# SpockBot Documentation

***Release 0.1.2***

**The SpockBot Project**

April 25, 2016

CHAPTER 1

# Welcome to SpockBot's documentation!

## 1.1 Installation

If you are looking to contribute to SpockBot please follow the instructions *here*.

1. Install cryptography first, instructions for that are here
2. Clone the repository locally:

```
git clone https://github.com/SpockBotMC/SpockBot.git
```

3. Install SpockBot:

```
cd SpockBot
python3 setup.py install
```

## 1.2 Getting Started

See the Basic Example Bot for basic usage of SpockBot.

Also see the Extra Repository for some more plugin examples.

## 1.3 Plugins

### 1.3.1 Core Plugins

#### Auth

##### Description

Provides authorization functions for Mojang's login and session servers

##### Events

Undocumented

#### Methods and Attributes

**class** spockbot.plugins.core.auth.**AuthCore**(*event*, *online_mode*, *auth_timeout*)

**auth_token**

**client_token**

**get_shared_secret**()

**get_username**()

**password**

**send_session_auth**(*pubkey_raw*, *server_id_raw*)

**set_auth_token**(*auth_token*)

**set_client_token**(*client_token*)

**set_password**(*password*)

**set_username**(*username*)

**shared_secret**

**start_session**()

**username**

### Event

#### Description

Provides the core event loop

#### Events

**event_start**
Fired before the first iteration of the event loop. The default StartPlugin depends on this event.

**event_tick**
Fired at the beginning of every iteration of the event loop. The default NetPlugin depends on this event.

**event_kill**
Fired after the last iteration of the event loop. Many plugins depend on this event for environment clean up purposes.

#### Methods and Attributes

### Net

#### Description

Provides an asynchronous, crypto and compression aware socket for connecting to servers and processing incoming packet data. Coordinates with the Timers plugin to honor wall-clock timers

#### Events

Undocumented

#### Methods and Attributes

**class** spockbot.plugins.core.net.**NetCore**(*sock*, *event*, *select*)

**connect**(*host='localhost'*, *port=25565*)

**disable_crypto**()

**enable_crypto**(*secret_key*)

**push**(*packet*)

**push_packet**(*ident*, *data*)

**read_packet**(*data=b''*)

**reset**(*sock*)

**set_comp_state**(*threshold*)

**set_proto_state**(*state*)

### Settings

#### Description

#### Events

Undocumented

#### Methods and Attributes

Undocumented

### Task Manager

#### Description

#### Events

Undocumented

#### Methods and Attributes

Undocumented

#### Ticker

##### Description

Registers timers to provide the necessary tick rates expected by MC servers

##### Events

**action_tick**
: This tick is for scheduling physical actions for the client, specifically those related to movement. Any plugin that wishes to move the client should schedule those movements by hooking into *action_tick* and calling the appropriate pathfinding or movement methods.

**physics_tick**
: This tick fires immediately after *action_tick* and signals that all movement actions have been scheduled and the client is ready to process the movement actions as well as world forces to resolve a position for the client. The default PhysicsPlugin depends on this event.

**client_tick**
: This tick fires immediately after *physics_tick* and signals that a new position has been resolved for the client and is ready to be sent to the server. The default MovementPlugin depends on this event.

#### Timer

##### Description

##### Events

Undocumented

##### Methods and Attributes

- *Auth* - Handling Minecraft account authentication
- *Event* - Provides an event system
- *Net* - Handles packets
- *Settings* - Allows custom settings handlers
- *Task Manager* - Create state machine tasks
- *Ticker* - Ticks 20 times a second
- *Timer* - Create timers to do stuff later

### 1.3.2 Helper Plugins

#### Channels

##### Description

Provides interface for Plugin Channels

#### Events

Undocumented

#### Methods and Attributes

**class** spockbot.plugins.helpers.channels.**ChannelsCore**(*net*)

**decode**(*structure*, *data*)

**encode**(*structure*, *data*)

**send**(*channel*, *data*)
Send a plugin channel message

### Chat

#### Description

Processes chat into easy to use events

#### Events

**chat**
Chat event was recieved

**Playload**

```
{'position': position, 'raw': json_data, 'text': text, 'type': chat_type, 'message': msg, 'name'
```

**position**
*int*

Where the text should be displayed

**raw**
*dict*

Raw json data from the chat message

**text**
*string*

The text of the chat message

**type**
*string*

The type of message (achievement, admin, announcement, emote, incoming, outgoing, text)

**message**
*string*

message

**uuid**
*string*

UUID of the player who sent the message

#### Methods and Attributes

**class** spockbot.plugins.helpers.chat.**ChatCore**(*net*)

**chat**(*message*)

**whisper**(*player*, *message*)

### ClientInfo

#### Description

ClientInfo is a central plugin for recording data about the client, e.g. Health, position, and some auxillary information like the player list. Plugins subscribing to ClientInfo's events don't have to independently track this information on their own.

#### Events

**client_login_success**
Client Info is done processing the Login Success packet

**client_join_game**
Client Info is done processing the Join Game packet

**Payload**

```
GameInfo()
```

**client_spawn_update**
Client Info is done processing the Spawn Position packet

**Payload**

```
Position()
```

**client_health_update**
When the players health changes

**Payload**

```
PlayerHealth()
```

**client_death**
When the player health becomes 0.0

**Payload**

```
PlayerHealth()
```

**client_position_update**
When the player is moved by the server

**Payload**

```
PlayerPosition()
```

**client_add_player**
Player added to the player list

**Payload**

```
PlayerListItem()
```

**client_update_player**
Player info is updated on the player list

**Payload**

```
PlayerListItem()
```

**client_remove_player**
Player removed from the player list

**Payload**

```
PlayerListItem()
```

#### Methods and Attributes

**class** spockbot.plugins.helpers.clientinfo.**ClientInfo**

**eid**
*int*

Entity ID of the player

**name**
*str*

Player's Username

**uuid**
*str*

Player's UUID

**abilities**
*Abilities*

Player's current movement state and speed

**game_info**
*GameInfo*

Information about the current world/server

**spawn_position**
*Position*

Players initial position

**health**
*PlayerHealth*

Player's health, food and saturation

**position**
*PlayerPosition*

Player's current position

**player_list**
*dict*

List of all players in the server

**eye_pos**
*PlayerPosition*

Player's eye position

**eye_pos**

**reset**()
Resets the information in ClientInfo

### Craft

#### Description

Craft items.

#### Events

Undocumented

#### Methods and Attributes

**class** spockbot.plugins.helpers.craft.**CraftPlugin**(*ploader*, *settings*)

**craft**(*item=None*, *meta=None*, *amount=1*, *recipe=None*, *parent=None*)
Starts a `craft_task`. Either `item` or `recipe` has to be given.

**Returns** The recipe used for crafting.

**Return type** Optional[Recipe]

**craft_task**(*recipe*, *amount=1*)
A task that crafts `amount` items with `recipe`.

**pl_announce = ('Craft',)**

**requires = ('Inventory', 'TaskManager')**

### Entities

#### Description

An entity tracker

#### Events

Undocumented

#### Methods and Attributes

**class** spockbot.plugins.helpers.entities.**EntitiesCore**

### Interact

#### Description

Interact with the world: - swing the arm, sneak, sprint, jump with a horse, leave the bed - look around - dig/place/use blocks - use the held (active) item - use/attack entities - steer vehicles - place and write signs - edit and sign books

By default, the client sends swing and look packets like the vanilla client. This can be disabled by setting the `auto_swing` and `auto_look` flags.

#### Events

Undocumented

#### Methods and Attributes

**class** spockbot.plugins.helpers.interact.**InteractPlugin**(*ploader*, *settings*)

**activate_item**()
Use (hold right-click) the item in the active slot. Examples: pull the bow, start eating once, throw an egg.

**attack_entity**(*entity*)

**cancel_digging**()

**click_block**(*pos*, *look_at_block=True*, *swing=True*, ***kwargs*)
Click on a block. Examples: push button, open window, make redstone ore glow

**Parameters**

- **face** (*int*) – side of the block on which the block is placed on
- **cursor_pos** (Vector3) – where to click inside the block, each dimension 0-15

**deactivate_item**()
Stop using (release right-click) the item in the active slot. Examples: shoot the bow, stop eating.

**dig_block**(*pos*)
Not cancelable.

**edit_book**(*pages*)
Set the pages of current book in hand

**finish_digging**()

**jump_horse**(*jump_boost=100*)

**jump_vehicle**()

**leave_bed**()

**look**(*yaw=0.0*, *pitch=0.0*, *radians=False*)

**look_at**(*pos*)

**look_at_rel**(*delta*)

**look_rel**(*d_yaw=0.0*, *d_pitch=0.0*, *radians=False*)

**mount_vehicle**(*entity*)

**open_inventory**()

**pl_announce = ('Interact',)**

**place_block**(*pos*, *sneak=True*, ***kwargs*)
Place a block next to `pos`. If the block at `pos` is air, place at `pos`.

**place_sign**(*pos*, *lines=[]*, ***place_block_kwargs*)
Place a sign block and write on it.

**requires = ('ClientInfo', 'Event', 'Inventory', 'Net', 'Channels')**

**sign_book**(*author*, *title*)
Sign current book in hand

**sneak**(*sneak=True*)

**sprint**(*sprint=True*)

**start_digging**(*pos*, *face=1*)

**steer_vehicle**(*left=0.0*, *forward=0.0*, *jump=False*, *unmount=False*)

**swing_arm**()

**unmount_vehicle**()

**unsneak**()

**unsprint**()

**use_bucket**(*pos*)
Using buckets is different from placing blocks. See "Special note on using buckets" in http://wiki.vg/Protocol#Player_Block_Placement

**use_entity**(*entity*, *cursor_pos=None*, *action=0*)
Uses (right-click) an entity to open its window. Setting `cursor_pos` sets `action` to "interact at".

**write_book**(*text*, *author=''*, *title=''*, *sign=False*)
Write text to the current book in hand, optionally sign the book

### Inventory

#### Description

The Inventory plugin keeps track of the inventory and provides simple inventory analysis and manipulation.

#### Events

Undocumented

#### Methods and Attributes

**class** spockbot.plugins.helpers.inventory.**InventoryCore**(*net_plugin*, *send_click*)
Handles operations with the player inventory.

**active_slot**

**click_slot**(*slot*, *right=False*)
Left-click or right-click the slot.

**Parameters** **slot** (Slot) – The clicked slot. Can be Slot instance or integer. Set to inventory.cursor_slot for clicking outside the window.

**close_window**()

**creative_set_slot**(*slot_nr=None*, *slot_dict=None*, *slot=None*)

**drop_slot**(*slot=None*, *drop_stack=False*)
Drop one or all items of the slot.

Does not wait for confirmation from the server. If you want that, use a Task and yield inventory.async.drop_slot() instead.

If slot is None, drops the cursor_slot or, if that's empty, the currently held item (active_slot).

**Parameters** **slot** (*Optional[Slot]*) – The dropped slot. Can be None, integer, or Slot instance.

**Returns** The action ID of the click

**Return type** int

**find_slot**(*wanted*, *slots=None*)
Searches the given slots or, if not given, active hotbar slot, hotbar, inventory, open window in this order.

**Parameters** **wanted** – function(Slot) or Slot or itemID or (itemID, metadata)

**Returns** The first slot containing the item or None if not found.

**Return type** Optional[Slot]

**find_slots**(*wanted*, *slots=None*)
Yields all slots containing the item. Searches the given slots or, if not given, active hotbar slot, hotbar, inventory, open window in this order.

**Parameters** **wanted** – function(Slot) or Slot or itemID or (itemID, metadata)

**inv_slots_preferred**
List of all available inventory slots in the preferred search order. Does not include the additional slots from the open window.

1.active slot

2.remainder of the hotbar

3.remainder of the persistent inventory

**select_active_slot**(*slot_or_hotbar_index*)

**total_stored**(*wanted*, *slots=None*)

Calculates the total number of items of that type in the current window or given slot range.

**Parameters** **wanted** – function(Slot) or Slot or itemID or (itemID, metadata)

### Keep Alive

#### Description

#### Events

Undocumented

#### Methods and Attributes

Undocumented

### Movement

#### Description

MovementPlugin provides a centralized plugin for controlling client movement so the client doesn't try to pull itself in a dozen directions.

#### Events

Undocumented

#### Methods and Attributes

**class** spockbot.plugins.helpers.movement.**MovementCore**(*plug*)

**current_path**

**current_target**

**final_target**

**is_moving**

**stop**()

### Pathfinding

#### Description

Very rough asychronous pathfinding plugin Implements the Lazy Theta* pathfinding algorithm

#### Events

Undocumented

#### Methods and Attributes

**class** spockbot.plugins.helpers.pathfinding.**PathfindingCore**(*start_path*)

### Physics

#### Description

A Physics module built from clean-rooming the Notchian Minecraft client

Collision detection and resolution is done by a Separating Axis Theorem implementation for concave shapes decomposed into Axis-Aligned Bounding Boxes. This isn't totally equivalent to vanilla behavior, but it's faster and Close Enough^TM

AKA this file does Minecraft physics

#### Events

Undocumented

#### Methods and Attributes

**class** spockbot.plugins.helpers.physics.**PhysicsCore**(*pos*, *vec*, *abilities*)

**jump**()

**move_angle**(*angle*, *radians=False*)

**move_target**(*vector*)

**move_vector**(*vector*)

**sprint**()

**walk**()

### Respawn

#### Description

#### Events

Undocumented

#### Methods and Attributes

Undocumented

### Start

#### Description

This plugin creates a convenient start() method and attaches it directly to the client. More complex bots will likely want to create their own initialization plugin, so StartPlugin stays out of the way unless you call the start() method. However, the start() method is very convenient for demos and tutorials, and illustrates the basic steps for initializing a bot.

#### Events

Undocumented

#### Methods and Attributes

Undocumented

### World

#### Description

Provides a very raw (but very fast) world map for use by plugins. Plugins interested in a more comprehensive world map view can use mcp.mapdata to interpret blocks and their metadata more comprehensively. Planned to provide light level interpretation based on sky light and time of day

#### Events

Undocumented

#### Methods and Attributes

**class** spockbot.plugins.helpers.world.**WorldData**(*dimension=0*)

**new_dimension**(*dimension*)

**reset**()

**update_time**(*data*)

- *Channels* - Process Plugin Channels
- *Chat* - Process chat into simple events
- *ClientInfo* - Keep track of information about the client/player
- *Craft* - Craft items
- *Entities* - Keep track of entities
- *Interact* - Interact with the world
- *Inventory* - Interact with inventories in Minecraft
- *Keep Alive* - Keep sending the keep alive packet
- *Movement* - Basic movement around the world
- *Pathfinding* - Pathfinding
- *Physics* - Reimplmentation of the Vanilla physics
- *Respawn* - Auto respawn the player on death
- *Start* - Helper start system for doing handshake and login
- *World* - Keep track of world data

## 1.4 Modules

### 1.4.1 spockbot package

#### Subpackages

##### spockbot.mcdata package

**Submodules**

**spockbot.mcdata.biomes module**
spockbot.mcdata.biomes.**get_biome**(*biome*)

**spockbot.mcdata.blocks module**
**class** spockbot.mcdata.blocks.**Block**(*meta=None*)
Bases: object

**bounding_box = None**

**diggable = True**

**display_name = 'Block'**

**drops = []**

**hardness = 0.0**

**harvest_tools = []**

**id = -1**

**material = None**

**name = 'block'**

**slipperiness = 0.6**

**stack_size = 0**

**variations = {}**

spockbot.mcdata.blocks.**block_ext**(**block_ids*)

spockbot.mcdata.blocks.**get_block**(*block*, *meta=0*, *init=True*)

**spockbot.mcdata.constants module** These constants are used in some plugins, but do not belong there. Some of them can later be extracted from minecraft-data

**spockbot.mcdata.items module**

**class** spockbot.mcdata.items.**Item**(*meta=None*)

Bases: object

**display_name = 'Item'**

**id = -1**

**name = 'item'**

**stack_size = 0**

**variations = {}**

spockbot.mcdata.items.**get_item**(*item*, *meta=0*, *init=True*)

**spockbot.mcdata.materials module**

spockbot.mcdata.materials.**get_material**(*name*)

**spockbot.mcdata.recipes module**

**class** spockbot.mcdata.recipes.**Recipe**(*raw*)

Bases: object

**ingredient_positions**

**Returns** In the form { (item_id, metadata) -> [(x, y, amount), ...] }

**Return type** dict

**total_ingredient_amounts**

**Returns** In the form { (item_id, metadata) -> amount }

**Return type** dict

**class** spockbot.mcdata.recipes.**RecipeItem**(*id*, *meta*, *amount*)

Bases: tuple

**__getnewargs__**()

Return self as a plain tuple. Used by copy and pickle.

**__getstate__**()

Exclude the OrderedDict from pickling

**static __new__** (*_cls*, *id*, *meta*, *amount*)
Create new instance of RecipeItem(id, meta, amount)

**__repr__** ()
Return a nicely formatted representation string

**amount**
Alias for field number 2

**id**
Alias for field number 0

**meta**
Alias for field number 1

spockbot.mcdata.recipes.**get_any_recipe**(*item*, *meta=None*)

spockbot.mcdata.recipes.**iter_recipes**(*item_id*, *meta=None*)

spockbot.mcdata.recipes.**reformat_item**(*raw*, *default_meta=None*)

spockbot.mcdata.recipes.**reformat_shape**(*shape*)

**spockbot.mcdata.utils module**

**class** spockbot.mcdata.utils.**BoundingBox**(*w*, *h*, *d=None*)
Bases: *spockbot.vector.Vector3*

**class** spockbot.mcdata.utils.**Info**
Bases: object

**get_dict**()

**set_dict**(*data*)

spockbot.mcdata.utils.**camel_case**(*text*)

spockbot.mcdata.utils.**clean_var**(*text*)
Turn text into a valid python classname or variable

spockbot.mcdata.utils.**find_by**(*key*, **args*)

spockbot.mcdata.utils.**snake_case**(*text*)

spockbot.mcdata.utils.**split_words**(*text*)

**spockbot.mcdata.windows module**

**class** spockbot.mcdata.windows.**BaseClick**
Bases: object

**apply**(*inv_plugin*)
Called by on_success(). Abstract method.

**Parameters** **inv_plugin** (InventoryPlugin) – inventory plugin instance

**cleanup_if_empty**(*slot*)

**copy_slot_type**(*slot_from*, *slot_to*)

**get_packet**(*inv_plugin*)
Called by send_click() to prepare the sent packet. Abstract method.

**Parameters** **inv_plugin** (InventoryPlugin) – inventory plugin instance

**mark_dirty**(*slot*)

**on_success**(*inv_plugin*, *emit_set_slot*)
Called when the click was successful and should be applied to the inventory.

**Parameters**

- **inv_plugin** (InventoryPlugin) – inventory plugin instance
- **emit_set_slot** (*func*) – function to signal a slot change, should be InventoryPlugin().emit_set_slot

**swap_slots**(*slot_a*, *slot_b*)

**transfer**(*from_slot*, *to_slot*, *max_amount*)

**class** spockbot.mcdata.windows.**DropClick**(*slot*, *drop_stack=False*)
Bases: *spockbot.mcdata.windows.BaseClick*

**apply**(*inv_plugin*)

**get_packet**(*inv_plugin*)

**class** spockbot.mcdata.windows.**SingleClick**(*slot*, *button=0*)
Bases: *spockbot.mcdata.windows.BaseClick*

**apply**(*inv_plugin*)

**get_packet**(*inv_plugin*)

**class** spockbot.mcdata.windows.**Slot**(*window*, *slot_nr*, *id=-1*, *damage=0*, *amount=0*, *enchants=None*)
Bases: object

**copy**()

**get_dict**()
Formats the slot for network packing.

**is_empty**

**matches**(*other*)

**move_to_window**(*window*, *slot_nr*)

**stacks_with**(*other*)

**class** spockbot.mcdata.windows.**SlotCursor**(*id=-1*, *damage=0*, *amount=0*, *enchants=None*)
Bases: *spockbot.mcdata.windows.Slot*

**class** spockbot.mcdata.windows.**Window**(*window_id*, *title*, *slot_count*, *inv_type=None*, *persistent_slots=None*, *eid=None*)
Bases: object

Base class for all inventory types.

**hotbar_slots**

**inv_data** = {}

**inv_type** = None

**inventory_slots**

**name** = None

**persistent_slots**

**window_slots**
All slots except inventory and hotbar. Useful for searching.

spockbot.mcdata.windows.**make_slot_check**(*wanted*)
Creates and returns a function that takes a slot and checks if it matches the wanted item.

**Parameters** **wanted** – function(Slot) or Slot or itemID or (itemID, metadata)

**Module contents**
spockbot.mcdata.**get_item_or_block**(*find*, *meta=0*, *init=True*)

### spockbot.mcp package

**Submodules**

**spockbot.mcp.bbuff module**
**class** spockbot.mcp.bbuff.**BoundBuffer**(*data=b''*)
Bases: object

**append**(*data*)

**buff = b''**

**cursor = 0**

**flush**()

**read**(*length*)

**recv**(*length*)

**revert**()

**save**()

**tell**()

**write**(*data*)
**exception** spockbot.mcp.bbuff.**BufferUnderflowException**
Bases: Exception

**spockbot.mcp.datautils module**
spockbot.mcp.datautils.**byte_to_hex**(*byte_str*)
spockbot.mcp.datautils.**pack**(*data_type*, *data*)
spockbot.mcp.datautils.**pack_fixed_point**(*mc_type*, *val*)
spockbot.mcp.datautils.**pack_metadata**(*data*)
spockbot.mcp.datautils.**pack_position**(*position*)
spockbot.mcp.datautils.**pack_slot**(*slot*)
spockbot.mcp.datautils.**pack_varint**(*val*)
spockbot.mcp.datautils.**pack_varlong**(*val*)
spockbot.mcp.datautils.**unpack**(*data_type*, *bbuff*)
spockbot.mcp.datautils.**unpack_fixed_point**(*mc_type*, *bbuff*)
spockbot.mcp.datautils.**unpack_metadata**(*bbuff*)
spockbot.mcp.datautils.**unpack_position**(*bbuff*)

spockbot.mcp.datautils.**unpack_slot**(*bbuff*)

spockbot.mcp.datautils.**unpack_varint**(*bbuff*)

spockbot.mcp.datautils.**unpack_varlong**(*bbuff*)

#### spockbot.mcp.mcdata module

#### spockbot.mcp.mcpacket module

**class** spockbot.mcp.mcpacket.**Packet**(*ident=[0, 1, 0], data=None*)

Bases: object

**clone**()

**decode**(*bbuff*, *proto_comp_state*)

**encode**(*proto_comp_state*, *proto_comp_threshold*, *comp_level=6*)

**new_ident**(*ident*)

**exception** spockbot.mcp.mcpacket.**PacketDecodeFailure**(*packet*, *pbuff*, *underflow=False*)

Bases: Exception

#### spockbot.mcp.mcpacket_extensions module

**spockbot.mcp.nbt module** Handle the NBT (Named Binary Tag) data format

**exception** spockbot.mcp.nbt.**MalformedFileError**

Bases: Exception

Exception raised on parse error.

**class** spockbot.mcp.nbt.**Tag**(*value=None*, *name=None*)

Bases: object

Tag, a variable with an intrinsic name.

**__repr__**()

Return a string (ascii formated for Python 2, unicode for Python 3) describing the class, name and id for debugging purposes.

**__str__**()

Return a string (ascii formated for Python 2, unicode for Python 3) with the result in human readable format. Unlike valuestr(), the result is recursive for iterators till at least one level deep.

**__unicode__**()

Return a unicode string with the result in human readable format. Unlike valuestr(), the result is recursive for iterators till at least one level deep.

**id = None**

**pretty_tree**(*indent=0*)

Return formated Unicode string of self, where iterable items are recursively listed in detail.

**tag_info**()

Return Unicode string with class, name and unnested value.

**valuestr**()

Return Unicode string of unnested value. For iterators, this returns a summary.

**class** spockbot.mcp.nbt.**TagByte**(*value=None*, *name=None*, *buffer=None*)
Bases: spockbot.mcp.nbt._TagNumeric

Represent a single tag storing 1 byte.

**fmt = <Struct object>**

**id = 1**

**class** spockbot.mcp.nbt.**TagByteArray**(*name=None*, *buffer=None*)
Bases: *spockbot.mcp.nbt.Tag*, collections.abc.MutableSequence

TagByteArray, comparable to a collections.UserList with an intrinsic name whose values must be bytes

**id = 7**

**insert**(*key*, *value*)

**valuestr**()

**class** spockbot.mcp.nbt.**TagCompound**(*buffer=None*)
Bases: *spockbot.mcp.nbt.Tag*, collections.abc.MutableMapping

TagCompound, comparable to a collections.OrderedDict with an intrinsic name

**id = 10**

**iteritems**()

**keys**()

**pretty_tree**(*indent=0*)

**valuestr**()

**class** spockbot.mcp.nbt.**TagDouble**(*value=None*, *name=None*, *buffer=None*)
Bases: spockbot.mcp.nbt._TagNumeric

Represent a single tag storing 1 IEEE-754 double precision floating point number.

**fmt = <Struct object>**

**id = 6**

**class** spockbot.mcp.nbt.**TagFloat**(*value=None*, *name=None*, *buffer=None*)
Bases: spockbot.mcp.nbt._TagNumeric

Represent a single tag storing 1 IEEE-754 floating point number.

**fmt = <Struct object>**

**id = 5**

**class** spockbot.mcp.nbt.**TagInt**(*value=None*, *name=None*, *buffer=None*)
Bases: spockbot.mcp.nbt._TagNumeric

Represent a single tag storing 1 int.

**fmt = <Struct object>**

**id = 3**

**class** spockbot.mcp.nbt.**TagIntArray**(*name=None*, *buffer=None*)
Bases: *spockbot.mcp.nbt.Tag*, collections.abc.MutableSequence

TagIntArray, comparable to a collections.UserList with an intrinsic name whose values must be integers

**id = 11**

**insert**(*key*, *value*)

**update_fmt**(*length*)
Adjust struct format description to length given

**valuestr**()

**class** spockbot.mcp.nbt.**TagList**(*type=None*, *value=None*, *name=None*, *buffer=None*)
Bases: *spockbot.mcp.nbt.Tag*, collections.abc.MutableSequence

TagList, comparable to a collections.UserList with an intrinsic name

**id = 9**

**insert**(*key*, *value*)

**pretty_tree**(*indent=0*)

**valuestr**()

**class** spockbot.mcp.nbt.**TagLong**(*value=None*, *name=None*, *buffer=None*)
Bases: spockbot.mcp.nbt._TagNumeric

Represent a single tag storing 1 long.

**fmt = <Struct object>**

**id = 4**

**class** spockbot.mcp.nbt.**TagShort**(*value=None*, *name=None*, *buffer=None*)
Bases: spockbot.mcp.nbt._TagNumeric

Represent a single tag storing 1 short.

**fmt = <Struct object>**

**id = 2**

**class** spockbot.mcp.nbt.**TagString**(*value=None*, *name=None*, *buffer=None*)
Bases: *spockbot.mcp.nbt.Tag*, collections.abc.Sequence

TagString, comparable to a collections.UserString with an intrinsic name

**id = 8**

### spockbot.mcp.yggdrasil module

**class** spockbot.mcp.yggdrasil.**YggdrasilCore**(*username=''*, *password=''*, *client_token=''*, *access_token=''*)
Bases: object

**authenticate**()
Generate an access token using an username and password. Any existing client token is invalidated if not provided.

**Returns** Response or error dict

**Return type** dict

**invalidate**()
Invalidate access tokens with a client/access token pair

**Returns** Empty or error dict

**Return type** dict

**login**()

**logout**()

**refresh**()
Generate an access token with a client/access token pair. Used access token is invalidated.

**Returns** Response or error dict

**Return type** dict

**signout**()
Invalidate access tokens with a username and password.

**Returns** Empty or error dict

**Return type** dict

**validate**()
Check if an access token is valid

**Returns** Empty or error dict

**Return type** dict

**ygg_url = 'https://authserver.mojang.com'**

**ygg_version = 1**

**Module contents**

### spockbot.plugins package

**Subpackages**

**spockbot.plugins.core package**

**Submodules**

**spockbot.plugins.core.auth module** Provides authorization functions for Mojang's login and session servers

**class** spockbot.plugins.core.auth.**AuthCore**(*event*, *online_mode*, *auth_timeout*)
Bases: object

**auth_token**

**client_token**

**get_shared_secret**()

**get_username**()

**password**

**send_session_auth**(*pubkey_raw*, *server_id_raw*)

**set_auth_token**(*auth_token*)

**set_client_token**(*client_token*)

**set_password**(*password*)

**set_username**(*username*)

**shared_secret**

**start_session**()

**username**

**class** spockbot.plugins.core.auth.**AuthPlugin**(*ploader*, *settings*)
Bases: *spockbot.plugins.base.PluginBase*

**defaults = {'online_mode': True, 'auth_quit': True, 'auth_timeout': 3, 'sess_quit': True}**

**events = {'auth_login_error': 'handle_auth_error', 'auth_session_error': 'handle_session_error'}**

**handle_auth_error**(*name*, *data*)

**handle_session_error**(*name*, *data*)

**pl_announce = ('Auth',)**

**requires = 'Event'**

spockbot.plugins.core.auth.**java_hex_digest**(*digest*)

**spockbot.plugins.core.event module** Provides the core event loop

**class** spockbot.plugins.core.event.**EventPlugin**(*ploader*, *settings*)
Bases: object

**emit**(*event*, *data=None*)

**event_loop**(*once=False*)

**kill**(**args*)

**pl_announce = ('Event',)**

**reg_event_handler**(*event*, *handler*)

**run_continuous**()

**run_once**()

**unreg_event_handler**(*event*, *handler*)

**spockbot.plugins.core.net module** Provides an asynchronous, crypto and compression aware socket for connecting to servers and processing incoming packet data. Coordinates with the Timers plugin to honor wall-clock timers

**class** spockbot.plugins.core.net.**AESCipher**(*shared_secret*)
Bases: object

**decrypt**(*data*)

**encrypt**(*data*)

**class** spockbot.plugins.core.net.**NetCore**(*sock*, *event*, *select*)
Bases: object

**connect**(*host='localhost'*, *port=25565*)

**disable_crypto**()

**enable_crypto**(*secret_key*)

**push**(*packet*)

**push_packet**(*ident*, *data*)

**read_packet**(*data=b''*)

**reset**(*sock*)

**set_comp_state**(*threshold*)

**set_proto_state**(*state*)

**class** spockbot.plugins.core.net.**NetPlugin**(*ploader*, *settings*)
Bases: *spockbot.plugins.base.PluginBase*

**check_quit**()

**defaults = {'sock_quit': True, 'bufsize': 4096}**

**events = {'select_send': 'handle_send', 'HANDSHAKE>Handshake': 'handle_handshake', 'LOGIN<Disconnect': 'han**

**handle_comp**(*name*, *packet*)
Handle Set Compression packets

**handle_disconnect**(*name*, *packet*)

**handle_err**(*name*, *error*)
Socket Error has occured

**handle_handshake**(*name*, *packet*)
Change to whatever the next state is going to be

**handle_hup**(*name*, *data*)
Socket has hung up

**handle_kill**(*name*, *data*)
Try to shutdown the socket politely

**handle_login_disconnect**(*name*, *packet*)

**handle_login_success**(*name*, *packet*)
Change to Play state

**handle_recv**(*name*, *fileno*)
Socket is ready to recieve data

**handle_select_err**(*name*, *fileno*)

**handle_send**(*name*, *fileno*)
Socket is ready to send data and send buffer has data to send

**pl_announce = ('Net',)**

**requires = ('Event', 'Select', 'Timers')**

**reset_sock**()

**tick**(*name*, *data*)

**spockbot.plugins.core.settings module**

**class** spockbot.plugins.core.settings.**PloaderFetch**(*plugins*, *plugin_settings*)
Bases: object

**get_plugin_settings**(*plugin*)

**get_plugins**()

**class** spockbot.plugins.core.settings.**SettingsPlugin**(*ploader*, *kwargs*)
Bases: object

**pl_announce = ('PloaderFetch',)**

**spockbot.plugins.core.taskmanager module**

**class** spockbot.plugins.core.taskmanager.**TaskManager**(*ploader*, *settings*)
Bases: *spockbot.plugins.base.PluginBase*

**pl_announce = ('TaskManager',)**

**requires = 'Event'**

**run_task**(*task*, *parent=None*, *name=None*)

**spockbot.plugins.core.ticker module** Registers timers to provide the necessary tick rates expected by MC servers

**class** spockbot.plugins.core.ticker.**TickerPlugin**(*ploader*, *settings*)
Bases: *spockbot.plugins.base.PluginBase*

**client_tick**()

**events = {'PLAY<Join Game': 'start_tickers'}**

**requires = ('Event', 'Timers')**

**start_tickers**(_, __)

**spockbot.plugins.core.timer module**

**Module contents**

**spockbot.plugins.helpers package**

**Submodules**

**spockbot.plugins.helpers.channels module** Provides interface for Plugin Channels

**class** spockbot.plugins.helpers.channels.**ChannelsCore**(*net*)
Bases: object

**decode**(*structure*, *data*)

**encode**(*structure*, *data*)

**send**(*channel*, *data*)
Send a plugin channel message

**class** spockbot.plugins.helpers.channels.**ChannelsPlugin**(*ploader*, *settings*)
Bases: *spockbot.plugins.base.PluginBase*

**events = {'PLAY<Plugin Message': 'handle_plugin_message'}**

**handle_plugin_message**(*name*, *packet*)

**pl_announce = ('Channels',)**

**requires = ('Event', 'Net')**

**spockbot.plugins.helpers.chat module** Processes chat into easy to use events

**class** spockbot.plugins.helpers.chat.**ChatCore**(*net*)
Bases: object

**chat**(*message*)

**whisper**(*player*, *message*)

**exception** spockbot.plugins.helpers.chat.**ChatParseError**
Bases: Exception

**class** spockbot.plugins.helpers.chat.**ChatPlugin**(*ploader*, *settings*)
Bases: *spockbot.plugins.base.PluginBase*

Emits chat events with position, raw, text, type, message, name, uuid.

position: Always one of spockbot.mcdata.constants's CHAT_POS_CHAT, CHAT_POS_SYSTEM_MESSAGE, CHAT_POS_ABOVE_HOTBAR.

raw: Always the JSON dict as received from the server.

text: The text (without formatting) of the chat message as the vanilla client would display it. Needs en_US.lang to be present in the active directory, otherwise some but not all messages are translated properly.

type: None or one of achievement, admin, announcement, emote, incoming, outgoing, text, which are the last part of the corresponding vanilla translation IDs.

If type is not None, message, name, uuid are set and an additional chat_<type> event is emitted. Otherwise, message, name, uuid are all None.

message: The message as it was typed by the sender.

name: The name of the sender.

uuid: The UUID of the sender, with dashes.

**events = {'PLAY<Chat Message': 'handle_chat'}**

**handle_chat**(*evt*, *packet*)

**load_translations**()

**pl_announce = ('Chat',)**

**render_chat**(*chat_data*)
Render the text as in the vanilla client. On a vanilla server, this uses the translations dict.

**requires = ('Event', 'Net')**

spockbot.plugins.helpers.chat.**parse_with_1_extra**(*json_data*)

**spockbot.plugins.helpers.clientinfo module** ClientInfo is a central plugin for recording data about the client, e.g. Health, position, and some auxillary information like the player list. Plugins subscribing to ClientInfo's events don't have to independently track this information on their own.

**class** spockbot.plugins.helpers.clientinfo.**Abilities**
Bases: *spockbot.mcdata.utils.Info*

**class** spockbot.plugins.helpers.clientinfo.**ClientInfo**
Bases: object

**eid**
*int*

Entity ID of the player

**name**
*str*

Player's Username

**uuid**
*str*

Player's UUID

**abilities**
*Abilities*

Player's current movement state and speed

**game_info**
*GameInfo*

Information about the current world/server

**spawn_position**
*Position*

Players initial position

**health**
*PlayerHealth*

Player's health, food and saturation

**position**
*PlayerPosition*

Player's current position

**player_list**
*dict*

List of all players in the server

**eye_pos**
*PlayerPosition*

Player's eye position

**eye_pos**

**reset**()
Resets the information in ClientInfo

**class** spockbot.plugins.helpers.clientinfo.**ClientInfoPlugin**(*ploader*, *settings*)
Bases: *spockbot.plugins.base.PluginBase*

**events = {'net_disconnect': 'handle_disconnect', 'PLAY<Update Health': 'handle_update_health', 'LOGIN<Login Suc**

**handle_attach_entity**(*name*, *packet*)

**handle_disconnect**(*name*, *data*)

**handle_game_state**(*name*, *packet*)

**handle_join_game**(*name*, *packet*)

**handle_login_success**(*name*, *packet*)

**handle_player_abilities**(*name*, *packet*)

**handle_player_list**(*name*, *packet*)

**handle_position_update**(*name*, *packet*)

**handle_server_difficulty**(*name*, *packet*)

**handle_spawn_position**(*name*, *packet*)

**handle_update_health**(*name*, *packet*)

**pl_announce = ('ClientInfo',)**

**requires = 'Event'**

**class** spockbot.plugins.helpers.clientinfo.**GameInfo**
Bases: *spockbot.mcdata.utils.Info*

**class** spockbot.plugins.helpers.clientinfo.**PlayerHealth**
Bases: *spockbot.mcdata.utils.Info*

**class** spockbot.plugins.helpers.clientinfo.**PlayerListItem**
Bases: *spockbot.mcdata.utils.Info*

**class** spockbot.plugins.helpers.clientinfo.**PlayerPosition**(**xyz*)
Bases: *spockbot.plugins.helpers.clientinfo.Position*

**class** spockbot.plugins.helpers.clientinfo.**Position**(**xyz*)
Bases: *spockbot.vector.Vector3*, *spockbot.mcdata.utils.Info*

Used for things that require encoding position for the protocol, but also require higher level vector functions.

**get_dict**()

**spockbot.plugins.helpers.craft module** Craft items.

**class** spockbot.plugins.helpers.craft.**CraftPlugin**(*ploader*, *settings*)
Bases: *spockbot.plugins.base.PluginBase*

**craft**(*item=None*, *meta=None*, *amount=1*, *recipe=None*, *parent=None*)
Starts a `craft_task`. Either `item` or `recipe` has to be given.

**Returns** The recipe used for crafting.

**Return type** Optional[Recipe]

**craft_task**(*recipe*, *amount=1*)
A task that crafts `amount` items with `recipe`.

**pl_announce = ('Craft',)**

**requires = ('Inventory', 'TaskManager')**

**spockbot.plugins.helpers.entities module** An entity tracker

**class** spockbot.plugins.helpers.entities.**EntitiesCore**
Bases: `object`

**class** spockbot.plugins.helpers.entities.**EntitiesPlugin**(*ploader*, *settings*)
Bases: *spockbot.plugins.base.PluginBase*

**events = {'PLAY<Destroy Entities': 'handle_destroy_entities', 'PLAY<Entity Teleport': 'handle_set_dict', 'PLAY<Enti**

**handle_destroy_entities**(*event*, *packet*)

**handle_join_game**(*event*, *packet*)

**handle_relative_move**(*event*, *packet*)

**handle_set_dict** (*event*, *packet*)

**handle_spawn_experience_orb** (*event*, *packet*)

**handle_spawn_global_entity** (*event*, *packet*)

**handle_spawn_mob** (*event*, *packet*)

**handle_spawn_object** (*event*, *packet*)

**handle_spawn_painting** (*event*, *packet*)

**handle_spawn_player** (*event*, *packet*)

**handle_unhandled** (*event*, *packet*)

**handle_velocity** (*event*, *packet*)

**pl_announce = ('Entities',)**

**requires = 'Event'**

**class** spockbot.plugins.helpers.entities.**ExpEntity**
Bases: *spockbot.plugins.helpers.entities.MCEntity*

**count = 0**

**x = 0**

**y = 0**

**z = 0**

**class** spockbot.plugins.helpers.entities.**GlobalEntity**
Bases: *spockbot.plugins.helpers.entities.MCEntity*

**global_type = 0**

**x = 0**

**y = 0**

**z = 0**

**class** spockbot.plugins.helpers.entities.**MCEntity**
Bases: *spockbot.mcdata.utils.Info*

**eid = 0**

**metadata = None**

**nbt = None**

**status = 0**

**class** spockbot.plugins.helpers.entities.**MobEntity**
Bases: *spockbot.plugins.helpers.entities.MovementEntity*

**head_pitch = 0**

**head_yaw = 0**

**metadata = None**

**mob_type = 0**

**velocity_x = 0**

**velocity_y = 0**

**velocity_z = 0**

**class** spockbot.plugins.helpers.entities.**MovementEntity**
Bases: *spockbot.plugins.helpers.entities.MCEntity*

**on_ground = True**

**pitch = 0**

**x = 0**

**y = 0**

**yaw = 0**

**z = 0**

**class** spockbot.plugins.helpers.entities.**ObjectEntity**
Bases: *spockbot.plugins.helpers.entities.MovementEntity*

**obj_data = 0**

**obj_type = 0**

**speed_x = 0**

**speed_y = 0**

**speed_z = 0**

**class** spockbot.plugins.helpers.entities.**PaintingEntity**
Bases: *spockbot.plugins.helpers.entities.MCEntity*

**direction = 0**

**location = {'z': 0, 'x': 0, 'y': 0}**

**title = ''**

**class** spockbot.plugins.helpers.entities.**PlayerEntity**
Bases: *spockbot.plugins.helpers.entities.MovementEntity*

**current_item = 0**

**metadata = None**

**uuid = 0**

**spockbot.plugins.helpers.interact module** Interact with the world: - swing the arm, sneak, sprint, jump with a horse, leave the bed - look around - dig/place/use blocks - use the held (active) item - use/attack entities - steer vehicles - place and write signs - edit and sign books

By default, the client sends swing and look packets like the vanilla client. This can be disabled by setting the `auto_swing` and `auto_look` flags.

**class** spockbot.plugins.helpers.interact.**InteractPlugin**(*ploader*, *settings*)
Bases: *spockbot.plugins.base.PluginBase*

**activate_item**()
Use (hold right-click) the item in the active slot. Examples: pull the bow, start eating once, throw an egg.

**attack_entity**(*entity*)

**cancel_digging**()

**click_block**(*pos*, *look_at_block=True*, *swing=True*, ***kwargs*)
Click on a block. Examples: push button, open window, make redstone ore glow

**Parameters**

- **face** (*int*) – side of the block on which the block is placed on
- **cursor_pos** (Vector3) – where to click inside the block, each dimension 0-15

**deactivate_item**()
Stop using (release right-click) the item in the active slot. Examples: shoot the bow, stop eating.

**dig_block**(*pos*)
Not cancelable.

**edit_book**(*pages*)
Set the pages of current book in hand

**finish_digging**()

**jump_horse**(*jump_boost=100*)

**jump_vehicle**()

**leave_bed**()

**look**(*yaw=0.0*, *pitch=0.0*, *radians=False*)

**look_at**(*pos*)

**look_at_rel**(*delta*)

**look_rel**(*d_yaw=0.0*, *d_pitch=0.0*, *radians=False*)

**mount_vehicle**(*entity*)

**open_inventory**()

**pl_announce = ('Interact',)**

**place_block**(*pos*, *sneak=True*, ***kwargs*)
Place a block next to `pos`. If the block at `pos` is air, place at `pos`.

**place_sign**(*pos*, *lines=[]*, ***place_block_kwargs*)
Place a sign block and write on it.

**requires = ('ClientInfo', 'Event', 'Inventory', 'Net', 'Channels')**

**sign_book**(*author*, *title*)
Sign current book in hand

**sneak**(*sneak=True*)

**sprint**(*sprint=True*)

**start_digging**(*pos*, *face=1*)

**steer_vehicle**(*left=0.0*, *forward=0.0*, *jump=False*, *unmount=False*)

**swing_arm**()

**unmount_vehicle**()

**unsneak**()

**unsprint**()

**use_bucket**(*pos*)
Using buckets is different from placing blocks. See "Special note on using buckets" in http://wiki.vg/Protocol#Player_Block_Placement

**use_entity**(*entity*, *cursor_pos=None*, *action=0*)
Uses (right-click) an entity to open its window. Setting `cursor_pos` sets `action` to "interact at".

**write_book**(*text*, *author=''*, *title=''*, *sign=False*)
Write text to the current book in hand, optionally sign the book

**spockbot.plugins.helpers.inventory module** The Inventory plugin keeps track of the inventory and provides simple inventory analysis and manipulation.

**class** spockbot.plugins.helpers.inventory.**InventoryCore**(*net_plugin*, *send_click*)
Bases: `object`

Handles operations with the player inventory.

**active_slot**

**click_slot**(*slot*, *right=False*)
Left-click or right-click the slot.

**Parameters** **slot** (`Slot`) – The clicked slot. Can be `Slot` instance or integer. Set to `inventory.cursor_slot` for clicking outside the window.

**close_window**()

**creative_set_slot**(*slot_nr=None*, *slot_dict=None*, *slot=None*)

**drop_slot**(*slot=None*, *drop_stack=False*)
Drop one or all items of the slot.

Does not wait for confirmation from the server. If you want that, use a `Task` and `yield inventory.async.drop_slot()` instead.

If `slot` is None, drops the `cursor_slot` or, if that's empty, the currently held item (`active_slot`).

**Parameters** **slot** (*Optional[Slot]*) – The dropped slot. Can be None, integer, or `Slot` instance.

**Returns** The action ID of the click

**Return type** int

**find_slot**(*wanted*, *slots=None*)
Searches the given slots or, if not given, active hotbar slot, hotbar, inventory, open window in this order.

**Parameters** **wanted** – function(Slot) or Slot or itemID or (itemID, metadata)

**Returns** The first slot containing the item or None if not found.

**Return type** Optional[Slot]

**find_slots**(*wanted*, *slots=None*)
Yields all slots containing the item. Searches the given slots or, if not given, active hotbar slot, hotbar, inventory, open window in this order.

**Parameters** **wanted** – function(Slot) or Slot or itemID or (itemID, metadata)

**inv_slots_preferred**
List of all available inventory slots in the preferred search order. Does not include the additional slots from the open window.

1.active slot

2.remainder of the hotbar

3.remainder of the persistent inventory

**select_active_slot**(*slot_or_hotbar_index*)

**total_stored**(*wanted*, *slots=None*)
Calculates the total number of items of that type in the current window or given slot range.

**Parameters** **wanted** – function(Slot) or Slot or itemID or (itemID, metadata)

**class** spockbot.plugins.helpers.inventory.**InventoryPlugin**(*ploader*, *settings*)
Bases: *spockbot.plugins.base.PluginBase*

**emit_open_window**(*_)

**emit_set_slot**(*slot*)

**events = {'PLAY<Confirm Transaction': 'handle_confirm_transaction', 'PLAY>Close Window': 'handle_close_windov**

**handle_close_window**(*event*, *packet*)

**handle_confirm_transaction**(*event*, *packet*)

**handle_held_item_change**(*event*, *packet*)

**handle_open_window**(*event*, *packet*)

**handle_set_slot**(*event*, *packet*)

**handle_window_items**(*event*, *packet*)

**handle_window_prop**(*event*, *packet*)

**pl_announce = ('Inventory',)**

**requires = ('Event', 'Net', 'Timers')**

**send_click**(*click*)
Sends a click to the server if the previous click has been confirmed.

**Parameters** **click** (BaseClick) – The click to send.

**Returns** the click's action ID if the click could be sent, None if the previous click has not been received and confirmed yet.

**set_slot**(*window_id*, *slot_nr*, *slot_data*)

### spockbot.plugins.helpers.keepalive module

**spockbot.plugins.helpers.movement module** MovementPlugin provides a centralized plugin for controlling client movement so the client doesn't try to pull itself in a dozen directions.

**class** spockbot.plugins.helpers.movement.**MovementCore**(*plug*)
Bases: object

**current_path**

**current_target**

**final_target**

**is_moving**

**stop**()

**class** spockbot.plugins.helpers.movement.**MovementPlugin**(*ploader*, *settings*)
Bases: *spockbot.plugins.base.PluginBase*

**follow_path**(*_*, *__*)

**new_path**(**xyz*)

**path_cb**(*result*)

**pl_announce = ('Movement',)**

**requires = ('ClientInfo', 'Event', 'Net', 'Pathfinding', 'Physics')**

**spockbot.plugins.helpers.pathfinding module** Very rough asychronous pathfinding plugin Implements the Lazy Theta* pathfinding algorithm

**class** spockbot.plugins.helpers.pathfinding.**Path**(*start_node*, *end_node*)
Bases: object

**calc_f_val**(*node*)

**class** spockbot.plugins.helpers.pathfinding.**PathNode**(**xyz*)
Bases: *spockbot.vector.Vector3*

**set**(*parent=None*, *is_fall=False*, *is_jump=False*)

**class** spockbot.plugins.helpers.pathfinding.**PathfindingCore**(*start_path*)
Bases: object

**class** spockbot.plugins.helpers.pathfinding.**PathfindingPlugin**(*ploader*, *settings*)
Bases: *spockbot.plugins.base.PluginBase*

**build_list_from_node**(*node*)

**check_for_bbox**(*pos*)

**check_node**(*node*, *offset*, *node_list*, *walk_fall=True*, *jump=True*)

**do_job**(*_=None*, *__=None*)

**find_valid_nodes**(*node*)

**get_block**(*pos*)

**pathfind**(*path*)

**pl_announce = ('Pathfinding',)**

**raycast_bbox**(*start*, *end*)

**requires = ('Event', 'World', 'Physics', 'ClientInfo', 'Timers')**

**start_path**(*pos*, *target*, *scb*, *fcb=None*)

**spockbot.plugins.helpers.physics module** A Physics module built from clean-rooming the Notchian Minecraft client

Collision detection and resolution is done by a Separating Axis Theorem implementation for concave shapes decomposed into Axis-Aligned Bounding Boxes. This isn't totally equivalent to vanilla behavior, but it's faster and Close Enough^TM

AKA this file does Minecraft physics

**class** spockbot.plugins.helpers.physics.**PhysicsCore**(*pos*, *vec*, *abilities*)
Bases: object

**jump**()

**move_angle**(*angle*, *radians=False*)

**move_target**(*vector*)

**move_vector**(*vector*)

**sprint**()

**walk**()

**class** spockbot.plugins.helpers.physics.**PhysicsPlugin**(*ploader*, *settings*)
Bases: *spockbot.plugins.base.PluginBase*

**apply_accel**()

**apply_drag**()

**apply_vector**(*mtv*)

**client_tick**(*name*, *data*)

**events = {'client_mount': 'suspend_physics', 'client_position_update': 'skip_physics', 'physics_tick': 'physics_tick', 'cl**

**get_block_slip**()

**get_mtv**()

**physics_tick**(*_*, *__*)

**pl_announce = ('Physics',)**

**requires = ('Event', 'ClientInfo', 'Net', 'World')**

**resume_physics**(*_=None*, *__=None*)

**skip_physics**(*_=None*, *__=None*)

**suspend_physics**(*_=None*, *__=None*)

**spockbot.plugins.helpers.respawn module**

**spockbot.plugins.helpers.start module** This plugin creates a convenient start() method and attaches it directly to the client. More complex bots will likely want to create their own initialization plugin, so StartPlugin stays out of the way unless you call the start() method. However, the start() method is very convenient for demos and tutorials, and illustrates the basic steps for initializing a bot.

**class** spockbot.plugins.helpers.start.**StartPlugin**(*ploader*, *settings*)
Bases: *spockbot.plugins.base.PluginBase*

**defaults = {'port': 25565, 'password': None, 'username': 'Bot', 'host': 'localhost'}**

**events = {'event_start': 'start_session_and_connect'}**

**requires = ('Auth', 'Event', 'Net')**

**start**(*host=None*, *port=None*)

**start_session_and_connect**(*_*, *__*)

**spockbot.plugins.helpers.world module** Provides a very raw (but very fast) world map for use by plugins. Plugins interested in a more comprehensive world map view can use mcp.mapdata to interpret blocks and their metadata more comprehensively. Planned to provide light level interpretation based on sky light and time of day

**class** spockbot.plugins.helpers.world.**WorldData**(*dimension=0*)
Bases: *spockbot.plugins.tools.smpmap.Dimension*

**new_dimension**(*dimension*)

**reset**()

**update_time**(*data*)

**class** spockbot.plugins.helpers.world.**WorldPlugin**(*ploader*, *settings*)
Bases: *spockbot.plugins.base.PluginBase*

**events = {'net_disconnect': 'handle_disconnect', 'PLAY<Block Change': 'handle_block_change', 'PLAY<Time Update**

**handle_block_change**(*name*, *packet*)
Block Change - Update a single block

**handle_chunk_data**(*name*, *packet*)
Chunk Data - Update World state

**handle_disconnect**(*name*, *data*)

**handle_map_chunk_bulk**(*name*, *packet*)
Map Chunk Bulk - Update World state

**handle_multi_block_change**(*name*, *packet*)
Multi Block Change - Update multiple blocks

**handle_new_dimension**(*name*, *packet*)
Join Game/Respawn - New Dimension

**handle_time_update**(*name*, *packet*)
Time Update - Update World Time

**handle_update_block_entity**(*event*, *packet*)

**handle_update_sign**(*event*, *packet*)

**pl_announce = ('World',)**

**requires = 'Event'**

**Module contents**

**spockbot.plugins.tools package**

**Submodules**

**spockbot.plugins.tools.collision module**
**class** spockbot.plugins.tools.collision.**MTVTest**(*world*, *bbox*)
Bases: object

**block_collision**(*pos*)

**check_collision**(*pos*, *vector*)
spockbot.plugins.tools.collision.**center_position**(*pos*, *bbox*)

spockbot.plugins.tools.collision.**check_axis**(*axis*, *min_a*, *max_a*, *min_b*, *max_b*)

spockbot.plugins.tools.collision.**gen_block_set**(*block_pos*, *xr=(-1, 2)*, *yr=(0, 3)*, *zr=(-1, 2)*)

spockbot.plugins.tools.collision.**uncenter_position**(*pos*, *bbox*)

**spockbot.plugins.tools.event module** Used for unregistering event handlers

**spockbot.plugins.tools.inventory_async module** Asynchronous task wrappers for inventory

**class** spockbot.plugins.tools.inventory_async.**InventoryAsync**(*inventory*)
Bases: object

**click_slot**(*slot*, *right=False*)

**click_slots**(**slots*)

**creative_set_slot**(*slot_nr=None*, *slot_dict=None*, *slot=None*)

**drop_slot**(*slot=None*, *drop_stack=False*)

**hold_item**(*wanted*)

**move_to_inventory**(**slots*)

**move_to_window**(**slots*)

**store_or_drop**()
Stores the cursor item or drops it if the inventory is full. Tip: look directly up or down before calling this, so you can pick up the dropped item when the inventory frees up again.

**Returns** The slot used to store it, or None if dropped.

**Return type** *Slot*

**swap_slots**(*a*, *b*)

**transfer_slots**(*source_slots*, *target_slots*)

spockbot.plugins.tools.inventory_async.**unpack_slots_list**(*slots*)

**spockbot.plugins.tools.smpmap module** Used for storing map data

Chunks are packed in X, Z, Y order The array walks down X, every 16 elements you enter a new Z-level ex. [0] - [15] are X = 0-15, Z = 0, Y = 0 [16] - [31] are X = 0-15, Z = 1, Y = 0 and so on

Every 256 elements you enter a new Y-level ex. [0]-[255] are X = 0-15, Z = 0-15, Y = 0 [256]-[511] are X = 0-15, Z = 0-15, Y = 1 and so on

**class** spockbot.plugins.tools.smpmap.**BannerData**(*nbt*)
Bases: *spockbot.plugins.tools.smpmap.BlockEntityData*

**class** spockbot.plugins.tools.smpmap.**BeaconData**(*nbt*)
Bases: *spockbot.plugins.tools.smpmap.BlockEntityData*

**class** spockbot.plugins.tools.smpmap.**BiomeData**
Bases: *spockbot.plugins.tools.smpmap.ChunkData*

A 16x16 array stored in each ChunkColumn.

**data = None**

**get**(*x*, *z*)

**length = 256**

**set**(*x*, *z*, *d*)

**class** spockbot.plugins.tools.smpmap.**BlockEntityData**(*nbt*)
Bases: object

**class** spockbot.plugins.tools.smpmap.**Chunk**
Bases: object

**class** spockbot.plugins.tools.smpmap.**ChunkColumn**
Bases: object

**unpack**(*buff*, *mask*, *skylight=True*, *continuous=True*)

**class** spockbot.plugins.tools.smpmap.**ChunkData**
Bases: object

**data = None**

**fill**()

**get**(*x*, *y*, *z*)

**length = 4096**

**pack**()

**set**(*x*, *y*, *z*, *data*)

**ty = 'B'**

**unpack**(*buff*)

**class** spockbot.plugins.tools.smpmap.**ChunkDataNibble**
Bases: *spockbot.plugins.tools.smpmap.ChunkData*

A 16x16x8 array for storing metadata, light or add. Each array element contains two 4-bit elements.

**get**(*x*, *y*, *z*)

**length = 2048**

**set**(*x*, *y*, *z*, *data*)

**class** spockbot.plugins.tools.smpmap.**ChunkDataShort**
Bases: *spockbot.plugins.tools.smpmap.ChunkData*

A 16x16x16 array for storing block IDs/Metadata.

**length = 8192**

**ty = 'H'**

**class** spockbot.plugins.tools.smpmap.**CommandBlockData**(*nbt*)
Bases: *spockbot.plugins.tools.smpmap.BlockEntityData*

**class** spockbot.plugins.tools.smpmap.**Dimension**(*dimension*)
Bases: object

A bunch of ChunkColumns.

**get_biome**(*x*, *z*)

**get_block**(*pos_or_x*, *y=None*, *z=None*)

**get_block_entity_data**(*pos_or_x*, *y=None*, *z=None*)
Access block entity data.

**Returns** BlockEntityData subclass instance or None if no block entity data is stored for that location.

**get_light**(*pos_or_x*, *y=None*, *z=None*)

**set_biome**(*x*, *z*, *data*)

**set_block**(*pos_or_x*, *y=None*, *z=None*, *block_id=None*, *meta=None*, *data=None*)

**set_block_entity_data**(*pos_or_x*, *y=None*, *z=None*, *data=None*)
Update block entity data.

**Returns** Old data if block entity data was already stored for that location, None otherwise.

**set_light**(*pos_or_x*, *y=None*, *z=None*, *light_block=None*, *light_sky=None*)

**unpack_bulk**(*data*)

**unpack_column**(*data*)

**class** spockbot.plugins.tools.smpmap.**FlowerPotData**(*nbt*)
Bases: *spockbot.plugins.tools.smpmap.BlockEntityData*

**class** spockbot.plugins.tools.smpmap.**HeadData**(*nbt*)
Bases: *spockbot.plugins.tools.smpmap.BlockEntityData*

**class** spockbot.plugins.tools.smpmap.**SignData**(*line_data*)
Bases: *spockbot.plugins.tools.smpmap.BlockEntityData*

**class** spockbot.plugins.tools.smpmap.**SpawnerData**(*nbt*)
Bases: *spockbot.plugins.tools.smpmap.BlockEntityData*

spockbot.plugins.tools.smpmap.**mapshort2id**(*data*)

### spockbot.plugins.tools.task module

**class** spockbot.plugins.tools.task.**Task**(*task*, *parent=None*, *name=None*)
Bases: object

**continue_with**(*func*)

**on_error**(*exception*)

**on_event**(*event*, *data*)

**on_success**(*data*)

**parse_response**(*response*)

**register**(*response*)

**run**(*task_manager*)

**tasktrace**
List of all parent tasks up to this one.

**Returns** List[Task]

**class** spockbot.plugins.tools.task.**TaskCallback**(*cb=None*, *eb=None*)
Bases: object

**on_error**(*error*)

**on_success**(*data*)

**exception** spockbot.plugins.tools.task.**TaskFailed**(*message*, **args*)
Bases: Exception

Raising this exception in any task stops it and signalizes the parent task that the task was aborted due to an error.

**message**
*str*

Description of the failure

**tasktrace**
*List[Task]*

List of all failed tasks since raising this error.

**prev_error**
*TaskFailed*

The previous error, if any. Provide via `with_error()`.

**__str__**()
Newline-separated text with all failed tasks and all previous errors.

**full_tasktrace**
List of all failed tasks caused by this and all previous errors.

**Returns** List[Task]

**with_error**(*prev_error*)
Sets the previous error and returns self.

When re-throwing a TaskFailed, you can provide a new, more high level failure description and pass along the previously failed tasks to still be able to reconstruct the full history of failed tasks.

#### Examples

Re-throw a TaskFailed with a new, more high level description.

```
>>> try:
...     raise TaskFailed('Low level', {'some': 1}, 'args')
... except TaskFailed as prev_err:
...     raise TaskFailed('High level').with_error(prev_err)
```

**Returns** TaskFailed

spockbot.plugins.tools.task.**accept**(*evt*, *data*)

spockbot.plugins.tools.task.**check_key**(*key*, *value*)
Generates a check function for a certain key-value pair.

Creates and returns a function that takes two arguments `(event, data)` and checks `data[key]` and `value` for equality.

This is supposed to be used as a check function generator for the `yield` statements in tasks.

#### Example

Wait for the next `player_join` event that has its `name` set to `Bob`, i.e. `data = {'name': 'Bob'}`.

```
>>> def my_task():
...     yield 'player_join', check_key('name', 'Bob')
```

### Module contents

### Submodules

**spockbot.plugins.base module**

**class** spockbot.plugins.base.**PluginBase**(*ploader*, *settings*)
Bases: object

A base class for cleaner plugin code.

Extending from PluginBase allows you to declare any requirements, default settings, and event listeners in a declarative way. Define the appropriate attributes on your subclass and enjoy cleaner code.

**defaults** = {}

**events** = {}

**requires** = ()

spockbot.plugins.base.**get_settings**(*defaults*, *settings*)

spockbot.plugins.base.**pl_announce**(**args*)

spockbot.plugins.base.**pl_event**(**args*)

**spockbot.plugins.loader module** Provides reasonably not-awful plugin loading

**class** spockbot.plugins.loader.**PluginLoader**(***kwargs*)
Bases: object

**provides**(*ident*, *obj*)

**requires**(*ident*, *hard=True*, *warning=None*)

**Module contents**

## Submodules

### spockbot.vector module

**class** spockbot.vector.**BaseVector**(**values*)
Bases: object

**init**(**args*)

**class** spockbot.vector.**CartesianVector**(**values*)
Bases: *spockbot.vector.BaseVector*

**ceil**()

**dist**(*other=None*)

**dist_cubic**(*other=None*)
Manhattan distance

**dist_sq**(*other=None*)
For fast length comparison

**dot_product**(*other*)

**floor**()

**iadd**(*other*)

**iceil**()

**idiv**(*other*)

**ifloor**()

**imul**(*other*)

**isub**(*other*)

**itruediv**(*other*)

**norm**()

**trunc**()

**zero**()

**class** spockbot.vector.**Vector3**(**xyz*)
Bases: *spockbot.vector.CartesianVector*

**get_dict**()

**init**(**args*)

**set_dict**(*data*)

**x**

**y**

**yaw_pitch**
Calculate the yaw and pitch of this vector

**z**

**class** spockbot.vector.**YawPitch**(**args*)
Bases: *spockbot.vector.BaseVector*

Store the yaw and pitch (in degrees)

**init**(**args*)

**pitch**
Pitch in degrees

**rpitch**
Pitch in radians

**ryaw**
Yaw in radians

**unit_vector**()
Generate a unit vector (norm = 1)

**yaw**
Yaw in degrees

### Module contents

## 1.5 Contributing

### 1.5.1 Fork, Clone

Fork the SpockBot repository, then clone your fork locally.

### 1.5.2 Setup

#### Configure remote

```
$ cd SpockBot
$ git remote add upstream git://github.com/SpockBotMC/SpockBot
```

#### Install development dependencies

```
$ pip3 install -r dev-requirements.txt
```

#### Install extra test interpreters

Installing the extra interpreters allows tox to test against multiple versions of python.

```
$ sudo apt-get install python2.7 python3.3 python3.4
```

### 1.5.3 Development

#### Create development branch

**Do not** work in the `master` branch, create a bug or feature branch and work from there. `master` should always be a mirror of upstream master.

```
$ git checkout -b feature-tacocat
```

Please use a more descriptive branch name than `feature-tacocat`

#### Hack away at the code

Have fun!

#### Docstrings

We use Google Style Python Docstrings, an example can be found here

#### Test your changes

Running `tox` will test that your changes pass all tests and follows the pep8 standard.

```
$ tox
```

#### Pull Request

If `tox` comes back with a success you are ready for a pull request. Commit your changes to your feature branch, push them to your fork and create a pull request.

### 1.5.4 Testing

We highly encourage writing tests, we use the pytest framework and you can run `tox` to test your newly written tests.

## 1.6 Authors

The following persons have contributed to SpockBot. The list is in the order of first contribution. For details on who contributed what, please refer to our Git repository.

- Nicholas Gamberini <nick@gamberini.email>
- Morgan Creekmore <morgan@morgancreekmore.me>
- gjum <code.gjum@gmail.com>
- Jasper Bok <hello@jasperbok.nl>
- Javier Domingo <javierdo1@gmail.com>
- Joshua Aune <luken@omner.org>

## 1.7 License

CHAPTER 2

# Indices and tables

- genindex
- modindex

## m



## p



## s



## v

# Symbols



# A



# B



# C

## D

## E



## F

## G



## H

## I



## J



## K



## L

## M



## N



## O

## P

## R



## S

## T



## U

## V



## W



## X



## Y



## Z