

Документация Telegram-плагина

????????? ????????	Telegram
??????????????	?????????????? ??????? ?? Metabot
???????	????????? ?????? (https://t.me/mr_result)
????? ??????????	02 ???????? 2023
????????????? ????? ??????????????	25 ??????? 2024

Описание

Этот плагин предоставляет удобный интерфейс для работы с Telegram Bot API. Он поддерживает отправку текстовых сообщений, фотографий, создание клавиатур и обработку ответов пользователя.

Основной класс TelegramMessage

Подключение и инициализация

```
let TelegramMessage = require('Common.Integrations.Telegram')
let msg = new TelegramMessage()
```

Основные параметры

```
msg.text = "Ваше сообщение" // Текст сообщения
msg.parse_mode = "HTML" // Режим форматирования (HTML или MarkdownV2)
msg.protect_content = true // Защита контента от пересылки
msg.keyboard = "Да[ yes]==Нет[ no]" // Создание клавиатуры
```

Форматы клавиатуры

- **Вертикальное разделение** — используйте `==`;

```
msg.keyboard = "Кнопка1[ btn1]==Кнопка2[ btn2]" // Кнопки будут расположены вертикально
```

- **Горизонтальное разделение** — используйте `||`;

```
msg.keyboard = "Кнопка1[ btn1]|| Кнопка2[ btn2]" // Кнопки будут расположены горизонтально
```

- **Комбинированное разделение;**

```
msg.keyboard = "Кнопка1[ btn1]|| Кнопка2[ btn2]==Кнопка3[ btn3]|| Кнопка4[ btn4]"
```

Специальные типы кнопок

- **Запрос контакта:**

```
msg.keyboard = "Отправить контакт[ telegram_contact]"
```

- **Запрос локации:**

```
msg.keyboard = "Отправить локацию[ telegram_location]"
```

- **Ссылки:**

```
msg.keyboard = "Посетить сайт[ <https://example.com>]"
```

- **Web App:**

```
msg.keyboard = "Открыть приложение{<https://webapp-url.com>}"
```

- **Ссылка на пользователя:**

```
msg.keyboard = "Написать админу[ tg: //user?id=123456789]"
```

Примеры использования

Простое меню с обработкой ответов

Код будет работать корректно только в команде [JS Callback](#).

```

let TelegramMessage = require('Common.Integrations.Telegram')
let msg = new TelegramMessage()
msg.text = "Выберите действие: "
msg.keyboard = "Информация[ info]==Помощь[ help]==Настройки[ settings]"
msg.parse_mode = "HTML"

switch (true) {
  case (isFirstImmediateCall):
    msg.send()
    return false

  case (msg.getMessagePayload()?.callback_data == "info"):
    msg.addReplyToText()
    bot.sendMessage("Информация о боте...")
    return false

  case (msg.getMessagePayload()?.callback_data == "help"):
    msg.addReplyToText()
    bot.sendMessage("Справка по использованию...")
    return false

  case (msg.getMessagePayload()?.callback_data == "settings"):
    msg.addReplyToText()
    bot.runScriptForLead(123, leadId) // Запуск скрипта настроек
    return false
}

```

Справочник методов

Основные методы отправки

- **send()** — отправка нового сообщения;
- **edit()** — редактирование существующего сообщения;
- **addReplyToText()** — добавление ответа к существующему сообщению;
- **removeInlineKeyboard()** — удаление клавиатуры.

Получение информации

- **getMessagePayload()** — получение информации о сообщении из вебхука:
 - **message_id** — ID сообщения;


- **text** — текст сообщения/кнопки;
- **input_type** — тип ввода ('write'/'press');
- **callback_data** — данные колбэка;
- **payload** — полные данные вебхука.

Обработка ответов пользователя

```
// Проверка типа ввода
if (msg.getMessagePayload()?.input_type == "write") {
    // Пользователь написал текст
}

// Проверка нажатия кнопки
if (msg.getMessagePayload()?.callback_data == "button_id") {
    // Пользователь нажал кнопку
}
```

Отладка

```
msg.debug("Отладочное сообщение") // Отправка отладочной информации.
 // Не отправляется пользователю но логируется в карточке лида
```

Лучшие практики

- Всегда проверяйте первый вызов:

```
if (isFirstImmediateCall) {
    msg.send()
    return false
}
```

- Добавляйте обработку текстовых сообщений:

```
if (msg.getMessagePayload()?.input_type == "write") {
    bot.sendMessage('Пожалуйста, используйте кнопки')
    return false
}
```

- Скрывайте клавиатуру после использования:

```
msg.addReplyToText() // После нажатия на кнопку записывает её в сообщение
```

Форматирование текста и entities

В плагине доступно два способа форматирования текста: **HTML** и **MarkdownV2**. По умолчанию параметр **parse_mode** не установлен — это сделано специально для возможности работы с **entities** (когда нужно сохранить форматирование текста, которое пользователь отправил в Telegram).

HTML форматирование

Документация: <https://core.telegram.org/bots/api#html-style>

```
msg.text = "<b>Жирный текст</b>\n<i>Курсив</i>\n<code>Моноширинный текст</code>"
msg.parse_mode = "HTML"
```

MarkdownV2 форматирование

Документация: <https://core.telegram.org/bots/api#markdownv2-style>

```
msg.text = "**Жирный текст**\n__Курсив__\n`Моноширинный текст`"
msg.parse_mode = "MarkdownV2"
```

Работа с entities

Entities используются, когда нужно сохранить форматирование текста, которое пользователь отправил в Telegram. Это особенно полезно при создании различных конструкторов сообщений или при пересылке сообщений с сохранением форматирования.

Пример сохранения форматированного сообщения пользователя:

```
let TelegramMessage = require('Common.Integrations.Telegram')
let msg = new TelegramMessage()

msg.text = `Введите текст для рассылки`
msg.keyboard = `⬅ Назад[${backScripts}]`
msg.parse_mode = 'HTML'

switch (true) {
  case (isFirstImmediateCall):
```

```

    msg.send()
    return false

case (msg.getMessagePayload()?.input_type == "write"):
    msg.removeInlineKeyboard()
    // Получаем webhook и извлекаем текст с entities
    let webhook = msg.getMessagePayload()
    let messageData = {
        "text": webhook?.payload?.message?.text,
        "entities": webhook?.payload?.message?.entities // Сохраняем entities для
сохранения форматирования
    }

    // Сохраняем сообщение с форматированием в базу данных
    lead.setJsonAttr("messageData", messageData)

    bot.run({
        script_id: nextScript
    })
    return false

default:
    msg.addReplyToText()
    bot.run({
        script_id: msg.getMessagePayload().callback_data
    })
    return true
}

```

В этом примере:

1. Когда пользователь отправляет форматированное сообщение, мы получаем не только сам текст, но и массив `entities`.
2. `Entities` содержат информацию о форматировании: жирный текст, курсив, ссылки и т.д.
3. Мы сохраняем и текст, и `entities`, чтобы позже можно было воспроизвести сообщение с точно таким же форматированием.
4. **Важно:** для работы с `entities` параметр **`parse_mode`** должен быть не установлен (режим по умолчанию).

Теперь после запоминания **`entities`** мы можем сделать его вывод таким образом:

```
let TelegramMessage = require('Common.Integrations.Telegram')

let msg = new TelegramMessage()

let messageData = lead.getJsonAttr("messageData")

msg.text = messageData?.text
msg.entities = messageData?.entities

// Если попробовать использовать parse_mode то форматирование будет некорректным
```

Работа с файлами

Плагин позволяет обрабатывать файлы, которые пользователи отправляют в Telegram. Вы можете получить URL файла и сохранить его в карточку лида, не отправляя обратно в Telegram.

Получение и обработка файлов

```
// Подключаем библиотеку
let TelegramMessage = require('Common.Integrations.Telegram')
let msg = new TelegramMessage()

// Все входящие файлы
let attachments = bot.getAllAttachments()

if (Boolean(attachments?.[0]?.url)) {

  uploadData = bot.downloadFileFromUrl(attachments[0].url)

  msg.debug('Пользователь отправил файл: ' + uploadData.url)
  msg.debug('Файл ID : ' + JSON.stringify(attachments[0]?.payload?.file_id)) // данная строка
  вспомогательная для
  [] // автоматизации вывода полученных
  lead.setAttr('file', uploadData.url) [] // файлов в других местах бота
  // Запуск скрипта...
  bot.runScriptByCodeForLead("recieveFile", lead.getData('id'))

  bot.stop()
```

```
}
```

В этом примере:

1. Проверяем наличие прикрепленных файлов с помощью **bot.getAllAttachments()**.
2. Если файл есть, скачиваем его через **bot.downloadFileFromUrl()**.
3. Логируем получение файла с помощью **msg.debug()**.
4. Сохраняем URL файла в атрибут лида.
5. Запускаем скрипт обработки файла.
6. Останавливаем текущий скрипт.

Дополнительный пример:

```
let TelegramMessage = require('Common.Integrations.Telegram')

let msg = new TelegramMessage()

msg.text = 'А теперь жду твоё фото '
msg.keyboard = ``
msg.parse_mode = 'HTML'

let scriptCode = bot.getCurrentScriptCode()
lead.setAttr("script_code", scriptCode)

let data = bot.getAllAttachments()

switch (true) {

    case (isFirstImmediateCall):

        msg.send()
        return false
        break

    case (Boolean(data?.[0]?.type != 'image')):

        bot.sendMessage('Отправь сжатое изображение. Такой формат не подходит')
        return false
        break

    case (Boolean(data?.[0]?.type == 'image')):
```



```
        uploadData = bot.downloadFileFromUrl(data[0].url)
        msg.debug(' Пользователь отправил файл: ' + uploadData.url)
        lead.setAttr(' photo', uploadData.url)
        bot.sendMessage(' Класс! Фото загрузил' )
        return true
        break

default:

        msg.addReplyToText()
        return true

}
```

Рекомендации по работе с файлами

- Всегда проверяйте наличие файлов перед их обработкой;
- Логируйте получение файлов для отладки;
- Сохраняйте URL файлов в атрибуты лида для дальнейшего использования;
- Запускайте отдельный скрипт для обработки файлов, чтобы разделить логику.

Версия #2

Ирина Петрова создал 3 March 2025 08:51:21

Ирина Петрова обновил 3 March 2025 09:32:09