MultiVoiceInput — сбор нескольких голосовых сообщений в одну коллекцию

Пакет: `Voice` **Полное имя компонента:** `Common.Voice.MultiVoiceInput` **Текущая версия в приложенном срезе:** `0.1` **Базовый компонент для понимания:** `VoiceInput`

Рекомендуем ознакомиться с базовым уроком по голосовому вводу: VoiceInput.

Что это

`MultiVoiceInput` — это высокоуровневый компонент Metabot для сбора **нескольких голосовых сообщений подряд** внутри сценария.

Он нужен, когда пользователю неудобно или неестественно укладывать весь ответ в одно голосовое.

Например, человек начал рассказывать задачу, потом вспомнил важную деталь, потом отправил ещё одно голосовое, потом ещё одно. Обычный `VoiceInput` ждёт одно сообщение и после распознавания сразу переводит сценарий дальше. `MultiVoiceInput` работает иначе: он собирает несколько голосовых в одну коллекцию, распознаёт каждое, сохраняет общий текст и завершает сбор только после команды вроде “готово”.

Базовый паттерн компонента в исходнике описан так: `collect()` ждёт `voice / video_note / audio`, отправляет файл в STT, добавляет результат в `collectionAttr`, обновляет `fullTextAttr`, ждёт следующее голосовое, а фраза “готово” завершает сбор и переводит сценарий в `successScript`.

Проще говоря:

```
VoiceInput = одно голосовое → один текст → следующий сценарий

MultiVoiceInput = несколько голосовых → коллекция текстов → общий full_text → следующий сценарий
```

Когда использовать MultiVoiceInput

`MultiVoiceInput` нужен там, где пользователь должен рассказать что-то свободно и развёрнуто, но может делать это частями.

Типовые кейсы:

```
AI Intake;
голосовая диагностика бизнеса;
бриф на разработку;
сбор обратной связи;
интервью эксперта;
профилирование пользователя;
разбор проблемы в поддержке;
онбординг партнёра;
подготовка к LLM-анализу;
сбор контекста перед routing / CRM / оффером.
```

Например, в AI Intake мы просим пользователя рассказать, что он хочет улучшить через AI, какие системы уже есть, где теряются деньги, время, заявки или знания. Человек может отправить 2–5 голосовых подряд, а уже после этого сценарий отправит общий текст в `LLMQuery`.

Чем MultiVoiceInput отличается от VoiceInput

`VoiceInput` — это базовый компонент голосового ввода. Он принимает одно голосовое сообщение, распознаёт его и сохраняет результат в один атрибут. Подробнее про философию, пользу, pipeline, настройки канала и базовый processor script смотрите в основной статье:

VoiceInput

`MultiVoiceInput` не заменяет этот урок. Он продолжает ту же архитектурную идею.

Разница такая:

```
VoiceInput:
- один пользовательский голосовой ответ;
- один targetAttr;
- один sourceAttr;
- после STT сразу successScript.

MultiVoiceInput:
- несколько голосовых подряд;
- collectionAttr со списком всех элементов;
- fullTextAttr с общим текстом;
- sourceUrlsAttr со списком источников;
- countAttr с количеством голосовых;
- statusAttr со статусом сбора;
- завершение по finish phrase, лимиту maxItems или лимиту длительности.
```

То есть `MultiVoiceInput` — это не просто “ещё один STT”. Это компонент для **голосовой сессии**, где пользователь может дать контекст кусками.

Общая схема работы

```
Сценарий
→ MultiVoiceInput.collect()
→ ожидание voice / video_note / audio
→ callback от мессенджера
→ получение файла
→ отправка в STT
→ callback от STT
→ добавление item в коллекцию
```

```
→ обновление full_text
→ ожидание следующего голосового
→ пользователь пишет “готово”
→ сбор завершается
→ переход в successScript
```

В рабочем AI Intake reference project компонент используется именно так: сценарий берёт текущий номер раунда, запускает сбор 1–5 голосовых, сохраняет транскрипты в round-specific attrs и после завершения переводит пользователя в сценарий LLM-анализа.

Где находится компонент

Подключение:

```
const MultiVoiceInput = require("Common.Voice.MultiVoiceInput")
```

Запуск:

```
return MultiVoiceInput.collect({...})
```

Есть alias:

```
MultiVoiceInput.start({...})
```

В текущем исходнике `start()` является алиасом на `collect()`.

Что нужно настроить перед использованием

Перед использованием `MultiVoiceInput` должны быть готовы те же базовые вещи, что и для `VoiceInput`.

1. Настройки канала

Для Telegram-контура проверьте, что канал обрабатывает голосовые штатно:

```
Реакция на аудио:
Штатная (NLP и меню)

Реакция на голосовые сообщения:
Штатная (NLP и меню)
```

Это базовое требование голосового ввода: если голосовые не попадают в сценарий, компонент не сможет их обработать. Основная статья по `VoiceInput` отдельно описывает эту настройку канала.

2. Processor script

Нужен системный сценарий, обычно:

```
MultiVoiceInput_Processor
```

Он должен содержать две команды.

3. Bot attrs

Минимально нужны:

```
METABOT_API_TOKEN
METABOT_SERVER_DOMAIN
OPENAI_API_KEY
```

`OPENAI_API_KEY` используется через `tokenKey` в STT-настройках. `METABOT_API_TOKEN` и `METABOT_SERVER_DOMAIN` нужны для callback-механики, как и в обычном `VoiceInput`.

Processor script

`MultiVoiceInput` работает через системный processor script. Он нужен потому, что компонент проходит несколько фаз: сначала ждёт сообщение от пользователя, потом получает файл, потом отправляет его в STT, потом ждёт async callback с текстом.

Processor script должен содержать две команды.

Команда 1. Callback от мессенджера

Тип команды:

```
Run JavaScript Callback
```

Код:

```
const MultiVoiceInput = require("Common.Voice.MultiVoiceInput")


return MultiVoiceInput.onCallback({ lead, isFirstImmediateCall })
```

Эта команда:

```
показывает wait-сообщение;
ждёт голосовое / video_note / audio;
проверяет finish phrases;
проверяет stop phrases;
проверяет help phrases;
проверяет add more phrases;
валидирует файл;
готовит текущий файл для STT.
```

Команда 2. Callback от STT

Тип команды:

```
Run asynchronous API-request
```

Код:

```
const MultiVoiceInput = require("Common.Voice.MultiVoiceInput")


return MultiVoiceInput.onSTT({ lead, isFirstImmediateCall })
```

Эта команда:

```
инициирует STT;
ждёт async response;
получает распознанный текст;
```

```
добавляет item в collection;
обновляет fullTextAttr;
обновляет countAttr;
если лимит не достигнут — возвращает пользователя к ожиданию следующего голосового;
если лимит достигнут — завершает сбор.
```

В reference project `MultiVoiceInput_Processor` устроен именно так: первая команда вызывает `onCallback`, вторая — `onSTT`.

Пример использования

```
const MultiVoiceInput = require("Common.Voice.MultiVoiceInput");

return MultiVoiceInput.collect({
  code: "demand_voice_diagnostic_intake",

  lead,

  successScript: "demand_voice_intake1_analyze",
  cancelScript: "demand_voice_diagnostic_cancelled",
  errorScript: "demand_voice_diagnostic_voice_error",

  collectionAttr: "demand_voice_intake1_items_json",
  fullTextAttr: "demand_voice_intake1_full_text",
  sourceUrlsAttr: "demand_voice_intake1_source_urls_json",
  countAttr: "demand_voice_intake1_count",
  statusAttr: "demand_voice_intake1_status",

  minItems: 1,
  maxItems: 5,
  maxTotalDurationSec: 900,

  extraAttrs: {
    active_agent: "metabot_intake",
    voice_context: "demand_voice_diagnostic",
    input_mode: "multi_voice_intake"
  },
```

```
processorScript: "MultiVoiceInput_Processor",

stt: {
  provider: "openai",
  options: {
    model: "whisper-1",
    language: "ru"
  },
  asyncResponse: true,
  tokenKey: "OPENAI_API_KEY"
},

constraints: {
  allow: {
    voice: true,
    video_note: true,
    audio: false
  },
  minDurationSec: 5,
  maxDurationSec: 600,
  maxFileSizeBytes: 20 * 1024 * 1024
},

messages: {
  wait: "□□Пришлите голосовое сообщение. Можно несколько подряд.",
  accepted: "□ Принял голосовое. Расшифровываю…",
  afterItem: "Принял. Можете прислать ещё одно голосовое или написать «готово».",
  wrong: "Нужна голосовуха. Если закончили — напишите «готово».",
  empty: "Пока нет ни одного голосового. Пришлите хотя бы одно сообщение.",
  canceled: "Ок, остановились.",
  finished: "Спасибо. Собрал контекст.",
  tooMany: "Я уже собрал достаточно материала. Сейчас соберу картину.",
  stillProcessing: "□ Ещё обрабатываю предыдущее голосовое…"
},

finishPhrases: [
  "готово",
  "всё",
  "все",
  "закончил",
```

```
    "закончила",
    "/done"
  ],

  stopPhrases: [
    "стоп",
    "stop",
    "отмена",
    "cancel",
    "я передумал",
    "/cancel"
  ]
});
```

В текущем исходнике обязательными являются `processorScript`, `successScript`, `cancelScript`, `collectionAttr`, `fullTextAttr` и `stt.tokenKey`; также проверяется, что `minItems >= 1`, а `maxItems >= minItems`.

Важное замечание по свежести примеров

В некоторых переходных примерах может встречаться вложенный блок:

```
voiceInput: {
  processorScript: "MultiVoiceInput_Processor",
  stt: {...},
  constraints: {...}
}
```

Но в текущем срезе компонента `Common.Voice.MultiVoiceInput` основные настройки `processorScript`, `stt` и `constraints` находятся на верхнем уровне конфигурации. Поэтому при новом внедрении сверяйте вызов с актуальной версией плагина на платформе.

Для production используйте свежую версию компонента из платформы, а приложенный reference project воспринимайте как рабочий срез на дату.

Что сохраняет компонент

`MultiVoiceInput` сохраняет несколько типов результата.

collectionAttr

JSON-массив всех распознанных голосовых.

Пример:

```
[
  {
    "index": 1,
    "type": "voice",
    "text": "Хочу внедрить AI в продажи...",
    "source_url": "https://...",
    "duration_sec": 42,
    "file_size": 123456,
    "file_id": "abc",
    "created_at": "2026-06-03T...",
    "transcribed_at": "2026-06-03T..."
  },
  {
    "index": 2,
    "type": "voice",
    "text": "Ещё важно, что у нас есть CRM...",
    "source_url": "https://...",
    "duration_sec": 31,
    "file_size": 100000,
    "file_id": "def",
    "created_at": "2026-06-03T...",
    "transcribed_at": "2026-06-03T..."
  }
]
```

В исходнике каждый распознанный элемент добавляется как item с индексом, типом файла, текстом, ссылкой на источник, длительностью, размером, file_id и timestamp-полями.

fullTextAttr

Обычный текстовый атрибут, где все голосовые объединены в один текст.

Формат текущей сборки:

```
[Голосовое 1]
Текст первого голосового

[Голосовое 2]
Текст второго голосового
```

Компонент строит `fullTextAttr` из массива `items`, соединяя распознанные тексты через пустую строку.

sourceUrlsAttr

JSON-массив ссылок на исходные голосовые файлы.

countAttr

Количество принятых голосовых.

statusAttr

Текущий статус сбора.

Возможные статусы:

```
active
collecting
completed
cancelled
```

Основные параметры collect()

code

Уникальный код сессии сбора.

Пример:

```
code: "demand_voice_diagnostic_intake"
```

Для повторяемых раундов удобно использовать динамический код:

```
const roundCode = `demand_intake_round${currentRound}`;
```

lead

Обязательный объект текущего лида.

```
lead
```

Если `lead` не передан, компонент выбросит ошибку.

processorScript

Код системного processor script.

```
processorScript: "MultiVoiceInput_Processor"
```

successScript

Сценарий, куда перейти после успешного завершения сбора.

```
successScript: "demand_intake_analyze_round"
```

cancelScript

Сценарий, куда перейти при отмене.

```
cancelScript: "demand_intake_cancelled"
```

errorScript

Сценарий для ошибки. В текущей конфигурации параметр есть в API и используется как часть маршрутов, но конкретную обработку ошибок нужно проверять по актуальной версии компонента.

collectionAttr

JSON-атрибут для массива всех голосовых.

```
collectionAttr: "demand_intake_round1_items_json"
```

fullTextAttr

Текстовый атрибут с объединённой расшифровкой.

```
fullTextAttr: "demand_intake_round1_full_text"
```

sourceUrlsAttr

JSON-атрибут со ссылками на исходные файлы.

```
sourceUrlsAttr: "demand_intake_round1_source_urls_json"
```

countAttr

Атрибут с количеством голосовых.

```
countAttr: "demand_intake_round1_count"
```

statusAttr

Атрибут со статусом.

```
statusAttr: "demand_intake_round1_status"
```

extraAttrs

Дополнительные атрибуты, которые компонент устанавливает в lead.

Пример:

```
extraAttrs: {
  active_agent: "metabot_intake",
  voice_context: "demand_intake_loop",
  input_mode: "multi_voice_intake_round"
}
```

В reference project через `extraAttrs` также сохраняются контекст раунда и entry-данные вроде route, funnel_id и campaign.

Лимиты

minItems

Минимальное количество голосовых, которое нужно принять до завершения.

```
minItems: 1
```

Если пользователь напишет “готово” до первого голосового, компонент покажет `messages.empty`.

maxItems

Максимальное количество голосовых.

```
maxItems: 5
```

Если пользователь достиг лимита, компонент показывает `messages.tooMany` и завершает сбор автоматически.

maxTotalDurationSec

Максимальная суммарная длительность всех голосовых.

```
maxTotalDurationSec: 900
```

Это полезно, чтобы пользователь не отправил слишком много материала в один раунд.

constraints.minDurationSec

Минимальная длительность одного файла.

```
minDurationSec: 5
```

constraints.maxDurationSec

Максимальная длительность одного файла.

```
maxDurationSec: 600
```

constraints.maxFileSizeBytes

Максимальный размер файла.

```
maxFileSizeBytes: 20 * 1024 * 1024
```

Компонент проверяет размер, минимальную и максимальную длительность файла и возвращает пользователю понятное предупреждение, если файл не проходит ограничения.

Типы файлов

По умолчанию:

```
constraints: {
  allow: {
    voice: true,
    video_note: true,
    audio: false
```

```
  }
}
```

Это значит:

```
voice — принимаем;
video_note — принимаем;
audio — не принимаем.
```

Если нужно принимать обычные аудиофайлы, включите:

```
audio: true
```

Но для Telegram voice-first сценариев обычно лучше начинать с `voice` и `video_note`.

STT-настройки

Пример:

```
stt: {
  provider: "openai",
  tokenKey: "OPENAI_API_KEY",
  asyncResponse: true,
  options: {
    model: "whisper-1",
    language: "ru"
  }
}
```

Поля:

```
provider — STT-провайдер;
tokenKey — имя bot attr, где лежит ключ провайдера;
asyncResponse — использовать async callback;
options.model — модель распознавания;
options.language — язык.
```

Если `stt.tokenKey` не указан, компонент выбросит ошибку конфигурации.

UX-сообщения

Блок `messages` отвечает за пользовательские тексты.

Основные поля:

```
messages: {
  wait: "□□Пришлите голосовое сообщение. Можно несколько подряд.",
  accepted: "□ Принял голосовое. Расшифровываю…",
  afterItem: "Принял. Можете прислать ещё одно голосовое или написать «готово».",
  wrong: "Нужна голосовуха. Если закончили — напишите «готово». Если хотите отменить — напишите «стоп».",
  empty: "Пока нет ни одного голосового. Пришлите хотя бы одно сообщение.",
  canceled: "Ок, остановились.",
  finished: "Спасибо. Собрал контекст. Сейчас разложу задачу по карте.",
  tooMany: "Я уже собрал достаточно материала. Сейчас соберу картину.",
  tooLongTotal: "Материала уже достаточно по длительности. Сейчас соберу картину.",
  stillProcessing: "□ Ещё обрабатываю предыдущее голосовое…",
  transcriptionEmpty: "⚠□ Не удалось получить текст из голосового. Попробуйте записать ещё раз.",
  help: "..."
}
```

Сценарист может переопределить только нужные сообщения. Остальные возьмутся из defaults.

Управляющие фразы

finishPhrases

Фразы завершения сбора:

```
finishPhrases: [
  "готово",
  "всё",
  "все",
```

```
  "закончил",
  "закончила",
  "/done"
]
```

Когда пользователь пишет такую фразу, компонент проверяет, есть ли минимум `minItems`, завершает сбор, ставит статус `completed` и запускает `successScript`.

stopPhrases

Фразы отмены:

```
stopPhrases: [
  "стоп",
  "stop",
  "отмена",
  "cancel",
  "я передумал",
  "/cancel"
]
```

При такой фразе компонент ставит статус `cancelled`, очищает активную конфигурацию и запускает `cancelScript`.

addMorePhrases

Фразы “добавить ещё”:

```
addMorePhrases: [
  "добавить ещё",
  "добавить еще",
  "еще",
  "ещё"
]
```

Компонент просто продолжает ждать голосовое.

helpPhrases

Фразы помощи:

```
helpPhrases: [
  "как записать голосовое",
  "помощь",
  "help",
  "/help"
]
```

Компонент отправляет `messages.help`.

В исходнике фразы нормализуются: текст приводится к нижнему регистру, лишние символы убираются, поддерживается точное совпадение и совпадение по началу фразы, чтобы варианты вроде “готово, собрать картину” тоже могли сработать.

Пример для AI Intake-раунда

```
const MultiVoiceInput = require("Common.Voice.MultiVoiceInput");

const currentRound = parseInt(lead.getAttr("demand_intake_round") || "1");
const roundCode = `demand_intake_round${currentRound}`;

return MultiVoiceInput.collect({
  code: roundCode,

  lead,

  successScript: "demand_intake_analyze_round",
  cancelScript: "demand_intake_cancelled",
  errorScript: "demand_intake_voice_error",

  collectionAttr: `demand_intake_round${currentRound}_items_json`,
  fullTextAttr: `demand_intake_round${currentRound}_full_text`,
  sourceUrlsAttr: `demand_intake_round${currentRound}_source_urls_json`,
  countAttr: `demand_intake_round${currentRound}_count`,
  statusAttr: `demand_intake_round${currentRound}_status`,

  minItems: 1,
```

```
maxItems: 5,
maxTotalDurationSec: 900,

extraAttrs: {
  active_agent: "metabot_intake",
  voice_context: "demand_intake_loop",
  input_mode: "multi_voice_intake_round",
  demand_intake_round: currentRound,
  demand_intake_round_code: roundCode
},

processorScript: "MultiVoiceInput_Processor",

stt: {
  provider: "openai",
  options: {
    model: "whisper-1",
    language: "ru"
  },
  asyncResponse: true,
  tokenKey: "OPENAI_API_KEY"
},

constraints: {
  allow: {
    voice: true,
    video_note: true,
    audio: false
  },
  minDurationSec: 5,
  maxDurationSec: 600,
  maxFileSizeBytes: 20 * 1024 * 1024
},

messages: {
  wait: "□□Ответьте голосом. Можно несколькими сообщениями подряд.",
  accepted: "□ Принял голосовое. Расшифровываю…",
  afterItem: "Принял. Можете записать ещё одно голосовое или нажать/написать «готово».",
  wrong: "Нужна голосовуха. Если хотите остановиться — нажмите/напишите «готово».",
  empty: "Пока нет ни одного голосового. Пришлите хотя бы одно сообщение.",
```

```
    canceled: "Ок, остановились.",
    finished: "Спасибо. Собрал контекст.",
    tooMany: "Я уже собрал достаточно материала. Сейчас обновлю картину.",
    stillProcessing: "⏳ Ещё обрабатываю предыдущее голосовое…"
  },

  finishPhrases: [
    "готово",
    "всё",
    "все",
    "закончил",
    "закончила",
    "дальше",
    "давай дальше",
    "/done"
  ]
});
```

В AI Intake reference project этот подход используется для каждого раунда: номер раунда читается из `demand_intake_round`, а результаты пишутся в `demand_intake_round{N}_items_json`, `demand_intake_round{N}_full_text`, `demand_intake_round{N}_source_urls_json`, `demand_intake_round{N}_count` и `demand_intake_round{N}_status`.

Что делать после MultiVoiceInput

После успешного завершения сбора обычно запускается сценарий анализа.

Например:

```
demand_intake_analyze_round
```

Он берёт:

```
demand_intake_round{N}_full_text
```

и отправляет этот текст в `LLMQuery`.

То есть `MultiVoiceInput` сам не анализирует смысл. Он только собирает и распознаёт голосовые. Анализ, JSON-сигнатура, follow-up, routing и CRM — это следующие слои сценария.

Правильное разделение:

```
MultiVoiceInput — собрать голосовые и full_text.
LLMQuery — понять смысл и вернуть JSON.
After-analysis — показать пользователю локализованный результат.
Routing — выбрать следующий шаг.
CRM / менеджер — получить внутренний бриф.
```

Как отлаживать

Отлаживать нужно по слоям.

1. Пользователь попал в collect script?

Проверить, что запустился сценарий, где вызывается:

```
MultiVoiceInput.collect({...})
```

2. Активная конфигурация создана?

Проверить lead attr:

```
__active_multi_voice_input
```

Если его нет, сбор не запущен или уже очищен.

3. Processor script существует?

Проверить сценарий:

```
MultiVoiceInput_Processor
```

В нём должны быть две команды: `onCallback` и `onSTT`.

4. Голосовое принято?

Проверить:

```
collectionAttr
countAttr
statusAttr
```

Если `countAttr = 0`, голосовое не добавилось в коллекцию.

5. STT вернул текст?

Проверить:

```
fullTextAttr
collectionAttr
```

Если в `collectionAttr` есть item без текста или `fullTextAttr` пустой, проблема на STT-слое.

6. Сохранились source URLs?

Проверить:

```
sourceUrlsAttr
```

Если пусто, возможно, проблема с получением ссылки на файл.

7. Завершение сработало?

Пользователь пишет:

```
готово
```

Проверить:

```
statusAttr = completed
```

и что запустился `successScript`.

8. Если пользователь пишет “готово”, но сбор не завершается

Проверить:

```
finishPhrases;
minItems;
countAttr;
__active_multi_voice_input;
processorScript;
onCallback.
```

Если `minItems = 1`, а голосовых ещё нет, компонент должен показать `messages.empty`.

9. Если бот пишет “нужна голосовуха”

Проверить:

```
тип входящего сообщения;
constraints.allow.voice;
constraints.allow.video_note;
constraints.allow.audio;
настройки канала;
payload мессенджера.
```

10. Если STT не стартует

Проверить:

```
stt.provider;
stt.tokenKey;
bot attr OPENAI_API_KEY;
Run asynchronous API-request;
METABOT_API_TOKEN;
METABOT_SERVER_DOMAIN.
```

Типовые ошибки

Ошибка 1. Создали collect script, но забыли processor

Без `MultiVoiceInput_Processor` компонент не сможет принять голос и получить STT callback.

Ошибка 2. Настроили processor как обычный JavaScript

Первая команда должна быть `Run JavaScript Callback`, вторая — `Run asynchronous API-request`.

Ошибка 3. Не задали collectionAttr или fullTextAttr

Компонент требует эти поля, потому что ему нужно сохранить коллекцию и общий текст.

Ошибка 4. Пользователь пишет “готово”, но ещё не отправил голосовое

При `minItems: 1` компонент не завершит сбор и покажет `messages.empty`.

Ошибка 5. Не локализовали UX под конкретный сценарий

Default-сообщения подходят для тестов, но в production лучше писать контекстно:

```
“Расскажите задачу голосом. Можно несколькими сообщениями подряд.”
```

Ошибка 6. Смешали MultiVoiceInput и LLM-анализ

`MultiVoiceInput` не должен решать бизнес-задачу. Его задача — собрать голосовые и подготовить текст.

Ошибка 7. Не проверили свежую версию компонента

Приложенный код — срез. Перед внедрением сверяйте актуальную версию плагина и параметры вызова.

Короткий production checklist

1. Канал принимает voice / video_note.
2. Создан MultiVoiceInput_Processor.
3. В processor есть onCallback.
4. В processor есть onSTT.
5. Настроены METABOT_API_TOKEN и METABOT_SERVER_DOMAIN.
6. Настроен OPENAI_API_KEY или другой tokenKey.
7. В collect() указан lead.
8. Указан successScript.
9. Указан cancelScript.
10. Указан collectionAttr.
11. Указан fullTextAttr.
12. Указаны sourceUrlsAttr / countAttr / statusAttr, если нужны для отладки.
13. minItems >= 1.
14. maxItems >= minItems.
15. Настроены finishPhrases.
16. Настроены stopPhrases.
17. Проверено, что fullTextAttr собирается из нескольких голосовых.
18. Проверено, что “готово” переводит в successScript.
19. Проверено, что после successScript данные уходят в LLMQuery / следующий сценарий.

Главный вывод

`MultiVoiceInput` нужен, когда одного голосового недостаточно.

Он позволяет превратить живой человеческий рассказ, разбитый на несколько сообщений, в один структурированный вход для следующего сценарного слоя.

Формула:

```
несколько голосовых
→ STT каждого сообщения
→ collectionAttr
→ fullTextAttr
→ successScript
→ LLMQuery / routing / CRM
```

Базовую философию голосового интерфейса, настройки канала и общую механику `VoiceInput` смотрите в основной документации:

```
https://docs.metabot24.ru/books/06-ai-komponenty-metabot-agent-stack/page/voice-input-golosovoi-interfeis-dlya-ai-voronok
```

`VoiceInput` даёт один голосовой ответ. `MultiVoiceInput` даёт голосовую сессию.

Версия #1
Artem Garashko создал 2026-06-03 17:10:41 UTC
Artem Garashko обновил 2026-06-04 12:59:13 UTC