# Vyper-Bot Documentation

***Release 3.2***

**Josh Bacon**

**Dec 19, 2018**

# Contents

# CHAPTER 1

# Api Documentation

**class API**

**Vyper Methods**

**configure**(*token*, *functions={}*, *debug=False*)

Configures the bot with the token and functions to run on updates.

**Parameters**

- **token** (*str*) – The token of the bot retrieved from the BotFather
- **functions** (*dict*) – A dictionary that has the message types and functions
- **debug** (*boolean*) – An experimental value that doesn't have a use yet, but will in later versions

**Raises** **ValueError** – if token is blank or not a string

**request**(*endpoint*, *parameters=None*, *file=None*)

Requests the endpoint from the telegram API. Shouldn't need to be called as the other functions run this more effectively.

**Parameters**

- **endpoint** (*str*) – The endpoint to be requested
- **parameters** (*dict*) – The parameters attached to the request
- **file** (*file*) – A file that is sent with the request

**Returns** Value retrieved from API, usually updates or a success dictionary

**Return type** `dict`

Valid parameter keys for `parameters`: `message` `edited_message` `channel_post` `edited_channel_post` `inline_query` `chosen_inline_result` `callback_query` `shipping_query` `pre_checkout_query`

**Telegram Commands**

**getMe**(*self*)
Retrieves the user object for the bot.

**Returns** The bot

**Return type** User Object

**getUpdates**(*self*)
Gets the updates that are pending an update. Will store the current update value in the file `lastupdate.vyper`.

**Returns** Doesn't return a value, but instead runs the corresponding function

**sendMessage**(*chat_id*, *text*, *parse_mode=None*, *disable_web_page_preview=False*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*)
Sends a message to the chat specified.

**Parameters**

- **chat_id** (*int*) – Chat id of the target chat
- **text** (*str*) – Text to send in the message
- **parse_mode** (*str*) – *Markdown* or *HTML*, depending on which formatting style is preferred
- **disable_web_page_preview** (*boolean*) – Whether to show a preview of the webpage on the message
- **disable_notification** (*boolean*) – Whether to disable the notification for the message
- **reply_to_message_id** (*int*) – The message id to make the message reply to
- **reply_markup** (*dict*) – The custom markup applied to the message, such as inline keyboards

**forwardMessage**(*chat_id*, *from_chat_id*, *message_id*, *disable_notification=False*)
Forwards a message from one chat to another.

**Parameters**

- **chat_id** (*int*) – Chat id of the target chat
- **from_chat_id** (*int*) – Chat id of the original chat
- **message_id** (*int*) – Message id of the message from the original chat
- **disable_notification** (*boolean*) – Whether to disable the notification for the message

**sendPhoto**(*chat_id*, *photo*, *caption=''*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*)
Sends a photo to the chat specified.

**Parameters**

- **chat_id** (*int*) – Chat id of the target chat
- **photo** (*file*) – Photo to send
- **caption** (*str*) – The text displayed with the photo
- **disable_notification** (*boolean*) – Whether to disable the notification for the message
- **reply_to_message_id** (*int*) – The message id to make the message reply to

- **`reply_markup`** (*`dict`*) – The custom markup applied to the message, such as inline keyboards

**`sendAudio`**(*chat_id*, *audio*, *caption=”*, *duration=None*, *performer=”*, *title=”*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*)
Sends an audio track to the chat specified.

**Parameters**

- **`chat_id`** (*`int`*) – Chat id of the target chat
- **`audio`** (*`file`*) – Audio track to send
- **`caption`** (*`str`*) – The text displayed with the audio
- **`duration`** (*`int`*) – The duration of the track
- **`performer`** (*`str`*) – Performer of the track
- **`title`** (*`str`*) – Title of the track
- **`disable_notification`** (*`boolean`*) – Whether to disable the notification for the message
- **`reply_to_message_id`** (*`int`*) – The message id to make the message reply to
- **`reply_markup`** (*`dict`*) – The custom markup applied to the message, such as inline keyboards

**`sendDocument`**(*chat_id*, *document*, *caption=”*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*)
Sends a document to the chat specified.

**Parameters**

- **`chat_id`** (*`int`*) – Chat id of the target chat
- **`document`** (*`file`*) – Audio track to send
- **`caption`** (*`str`*) – The text displayed with the audio
- **`disable_notification`** (*`boolean`*) – Whether to disable the notification for the message
- **`reply_to_message_id`** (*`int`*) – The message id to make the message reply to
- **`reply_markup`** (*`dict`*) – The custom markup applied to the message, such as inline keyboards

**`sendVideo`**(*self*, *chat_id*, *video*, *duration=None*, *width=None*, *height=None*, *caption=”*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*)
Sends a video to the chat specified.

**Parameters**

- **`chat_id`** (*`int`*) – Chat id of the target chat
- **`audio`** (*`file`*) – Audio track to send
- **`duration`** (*`int`*) – The duration of the video
- **`width`** (*`int`*) – Width of the video
- **`height`** (*`int`*) – Height of the video
- **`caption`** (*`str`*) – The text displayed with the video

- **disable_notification** (*boolean*) – Whether to disable the notification for the message
- **reply_to_message_id** (*int*) – The message id to make the message reply to
- **reply_markup** (*dict*) – The custom markup applied to the message, such as inline keyboards

**sendVoice** (*self*, *chat_id*, *voice*, *caption=None*, *duration=None*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*)
Sends a voice message to the chat specified.

**Parameters**

- **chat_id** (*int*) – Chat id of the target chat
- **voice** (*file*) – Audio to send (.ogg)
- **caption** (*str*) – The text displayed with the audio
- **duration** (*int*) – The duration of the message
- **disable_notification** (*boolean*) – Whether to disable the notification for the message
- **reply_to_message_id** (*int*) – The message id to make the message reply to
- **reply_markup** (*dict*) – The custom markup applied to the message, such as inline keyboards

**sendVideoNote** (*self*, *chat_id*, *video_note*, *length=None*, *duration=None*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*)
Sends a video note to the chat specified.

**Parameters**

- **chat_id** (*int*) – Chat id of the target chat
- **video_note** (*file*) – Video to send
- **duration** (*int*) – The duration of the track
- **length** (*int*) – Video width and height
- **disable_notification** (*boolean*) – Whether to disable the notification for the message
- **reply_to_message_id** (*int*) – The message id to make the message reply to
- **reply_markup** (*dict*) – The custom markup applied to the message, such as inline keyboards

**sendLocation** (*chat_id*, *latitude*, *longitude*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*)
Sends a location to the chat specified.

**Parameters**

- **chat_id** (*int*) – Chat id of the target chat
- **latitude** (*float*) – Latitude of the location
- **longitude** (*float*) – Longitude of the location
- **disable_notification** (*boolean*) – Whether to disable the notification for the message
- **reply_to_message_id** (*int*) – The message id to make the message reply to

- **reply_markup** (*dict*) – The custom markup applied to the message, such as inline keyboards

**sendVenue** (*chat_id*, *latitude*, *longitude*, *title*, *address*, *foursquare_id=”*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*)
Sends a venue to the chat specified.

**Parameters**

- **chat_id** (*int*) – Chat id of the target chat
- **latitude** (*float*) – Latitude of the venue
- **longitude** (*float*) – Longitude of the venue
- **title** (*str*) – Name of the venue
- **address** (*str*) – Address of the venue
- **foursquare_id** (*str*) – Foursquare id of the venue
- **disable_notification** (*boolean*) – Whether to disable the notification for the message
- **reply_to_message_id** (*int*) – The message id to make the message reply to
- **reply_markup** (*dict*) – The custom markup applied to the message, such as inline keyboards

**sendContact** (*chat_id*, *phone_number*, *first_name*, *last_name=”*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*)
Sends a contact to the chat specified.

**Parameters**

- **chat_id** (*int*) – Chat id of the target chat
- **phone_number** (*str*) – Latitude of the contact
- **first_name** (*str*) – Longitude of the contact
- **last_name** (*str*) – Name of the contact
- **disable_notification** (*boolean*) – Whether to disable the notification for the message
- **reply_to_message_id** (*int*) – The message id to make the message reply to
- **reply_markup** (*dict*) – The custom markup applied to the message, such as inline keyboards

**sendChatAction** (*chat_id*, *action*)
Sends the bot’s current status to the chat.

**Parameters**

- **chat_id** (*int*) – Chat id of the target chat
- **action** (*ChatAction*) – Type of action to broadcast

**getUserProfilePhotos** (*user_id*, *offset=None*, *limit=None*)
Returns an array of profile photos from the target user

**Parameters**

- **user_id** (*int*) – The user id of the target user
- **offset** (*int*) – The first photo to be returned

- **limit** (*int*) – The maximum number of photos to be retrieved

**getFile** (*file_id*)
Gets simple file information to be downloaded from `https://api.telegram.org/file/bot<token>/<file_path>`.

**Parameters** **file_id** (*str*) – File identifier

**Administrative Commands**

**kickChatMember** (*chat_id*, *user_id*, *until_date=0*)
Kicks a chat member until the date specified or until unbanned.

**Parameters**

- **chat_id** (*int*) – The id of the target chat
- **user_id** (*int*) – The id of the target user
- **until_date** (*int*) – The date in unix time that the user will be unbanned

**unbanChatMember** (*chat_id*, *user_id*)
Unbans a chat member from a chat.

**Parameters**

- **chat_id** (*int*) – The id of the target chat
- **user_id** (*int*) – The id of the target user

**restrictChatMember** (*chat_id*, *user_id*, *until_date=0*, *can_send_messages=True*, *can_send_media_messages=True*, *can_send_other_messages=True*, *can_add_web_page_previews=True*)
Restricts a chat member's permissions in a chat.

**Parameters**

- **chat_id** (*int*) – The id of the target chat
- **user_id** (*int*) – The id of the target user
- **until_date** (*int*) – The date in unix time that the user will be unbanned
- **can_send_messages** (*boolean*) – Whether a user can send messages
- **can_send_media_messages** (*boolean*) – Whether a user can send media messages
- **can_send_other_messages** (*boolean*) – Whether a user can send other messages
- **can_add_web_page_previews** (*boolean*) – Whether a user can create web page previews

**promoteChatMember** (*chat_id*, *user_id*, *can_change_info=False*, *can_post_messages=False*, *can_edit_messages=False*, *can_delete_messages=False*, *can_invite_users=False*, *can_restrict_members=False*, *can_pin_messages=False*, *can_promote_members=False*)
Promotes a chat member

**Parameters**

- **chat_id** (*int*) – The id of the target chat
- **user_id** (*int*) – The id of the target user
- **can_change_info** (*boolean*) – Whether a user can change group info
- **can_post_messages** (*boolean*) – Whether a user can make channel posts

- **`can_edit_messages`** (*`boolean`*) – Whether a user can edit other messages in a channel
- **`can_delete_messages`** (*`boolean`*) – Whether a user can delete other users' messages
- **`can_invite_users`** (*`boolean`*) – Whether a user can invite members to the group
- **`can_restrict_members`** (*`boolean`*) – Whether a user can restrict members in the group
- **`can_pin_messages`** (*`boolean`*) – Whether a user can pin messages
- **`can_promote_members`** (*`boolean`*) – Whether a user can promote users

**`leaveChat`**(*chat_id*)

Makes the bot leave the target chat

**Parameters** **`chat_id`** (*`int`*) – The id of the target chat

**`getChat`**(*chat_id*)

Returns information on the target chat

**Parameters** **`chat_id`** (*`int`*) – The id of the target chat

**`getChatAdministrators`**(*chat_id*)

Returns list of administrators in the target chat

**Parameters** **`chat_id`** (*`int`*) – The id of the target chat

**`getChatMembersCount`**(*chat_id*)

Returns number of members in the target chat

**Parameters** **`chat_id`** (*`int`*) – The id of the target chat

**`getChatMember`**(*chat_id*, *user_id*)

Returns information on the target chat member

**Parameters**

- **`chat_id`** (*`int`*) – The id of the target chat
- **`user_id`** (*`int`*) – The id of the target user

**`answerCallbackQuery`**(*callback_query_id*, *text=''*, *show_alert=False*, *url=''*, *cache_time=None*)

Sends an answer to a callback query from an inline keyboard.

**Parameters**

- **`callback_query_id`** (*`str`*) – Callback query id
- **`text`** (*`str`*) – Text for the notification
- **`show_alert`** (*`boolean`*) – Shows an alert instead of a notification
- **`url`** (*`str`*) – Url of game or to open bot with parameter
- **`cache_time`** (*`str`*) – Time to cache the query on the client

**`editMessageText`**(*text*, *chat_id=None*, *message_id=None*, *inline_message_id=None*, *parse_mode=None*, *disable_web_page_preview=False*, *reply_markup=None*)

Edits a message from a chat.

**Parameters**

- **`text`** (*`str`*) – New text for the message
- **`chat_id`** (*`int`*) – Chat id of the target chat

- **message_id** (*int*) – Message id in the target chat
- **inline_message_id** (*str*) – Inline message id in the target chat
- **parse_mode** (*str*) – *Markdown* or *HTML*, depending on which formatting style is preferred
- **disable_notification** (*boolean*) – Whether to disable the notification for the message
- **reply_markup** (*dict*) – The custom markup applied to the message, such as inline keyboards

**editMessageCaption** (*chat_id=None*, *message_id=None*, *inline_message_id=None*, *caption=None*, *reply_markup=None*)

Edits a caption from a chat.

**Parameters**

- **chat_id** (*int*) – Chat id of the target chat
- **message_id** (*int*) – Message id in the target chat
- **inline_message_id** (*str*) – Inline message id in the target chat
- **parse_mode** (*str*) – *Markdown* or *HTML*, depending on which formatting style is preferred
- **reply_markup** (*dict*) – The custom markup applied to the message, such as inline keyboards

**editMessageReplyMarkup** (*chat_id=None*, *message_id=None*, *inline_message_id=None*, *reply_markup=None*)

Edits the reply markup on a message.

**Parameters**

- **chat_id** (*int*) – Chat id of the target chat
- **message_id** (*int*) – Message id in the target chat
- **inline_message_id** (*str*) – Inline message id in the target chat
- **reply_markup** (*dict*) – The custom markup applied to the message, such as inline keyboards

**deleteMessage** (*chat_id*, *message_id*)

Deletes a message from a chat.

**Parameters**

- **chat_id** (*int*) – Chat id of the target chat
- **message_id** (*int*) – Message id in the target chat

**answerInlineQuery** (*inline_query_id*, *results*, *cache_time=None*, *is_personal=False*, *next_offset="*, *switch_pm_text="*, *switch_pm_parameter="*)

Answers an inline query

**Parameters**

- **inline_query_id** (*str*) – The id of the inline query
- **results** – The results to send to the user
- **cache_time** (*integer*) – The time to cache the results on the server
- **is_personal** (*boolean*) – Should results be cached server side only for that user

- **next_offset** (*str*) – Offset a client should send in the next query to recieve more results
- **switch_pm_text** (*str*) – Clients display button with specified text that switches to private chat
- **switch_pm_parameter** (*str*) – The parameter for the /start message sent when the button is pressed

**sendInvoice**(*chat_id*, *title*, *description*, *payload*, *provider_token*, *start_parameter*, *currency*, *prices*, *photo_url=”*, *photo_size=None*, *photo_height=None*, *photo_width=None*, *need_name=False*, *need_phone_number=False*, *need_email=False*, *need_shipping_address=False*, *is_flexible=False*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*)

Sends an invoice to the user.

**Parameters**

- **chat_id** (*int*) – Private chat id
- **title** (*str*) – Product name
- **description** (*str*) – Product description
- **payload** (*str*) – Invoice payload
- **provider_token** (*str*) – Payment token from BotFather
- **start_parameter** (*str*) – Deep linking parameter when used as a /start parameter
- **currency** (*str*) – Three letter currency code
- **prices** (*list*) – Array of prices
- **photo_url** (*str*) – Product photo
- **photo_size** (*int*) – Photo size
- **photo_width** (*int*) – Photo width
- **photo_height** (*int*) – Photo height
- **need_name** (*boolean*) – Needs full name to complete order
- **need_phone_number** (*boolean*) – Needs phone number to complete order
- **need_email** (*boolean*) – Needs email to complete order
- **need_shipping_address** (*boolean*) – Needs shipping address to complete order
- **is_flexible** (*boolean*) – Final price depends on shipping method
- **disable_notification** (*boolean*) – Disable the notification
- **reply_to_message_id** (*int*) – Message id to reply to
- **reply_markup** (*dict*) – The inline keyboard applied to the message

**answerShippingQuery**(*shipping_query_id*, *ok*, *shipping_options=None*, *error_message=”*)

If `is_flexible` and `need_shipping_address` are in the invoice, sends an update.

**Parameters**

- **shipping_query_id** (*str*) – The shipping query id
- **ok** (*boolean*) – Is the address ok
- **shipping_options** (*list*) – Sends the shipping options

- **`error_message`** (*`str`*) – The error message to send to the user as a reason for the shipping to fail.

**`answerPreCheckoutQuery`**(*pre_checkout_query_id*, *ok*, *error_message=''*)
After shipping and payment details are confirmed, send a confirmation.

**Parameters**

- **`pre_checkout_query_id`** (*`str`*) – The precheckout query id
- **`ok`** (*`boolean`*) – Is the order ok
- **`error_message`** (*`str`*) – The error message to send to the user as a reason for the order to fail.

**`sendGame`**(*chat_id*, *game_short_name*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*)
Sends a game to users in a chat.

**Parameters**

- **`chat_id`** (*`int`*) – The id of the target chat
- **`game_short_name`** (*`str`*) – The id of the target chat
- **`disable_notification`** (*`boolean`*) – Disable the notification
- **`reply_to_message_id`** (*`int`*) – Message id to reply to
- **`reply_markup`** (*`dict`*) – The inline keyboard applied to the message

**`exportChatInviteLink`**(*chat_id*)
Exports an invite link

**Parameters** **`chat_id`** (*`int`*) – The id of the target chat

**`setChatPhoto`**(*chat_id*, *photo*)
Sets the chat photo

**Parameters**

- **`chat_id`** (*`int`*) – The id of the target chat
- **`photo`** (*`file`*) – The photo to set as the chat photo

**`deleteChatPhoto`**(*chat_id*)
Deletes a chat photo

**Parameters** **`chat_id`** (*`int`*) – The id of the target chat

**`setChatTitle`**(*chat_id*, *title*)
Sets the chat title

**Parameters**

- **`chat_id`** (*`int`*) – The id of the target chat
- **`title`** (*`str`*) – The text to set as the chat title

**`setChatDescription`**(*chat_id*, *description*)
Sets the chat description

**Parameters**

- **`chat_id`** (*`int`*) – The id of the target chat
- **`description`** (*`str`*) – The text to set as the chat description

**pinChatMessage** (*chat_id*, *message_id*, *disable_notification=False*)
Sets the chat description

**Parameters**

- **chat_id** (*int*) – The id of the target chat
- **message_id** (*int*) – The id of the target message
- **disable_notification** (*boolean*) – Disable the notification

**unpinChatMessage** (*chat_id*)
Unpins the message in the target chat

**Parameters** **chat_id** (*int*) – The id of the target chat

**getStickerSet** (*name*)
Returns the sticker set with the name specified

**Parameters** **name** (*str*) – The name of the sticker set

**sendSticker** (*chat_id*, *sticker*, *disable_notification=False*, *reply_to_message_id=None*, *reply_markup=None*)

**uploadStickerFile** (*self*)
Uploads a new sticker

**Parameters**

- **user_id** (*int*) – The user id of the sticker owner
- **png_sticker** (*file*) – The png file with at least one side of 512px

**createNewStickerSet** (*user_id*, *name*, *title*, *png_sticker*, *emojis*, *contains_masks=False*, *mask_position=None*)
Uploads a new sticker

**Parameters**

- **user_id** (*int*) – The user id of the sticker owner
- **name** (*str*) – The short name of the sticker set
- **title** (*str*) – The title of the sticker set
- **png_sticker** (*file*) – The png file with at least one side of 512px
- **emojis** (*str*) – Emoji to correspond to a sticker
- **contains_masks** (*boolean*) – Set of mask stickers should be created
- **mask_position** (*dict*) – Position of the mask

**addStickerToSet** (*user_id*, *name*, *png_sticker*, *emojis*, *mask_position=None*)
Adds sticker to set

**Parameters**

- **user_id** (*int*) – The user id of the sticker owner
- **name** (*str*) – The short name of the sticker set
- **png_sticker** (*file*) – The png file with at least one side of 512px
- **emojis** (*str*) – Emoji to correspond to a sticker
- **mask_position** (*dict*) – Position of the mask

**`setStickerPositionInSet`**(*sticker*, *position*)
Moves sticker to position in the set

**Parameters**

- **`sticker`** (*`str`*) – File id of sticker
- **`position`** (*`int`*) – New sticker position, zero-based

**`deleteStickerFromSet`**(*sticker*)
Deletes sticker from set.

**Parameters** **`sticker`** (*`str`*) – File id of sticker

**`class ChatAction`**(*Enum*)

**`TYPING`**

**`PHOTO`**

**`UVIDEO`**

**`RVIDEO`**

**`UAUDIO`**

**`RAUDIO`**

**`DOCUMENT`**

**`LOCATION`**

**`UVIDNOTE`**

**`RVIDNOTE`**

# CHAPTER 2

# Basebot

As of `version 3.3`, vyper-bot comes with template bots to make getting started easier. The most basic one being `basebot`. It provides a very simple framework for your bot.

## 2.1 Simple Setup

```
from vyper import basebot

class MyBot(basebot.Basebot):
    def message(self, msg):
        msg = msg['message']
        if msg['text'] == '/ping':
            self.sendMessage(msg['chat']['id'], 'PONG')

bot = MyBot('123456:ABC-DEF1234ghIkl-zyx57W2v1u123ew11', start_loop=True)
```

This is perfectly valid code, which will run the message function when a message is sent, and will reply with `PONG` if the message is `/ping`. Using this style of bot makes the setup slightly less tedious than it would be by using only the base API, as it does all the background work for you.

**class** `basebot.`**Basebot**(*token*, *debug=False*, *start_loop=False*, *loop_time=0.05*)

Inherits `vyper.API`, allowing you to use any of the API methods from within the bot.

**Members** start_message

**start_loop**(*loop_time*)

This method will provide an easier way to create the update loop.

**Parameters** **loop_time** (*float*) – The time in between cycles of the loop

**set_functions**(*functions*)

Sets the functions that will run when an update is received. Defaults to the functions that are setup already, which are designed to be overwritten.

**Parameters** **functions** (*dict*) – The dictionary of functions that will be run

**message**(*msg*)
Called when a message is received. Overwrite this in your child bot.

**edited_message**(*msg*)
Called when a message is edited. Overwrite this in your child bot.

**channel_post**(*msg*)
Called when a message is posted to a channel. Overwrite this in your child bot.

**edited_channel_post**(*msg*)
Called when a message is edited in a channel. Overwrite this in your child bot.

**inline_query**(*msg*)
Called when an inline query is received. Overwrite this in your child bot.

**chosen_inline_result**(*msg*)
Called when an inline query result is is chosen. Overwrite this in your child bot.

**callback_query**(*msg*)
Called when a callback is received. Overwrite this in your child bot.

**shipping_query**(*msg*)
Called when a shipping query is received. Overwrite this in your child bot.

**pre_checkout_query**(*msg*)
Called when a pre-checkout query is received. Overwrite this in your child bot.

CHAPTER 3

# Plugin Bots

If you want to make a bot using a plugin system, `PluginBot` is the way to go. Rather than having to develop a system to decide which command to use, you can just create a class that extends `Plugin` and it will be automatically detected. If you run a bot using the `PluginBot` base, it will automatically create a folder called plugins, which will detect any file and plugin within that folder. What this means is that you won't have to import any files and it will find them for you. I've found that this saves a lot of grief trying to figure out why a command won't be detected properly, as it should be detected no matter what.

```
from vyper import pluginbot

class TutorialBot(pluginbot.PluginBot):
    def message(self, msg):
        msg = msg['message']
                self.test_plugins(msg)

if __name__ == '__main__':
    bot = VyperBot('123456:ABC-DEF1234ghIkl-zyx57W2v1u123ew11', start_loop=True, list_
↪plugins=True)
```

```
from vyper import pluginbot

class Help(pluginbot.Plugin):
    def message(self, msg):
        if msg['text'] == '/help':
            self.bot.sendMessage(msg['chat']['id'], 'This is the help command')
```

**class** `pluginbot.`**`PluginBot`**(*token*, *debug=False*, *start_loop=False*, *loop_time=.05*, *ping=True*, *list_plugins=False*)

Inherits `pluginbot.BaseBot`, allowing you to use any of the API methods from within the bot.

**Members** start_message

**`start_loop`**(*loop_time*)

This method will provide an easier way to create the update loop.

**Parameters** **`loop_time`** (*`float`*) – The time in between cycles of the loop

**set_functions**(*functions*)
Sets the functions that will run when an update is received. Defaults to the functions that are setup already, which are designed to be overwritten.

**Parameters** **functions** (`dict`) – The dictionary of functions that will be run

**test_plugins**(*msg*)
Tests all the plugins and runs the `message()` method in them to scan them. This is bound to be changed in a future update to automatically test for a plugin based on a custom variable, but it doesn't do that yet.

**Parameters** **msg** – The message object that will be passed to the other end of the function. Can be a dictionary or a `types.Message` object, depending on personal preference. Note that the `types.Message` object must be created by calling `msg = types.build(msg)` and will return a dot operator seperated object.

**_get_plugins**()
Gets a list of all the plugins that are found in the bot. You shouldn't need to run this, but it gets run when testing the plugins automatically. Returns as a generator, so the plugins can be easily iterated over.

**Returns** The plugins

**Return type** Generator

**message**(*msg*)
Called when a message is received. Overwrite this in your child bot.

**edited_message**(*msg*)
Called when a message is edited. Overwrite this in your child bot.

**channel_post**(*msg*)
Called when a message is posted to a channel. Overwrite this in your child bot.

**edited_channel_post**(*msg*)
Called when a message is edited in a channel. Overwrite this in your child bot.

**inline_query**(*msg*)
Called when an inline query is received. Overwrite this in your child bot.

**chosen_inline_result**(*msg*)
Called when an inline query result is is chosen. Overwrite this in your child bot.

**callback_query**(*msg*)
Called when a callback is received. Overwrite this in your child bot.

**shipping_query**(*msg*)
Called when a shipping query is received. Overwrite this in your child bot.

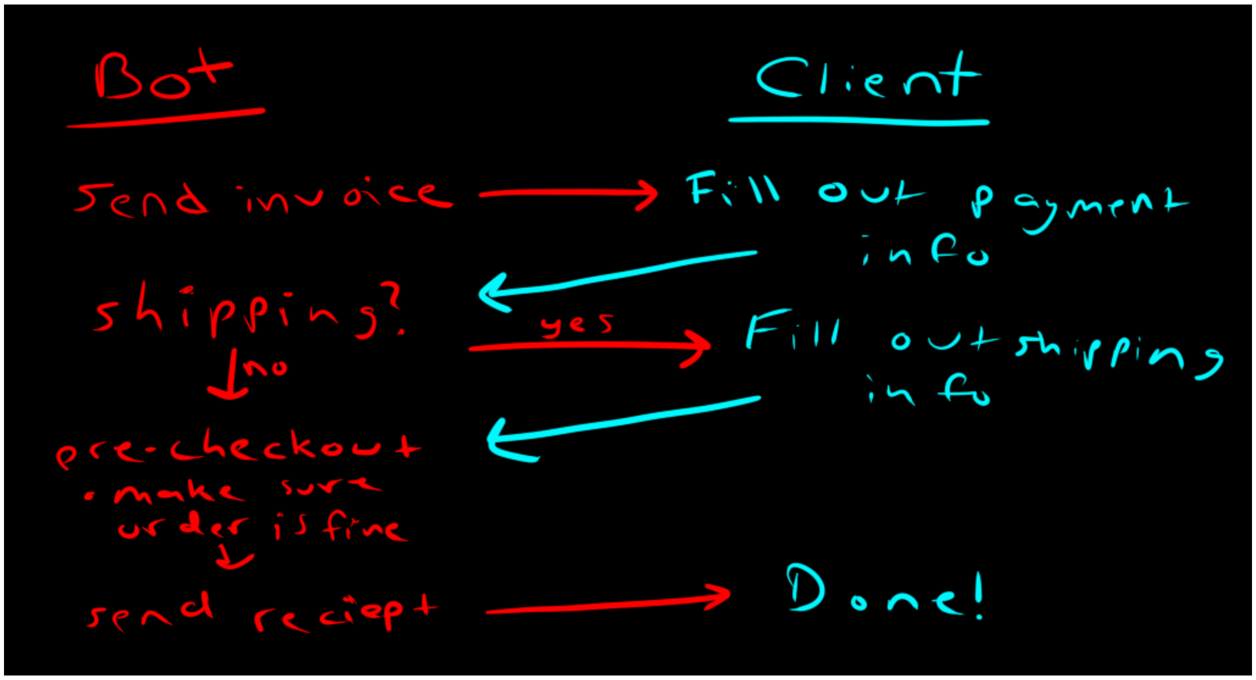
**pre_checkout_query**(*msg*)
Called when a pre-checkout query is received. Overwrite this in your child bot.

CHAPTER 4

# Payments

## 4.1 The Telegram Payment System

When first trying to figure out the payment system built into Telegram, it can be extremely daunting. My first time trying to learn how it worked wasn't extremely difficult, but still took some getting used to, as well as some custom functions to make life easier. After you figure it out for the first time though, it just clicks on how it works.



Above is the general idea of what has to happen to make a payment go through properly. It's really only 2 steps on the bot side of things if the product doesn't need delivery, so it can be super simple. Adding shipping information only requires one extra step, which is just confirming that the location can actually be shipped to (kind of important).

## 4.2 Vyper Payments

**Note:** If you don't have a payment provider code yet, make sure to head over to BotFather and get one, or none of this will work.

The payments in vyper work in the same order as in the Telegram system (seeing as it's what it runs on), but I've added a few extra functions to make things easier to keep track of. When using the default system, you need to define a "payload" for the order, which took me a while to realize is just a string that can be tied to the order that's being processed. This makes it easier to keep track of the payment, but can be confusing to come up with. If you use the `payments.Item` class, you don't have to worry about quite a bit of the information in the payment.

To make an item, all you need to do is create the item file, then pass the result of the invoice method into the sendInvoice() method. I'll be using an example for a pluginbot design.

**Important:** If you don't split the result of `invoice()` into the invoice and payload, everything will break. Also, make sure to deconstruct the invoice into all the parameters using an asterisk. `self.bot.sendInvoice(*invoice)` This is because it's returned as a tuple of all the values needed for the invoice, so if you don't break it down, it will break horribly.

```
stripe = 'STRIPE TEST CODE RETRIEVED FROM BOTFATHER'
item = payments.Item('Test Item', 'Test Description', stripe, prices=[payments.
↪LabeledPrice('Item', 500)])

class Pay(pluginbot.Plugin):
    def message(self, msg):
        if msg['text'] == '/pay':
            invoice, payload = item.invoice(msg)
            self.bot.sendInvoice(*invoice)
```

As you can see, the `payments.Item` format makes it so only 4 arguments are needed to send the item, as the rest can be generated easily. Normally, it would take 8 different arguments to send an invoice to the user, but I automatically fill the currency, start_parameter, payload, and chat_id for you. The start_parameter will just be the name of the item, with spaces stripped off of it, and the currency will be defaulted to USD, but can be changed in the constructor. For the payload, I generate a string using the user's id and the start parameter, as well as appending the unix timestamp to the end, creating a payload that looks similar to `123456789TestProduct1416667432`. This makes it both easy to identify, as well as unique for every user, so you won't have to worry about duplicates. The only way a duplicate could be created is if a user tried to send multiple orders in the exact same second, which is extremely unlikely.

# CHAPTER 5

# Version History

**Version 5.0.2 (November 6, 2018)**

- Starting work on API again
- Bug fixes

**Version 4.1 (September 12, 2017)**

- Added logging decorator to be added to the main class
- Added a few missing types
- Bug Fixes

**Version 4.0 (September 9, 2017)**

- Created version log
- Added payment API
- Added BaseBot and PluginBot, for easy startup
- Added types system, preparing for future, as well as allowing for message dictionaries to be converted into objects
- Added keyboard generator using excel, still buggy, would not recommend using unless testing.

**Previous Versions**

- Added support for every function available through the Telegram Bot API
- Created documentation to make getting started easier
- Updated to version 3.3 of the Bot API

# CHAPTER 6

# Installation

To install using pip, simply run

```
pip3 install vyper-bot
```

in a terminal. If you already have it installed but want to check for a new version, run

```
pip3 install vyper-bot --upgrade
```

**Note:** If you get an error saying that pip3 is not recognized as an internal or external command, try running `py -m -3 pip install vyper-bot`. This can occur if python isn't installed correctly or if it isn't added to the $PATH on windows.

CHAPTER 7

# Getting Started

When first starting to get your bot up and running, you must start by getting a bot key from the BotFather. This key can be retrieved by sending `/newbot` in a message to BotFather. After answering a few questions, you will be rewarded with a key that will look very similar to `123456:ABC-DEF1234ghIkl-zyx57W2v1u123ew11`.

The most simple program that you can make is one that only has an object representing the API. I prefer to name this object `bot` as this makes it easier to reference later when making calls.

```
from vyper import vyper

bot = vyper.API()
```

This API object is fully functional, but won't do anything for a few reasons. First of all, we aren't asking it to do anything, but also, the bot key isn't put into the API yet. All you have to do to make the key part of the bot is to take this line

```
bot = vyper.API()
```

and turn it into

```
bot = vyper.API().configure('botkey')
```

where `botkey` is your key that you got from BotFather.

**Important:** Make sure you change the botkey or your code WILL NOT WORK!

Now, you can run any of the methods from the Telegram API using the following code as an example.

```
# Sends a message to the chat id or chat username.
bot.sendMessage(chat_id, 'Test message')
```

This will send the message `Test Message` to the chat that you specify.

# CHAPTER 8

# Getting Updates

One of the biggest requirements of a telegram bot is the ability to get the messages that are sent to a group so that it can send a message back in response to commands. I've made the process of getting updates extremely simple, just modifying one of the existing lines. All that needs to be done is to create a method that will be run from a message being sent.

```
from vyper import vyper
import time

def on_message(msg):
    msg = msg['message']
    bot.sendMessage(msg['chat']['id'], 'I have recieved your message')

bot = vyper.API().configure('123456:ABC-DEF1234ghIkl-zyx57W2v1u123ew11', functions={
↪'message': on_message})

while True:
    bot.getUpdates()
    time.sleep(.05)
```

**Note:**

**You can use any of the following as a key in the functions dictionary:** `message` `edited_message` `channel_post` `edited_channel_post` `inline_query` `chosen_inline_result` `callback_query` `shipping_query` `pre_checkout_query`

# Index