.5 and 2.0 by Notchian clients

Player List Header And Footer

This packet may be used by custom servers to display additional information above/below the player list. It is never sent by the Notchian server.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x4A	Play	Client	Header	Chat	To remove the header, send a empty translatable component: {"translate":""}
			Footer	Chat	To remove the footer, send a empty translatable component: {"translate":""}

Collect Item

Sent by the server when someone picks up an item lying on the ground — its sole purpose appears to be the animation of the item flying towards you. It doesn't destroy the entity in the client memory, and it doesn't add it to your inventory. The server only checks for items to be picked up after each Player Position (and Player Position And Look) packet sent by the client. The collector entity can be any entity, it does not have to be a player.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x4B	Play	Client	Collected Entity ID	VarInt	
			Collector Entity ID	VarInt	
			Pickup Item Count	VarInt	Seems to be 1 for XP orbs, otherwise the number of items in the stack.

Entity Teleport

This packet is sent by the server when an entity moves more than 8 blocks.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x4C	Play	Client	Entity ID	VarInt	
			X	Double	
			Y	Double	
			Z	Double	
			Yaw	Angle	New angle, not a delta
			Pitch	Angle	New angle, not a delta
			On Ground	Boolean	

Advancements

Packet ID	State	Bound To	Field Name		Field Type		Notes
0x4D	Play	Client	Reset/Clear		Boolean		Whether to reset/clear the current advancements
			Mapping size		VarInt		Size of the following array
			Advancement mapping	Key	Array	Identifier	The identifier of the advancement
				Value		Advancement	See below
			List size		VarInt		Size of the following array
			Identifiers		Array of Identifier		The identifiers of the advancements that should be removed
			Progress size		VarInt		Size of the following array
			Progress mapping	Key	Array	Identifier	The identifier of the advancement
				Value		Advancement progress	See below

Advancement structure:

Field Name		Field Type		Notes
Has parent		Boolean		Indicates whether the next field exists.
Parent id		Optional Identifier		The identifier of the parent advancement.
Has display		Boolean		Indicates whether the next field exists
Display data		Optional advancement display		See below.
Number of criteria		VarInt		Size of the following array
Criteria	Key	Array	Identifier	The identifier of the criterion
	Value		**Void**	There is *no* content written here. Perhaps this will be expanded in the future?
Array length		VarInt		Number of arrays in the following array
Requirements	Array length 2	Array	VarInt	Number of elements in the following array
	Requirement		Array of String	Array of required criteria

Advancement display:

Field Name	Field Type	Notes
Title	Chat	
Description	Chat	
Icon	Slot	
Frame type	VarInt enum	0 = `task`, 1 = `challenge`, 2 = `goal`
Flags	Integer	0x1: has background texture; 0x2: `show_toast`; 0x4: `hidden`
Background texture	Optional Identifier	Background texture location. Only if flags indicates it.
X coord	Float	
Y coord	Float	

Advancement progress:

Field Name		Field Type		Notes
Size		VarInt		Size of the following array
Criteria	Criterion identifier	Array	Identifier	The identifier of the criterion.
	Criterion progress		Criterion progress	

Criterion progress:

Field Name	Field Type	Notes
Achieved	Boolean	If true, next field is present
Date of achieving	Optional Long	As returned by `Date.getTime` (https://docs.oracle.com/javase/6/docs/api/java/util/Date.html#getTime())

Entity Properties

Sets attributes (https://minecraft.wiki/w/Attribute) on the given entity.

Packet ID	State	Bound To	Field Name		Field Type		Notes
0x4E	Play	Client	Entity ID		VarInt		
			Number Of Properties		Int		Number of elements in the following array
			Property	Key	Array	String (64)	See below
				Value		Double	See below
				Number Of Modifiers		VarInt	Number of elements in the following array
				Modifiers		Array of Modifier Data	See Attribute#Modifiers (https://minecraft.wiki/w/Attribute%23Modifiers). Modifier Data defined below.

Known Key values (see also Attribute#Modifiers (https://minecraft.wiki/w/Attribute%23Modifiers)):

Key	Default	Min	Max	Label
generic.maxHealth	20.0	0.0	1024.0	Max Health
generic.followRange	32.0	0.0	2048.0	Follow Range
generic.knockbackResistance	0.0	0.0	1.0	Knockback Resistance
generic.movementSpeed	0.699999988079071	0.0	1024.0	Movement Speed
generic.attackDamage	2.0	0.0	2048.0	Attack Damage
generic.attackSpeed	4.0	0.0	1024.0	Attack Speed
generic.flyingSpeed	0.4000000059604645	0.0	1024.0	Flying Speed
horse.jumpStrength	0.7	0.0	2.0	Jump Strength
zombie.spawnReinforcements	0.0	0.0	1.0	Spawn Reinforcements Chance
generic.reachDistance	5.0	0.0	1024.0	Player Reach Distance (Forge only)
forge.swimSpeed	1.0	0.0	1024.0	Swimming Speed (Forge only)

Modifier Data structure:

Field Name	Field Type	Notes
UUID	UUID	
Amount	Double	May be positive or negative
Operation	Byte	See below

The operation controls how the base value of the modifier is changed.

- 0: Add/subtract amount
- 1: Add/subtract amount percent of the current value
- 2: Multiply by amount percent

All of the 0's are applied first, and then the 1's, and then the 2's.

Entity Effect

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x4F	Play	Client	Entity ID	VarInt	
			Effect ID	Byte	See this table (https://minecraft.wiki/w/Status_effect%23List_of_effects)
			Amplifier	Byte	Notchian client displays effect level as Amplifier + 1
			Duration	VarInt	Seconds
			Flags	Byte	Bit field, see below.

Within flags:

- 0x01: Is ambient - was the effect spawned from a beacon? All beacon-generated effects are ambient. Ambient effects use a different icon in the HUD (blue border rather than gray). If all effects on an entity are ambient, the "Is potion effect ambient" living metadata field should be set to true. Usually should not be enabled.
- 0x02: Show particles - should all particles from this effect be hidden? Effects with particles hidden are not included in the calculation of the effect color, and are not rendered on the HUD (but are still rendered within the inventory). Usually should be enabled.

Serverbound

Teleport Confirm

Sent by client as confirmation of Player Position And Look.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x00	Play	Server	Teleport ID	VarInt	The ID given by the Player Position And Look packet

Tab-Complete (serverbound)

Sent when the user presses *tab* while writing text.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x01	Play	Server	Text	String (32767)	All text behind the cursor (e.g. to the left of the cursor in left-to-right languages like English)
			Assume Command	Boolean	If true, the server will parse Text as a command even if it doesn't start with a `/`. Used in the command block GUI.
			Has Position	Boolean	
			Looked At Block	Optional Position	The position of the block being looked at. Only sent if Has Position is true.

Chat Message (serverbound)

Used to send a chat message to the server. The message may not be longer than 256 characters or else the server will kick the client.

If the message starts with a `/`, the server will attempt to interpret it as a command. Otherwise, the server will broadcast the same chat message to all players on the server (including the player that sent the message), prepended with player's name. Specifically, it will respond with a translate chat component, "`chat.type.text`" with the first parameter set to the display name of the player (including some chat component logic to support clicking the name to send a PM) and the second parameter set to the message. See processing chat for more information.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x02	Play	Server	Message	String (256)	The client sends the raw input, not a Chat component

Client Status

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x03	Play	Server	Action ID	VarInt Enum	See below

Action ID values:

Action ID	Action	Notes
0	Perform respawn	Sent when the client is ready to complete login and when the client is ready to respawn after death.
1	Request stats	Sent when the client opens the Statistics menu

Client Settings

Sent when the player connects, or when settings are changed.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x04	Play	Server	Locale	String (16)	e.g. en_GB
			View Distance	Byte	Client-side render distance, in chunks
			Chat Mode	VarInt Enum	0: enabled, 1: commands only, 2: hidden. See processing chat for more information.
			Chat Colors	Boolean	“Colors” multiplayer setting
			Displayed Skin Parts	Unsigned Byte	Bit mask, see below
			Main Hand	VarInt Enum	0: Left, 1: Right

Displayed Skin Parts flags:

- Bit 0 (0x01): Cape enabled
- Bit 1 (0x02): Jacket enabled
- Bit 2 (0x04): Left Sleeve enabled
- Bit 3 (0x08): Right Sleeve enabled
- Bit 4 (0x10): Left Pants Leg enabled
- Bit 5 (0x20): Right Pants Leg enabled
- Bit 6 (0x40): Hat enabled

The most significant bit (bit 7, 0x80) appears to be unused.

Confirm Transaction (serverbound)

If a transaction sent by the client was not accepted, the server will reply with a Confirm Transaction (clientbound) packet with the Accepted field set to false. When this happens, the client must send this packet to apologize (as with movement), otherwise the server ignores any successive transactions.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x05	Play	Server	Window ID	Byte	The ID of the window that the action occurred in
			Action Number	Short	Every action that is to be accepted has a unique number. This number is an incrementing integer (starting at 1) with separate counts for each window ID.
			Accepted	Boolean	Whether the action was accepted

Enchant Item

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x06	Play	Server	Window ID	Byte	The ID of the enchantment table window sent by Open Window
			Enchantment	Byte	The position of the enchantment on the enchantment table window, starting with 0 as the topmost one

Click Window

This packet is sent by the player when it clicks on a slot in a window.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x07	Play	Server	Window ID	Unsigned Byte	The ID of the window which was clicked. 0 for player inventory.
			Slot	Short	The clicked slot number, see below
			Button	Byte	The button used in the click, see below
			Action Number	Short	A unique number for the action, implemented by Notchian as a counter, starting at 1 (different counter for every window ID). Used by the server to send back a Confirm Transaction (clientbound).
			Mode	VarInt Enum	Inventory operation mode, see below
			Clicked item	Slot	The clicked slot. Has to be empty (item ID = -1) for drop mode.

See Inventory for further information about how slots are indexed.

When right-clicking on a stack of items, half the stack will be picked up and half left in the slot. If the stack is an odd number, the half left in the slot will be smaller of the amounts.

The distinct type of click performed by the client is determined by the combination of the Mode and Button fields.

Mode	Button	Slot	Trigger
0	0	Normal	Left mouse click
	1	Normal	Right mouse click
1	0	Normal	Shift + left mouse click
	1	Normal	Shift + right mouse click *(identical behavior)*
2	0	Normal	Number key 1
	1	Normal	Number key 2
	2	Normal	Number key 3
	⋮	⋮	⋮
	8	Normal	Number key 9
3	2	Normal	Middle click, only defined for creative players in non-player inventories.
4	0	Normal*	Drop key (Q) (* Clicked item is different, see above)
	1	Normal*	Ctrl + Drop key (Ctrl-Q) *(drops full stack)*
	0	-999	Left click outside inventory holding nothing *(no-op)*
	1	-999	Right click outside inventory holding nothing *(no-op)*
5	0	-999	Starting left mouse drag
	4	-999	Starting right mouse drag
	8	-999	Starting middle mouse drag, only defined for creative players in non-player inventories. (Note: the vanilla client will still incorrectly send this for non-creative players - see MC-46584 (https://bugs.mojang.com/browse/MC-46584))
	1	Normal	Add slot for left-mouse drag
	5	Normal	Add slot for right-mouse drag
	9	Normal	Add slot for middle-mouse drag, only defined for creative players in non-player inventories. (Note: the vanilla client will still incorrectly send this for non-creative players - see MC-46584 (https://bugs.mojang.com/browse/MC-46584))
	2	-999	Ending left mouse drag
	6	-999	Ending right mouse drag
	10	-999	Ending middle mouse drag, only defined for creative players in non-player inventories. (Note: the vanilla client will still incorrectly send this for non-creative players - see MC-46584 (https://bugs.mojang.com/browse/MC-46584))
6	0	Normal	Double click

Starting from version 1.5, “painting mode” is available for use in inventory windows. It is done by picking up stack of something (more than 1 item), then holding mouse button (left, right or middle) and dragging held stack over empty (or same type in case of right button) slots. In that case client sends the following to server after mouse button release (omitting first pickup packet which is sent as usual):

1. packet with mode 5, slot -999, button (0 for left | 4 for right);
2. packet for every slot painted on, mode is still 5, button (1 | 5);
3. packet with mode 5, slot -999, button (2 | 6);

If any of the painting packets other than the “progress” ones are sent out of order (for example, a start, some slots, then another start; or a left-click in the middle) the painting status will be reset.

The server will send back a Confirm Transaction packet. If the click was not accepted, the client must send a matching serverbound confirm transaction packet before sending more Click Window packets, otherwise the server will reject them silently. The Notchian server also sends a Window Items packet for the open window and Set Slot packets for the clicked and cursor slot, but only when the click was not accepted, probably to resynchronize client and server.

Close Window (serverbound)

This packet is sent by the client when closing a window.

Notchian clients send a Close Window packet with Window ID 0 to close their inventory even though there is never an Open Window packet for the inventory.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x08	Play	Server	Window ID	Unsigned Byte	This is the ID of the window that was closed. 0 for player inventory.

Plugin Message (serverbound)

> *Main article: Plugin channels*

Mods and plugins can use this to send their data. Minecraft itself uses a number of plugin channels. These internal channels are prefixed with `MC|`.

More documentation on this: http://dinnerbone.com/blog/2012/01/13/minecraft-plugin-channels-messaging/ (http://dinnerbone.com/blog/2012/01/13/minecraft-plugin-channels-messaging/)

Note that the length of Data is known only from the packet length, since the packet has no length field of any kind.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x09	Play	Server	Channel	String (20)	Name of the plugin channel used to send the data
			Data	Byte Array	Any data, depending on the channel. `MC|` channels are documented here. The length of this array must be inferred from the packet length.

Use Entity

This packet is sent from the client to the server when the client attacks or right-clicks another entity (a player, minecart, etc).

A Notchian server only accepts this packet if the entity being attacked/used is visible without obstruction and within a 4-unit radius of the player's position.

Note that middle-click in creative mode is interpreted by the client and sent as a Creative Inventory Action packet instead.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x0A	Play	Server	Target	VarInt	
			Type	VarInt Enum	0: interact, 1: attack, 2: interact at
			Target X	Optional Float	Only if Type is interact at
			Target Y	Optional Float	Only if Type is interact at
			Target Z	Optional Float	Only if Type is interact at
			Hand	Optional VarInt Enum	Only if Type is interact or interact at; 0: main hand, 1: off hand

Keep Alive (serverbound)

The server will frequently send out a keep-alive, each containing a random ID. The client must respond with the same packet.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x0B	Play	Server	Keep Alive ID	Long	

Player

This packet as well as Player Position, Player Look, and Player Position And Look are called the “serverbound movement packets”. Vanilla clients will send Player Position once every 20 ticks even for a stationary player.

This packet is used to indicate whether the player is on ground (walking/swimming), or airborne (jumping/falling).

When dropping from sufficient height, fall damage is applied when this state goes from false to true. The amount of damage applied is based on the point where it last changed from true to false. Note that there are several movement related packets containing this state.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x0C	Play	Server	On Ground	Boolean	True if the client is on the ground, false otherwise

Player Position

Updates the player's XYZ position on the server.

Checking for moving too fast is achieved like this:

- Each server tick, the player's current position is stored
- When a player moves, the changes in x, y, and z coordinates are compared with the positions from the previous tick (Δx, Δy, Δz)
- Total movement distance squared is computed as $\Delta x^2 + \Delta y^2 + \Delta z^2$
- The expected movement distance squared is computed as $velocityX^2 + veloctyY^2 + velocityZ^2$
- If the total movement distance squared value minus the expected movement distance squared value is more than 100 (300 if the player is using an elytra), they are moving too fast.

If the player is moving too fast, it will be logged that "<player> moved too quickly! " followed by the change in x, y, and z, and the player will be teleported back to their current (before this packet) serverside position.

Also, if the absolute value of X or the absolute value of Z is a value greater than 3.2×10^7, or X, Y, or Z are not finite (either positive infinity, negative infinity, or NaN), the client will be kicked for "Invalid move player packet received".

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x0D	Play	Server	X	Double	Absolute position
			Feet Y	Double	Absolute feet position, normally Head Y - 1.62
			Z	Double	Absolute position
			On Ground	Boolean	True if the client is on the ground, false otherwise

Player Position And Look (serverbound)

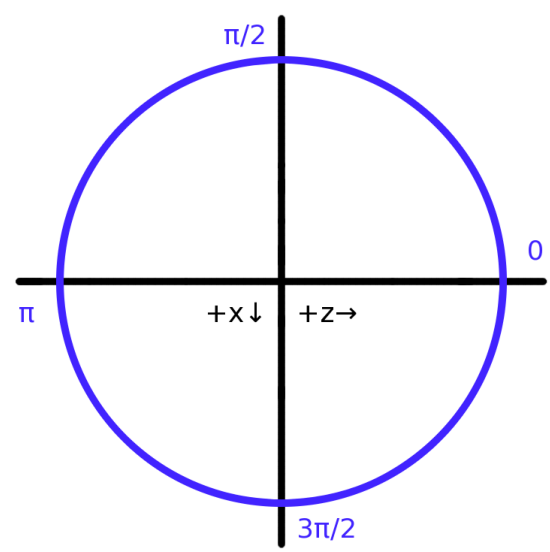
A combination of Player Look and Player Position.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x0E	Play	Server	X	Double	Absolute position
			Feet Y	Double	Absolute feet position, normally Head Y - 1.62
			Z	Double	Absolute position
			Yaw	Float	Absolute rotation on the X Axis, in degrees
			Pitch	Float	Absolute rotation on the Y Axis, in degrees
			On Ground	Boolean	True if the client is on the ground, false otherwise

Player Look

Updates the direction the player is looking in.

Yaw is measured in degrees, and does not follow classical trigonometry rules. The unit circle of yaw on the XZ-plane starts at (0, 1) and turns counterclockwise, with 90 at (-1, 0), 180 at (0,-1) and 270 at (1, 0). Additionally, yaw is not clamped to between 0 and 360 degrees; any number is valid, including negative numbers and numbers greater than 360.



The unit circle for yaw

Pitch is measured in degrees, where 0 is looking straight ahead, -90 is looking straight up, and 90 is looking straight down.

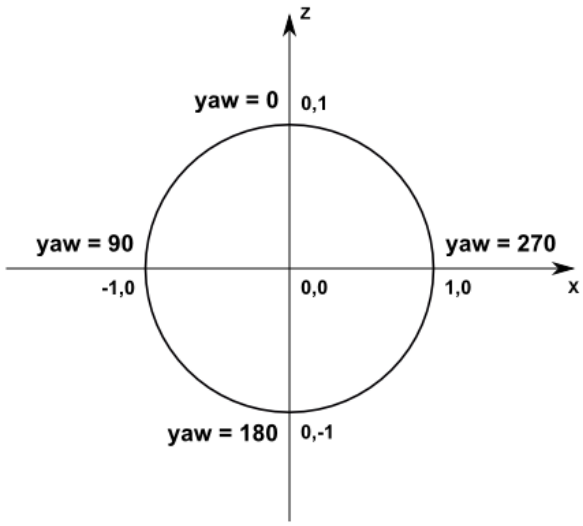
The yaw and pitch of player (in degrees), standing at point (x0, y0, z0) and looking towards point (x, y, z) can be calculated with:

```
dx = x-x0
dy = y-y0
dz = z-z0
r = sqrt( dx*dx + dy*dy + dz*dz )
yaw = -atan2(dx,dz)/PI*180
if yaw < 0 then
    yaw = 360 + yaw
pitch = -arcsin(dy/r)/PI*180
```

You can get a unit vector from a given yaw/pitch via:

```
x = -cos(pitch) * sin(yaw)
y = -sin(pitch)
z =  cos(pitch) * cos(yaw)
```

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x0F	Play	Server	Yaw	Float	Absolute rotation on the X Axis, in degrees
			Pitch	Float	Absolute rotation on the Y Axis, in degrees
			On Ground	Boolean	True if the client is on the ground, False otherwise



The unit circle of yaw, redrawn

Vehicle Move (serverbound)

Sent when a player moves in a vehicle. Fields are the same as in Player Position And Look. Note that all fields use absolute positioning and do not allow for relative positioning.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x10	Play	Server	X	Double	Absolute position (X coordinate)
			Y	Double	Absolute position (Y coordinate)
			Z	Double	Absolute position (Z coordinate)
			Yaw	Float	Absolute rotation on the vertical axis, in degrees
			Pitch	Float	Absolute rotation on the horizontal axis, in degrees

Steer Boat

Used to *visually* update whether boat paddles are turning. The server will update the Boat entity metadata to match the values here.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x11	Play	Server	Right paddle turning	Boolean	
			Left paddle turning	Boolean	

Right paddle turning is set to true when the left button or forward button is held; left paddle turning is set to true when the right button or forward button is set to true.

Craft Recipe Request

A replacement for Prepare Crafting Grid. It appears to behave more or less the same, but the client does not specify where to move the items.

This packet is sent when a player clicks a recipe in the crafting book that is craftable (white border).

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x12	Play	Server	Window ID	Byte	
			Recipe	VarInt	A recipe ID
			Make all	Boolean	Affects the amount of items processed; true if shift is down when clicked

Player Abilities (serverbound)

The latter 2 bytes are used to indicate the walking and flying speeds respectively, while the first byte is used to determine the value of 4 booleans.

The vanilla client sends this packet when the player starts/stops flying with the Flags parameter changed accordingly. All other parameters are ignored by the vanilla server.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x13	Play	Server	Flags	Byte	Bit mask. 0x08: damage disabled (god mode), 0x04: can fly, 0x02: is flying, 0x01: is Creative
			Flying Speed	Float	
			Walking Speed	Float	

Player Digging

Sent when the player mines a block. A Notchian server only accepts digging packets with coordinates within a 6-unit radius between the center of the block and 1.5 units from the player's feet (*not* their eyes).

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x14	Play	Server	Status	VarInt Enum	The action the player is taking against the block (see below)
			Location	Position	Block position
			Face	Byte Enum	The face being hit (see below)

Status can be one of seven values:

Value	Meaning	Notes
0	Started digging	
1	Cancelled digging	Sent when the player lets go of the Mine Block key (default: left click)
2	Finished digging	Sent when the client thinks it is finished
3	Drop item stack	Triggered by using the Drop Item key (default: Q) with the modifier to drop the entire selected stack (default: depends on OS). Location is always set to 0/0/0, Face is always set to -Y.
4	Drop item	Triggered by using the Drop Item key (default: Q). Location is always set to 0/0/0, Face is always set to -Y.
5	Shoot arrow / finish eating	Indicates that the currently held item should have its state updated such as eating food, pulling back bows, using buckets, etc. Location is always set to 0/0/0, Face is always set to -Y.
6	Swap item in hand	Used to swap or assign an item to the second hand. Location is always set to 0/0/0, Face is always set to -Y.

The Face field can be one of the following values, representing the face being hit:

Value	Offset	Face
0	-Y	Bottom
1	+Y	Top
2	-Z	North
3	+Z	South
4	-X	West
5	+X	East

Entity Action

Sent by the client to indicate that it has performed certain actions: sneaking (crouching), sprinting, exiting a bed, jumping with a horse, and opening a horse's inventory while riding it.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x15	Play	Server	Entity ID	VarInt	Player ID
			Action ID	VarInt Enum	The ID of the action, see below
			Jump Boost	VarInt	Only used by the “start jump with horse” action, in which case it ranges from 0 to 100. In all other cases it is 0.

Action ID can be one of the following values:

ID	Action
0	Start sneaking
1	Stop sneaking
2	Leave bed
3	Start sprinting
4	Stop sprinting
5	Start jump with horse
6	Stop jump with horse
7	Open horse inventory
8	Start flying with elytra

Leave bed is only sent when the “Leave Bed” button is clicked on the sleep GUI, not when waking up due today time.

Open horse inventory is only sent when pressing the inventory key (default: E) while on a horse — all other methods of opening a horse's inventory (involving right-clicking or shift-right-clicking it) do not use this packet.

Steer Vehicle

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x16	Play	Server	Sideways	Float	Positive to the left of the player
			Forward	Float	Positive forward
			Flags	Unsigned Byte	Bit mask. 0x1: jump, 0x2: unmount

Also known as 'Input' packet.

Crafting Book Data

Packet ID	State	Bound To	Field Name		Field Type	Notes
0x17	Play	Server	Type		VarInt	Determines the format of the rest of the packet
			Type	**Field Name**		
			0: Displayed Recipe	Recipe ID	Int	The internal id of the displayed recipe.
			1: Crafting Book Status	Crafting Book Open	Boolean	Whether the player has the crafting book currently opened/active.
				Crafting Filter	Boolean	Whether the player has the crafting filter option currently active.

The Crafting Book Status type is sent when the player closes its inventory.

Resource Pack Status

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x18	Play	Server	Result	VarInt Enum	0: successfully loaded, 1: declined, 2: failed download, 3: accepted

Advancement Tab

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x19	Play	Server	Action	VarInt enum	0: Opened tab, 1: Closed screen
			Tab ID	Optional identifier	Only present if action is Opened tab

Held Item Change (serverbound)

Sent when the player changes the slot selection

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x1A	Play	Server	Slot	Short	The slot which the player has selected (0–8)

Creative Inventory Action

While the user is in the standard inventory (i.e., not a crafting bench) in Creative mode, the player will send this packet.

Clicking in the creative inventory menu is quite different from non-creative inventory management. Picking up an item with the mouse actually deletes the item from the server, and placing an item into a slot or dropping it out of the inventory actually tells the server to create the item from scratch. (This can be verified by clicking an item that you don't mind deleting, then severing the connection to the server; the item will be nowhere to be found when you log back in.) As a result of this implementation strategy, the "Destroy Item" slot is just a client-side implementation detail that means "I don't intend to recreate this item.". Additionally, the long listings of items (by category, etc.) are a client-side interface for choosing which item to create. Picking up an item from such listings sends no packets to the server; only when you put it somewhere does it tell the server to create the item in that location.

This action can be described as "set inventory slot". Picking up an item sets the slot to item ID -1. Placing an item into an inventory slot sets the slot to the specified item. Dropping an item (by clicking outside the window) effectively sets slot -1 to the specified item, which causes the server to spawn the item entity, etc.. All other inventory slots are numbered the same as the non-creative inventory (including slots for the 2x2 crafting menu, even though they aren't visible in the vanilla client).

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x1B	Play	Server	Slot	Short	Inventory slot
			Clicked Item	Slot	

Update Sign

This message is sent from the client to the server when the “Done” button is pushed after placing a sign.

The server only accepts this packet after Open Sign Editor, otherwise this packet is silently ignored.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x1C	Play	Server	Location	Position	Block Coordinates
			Line 1	String (384)	First line of text in the sign
			Line 2	String (384)	Second line of text in the sign
			Line 3	String (384)	Third line of text in the sign
			Line 4	String (384)	Fourth line of text in the sign

Animation (serverbound)

Sent when the player's arm swings.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x1D	Play	Server	Hand	VarInt Enum	Hand used for the animation. 0: main hand, 1: off hand.

Spectate

Teleports the player to the given entity. The player must be in spectator mode.

The Notchian client only uses this to teleport to players, but it appears to accept any type of entity. The entity does not need to be in the same dimension as the player; if necessary, the player will be respawned in the right world. If the given entity cannot be found (or isn't loaded), this packet will be ignored. It will also be ignored if the player attempts to teleport to themselves.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x1E	Play	Server	Target Player	UUID	UUID of the player to teleport to (can also be an entity UUID)

Player Block Placement

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x1F	Play	Server	Location	Position	Block position
			Face	VarInt Enum	The face on which the block is placed (as documented at Player Digging)
			Hand	VarInt Enum	The hand from which the block is placed; 0: main hand, 1: off hand
			Cursor Position X	Float	The position of the crosshair on the block, from 0 to 1 increasing from west to east
			Cursor Position Y	Float	The position of the crosshair on the block, from 0 to 1 increasing from bottom to top
			Cursor Position Z	Float	The position of the crosshair on the block, from 0 to 1 increasing from north to south

Upon placing a block, this packet is sent once.

The Cursor Position X/Y/Z fields (also known as in-block coordinates) are calculated using raytracing. The unit corresponds to sixteen pixel in the default resource pack. For example, let's say a slab is being placed against the south face of a full block. The Cursor Position X will be higher if the player was pointing near the right (east) edge of the face, lower if pointing near the left. The Cursor Position Y will be used to determine whether it will appear as a bottom slab (values 0.0–0.5) or as a top slab (values 0.5-1.0). The Cursor Position Z should be 1.0 since the player was looking at the southernmost part of the block.

Use Item

Sent when pressing the Use Item key (default: right click) with an item in hand.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x20	Play	Server	Hand	VarInt Enum	Hand used for the animation. 0: main hand, 1: off hand.

Status

Main article: Server List Ping

Clientbound

Response

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x00	Status	Client	JSON Response	String (32767)	See Server List Ping#Response

Pong

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x01	Status	Client	Payload	Long	Should be the same as sent by the client

Serverbound

Request

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x00	Status	Server	*no fields*		

Ping

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x01	Status	Server	Payload	Long	May be any number. Notchian clients use a system-dependent time value which is counted in milliseconds.

Login

The login process is as follows:

1. C→S: Handshake with Next State set to 2 (login)
2. C→S: Login Start
3. S→C: Encryption Request
4. Client auth
5. C→S: Encryption Response

6. Server auth, both enable encryption
7. S→C: Set Compression (optional)
8. S→C: Login Success

Set Compression, if present, must be sent before Login Success. Note that anything sent after Set Compression must use the Post Compression packet format.

For unauthenticated and localhost connections (either of the two conditions is enough for an unencrypted connection) there is no encryption. In that case Login Start is directly followed by Login Success.

See Protocol Encryption for details.

Clientbound

Disconnect (login)

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x00	Login	Client	Reason	Chat	

Encryption Request

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x01	Login	Client	Server ID	String (20)	Appears to be empty
			Public Key Length	VarInt	Length of Public Key
			Public Key	Byte Array	
			Verify Token Length	VarInt	Length of Verify Token. Always 4 for Notchian servers.
			Verify Token	Byte Array	A sequence of random bytes generated by the server

See Protocol Encryption for details.

Login Success

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x02	Login	Client	UUID	String (36)	Unlike in other packets, this field contains the UUID as a string with hyphens.
			Username	String (16)	

This packet switches the connection state to play.

Set Compression

Enables compression. If compression is enabled, all following packets are encoded in the compressed packet format. Negative values will disable compression, meaning the packet format should remain in the uncompressed packet format. However, this packet is entirely optional, and if not sent, compression will also not be enabled (the notchian server does not send the packet when compression is disabled).

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x03	Login	Client	Threshold	VarInt	Maximum size of a packet before it is compressed

Serverbound

Login Start

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x00	Login	Server	Name	String (16)	Player's Username

Encryption Response

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x01	Login	Server	Shared Secret Length	VarInt	Length of Shared Secret
			Shared Secret	Byte Array	
			Verify Token Length	VarInt	Length of Verify Token
			Verify Token	Byte Array	

See Protocol Encryption for details.