# Common.AI.ImageGen — Генерация изображений

**Автор:** Art Yg
**Версия:** 1.0

`ImageGen` — универсальный плагин для **асинхронной генерации изображений** через внешний Webhook Processor, с сохранением результата в lead (URL и/или base64) и поддержкой сценарного выхода (success/error/timeout).

Плагин спроектирован так, чтобы работать в **двухфазном режиме**, как `LLMQuery`:

- **PHASE 1** — отправить запрос (async) и выйти из скрипта
- **PHASE 2** — обработать callback-ответ от процессора и продолжить сценарий

---

# Зачем он существует

В Metabot-сценариях нельзя считать генерацию изображения “быстрой синхронной функцией”:

- внешние провайдеры отвечают с задержкой
- ответ может потеряться
- пользователь может продолжать писать, пока мы ждём
- результат может прийти не в том формате (URL vs b64)
- иногда важен **жёсткий контроль ожидания** (timeout), иначе сценарий “повиснет” навсегда

`ImageGen` стандартизирует этот поток:

- отправляет запрос **через Webhook Processor** (Remote transport)
- ждёт callback и распознаёт “это callback или пользовательский ввод”
- сохраняет результат в атрибуты lead
- может уйти в `successScript` (опционально)
- может уйти в `error.script` / `timeout` (если настроены)

---

# Что использует внутри

`ImageGen` — это обёртка, которая опирается на инфраструктурные компоненты:

- **Common.Remote.RemoteApiCall** — транспорт, отправляет запрос в Webhook Processor и включает `asyncResponse`
- **Webhook Processor** — внешний слой, который делает реальный HTTP-запрос к провайдеру и возвращает callback
- **Common.Platform.AsyncFallback** — таймаут/фоллбек-оркестрация (используется внутри `ImageGen` через `timeout`)
- **Common.AI.Prompts** — компонент резолвинга промптов из таблицы и сборки массивов `system/user`, включая ссылки вида `$alias`
- **локальная таблица провайдеров внутри ImageGen** — mapping baseUrl/endpoint/method

Важно: `RemoteApiCall` как транспорт **в принципе поддерживает любых провайдеров**, но текущая версия `ImageGen` по умолчанию заточена под OpenAI Images API.

---

# Поддерживаемые провайдеры

На текущий момент поддержан:

- `openai`

Если вам нужен другой провайдер (например, Replicate, Stability, Midjourney proxy, внутренний сервис) — **свяжитесь с командой**, и мы расширим плагин.

> Примечание по архитектуре: сейчас таблица провайдеров (`PROVIDERS`) находится **внутри плагина**. При необходимости её можно вынести наружу (в конфиг бота/таблицу/отдельный реестр), чтобы вы могли подключать свои провайдеры без изменения кода плагина.

---

# Что сохраняет

В зависимости от настроек:

- `save.urlAttr` — URL изображения (если провайдер вернул url)
- `save.b64Attr` — base64 JSON (`b64_json`), если пришёл именно он
- `save.rawJsonAttr` — сырой payload ответа (для диагностики)

---

# Двухфазный протокол выполнения

## PHASE 1 — отправка запроса

Когда `isFirstImmediateCall = true`:

1. Инициализирует timeout-policy через `Common.Platform.AsyncFallback` (если задан `timeout`)
2. Берёт токен из атрибута бота (по `auth.tokenKey`)
3. Собирает итоговый prompt:
   - если задан `prompts` — собирает `system[] + user[]`
   - элементы массива могут быть:
     - inline строка
     - ссылка `$alias` (берётся из таблицы `promptTable` для `agentName`)
   - если в `prompts` используются `$alias`, то **обязательны** `agentName` и `promptTable`
4. Формирует request body под `/images/generations`
5. Отправляет запрос через `RemoteApiCall.send(..., asyncResponse: true)`
6. (опционально) показывает `messages.wait`
7. Возвращает `false` — сценарий “выходит” и ждёт callback

---

## PHASE 2 — обработка callback

Когда `isFirstImmediateCall = false`:

1. Проверяет, что это действительно callback от процессора (`payload.is_async_response`)
2. Если это не callback (пользователь что-то написал):
   - показывает `messages.processing`
   - возвращает `false`
3. Если callback:
   - снимает таймаут-job через `AsyncFallback.unschedule()`
   - парсит `payload.content`
   - извлекает `url` или `b64_json`

- сохраняет в lead
- если включён `requireUrl` и url нет → ошибка

4. Если задан `successScript` — делает `bot.run(successScript)`, иначе возвращается `true`

---

# Конфигурация

Ниже описаны ключевые блоки `ImageGen.run()`.

---

# Provider + Auth

```
provider: "openai",
auth: { tokenKey: "OPENAI_API_KEY" }
```

- `provider` — ключ провайдера (сейчас актуально `openai`)
- `auth.tokenKey` — имя атрибута бота, где лежит API ключ

---

# Prompts (таблица + массивы)

`ImageGen` поддерживает интерфейс промптов “на выроста” — как в LLMQuery: массивы промптов и табличные ссылки.

```
agentName: "orion",
promptTable: "gpt_prompts",

prompts: {
  system: ["$avatar_brief_generator"],
  user: ["...основной запрос..."]
}
```

**Принципы:**

- `prompts.system` и `prompts.user` — **массивы строк**.
- элементы массива могут быть:
  - обычной строкой (inline prompt)
  - ссылкой вида `$alias` — тогда текст берётся из `promptTable` для указанного `agentName`
- если вы используете `$alias`, то:

- `agentName` **обязателен**
- `promptTable` **обязателен**
- иначе это конфигурационная ошибка (плагин бросает `throw`, чтобы не было “тихих” генераций без системного слоя)

> Примечание: OpenAI Images API принимает один `prompt` (строкой), поэтому `ImageGen` **склеивает массивы** в итоговый текст (обычно `system` + пустая строка + `user`). Это сделано для унификации с LLMQuery и поддержки других провайдеров/форматов в будущем.

---

# Image параметры

```
image: {
  model: "dall-e-3",
  n: 1,
  size: "1024x1792",
  quality: "standard",
  style: "natural",
  background: null,
  response_format: "url"
}
```

Практика по форматам результата:

- **dall-e-2 / dall-e-3**: можно запросить `response_format: "url"` и получить временный URL
- *gpt-image- модели**: часто отдают `b64_json` (URL может не прийти)

---

# requireUrl

```
requireUrl: true
```

Если `true`, то отсутствие URL считается ошибкой (даже если пришёл b64).

Это удобно для MVP-потока “дай URL, а скачивание/сохранение сделаю потом”.

---

# Таймаут (fallback)

```
timeout: {
  seconds: 120,
  script: "Orion_Image_Timeout"
}
```

Если callback **не пришёл** за указанное время — планировщик Metabot запускает `timeout.script`.

Таймаут реализован через `Common.Platform.AsyncFallback` **внутри** `ImageGen`.

---

# Namespace для fallback

```
fallback: {
  namespace: "orion_image_reflection"
}
```

Нужен, чтобы несколько асинхронных операций не конфликтовали.

Если не задан, будет auto:

- `imagegen_${code.toLowerCase()}`

---

# UX сообщения

```
messages: {
  wait: "   Генерируем…",
  processing: "⏳ Подожди, ещё формируется…",
  error: "⚠ Не удалось получить изображение"
}
```

- `wait` — показываем при отправке (PHASE 1)
- `processing` — показываем если пользователь пишет во время ожидания (PHASE 2, но это не callback)
- `error` — сообщение по умолчанию при ошибке (если нет `error.script`)

---

# Ошибки

```
error: {
  script: "Orion_Image_Error",
  flagAttr: "orion_image_error",
  reasonAttr: "orion_image_error_reason"
}
```

- `flagAttr` / `reasonAttr` — сохранить состояние ошибки в lead
- `script` — куда перейти при ошибке (опционально)

---

# Успешный выход

```
successScript: "Orion_Image_Ready"
```

Если не указан — `ImageGen.run()` возвращается в ту же точку (`return true`), без перехода.

---

# Таймаут: встроенный vs внешний

В большинстве сценариев таймаут проще и чище задавать **внутри** `ImageGen` через `timeout`, потому что плагин сам использует `Common.Platform.AsyncFallback`.

Внешнее управление таймаутом имеет смысл, если:

- вы хотите один общий timeout на несколько async-операций
- вы централизованно оркестрируете несколько вызовов с одним namespace/policy

---

# Примеры использования

# Пример 1 — как используется в Orion: system prompt из таблицы + timeout + error + URL (MVP)

```
/**
 * orion_profiling_reflection_image
 *
 * Назначение:
 * - асинхронно сгенерировать вертикальный "Operator Reflection" образ
 * - сохранить URL в лид (скачивание/сохранение — отдельным шагом)
 * - timeout + error внутри ImageGen (как у LLMQuery)
 */

const ImageGen = require("Common.AI.ImageGen");

return ImageGen.run({
  lead,
  isFirstImmediateCall,

  code: "OrionActorImage",

  // Provider/Auth
  provider: "openai",
  auth: { tokenKey: "OPENAI_API_KEY" },

  // Prompts из таблицы + inline
  agentName: "orion",
  promptTable: "gpt_prompts",
  prompts: {
    // системный слой (табличный)
    system: ["$avatar_brief_generator"],

    // основной запрос (inline)
    user: [`
Create a vertical codex-grade mythotech Operator icon in Aurum Void aesthetic.

Aurum Void: cold matte gold schematic lines (axes, nodes, ritual UI glyphs) over deep graphite
```

```
void.
No glossy sci-fi, no neon, no superhero vibe, no humor.

Faceless figure (hood/shadow/mask/void), calm, stable, centered on a strong vertical axis.
Heavy materials: carbon composite armor/robe, matte metal, dense fabric, worn realistic
textures.
Subtle fog/particles for depth.

This is NOT a human portrait. It is an operational manifestation of a system entrepreneur /
product leader at the scaling stage.
Output: a visual artifact, not an illustration.
    `.trim()]
  },

  // Таймаут (fallback)
  timeout: {
    seconds: 120,
    script: "Orion_Image_Timeout"
  },

  // Ошибки (как в LLMQuery)
  error: {
    script: "Orion_Image_Error",
    flagAttr: "orion_image_error",
    reasonAttr: "orion_image_error_reason"
  },

  // Namespace чтобы не конфликтовать
  fallback: {
    namespace: "orion_image_reflection"
  },

  // MVP: хотим именно URL
  requireUrl: true,

  image: {
    model: "dall-e-3",
    size: "1024x1792",
    quality: "standard",
    style: "natural",
```

```
    response_format: "url"
  },

  messages: {
    wait: "   ORION формирует визуальное отражение…",
    processing: "  Подожди, образ ещё куется…",
    error: "⚠  Не удалось получить изображение"
  },

  save: {
    urlAttr: "orion_actor_image_url",
    b64Attr: "orion_actor_image_b64",
    rawJsonAttr: "orion_actor_image_payload"
  }

  // successScript: "Orion_Image_Ready" // опционально
});
```

---

# Пример 2 — минимальный сценарий с таблицей промптов, без successScript

Подходит, когда вы хотите вернуться “в ту же точку” и решать дальше в текущем шаге.

```
const ImageGen = require("Common.AI.ImageGen");

return ImageGen.run({
  lead,
  isFirstImmediateCall,

  code: "OperatorIcon",

  provider: "openai",
  auth: { tokenKey: "OPENAI_API_KEY" },

  agentName: "orion",
  promptTable: "gpt_prompts",
  prompts: {
```

```
    system: ["$avatar_brief_generator"],
    user: ["Create a vertical mythotech Operator icon in Aurum Void aesthetic..."]
  },

  timeout: { seconds: 90, script: "Image_Timeout" },
  error: { script: "Image_Error" },

  requireUrl: true,

  image: {
    model: "dall-e-3",
    size: "1024x1792",
    response_format: "url"
  },

  save: {
    urlAttr: "operator_icon_url",
    rawJsonAttr: "operator_icon_payload"
  }
});
```

---

# Пример 3 — внешний контроль timeout через AsyncFallback (продвинутый режим)

Этот способ полезен, если:

- у вас одна общая политика таймаутов
- вы управляете фоллбеками централизованно (например, несколько разных async-операций в одном шаге)

```
const ImageGen = require("Common.AI.ImageGen");
const AsyncFallback = require("Common.Platform.AsyncFallback");

if (isFirstImmediateCall) {
  AsyncFallback.configure({
    lead,
```

```
    namespace: "orion_image_reflection",
    timeout: { seconds: 120, script: "Orion_Image_Timeout" },
    error: { flagAttr: "orion_image_error", reasonAttr: "orion_image_error_reason" }
  }).schedule();
}

const res = ImageGen.run({
  lead,
  isFirstImmediateCall,

  code: "OrionActorImage",
  provider: "openai",
  auth: { tokenKey: "OPENAI_API_KEY" },

  agentName: "orion",
  promptTable: "gpt_prompts",
  prompts: {
    system: ["$avatar_brief_generator"],
    user: ["Create a vertical mythotech Operator icon in Aurum Void aesthetic..."]
  },

  // timeout внутри не задаём, потому что контролируем снаружи
  error: {
    script: "Orion_Image_Error",
    flagAttr: "orion_image_error",
    reasonAttr: "orion_image_error_reason"
  },

  requireUrl: true,

  image: {
    model: "dall-e-3",
    size: "1024x1792",
    response_format: "url"
  },

  save: {
    urlAttr: "orion_actor_image_url",
    rawJsonAttr: "orion_actor_image_payload"
  }
```

```
});

if (!isFirstImmediateCall && res === true) {
  AsyncFallback.configure({
    lead,
    namespace: "orion_image_reflection"
  }).unschedule();
}

return res;
```

---

# Частые сценарии отказов и как реагировать

- **send_failed** — процессор не принял запрос / RemoteApiCall не отработал → `error.script` или retry
- **provider_error** — провайдер вернул ошибку (например, HTTP != 200) → смотреть `save.rawJsonAttr`, логировать, предлагать retry
- **url_missing** при `requireUrl=true` → либо переключиться на `b64_json`, либо поменять модель/response_format
- **no_result** — ответ пришёл, но данных нет → сохранить `rawJsonAttr` и смотреть, что вернул провайдер/процессор
- **timeout** (через `timeout.script`) → отдельный timeout-script может предложить повторить генерацию или вернуться в меню

---

# Итог

`Common.AI.ImageGen` — стандартизированный способ подключить генерацию изображений в сценарии Metabot:

- двухфазный async flow
- сохранение результата в lead
- UX на случай “пользователь пишет во время ожидания”
- встроенный timeout через `Common.Platform.AsyncFallback`
- работа с промптами как с массивами, включая табличные `$alias` через `Common.AI.Prompts`
- расширяемая система провайдеров (сейчас OpenAI)

Версия #5
Artem Garashko создал 1 February 2026 10:57:26
Artem Garashko обновил 1 February 2026 12:04:33