Как трассировать и отлаживать сложные процессы и функционал

> ⚠ **Важно перед началом**
>
> Ниже приведён **реальный рабочий плагин**, который используется в проектах Metabot для трассировки и отладки сложной логики.
>
> Этот код:
>
> - не является учебным примером «в вакууме»
> - не упрощён специально для урока
> - отражает **практический инженерный подход**, применяемый в боевых сценариях
>
> Вы можете:
>
> - использовать этот плагин **как есть** в своих проектах
> - брать его **в качестве основы** для собственных трассировщиков
> - адаптировать под свои таблицы, поля и процессы
>
> ⚠ **Обратите внимание:** версия плагина, опубликованная ниже, может **отличаться от версии**, используемой в текущем продакшене. В боевых окружениях код может эволюционировать, дополняться или оптимизироваться под конкретные задачи.
>
> Рассматривайте этот материал как:
>
> - пример архитектурного подхода
> - рабочий шаблон
> - отправную точку для собственных решений
>
> Ниже — описание принципов и **полный исходный код плагина**.

Универсальный трассировщик событий

Автор: Art Yg

Tracer предназначен для записи любых диагностических событий в таблицы базы данных:

- ошибок
- проверок
- ветвлений логики
- внутренних состояний
- отладочной информации

Tracer **не знает**:

- что такое сессия
- что такое скрипт, команда или триггер
- какие поля считаются «правильными»

Он записывает **ровно те данные, которые ему передали**.

Основные принципы

- **Stateless** — не хранит состояние
- **Schema-agnostic** — не требует фиксированной схемы
- **Zero mandatory fields** — нет обязательных полей
- **Opt-in** — работает только если включён
- **Side-effect only** — не влияет на выполнение кода

Tracer можно удалить из проекта — бизнес-логика продолжит работать.

Минимальные требования

Для начала работы достаточно:

1. Создать любую таблицу в БД (даже без полей, кроме `id`)
2. Добавить атрибут бота `TRACER_CONFIG`

3. Вызвать `trace()`

Рекомендуемое, но **не обязательное** поле таблицы:

- `created_at` с автозаполнением (`NOW`)

Tracer **не управляет временем**. Если поле есть — БД заполнит его автоматически. Если нет — запись всё равно создаётся.

Конфигурация

Tracer настраивается через один JSON в атрибутах бота: `TRACER_CONFIG`.

```
{
  "navigation": {
    "enabled": true,
    "table": "nav_trace"
  },
  "ai": {
    "enabled": false,
    "table": "ai_trace"
  }
}
```

- каждый tracer имеет имя (`navigation`, `ai`, `api` и т.д.)
- у каждого tracera своя таблица (или общая)
- выключенный tracer ничего не пишет

Использование

```
const Tracer = require("Common.DX.Tracer");

Tracer.trace("navigation", {
  category: "NAVIGATION",
  component: "Actor",
  action: "hasAchievement",
  level: "OK",
```

```
  payload: {
    actor_id: 42,
    achievement: "first_step",
    result: true
  }
});
```

Если tracer выключен — метод молча завершится.

Методы Tracer

Tracer предоставляет несколько эквивалентных методов:

- `trace(name, data)` — базовый метод записи
- `log(name, data)` — алиас для читаемости
- `info(name, data)` — добавляет `level: "INFO"`
- `error(name, data)` — добавляет `level: "ERROR"`

Все методы:

- не выбрасывают ошибок
- не изменяют переданные данные
- не влияют на бизнес-логику

Данные события

Tracer принимает **любой объект**.

Все поля:

- опциональны
- именуются произвольно
- записываются «как есть»

Рекомендуемые (но не обязательные):

- `category` — область (NAVIGATION, AI, API)
- `component` — компонент
- `action` — действие

- `source` — источник (system, user, webhook)
- `level` — уровень ошибки
- `payload` — любые данные (тип поля TEXTAREA)

Если таблица не содержит поле — БД вернёт ошибку. В таком случае используйте `payload`.

Работа со временем

Tracer:

- не добавляет timestamp
- не требует поля времени
- позволяет передать своё время

```
{
  event_time: "2026-01-16T12:00:00Z"
}
```

или

```
{
  created_at: "2026-01-16T12:00:00Z"
}
```

Когда использовать

- метод возвращает `true / false`, но нужна диагностика
- не хочется усложнять ответы ошибками
- важно понять, **почему** логика не сработала
- нужна отладка без влияния на сценарии

Когда не использовать

- как бизнес-лог
- как аудит-лог

- как аналитику или метрики

Итог

`Common.DX.Tracer` — простой и ненавязчивый способ видеть, что происходит внутри системы.

Никакой магии. Никаких обязательств. Никакой боли.

```
/**
 * Common.DX.Tracer
 *
 * Universal stateless event tracer for Metabot.
 *
 * Version: 1.0.0
 * Author: Art Yg
 *
 * Principles:
 *  - Stateless
 *  - Schema-agnostic
 *  - Zero mandatory fields
 *  - Opt-in only
 *  - Side-effect only
 *
 * Tracer writes exactly what it is given.
 * If disabled — does nothing.
 */

class Tracer {

  /**
   * Internal: get tracer config from bot attributes
   *
   * Expected bot attr: TRACER_CONFIG (JSON)
   *
   * Example:
   * {
   *   "navigation": { "enabled": true, "table": "nav_trace" },
   *   "ai":         { "enabled": false, "table": "ai_trace" }
   * }
```

```
 *
 * @returns {Object}
 */
static _getConfig() {
  try {
    const raw =
      bot.getJsonAttr('TRACER_CONFIG') ||
      bot.getAttr('TRACER_CONFIG');

    if (!raw) return {};

    if (typeof raw === 'object') return raw;

    return JSON.parse(raw);
  } catch (e) {
    // Tracer must never break execution
    return {};
  }
}

/**
 * Check whether tracer is enabled
 *
 * @param {string} tracerName
 * @returns {Object|null} tracer config or null
 */
static _getTracer(tracerName) {
  if (!tracerName) return null;

  const cfg = Tracer._getConfig();
  const tracer = cfg[tracerName];

  if (!tracer) return null;
  if (!tracer.enabled) return null;
  if (!tracer.table) return null;

  return tracer;
}

/**
 * Write trace event
```

```
   *
   * @param {string} tracerName - logical tracer name (navigation, ai, api, etc)
   * @param {Object} data - arbitrary payload
   *
   * @returns {boolean} true if written, false otherwise
   */
  static trace(tracerName, data = {}) {
    const tracer = Tracer._getTracer(tracerName);
    if (!tracer) return false;

    // We do NOT mutate data
    // We do NOT enforce schema
    // We do NOT inject fields
    try {
      table.createItem(tracer.table, data);
      return true;
    } catch (e) {
      // Silent failure by design
      return false;
    }
  }

  /**
   * Alias for semantic readability
   * (optional, but often nice)
   */
  static log(tracerName, data = {}) {
    return Tracer.trace(tracerName, data);
  }

  /**
   * Explicit error tracer helper
   * (pure sugar, no logic)
   */
  static error(tracerName, data = {}) {
    return Tracer.trace(tracerName, {
      ...data,
      level: data.level || 'ERROR'
    });
  }
```

```
  /**
   * Explicit info tracer helper
   */
  static info(tracerName, data = {}) {
    return Tracer.trace(tracerName, {
      ...data,
      level: data.level || 'INFO'
    });
  }
}

module.exports = Tracer;
```

Версия #5
Artem Garashko создал 16 January 2026 16:43:50
Artem Garashko обновил 16 January 2026 17:39:42