PhpBotFramework Documentation

Release 3.0.0

Danilo Spinella, Dom Corvasce

Sep 23, 2017

Get started:

1 Overview **1**
- 1.1 Features 1
- 1.2 Requirements 1
- 1.3 Installation 2
- 1.4 Made with PhpBotFramework 2
- 1.5 Authors 2
- 1.6 License 2

2 Quickstart **5**
- 2.1 Create a bot 5
- 2.2 Answer messages 5
- 2.3 Answer updates 5
- 2.4 Answer the /start command 6
- 2.5 Getting updates 6

3 Database **7**
- 3.1 Connecting the database 7
- 3.2 Broadcast message 7

4 Logging **9**
- 4.1 BotName 9
- 4.2 Chat logging 9

5 Chat_id **11**
- 5.1 withChatId 11
- 5.2 useChatId 11

6 Payments **13**
- 6.1 What are Payments API? 13
- 6.2 Prepare the playground 13
- 6.3 Configure Payments credentials 13
- 6.4 Create an invoice 14
- 6.5 Shipping & Checkout 14

7 Commands **15**
- 7.1 MessageCommands 15
- 7.2 CallbackCommand 15

7.3 AdminCommand 16
7.4 MultiCharacterCommand 16

8 Create an Inline Keyboard **17**

9 Inline Queries **19**
9.1 Results 19
9.2 Sending the results 19
9.3 Types 20

10 Indices and tables **21**

CHAPTER 1

Overview

Features

- Modular
- Flexible HTTP requests with Guzzle
- Fast and easy to use
- Implements `getUpdates` and `webhook`
- Command-handle system for messages and callback queries
- Easy **inline keyboard** creation
- InlineQuery cretion
- SQL Database facilities
- Localization
- Save status for interactive chats
- Upload local files
- Logging

Requirements

1. Php 7.0 or greater
2. Composer

Webhook requirements

3. Webserver
4. SSL certificate

Localization requirements

5. SQL Database
6. Redis database (Optional)

Installation

In order to install PhpBotFramework you need to install composer:

```
curl -sS https://getcomposer.org/installer | php
```

Then set the framework as a requirement for the bot and install the dependencies:

```
php composer.phar require danyspin97/php-bot-framework
php composer.phar install --no-dev
```

Alternatively, you can set the depency in the `composer.json` of your bot:

```
{
   "require": {
      "danyspin97/php-bot-framework": "*"
   }
}
```

After installing it, you need to require the Composer's autoloader:

```
require 'vendor/autoload.php';
```

Made with PhpBotFramework

- MyAddressBookBot: Try it on Telegram
- Giveaways_Bot: Try it on Telegram

Authors

This framework is developed and mantained by Danilo Spinella and Dom Corvasce.

License

PhpBotFramework is released under the GNU Lesser General Public License v3.

CHAPTER 2

Quickstart

Create a bot

```
<?php
// Require the Composer's autoloader
require 'vendor/autoload.php';

// Create the bot object
$bot = new PhpBotFramework\Bot("token");
```

Answer messages

```
$bot->answerUpdate["message"] = function ($bot, $message) {

    // Reply as an echo bot
    $bot->sendMessage($message->getText());

};
```

Answer updates

As for messages, answering other updates requires the assignment of a function to `Bot::answerUpdate["entity"]`.

The function assigned must take 2 arguments:

`$bot` the bot object

`$entity` the entity attached to the update

Example: answer all CallbackQuery removing the loading circle.

```
$bot->answerUpdate["callback_query"] = function ($bot, $callback_query) {
    $bot->answerCallbackQuery();
}
```

You can find all possible updates here.

Answer the /start command

```
// What the bot will answer?
$start_closure = function ($bot, $message) {
    $bot->sendMessage("Hello stranger. This is my start message");
};

// Register the command
$bot->addCommand(new Commands\MessageCommand("start", $start_closure));
```

For a complete list of Commands, checkout the Command List.

Getting updates

There are two mutually exclusive ways of receiving updates for your bot: - getUpdates - Webhooks

getUpdates

At the end of your bot script add:

```
$bot->run(GETUPDATES);
```

Your bot will start asking updates to Telegram and will process them using `Bot::answerUpdate` and Commands.

Webhooks

Warning: *This method requires both a webserver and a SSL certificate.*

Add this line at the end of your bot script:

```
$bot->run(WEBHOOK);
```

CHAPTER 3

Database

Connecting the database

PhpBotFramework uses a simple wrapper to handle the database.

Connection using the wrapper:

```
$bot->database->connect([
    'adapter' => 'pgsql',
    'username' => 'sysuser',
    'password' => 'mypassword',
    'dbname' => 'my_bot_db'
]);
```

Or if you connect using PDO, pass the PDO object to the framework to use the facilities:

```
$bot->database->pdo = $yourPdoObject;
```

Then you can access your PDO object using:

```
$bot->getPdo();
```

Broadcast message

If you want to update your users with the bot changelog, or telling them an important news you can use the `Database::broadcastMessage` which will do the job for you:

```
$bot->broadcastMessage("Checkout my new bot @DonateBot.")
```

This method takes the same parameters as `Bot::sendMessage`.

For working the database must have a `"User"` table with a `chat_id` row.

CHAPTER 4

Logging

PhpBotFramework implements logging features.

It automatically creates a logging in the bot folder if it is using `getUpdates`, otherwise it uses the syslog.

BotName

To change the bot name used in the log:

```
$bot->setBotName("MyBot");
```

Chat logging

In addition to file log and syslog you can setup a chat where all error messages will be sent:

```
$bot->setChatLog("35818591");
```

CHAPTER 5

Chat_id

The framework saves the `chat_id` of the current user (for private chats), group or channel based on where the update comes from.

All the Bot API methods which don't take `$chat_id` as a parameter will target the current chat.

To target another chat you can use:

- useChatId - withChatId

withChatId

This framework method will call the requested Bot API method using the choosed chat_id, without changing the current one.

```
$contact_command = new PhpBotFramework\Commands\MessageCommand("contact",
        function ($bot, $message) {
            // This message will be sent to whom pressed /contact
            $bot->sendMessage("My creator has been called");

            // This message will always be sent to @another_username
            $bot->withChatId("31285239382", "sendMessage", "Someone is calling for you
↪");
        });
```

useChatId

This method will execute all the logic inside assuming the `chat_id` is the one choosed instead of the current.

```
$bot->useChatId("25929619",
        function() use ($bot) {
            $bot->sendMessage("This is a new message");
            $bot->sendPhoto("logo.png");
        });
```

All methods inside the anonymous function will target the choosed `chat_id`. After the method will be called, the current chat will be the same as before.

CHAPTER 6

Payments

What are Payments API?

Recently, Telegram released the Payments API which allows bots (and developers) to receive money from users without need to leave the chat.

The payments are managed by third-party services:

- Stripe
- Yandex.Money
- Payme

While the API makes easy for a developer to know if (s)he was paid or not, it's not the same for the users who doesn't have no warranty.

PhpBotFramework 3.x has introduced the support to the Payments API and the usage is really straightforward.

Prepare the playground

You can enable Payments API for your bot directly from **@BotFather**.

Like always, Telegram explains in-depth how to do so and we can't do nothing better than link to its documentation.

Configure Payments credentials

PhpBotFramework comes with a method named **setPayment** which's used to set the necessary data used by the bot to receive money (the provider token and the money currency).

```
$bot->setPayment(getenv('PAYMENT_TOKEN'), 'EUR');
```

The money currency should be represented following **ISO 4217 currency code**.

Learn more here.

Create an invoice

A Telegram **invoice** is a special message which includes a form the user needs to fill in order to send money to the bot.

PhpBotFramework provides the **sendInvoice** method; here's the basic usage:

```
$bot->sendInvoice('Donation', 'Basic Donation', 'basicDonation', ['Donation' => 1]);
```

Using the code above, you're going to get something like:

You can define various prices to pay:

```
$bot->sendInvoice('Donation', 'Basic Donation', 'basicDonation', ['Donation' => 1,
↪'Plus' => 1.5]);
```

And you can pass additional parameters to 'sendInvoice'. Here's the complete list.

```
$bot->sendInvoice('Donation', 'Basic Donation', 'basicDonation', ['Donation' => 1], [
↪'is_flexible' => true]);
```

Shipping & Checkout

When the user fills the form and the payment is ok, we need a way to tell the bot what to do next.

PhpBotFramework integrates the **answerPreCheckoutQuery** method which takes the incoming **pre_checkout_query** (managed through answerUpdate) and answer it by returning greetings, errors or any kind of response.

```
$bot->answerUpdate['pre_checkout_query'] = function ($bot, $pre_checkout_query) {
  // Telegram uses a custom way to define the amount of money handled.
  // For instance, 1 EUR is represented like 100.
  $money_received = $pre_checkout_query['total_amount'] / 100;

  // For logging purpose.
  echo "Received '$money_received EUR'";

  $bot->sendMessage('Thanks for your donation!');
  $bot->answerPreCheckoutQuery(true);
};
```

As we said, we can return an error if something goes wrong:

```
$bot->answerPreCheckoutQuery(false, 'I am too rich to allows other donations');
```

We can also return additional delivery costs if needed through **answerShipping**.

```
$bot->answerShipping(true, '', ['FedEx' => 3.99, 'USPS' => 4.20]);
```

CHAPTER 7

Commands

When a command get triggered, the associated function get called.

Commands are checked in order of priority (based on the type of the commands).

MessageCommands

MessageCommands get triggered when the message received contains a `bot_command` at the start.

```
$start_command = new PhpBotFramework\Commands\MessageCommands("start",
        function ($bot, $message) {
            $bot->sendMessage("You just hit /start");
        });
```

CallbackCommand

CallbackCommands get triggered when an inline button containing the corresponding data is hit by the user.

```
$help_callback = new PhpBotFramework\Commands\CallbackCommand("help",
        function ($bot, $message) {
            // Edit the message which contains the inline button
            $bot->editMessageText("This message now contains helpful information");
            // Don't forget to call Bot::answerCallbackQuery to remove the updating␣
↪circle in the button
            $bot->answerCallbackQuery();
        }
);
```

AdminCommand

Admin command are valid only for selected id.

```
$admin_command = new PhpBotFramework\Commands\AdminCommand("data",
        function ($bot, $message) {
            $bot->sendMessage("Important data sent here");
        },
        // user_id of admins
        [ 2501295, 25912395 ]
);
```

MultiCharacterCommand

MultiCharacterCommand get triggered by messages what contains the selected keyword, prefixed by one of the wanted characters:

```
   $about_command = new PhpBotFramework\Commands\MessageCommands("about",
           function ($bot, $message) {
               $bot->sendMessage("This bot was made by BotFather.");
           },
           ['!', '.', '/']);


Either the messages starting with ``/start``, ``.start`` and ``!start`` will trigger␣
↪this command.
```

CHAPTER 8

Create an Inline Keyboard

Inline keyboard are special objects that can be sent along with messages.

`Bot::$keyboard` is a wrapper for inline keyboard creation:

```
// Answer /about messages
$bot->addCommand(new Commands\MessageCommand("about", function($bot, $message)
        {
            // Create an inline keyboard button with a link to a site
            $bot->keyboard->addButton("Link", "url", "example.com");

            // then send it with a message
            $bot->sendMessage("Visit our website!", $bot->keyboard->get());
        }
    )
);
```

For more information and guides about Inline Keyboard have a look here

All button added will be on the same row. Use:

```
$bot->keyboard->nextRow();
```

to switch to the next row.

You can also add more buttons on the same row using `Keyboard::addLevelButtons`:

```
$bot->keyboard->addLevelButtons(
         [
            'text' => 'Button1',
            'callback_data' => 1
         ],
         [
            'text' => 'Button2',
            'callback_data' => 2
         ]
     );
```

This method will automatically change row after being called.

CHAPTER 9

Inline Queries

Inline queries have been added in January 2016.

The user can call your bot by simply writing its username in the chat and a query.

After having enabled the inline mode enabled (by sending `/setinline` to @BotFather) the bot will start receiving `inline_query` updates.

Use the method `answerInlineQuery` to asnwer these updates. It requires an array of results to show to the user.

Results

Let's create the results:

```
$bot->answerUpdate["inline_query"] = function ($bot, $message) {

    $bot->results->newArticle("Result1", "This is the first result.");

    $bot->results->newArticle("Result2", "This is the second result.");

};
```

Important

Remember that the parameter `$reply_markup` must not be encoded in JSON.

If you cannot see the answer of the bots but you don't get any error message neither, check the `$reply_markup` parameter.

Sending the results

When we have added all the results, we are ready to send them:

```
$bot->answerUpdate["inline_query"] = function ($bot, $message) {

    // Creation of the results
    ...

    $bot->answerInlineQuery($bot->results->get());

};
```

Types

The type used in the example is `article`, have a look here for all types.

To create a result of a different type you can use `addResult`:

```
$bot->answerUpdate["inline_query"] = function ($bot, $message) {

    $bot->results->addResult([
            'type' => 'photo',
            'photo_url' => 'https://www.gstatic.com/webp/gallery/1.jpg',
            'thumb_url' => 'https://www.gstatic.com/webp/gallery/1.jpg'
         ]);

    $bot->answerInlineQuery($bot->results->get());

 };
```

To add multiple results simultaneously:

```
$bot->answerUpdate["inline_query"] = function ($bot, $message) {

    $bot->results->addResults([
            [
                'type' => 'photo',
                'photo_url' => 'https://www.gstatic.com/webp/gallery/1.jpg',
                'thumb_url' => 'https://www.gstatic.com/webp/gallery/1.jpg'
            ],
            [
                'type' => 'photo',
                'photo_url' => 'https://www.gstatic.com/webp/gallery/2.jpg',
                'thumb_url' => 'https://www.gstatic.com/webp/gallery/2.jpg'
            ]
    ]);

    $bot->answerInlineQuery($bot->results->get());

 };
```

CHAPTER 10

Indices and tables

- genindex
- modindex
- search