# Common.AI.Prompts — Универсальный резолвер и сборщик промптов

**Автор:** Art Yg
**Версия:** 1.0

`Prompts` — инфраструктурный helper для **унифицированной работы с промптами** в сценариях Metabot и внутри других плагинов (например, `Common.AI.ImageGen`, `LLMQuery/LMClient`).

Он решает типовую боль: **не держать промпты в коде**, не копировать одно и то же “склеивание”, и иметь единый механизм:

- подтянуть промпт из таблицы по ссылке (`$name`, `@name`)
- применить макросы на основе `lead` и `bot`
- нормализовать вход (строка/массив/что угодно)
- собрать итоговый текст промпта из блоков (`system/user/last`)

---

# Зачем он существует

В продуктовых сценариях промпты обычно:

- растут и ветвятся по агентам (`agentName`)
- переезжают в таблицы/консоль управления (а не в JS)
- используют переменные окружения (lead/bot attrs)
- собираются из нескольких частей (“системный каркас” + “задача” + “контекст”)

Без `Prompts` это превращается в дублирование:

- вручную ходим в `table.find(...)`
- вручную проверяем, что `$alias` нашли
- вручную подставляем `lead.getAttr(...)`
- вручную склеиваем массивы

`Prompts` стандартизирует это как **маленький инфраструктурный слой**, который можно подключать где угодно.

---

# Основные принципы

- **Lead-first** — `lead` передаётся явно (без неявной магии).
- **Bot runtime** — `bot` берётся из глобального окружения runtime.
- **Refs only if asked** — в таблицу лезем **только если** ты используешь `$...` или `@...`.
- **Strict by default** — если промпт по ссылке не найден → ошибка (чтобы не генерить “пустоту” молча).
- **Composable** — подходит и как отдельный helper, и как зависимость внутри других плагинов.

---

# Минимальные требования

Для работы в “inline-режиме” достаточно:

1. `lead`
2. вызвать `Prompts.buildText(...)`

Для работы с табличными ссылками (`$...` / `@...`) дополнительно нужно:

1. наличие `table.find` в runtime
2. таблица промптов (`promptTable`)
3. агент (`agentName`) — **обязателен только для** `$name`

---

# Что умеет Prompts

## 1) Ссылки на промпты из таблицы

Поддерживаются два типа ref:

- `$name` — промпт из таблицы для **конкретного агента**
- `@name` — промпт из таблицы для **общего агента** `<<common>>`

Пример:

- `$avatar_brief_generator` → `agentName = "orion"`
- `@safety_rules` → `agentName` не нужен (используется `<<common>>`)

---

# 2) Макросы (переменные из lead и bot)

`Prompts` умеет подставлять значения из атрибутов:

- `{{$key}}` → из `lead` (attr или json)
- `{{$$key}}` → из `bot` (attr или json)

Поведение:

- если найден json-атрибут → подставляет `JSON.stringify(value)`
- иначе подставляет строковый `getAttr`
- если не найдено → подставляет пустую строку

---

# 3) Нормализация входов и сборка блоков

На вход можно дать:

- `null/undefined` → станет `[]`
- `string` → станет `[string]`
- `array` → останется array
- `object/number` → станет `[String(value)]`

Дальше `Prompts` собирает:

- `system[]`
- `user[]`
- `last[]`
- `all[]` = `system + user + last`

И может вернуть:

- либо blocks (`buildBlocks`)
- либо финальную строку (`buildText`) через `join("\n\n")`

---

# Конфигурация

Все методы используют общие опции.

# DEFAULTS

```
{
  promptTable: "gpt_prompts",
  agentName: null,   // обязателен для "$name"
  strict: true,      // если промпт не найден → throw
  applyMacros: true
}
```

# Важные правила

- `agentName` **обязателен только** если ты используешь `$name`
- `promptTable` обязателен для `$name` и `@name`
- если нет `$` / `@` refs — можно вообще не передавать `agentName/promptTable`

---

# API Prompts

# Prompts.toArray(value)

Нормализует значение в массив строк.

Используется внутри, но можно использовать и снаружи.

---

# Prompts.resolveOne(ref, opts)

Резолвит одну строку:

- `$name` → ищет в `promptTable` по `agentName`
- `@name` → ищет в `promptTable` по агенту `<<common>>`
- обычная строка → возвращается как есть

# Prompts.resolveMany(list, opts)

Резолвит список refs/строк → массив строк.

---

# Prompts.applyMacros(str, lead)

Применяет:

- `{{$key}}` из lead
- `{{$$key}}` из bot

---

# Prompts.buildBlocks(input, opts)

Собирает блоки по секциям:

```
{
  system: [],
  user: [],
  last: [],
  all: []
}
```

---

# Prompts.buildText(input, opts)

Собирает итоговый prompt как строку:

- вызывает `buildBlocks`
- склеивает `all.join("\n\n")`

---

# Табличный формат промптов

`Prompts` ожидает, что таблица (`promptTable`) содержит хотя бы поля:

- `agent_name`
- `name`
- `prompt`

Запрос выполняется через:

```
table.find(promptTable, ["prompt"], [
  ["agent_name", agent],
  ["name", name]
]);
```

---

# Использование

Ниже примеры именно “для статьи”: чтобы читатель увидел, что есть несколько режимов и зачем это.

---

# Примеры

## Пример 1 — Inline: без таблиц, без агента

Подходит для простых сценариев и MVP.

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

const prompt = Prompts.buildText(
  {
    system: [
      "You are a strict image generator. Output must be cinematic, realistic, and calm."
    ],
    user: [
      "Create a vertical mythotech Operator icon in Aurum Void aesthetic."
    ]
  },
```

```
  { lead } // agentName/promptTable не нужны
);

// prompt — готовая строка, без обращений к таблицам
```

---

# Пример 2 — Табличный промпт агента: `$alias`

Если используешь `$...`, то **agentName обязателен**, иначе будет throw.

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

const prompt = Prompts.buildText(
  {
    system: [
      "$avatar_brief_generator" // берём из таблицы для агента orion
    ],
    user: [
      "Output should be a codex-grade artifact. No neon. No superhero vibe."
    ]
  },
  {
    lead,
    agentName: "orion",
    promptTable: "gpt_prompts"
  }
);
```

---

# Пример 3 — Общий промпт: `@alias` (agentName не нужен)

`@name` всегда читается из агента `<<common>>`.

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

```
const prompt = Prompts.buildText(
  {
    system: [
      "@safety_rules",
      "@style_aurum_void"
    ],
    user: [
      "Create an abstract Operator icon."
    ]
  },
  {
    lead,
    promptTable: "gpt_prompts"
  }
);
```

# Пример 4 — Макросы: подтягиваем контекст из lead и bot

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

// допустим:
// lead.getAttr("actor_stage") = "scaling"
// bot.getAttr("BRAND_TONE") = "discipline, meaning, depth"

const prompt = Prompts.buildText(
  {
    system: [
      "Tone: {{$$BRAND_TONE}}"
    ],
    user: [
      "Stage: {{$actor_stage}}",
      "Create a vertical Operator artifact."
    ]
  },
  {
    lead,
```

```
    applyMacros: true
  }
);
```

---

# Пример 5 — Сборка blocks отдельно (для отладки/логирования)

Иногда полезно видеть, какие блоки получились до склейки.

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

const blocks = Prompts.buildBlocks(
  {
    system: ["@style_aurum_void", "$avatar_brief_generator"],
    user: ["Generate an icon for current stage: {{$actor_stage}}"]
  },
  {
    lead,
    agentName: "orion",
    promptTable: "gpt_prompts"
  }
);

// blocks.system / blocks.user / blocks.all — можно сохранить в lead или tracer
```

---

# Пример 6 — Мягкий режим (strict=false)

Иногда нужен режим “не падать”, а вернуть сообщение/заглушку.

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

const prompt = Prompts.buildText(
  {
    system: ["$missing_alias"],
    user: ["Create an icon."]
  },
```

```
  {
    lead,
    agentName: "orion",
    promptTable: "gpt_prompts",
    strict: false
  }
);

// В strict=false при отсутствии промпта вернётся текст-предупреждение (а не throw)
```

---

# Использование внутри других плагинов

`Common.AI.Prompts` специально сделан как “маленький кусок инфраструктуры”, чтобы:

- не тащить целиком LMClient ради таблиц промптов
- не повторять код резолва в каждом AI-плагине
- унифицировать поведение `$alias` / `@alias` и макросов

## Где он уже естественно применяется

- `Common.AI.ImageGen` — чтобы собирать итоговый `image.prompt` из блоков и таблиц
- потенциально любой “AI transport plugin”: STT, TTS, embeddings, classification, etc.

---

# Частые ошибки и как их избежать

## 1) Использовали `$alias`, но не указали agentName

Это **ошибка по контракту**, будет throw:

- потому что `$alias` — агентный промпт
- без `agentName` невозможно определить неймспейс

Решение: передай `agentName`.

---

# 2) Использовали `$alias` / `@alias`, но не указали promptTable

Это тоже ошибка:

- потому что непонятно, откуда искать

Решение: передай `promptTable` (или оставь дефолт `"gpt_prompts"`).

---

# 3) Хотели “просто текст”, но случайно начали строку с `$`

Если текст реально должен начинаться с `$`, то сейчас это будет воспринято как ref.

Практический паттерн: не начинай “сырой текст” с `$` / `@`. Если прям надо — лучше добавить пробел или явную экранировку на уровне твоего контента.

---

# Итог

`Common.AI.Prompts` — это базовый инфраструктурный helper, который:

- даёт единый формат сборки промптов (`system/user/last`)
- вытаскивает промпты из таблиц (`$agent`, `@common`)
- подставляет переменные из lead/bot
- снижает дублирование и делает AI-плагины проще и чище

Он может использоваться:

- **как самостоятельный utility** в сценариях
- **как зависимость** внутри более крупных AI-плагинов (например, `ImageGen`)

Если в твоей архитектуре дальше появятся новые источники промптов (реестр провайдеров, JSON-конфиги, версии промптов, AB-тесты) — вот этот слой и будет правильным местом для расширения. И это как раз тот случай, где можно внезапно сделать себе ловушку, если начать “подмешивать” сюда бизнес-логику — держи его инфраструктурным, иначе потом будет боль.

---

Версия #2
Artem Garashko создал 1 February 2026 11:39:19
Artem Garashko обновил 1 February 2026 11:49:30