

# ??????????

API SDK, команды и протокол обмена с сервером

- [Справочник API](#)
- [Обработка команд](#)
- [Протокол обмена с сервером](#)

# ????????? API

Справочник публичного интерфейса SDK `artex-robot` версии 0.1.1.

## ????? Robot

```
from artex_robot import Robot
robot = Robot(conf: dict = {})
```

Создаёт робота, загружает конфигурацию из `params.json` и словаря `conf` (приоритет у `conf`), инициализирует журнал и Vault. Соединение с сервером **не** устанавливается — для этого вызовите `init_connector()`. Параметры — на странице [Конфигурация](#).

## ?????????

### @robot.com\_handler

Регистрирует обработчик команд. **Обязателен** — должен быть объявлен до вызова `init_connector()`.

```
@robot.com_handler
def on_command(type_command: str, command, author=None):
    ...
```

| Аргумент                  | Описание                                                                                                                    |
|---------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <code>type_command</code> | Тип команды: <code>com</code> , <code>force_com</code> , <code>sys_com</code> и др. (см. <a href="#">Обработка команд</a> ) |
| <code>command</code>      | Сама команда — строка или структура, которую отправил оператор/модель                                                       |
| <code>author</code>       | Идентификатор автора команды (оператор или модель ИИ)                                                                       |

Обработчик вызывается в потоке Socket.IO-клиента, а не в основном цикле. Долгие действия выполняйте асинхронно (см. пример колёсного робота на странице [Примеры роботов](#)).

## @robot.telemetry(key)

Помечает функцию как источник телеметрии. При каждом вызове функции её результат записывается в Vault под ключом `key`.

```
@robot.telemetry("battery")
def read_battery():
    return adc.read_voltage()
```

- Результат должен сериализоваться в JSON-совместимый вид: число, строка, bool, список, словарь.
- На сервер отправляется **весь** словарь телеметрии, например `{"battery": 11.8, "speed": 0.4}`.
- Несколько функций с разными ключами дополняют общий словарь.
- Функция-обёртка ничего не возвращает — чтобы использовать значение ещё и локально, прочитайте датчик отдельно.

## @robot.video(cam\_id)

Помечает функцию как источник видео для камеры `cam_id`. Результат функции — кадр OpenCV (numpy-массив BGR) — сжимается в JPEG (качество 80) и записывается в Vault.

```
@robot.video("front")
def send_front(frame):
    return frame
```

- Функция **не должна** возвращать `None` — это вызовет ошибку OpenCV.
- Несколько камер — несколько функций с разными `cam_id`; на сервер уходит словарь `{"front": <jpeg>, "rear": <jpeg>}`.
- Если `cam_id` содержит `led` или `rgb_strip`, значение **не** кодируется в JPEG и передаётся как есть. Так LED-устройства передают своё текущее состояние (например, массив цветов светодиодной ленты).

## @robot.mainloop

Регистрирует функцию основного цикла. **Обязателен.**

```
@robot.mainloop
def main_loop(context: dict):
    ...
```

Функция вызывается бесконечно с паузой `max_main_freq` секунд. Аргумент `context` — словарь контекста робота (см. ниже). Исключения внутри функции не перехватываются SDK и завершают программу — обрабатывайте их сами.

## ??????

| Метод                                                                                                               | Описание                                                                                                                                                                                                                                                                                         |
|---------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>robot.init_connector()</code>                                                                                 | Подключается к серверу <code>server:port</code> , namespace <code>/robot</code> , передаёт паспорт устройства. Включено авто-переподключение: 5 попыток, задержка 1-5 с. Если сервер недоступен при первом подключении — выбрасывает исключение <code>socketio.exceptions.ConnectionError</code> |
| <code>robot.start()</code>                                                                                          | Запускает основной цикл. Блокирует выполнение. По Ctrl+C корректно останавливает потоки и отключается                                                                                                                                                                                            |
| <code>robot.stop()</code>                                                                                           | Останавливает отправку телеметрии и видео и отключается от сервера                                                                                                                                                                                                                               |
| <code>robot.set_context(key, value)</code>                                                                          | Кладёт объект в контекст робота                                                                                                                                                                                                                                                                  |
| <code>robot[key] = value</code> , <code>robot[key]</code> , <code>del robot[key]</code> , <code>key in robot</code> | То же через синтаксис словаря                                                                                                                                                                                                                                                                    |

## ????????

Контекст — словарь для объектов, которые нужны основному циклу и обработчикам: камеры, драйверы моторов, последние команды, состояние автомата.

```
robot["cam"] = cv2.VideoCapture(0)
robot["motors"] = MotorDriver()

@robot.mainloop
def loop(ctx):
    ok, frame = ctx["cam"].read()
```

## ?????? robot.log

```
robot.log("INFO", "Моторы инициализированы")
robot.log("WARN", "Низкий заряд")
robot.log("ERROR", "Нет ответа от контроллера")
```

```
robot.log.set_level("skip_TODO")    # скрыть служебные TODO-сообщения
robot.log.set_type("file")          # писать в all.log / error.log
```

| Уровень | Использование                                                                                            |
|---------|----------------------------------------------------------------------------------------------------------|
| INFO    | Штатные события                                                                                          |
| WARN    | Предупреждения                                                                                           |
| ERROR   | Ошибки — при <code>logger_type = file</code> дублируются в <code>error.log</code>                        |
| TODO    | Служебные сообщения SDK (выводятся на каждой итерации цикла, скрываются уровнем <code>skip_TODO</code> ) |

Файлы журнала создаются в текущем рабочем каталоге. Сервер может запросить их удалённо (см. [Протокол обмена с сервером](#)).

???????????? ???? (???)  
????????????????????????????)

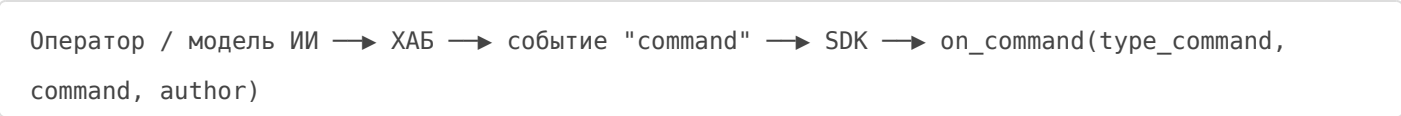
| Атрибут     | Назначение                                                                                                                                                                              |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| robot.vault | Хранилище: <code>set telem(key, val)</code> , <code>set_video(cam_id, frame)</code> , <code>get smeta()</code> , <code>get_dmeta(key)</code>                                            |
| robot.sio   | Коннектор; <code>robot.sio.sio</code> — клиент <code>socketio.Client</code> , <code>robot.sio.emit(event, data)</code> — отправка произвольного события в namespace <code>/robot</code> |

Пример прямой записи телеметрии без декоратора: `robot.vault.set telem("temp", 36.6)`.

# ????????? ???????

Команды приходят роботу от оператора (веб-интерфейс) или от модели ИИ через сервер. SDK разбирает сообщение и вызывает ваш обработчик `@robot.com_handler`.

## ??? ?????????? ?????????? ?? ?????????



Сообщение от сервера имеет вид:

```
{
  "author": "operator-7",
  "payload": { "com": "forward" }
}
```

Ключ внутри `payload` — **тип команды**, значение — **сама команда**. SDK передаёт их в обработчик как `type_command="com"`, `command="forward"`, `author="operator-7"`.

Команды принимаются только после того, как сервер прислал сигнал готовности (`Ready` в heartbeat). Команды, пришедшие раньше, игнорируются.

## ????? ??????????

| Тип                    | Кто обрабатывает  | Назначение                                                                                                  |
|------------------------|-------------------|-------------------------------------------------------------------------------------------------------------|
| <code>com</code>       | Ваш обработчик    | Обычная команда управления                                                                                  |
| <code>force_com</code> | Ваш обработчик    | Приоритетная команда — выполнить немедленно, прервав текущее действие (например, экстренная остановка)      |
| <code>sys_com</code>   | Ваш обработчик    | Системная команда (перезагрузка, калибровка, смена режима)                                                  |
| <code>file</code>      | SDK автоматически | Запрос журналов: <code>log</code> , <code>error</code> , <code>clear</code> (см. <a href="#">Протокол</a> ) |



```

elif command == "disarm": motors.power_off()
elif isinstance(command, str) and command.startswith("speed:"):
    motors.set_speed(int(command.split(":", 1)[1]) / 100)
else:
    robot.log("WARN", f"Неизвестная команда {command}")
    return
ctx["last_command_time"] = time.monotonic()

@robot.com_handler
def on_command(type_command, command, author=None):
    robot.log("INFO", f"{type_command}:{command} от {author}")

    # Остановка – всегда и немедленно, от любого источника
    if command in STOP_WORDS or type_command == "force_com":
        apply(command, robot.context)
        return

    if type_command == "com":
        apply(command, robot.context)
    elif type_command == "sys_com":
        handle_system(command)

```

?????????????: ?????????? ???  
 ??????? ???????

Если связь прервалась в момент движения, робот продолжит ехать с последней командой. Добавьте в основной цикл «сторожевой таймер»: если команд не было дольше заданного времени — остановка.

```

COMMAND_TIMEOUT = 0.5    # секунд

@robot.mainloop
def loop(ctx):
    last = ctx.get("last_command_time", 0)
    if time.monotonic() - last > COMMAND_TIMEOUT:
        ctx["motors"].drive(0, 0)

```

Для режима, когда оператор удерживает кнопку «вперёд», интерфейс или модель должны повторять команду чаще, чем раз в `COMMAND_TIMEOUT`. Для дискретных команд (движение до следующей команды) увеличьте таймаут или привяжите остановку к событию отключения.

????????? ?????????? ??? ????????

Правило приоритета применяется на сервере. Если роботу нужно учитывать его и на борту (например, при прямом подключении модели), различайте источники по `author`:

```
OPERATOR_HOLD = 2.0 # секунд после команды оператора модель не управляет

@robot.com_handler
def on_command(type_command, command, author=None):
    now = time.monotonic()
    is_model = str(author).startswith("model")
    if is_model and now - robot.context.get("operator_time", 0) < OPERATOR_HOLD:
        return # оператор недавно вмешался
    if not is_model:
        robot.context["operator_time"] = now
    apply(command, robot.context)
```

Префикс `model` в идентификаторе автора — соглашение для примера; используйте идентификаторы, которые выдаёт ваша платформа.

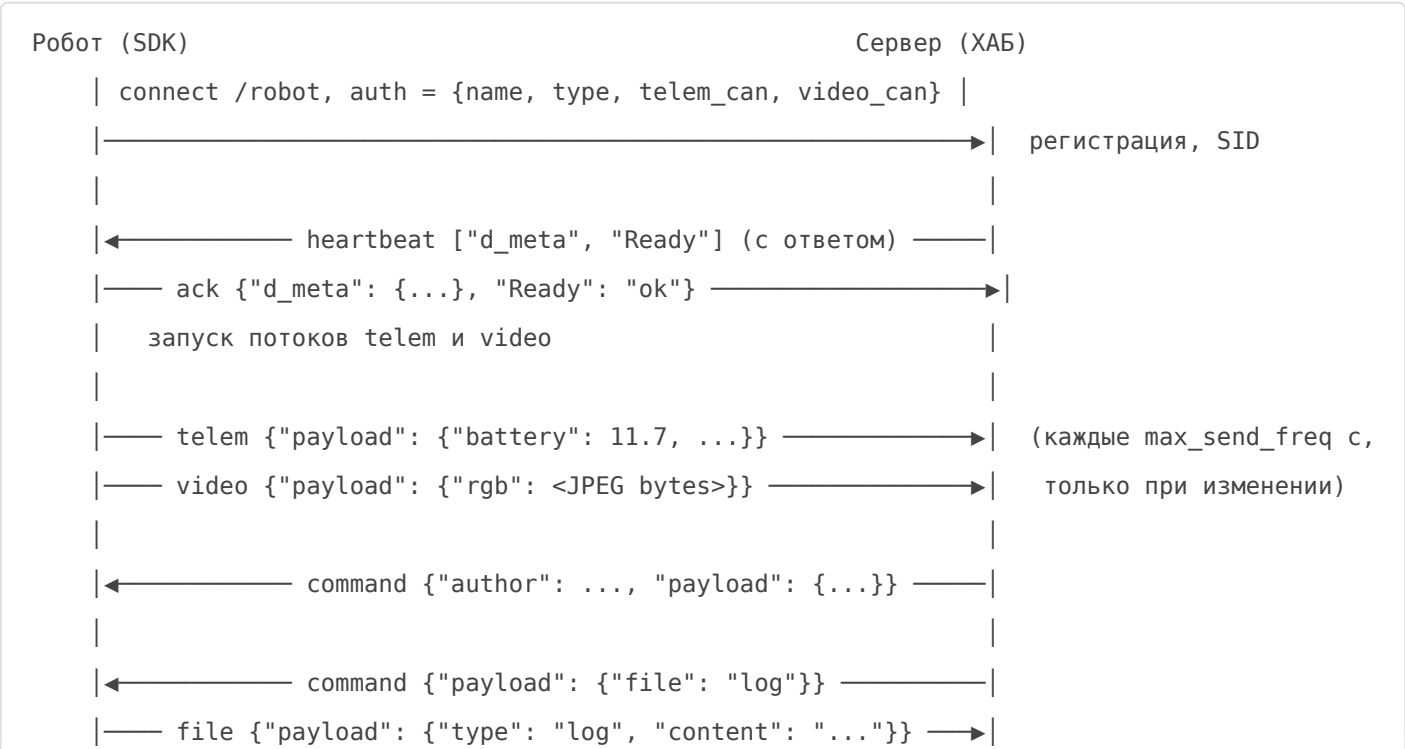
# ????????? ?????? ? ??????????

Описание протокола обмена между SDK `artex-robot` (версия 0.1.1) и сервером ARTEX. Страница полезна тем, кто реализует собственного клиента (на другом языке, для микроконтроллера или симулятора) или серверную часть.

## ??????????

| Параметр        | Значение                                                                                   |
|-----------------|--------------------------------------------------------------------------------------------|
| Протокол        | Socket.IO (Engine.IO поверх WebSocket / HTTP long-polling)                                 |
| Адрес           | <code>&lt;server&gt;:&lt;port&gt;</code> , по умолчанию <code>http://artex.arcon:80</code> |
| Namespace       | <code>/robot</code>                                                                        |
| Авторизация     | Объект <code>auth</code> при подключении = паспорт устройства                              |
| Переподключение | Автоматическое: до 5 попыток, задержка 1 с, максимум 5 с                                   |

## ????????????????????????????????????



|  
| disconnect

??????????

При подключении клиент передаёт в `auth` паспорт устройства:

```
{ "name": "tank-01", "type": "Robot", "telem_can": true, "video_can": true }
```

???????? ?? ????????? ? ???????

`heartbeat` (? ??????????????)

Данные — список запросов. Робот возвращает словарь ответов (Socket.IO ack).

| Запрос                | Ответ                                                                                              | Побочный эффект                                                                                    |
|-----------------------|----------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------|
| <code>"d_meta"</code> | Состояние: <code>{"sid": ..., "ready": "init"   "ok", "telem_run": bool, "video_run": bool}</code> | —                                                                                                  |
| <code>"Ready"</code>  | <code>"ok"</code>                                                                                  | Робот переходит в состояние <code>ready = ok</code> и запускает потоки отправки телеметрии и видео |
| другое                | —                                                                                                  | Предупреждение в журнале                                                                           |

Пример ответа: `{"d_meta": {"sid": null, "ready": "ok", "telem_run": true, "video_run": true}, "Ready": "ok"}`

`token_ask` (? ??????????????)

Запрос идентификатора. Робот возвращает значение `sid` из своего состояния.

`command`

```
{ "author": "<id автора>", "payload": { "<тип>": <команда> } }
```

Обрабатывается только в состоянии `ready = ok`. Типы `file` и `web_settings` обрабатываются SDK; остальные передаются в `@robot.com_handler`. См. [Обработка команд](#).

???????? ?? ?????? ? ????????

## telem

```
{ "payload": { "battery": 11.7, "speed": 0.42, "imu": [0.01, -0.02, 9.81] } }
```

Полный текущий словарь телеметрии. Отправляется, только если с прошлой отправки что-то записывалось в телеметрию.

## video

```
{ "payload": { "rgb": "<bytes: JPEG>", "rear": "<bytes: JPEG>" } }
```

Словарь «камера → кадр». Кадры — JPEG (качество 80) в виде бинарных данных Socket.IO. Для ключей, содержащих `led` или `rgb_strip`, значение передаётся как есть (без JPEG).

## file

Ответ на команду `{"file": "log"}` или `{"file": "error"}`:

```
{ "payload": { "type": "log", "content": "<текст журнала>" } }
```

Команда `{"file": "clear"}` очищает журналы и ответа не отправляет.

????????

- Телеметрия и видео отправляются отдельными потоками с паузой `max_send_freq` секунд (по умолчанию 0,05 с).
- Отправляется только изменённое с момента прошлой отправки, поэтому фактическая частота ограничена частотой основного цикла робота.

???????? ?????????? ????????

Для отладки робота без платформы можно запустить имитатор сервера (Python, `pip install "python-socketio aiohttp"`):

```

import asyncio, socketio
from aiohttp import web

sio = socketio.AsyncServer(async_mode="aiohttp")
app = web.Application(); sio.attach(app)

@sio.on("connect", namespace="/robot")
async def connect(sid, environ, auth):
    print("Подключён:", auth)
    async def handshake():
        await asyncio.sleep(0.5)
        ans = await sio.call("heartbeat", ["d_meta", "Ready"], to=sid, namespace="/robot")
        print("heartbeat:", ans)
        await sio.emit("command", {"author": "tester", "payload": {"com": "forward"}},
                        to=sid, namespace="/robot")
    asyncio.create_task(handshake())

@sio.on("telem", namespace="/robot")
async def telem(sid, data):
    print("telem:", data["payload"])

@sio.on("video", namespace="/robot")
async def video(sid, data):
    print("video:", {cam: len(frame) for cam, frame in data["payload"].items()}, "байт")

web.run_app(app, port=8765)

```

Запустите его и укажите в `params.json` работа `"server": "http://localhost", "port": 8765`.

В целевой архитектуре платформы предусмотрены также передача видео по RTSP с преобразованием в WebRTC и подключение IoT-датчиков по MQTT. Описание — в книге «Сервер управления роботами», страница [Протоколы и потоки данных](#).