Форматированные сообщения в Telegram в Metabot

Быстрый гайд + пример собственного плагина

В Metabot сообщения в мессенджеры можно отправлять не только через стандартные команды конструктора, но и **программно** — напрямую из JavaScript-кода, используя плагины.

Это особенно полезно, когда вам нужно:

- гибко управлять форматированием текста;
- выбирать parse mode (HTML / Markdown / MarkdownV2);
- централизовать логику отправки сообщений;
- подготовить основу под мультиканальную отправку (Telegram, WhatsApp, WebChat и др.).

В этом гайде разберём, как отправлять **форматированные сообщения в Telegram** с помощью плагина `Common.Helpers.SendFormattedMessage`.

Зачем отдельный плагин для форматированных сообщений

На первый взгляд Telegram позволяет просто вызвать `bot.sendMessage`. Но на практике быстро появляются проблемы:

- разные режимы форматирования (`HTML`, `Markdown`, `MarkdownV2`);

- необходимость экранирования символов (особенно в `MarkdownV2`);
- опции Telegram API (например, превью ссылок);
- обработка ошибок и возврат результата выполнения;
- повторное использование кода в разных сценариях и модулях.

Поэтому в Metabot мы выносим эту логику в **отдельный helper-плагин**, который:

- инкапсулирует работу с Telegram API;
- принимает текст + формат;
- возвращает стандартизированный результат выполнения.

Как использовать `sendFormattedMessage`

Подключение плагина

В любом JS-скрипте (endpoint, custom-command, handler кнопки и т.д.):

```
const { sendFormattedMessage } = require('Common.Helpers.SendFormattedMessage')
```

Пример отправки форматированного сообщения

```
let message = `
*Я — Орион.*
Я — *навигатор Корпуса Операторов*.

Я помогаю людям *ориентироваться в сложных системах*:
технологиях, ролях, командах и *собственных траекториях*.

*Я не продаю курсы* и *не обещаю быстрых результатов*.

Если ты здесь —
значит, тебе *интересна сложность*.
```

```
Для начала мне нужно немного понять:
*зачем ты пришёл*
и *в какой форме ты сейчас живёшь*.

Это займёт *пару минут*.`

sendFormattedMessage(message, 'Markdown')
```

Здесь:

- текст оформлен в **Telegram Markdown**;
- формат явно передаётся вторым аргументом;
- сообщение отправляется **напрямую через ядро**, минуя визуальный конструктор.

Поддерживаемые форматы Telegram

Плагин поддерживает стандартные режимы Telegram API:

Формат	Когда использовать	Особенности
`HTML` (по умолчанию)	когда нужен стабильный контроль	не требует сложного экранирования
`Markdown`	для простого форматирования	ограниченный синтаксис
`MarkdownV2`	когда нужен расширенный Markdown	требует экранирования почти всех спецсимволов

Практическая рекомендация

- **HTML** — самый безопасный и предсказуемый вариант.
- **Markdown** — удобен для быстрых текстов и прототипов.
- **MarkdownV2** — мощный, но сложный, используйте осознанно.

Подробные правила форматирования смотрите в официальной документации Telegram.

Где находится код плагина

Плагин размещается в общем пространстве helpers:

```
Common.Helpers.SendFormattedMessage
```

Структура плагина следующая:

- экспортируемая функция `sendFormattedMessage(message, parseMode, options)`
- внутри:
  - формирование `apiAdditionalParams` для Telegram (`parse_mode`, `disable_web_page_preview`);
  - вызов `bot.sendMessage(...)`;
  - обработка ошибок через стандартный response-модуль.

```
/**
 * Плагин для отправки форматированных сообщений в мессенджер (например, Telegram).
 * Автор: @ArtemGarashko
 * Версия: 1.0
 * Дата последнего обновления: 24.02.2025
 *
 * Этот плагин используется для отправки форматированных сообщений в мессенджер.
 * Рекомендуется использовать формат HTML т.к. он не требует эскейпирования символов в отличие от MarkdownV2.
 * В случае ошибки при отправке сообщения, плагин использует Response для формирования ответа.
 */

// Подключаем Response плагин для обработки ответов
const { getErrorResponse, getSuccessResponse } = require('Common.Utils.Response');

/**
 * Функция для отправки форматированного сообщения в мессенджер.
 * Использует Response плагин для обработки ответов.
 *
 * Примеры для каждого parseMode:
 * - 'HTML': <b>Привет</b> мир! — Выведет "Привет" жирным шрифтом.
 * - 'Markdown': *Привет* мир! — Выведет "Привет" курсивом.
 * @param {string} message - Текст сообщения, которое вы хотите отправить.
 * @param {string} [parseMode='HTML'] - Форматирование текста. Может быть:
 *  - 'HTML' (по умолчанию)
```

```
 *  - 'Markdown'
 *  - 'MarkdownV2'
 * @param {Object} [options={}] - Дополнительные параметры.
 * @param {boolean} [options.disableLinksPreview=true] - Отключить ли превью ссылок (по
умолчанию true).
 * @returns {Object} - Возвращает успешный или ошибочный ответ в зависимости от результата.
 */
function sendFormattedMessage(message, parseMode = 'HTML', options = {}) {
    // Получаем значение для отключения превью ссылок из options (по умолчанию true)
    const disableLinksPreview = options.disableLinksPreview !== undefined ?
options.disableLinksPreview : true;

    // Устанавливаем параметры для API Telegram
    let apiAdditionalParams = {
        "parse_mode": parseMode,
        "disable_web_page_preview": disableLinksPreview
    };

    try {
        // Отправляем сообщение через Telegram API
        bot.sendMessage(message, null, null, apiAdditionalParams);

        // Возвращаем успешный ответ через Response плагин
        return getSuccessResponse({ message: 'Сообщение отправлено успешно' });
    } catch (error) {
        // В случае ошибки возвращаем ошибку через Response плагин
        return getErrorResponse('Не удалось отправить сообщение', error.message);
    }
}

module.exports = {
    sendFormattedMessage
}
```

> ⚠ Исходный код, приведённый выше, актуален на момент публикации статьи. На серверах Metabot может использоваться более новая версия плагина с изменениями и доработками. Используйте код как ориентир и архитектурный шаблон.

Возврат результата и обработка ошибок

Важно: плагин **не просто отправляет сообщение**, но и возвращает результат выполнения.

Для этого используется общий модуль:

```
Common.Utils.Response
```

- `getSuccessResponse(...)`
- `getErrorResponse(...)`

Таким образом:

- при успехе вы получаете `{ success: true, ... }`;
- при ошибке — `{ success: false, error: true, message: ... }`.

Это удобно, если:

- вы хотите анализировать результат отправки;
- логировать ошибки;
- строить цепочки логики (например, fallback-сценарии).

Подробный разбор этого подхода будет в отдельной статье → *«Стандартизация успешных ответов и ошибок в Metabot»*.

Почему это low-code, а не просто JS

Обратите внимание на архитектурный момент:

- мы **не используем** визуальную команду конструктора;
- мы вызываем отправку сообщения **программно**;
- но при этом:
  - работаем внутри экосистемы Metabot;

- используем общий стандарт ответов;
- можем переиспользовать код в любых сценариях.

Это и есть философия Metabot:

> low-code как база + full-code там, где это действительно нужно.

Как расширять плагин дальше

Текущая версия — минималистичная, но она легко расширяется:

- добавить Telegram-специфичные опции (`reply_markup`, `entities`, `protect_content`);
- прокинуть дополнительные `options` без изменения API;
- реализовать аналогичные helpers под:
  - WhatsApp,
  - VK,
  - WebChat,
  - email,
  - push-уведомления.

По сути, `sendFormattedMessage` — это **точка абстракции**, от которой удобно строить мультиканальную систему сообщений.

Версия #5
Artem Garashko создал 2026-01-02 08:42:18 UTC
Artem Garashko обновил 2026-01-02 09:11:19 UTC