ASM STT v3.3.0 API

1. Передача данных

API поддерживает REST синхронные и асинхронные запросы, передачу JSON данных, поддержку формата msgpack.

> ❗ Данные, запакованные msgpack из **python 2.x** - не поддерживаются
> ❗ При передаче данных в JSON, без запаковки msgpack - бинарные данные должны быть закодированы base64

Заголовок Content-Type

```
Content-Type: application/json
Content-Type: application/x-msgpack
```

Заголовок Accept

```
Accept: application/json
Accept: application/x-msgpack
```

Если заголовки не указаны

- формат входных данных определяется автоматически
- формат ответа соответствует формату входящего запроса
- для GET запросов ответ выдаётся как plaintext

GET запросы

Для GET запросов вместо заголовков можно указать тип данных в url.

```
http://<address>/<method>?format=msgpack
http://<address>/<method>?format=json
```

Аудио-данные

Сервис работает с форматом **pcm_s16le, mono, 8000 Hz, 16 bit, 128 kb/s**.
Если не указано отдельно - входные данные переконвертируются в нужный формат с помощью ffmpeg.

> ❗ Хранение и передача данных в формате отличном от вышеуказанного - может ухудшить качество распознавания.

2. Получение данных

Если запрос неуспешен - HTTP код ответа соответствует ошибке, в теле ответа - подробное описание ошибки.
Разметка в данных возвращается во фреймах.

> ❗ 1 секунда = 8000 фреймов

3. Методы информации о сервисе

3.1 Список доступных методов и моделей (метод /models)

Получить список доступных методов и моделей можно с помощью GET или POST запросов.

запрос GET:

```
HOST = '127.0.0.1:8072'
```

```
APIKEY = 'apikey'

url = 'http://{}/models?format=msgpack&apikey={}'.format(HOST, APIKEY)
r = requests.get(url)

# получение результата
if r.status_code == 200:
    result = msgpack.unpackb(r.content, raw=False)
```

запрос POST:

```
HOST = '127.0.0.1:8072'
APIKEY = 'apikey'

url = 'http://{}/models'.format(HOST)
data = {
    'apikey': str,  # Персональный ключ
    # далее опционально
    'api': str,     # Имя метода, (по умолчанию результат вернётся для всех доступных методов)
}
packed_data = msgpack.packb(data, use_bin_type=True)
r = requests.post(url, data=packed_data)

# получение результата
if r.status_code == 200:
    result = msgpack.unpackb(r.content, raw=False)
```

результат:

```
#Если не указано имя метода – результат: словарь словарей
{
    method_name: {
        model_name: threads,
        ...
    }, ...
}

#Если указано имя метода – результат: словарь
{
    model_name: threads,
    ...
}
```

4. Методы распознавания речи

4.1 Формат фрагмента обработанных данных

```
{
    "parent_id": str,        # [опционально] id запроса, полученный от клиента
    "task_id": str,          # id запроса во фронтенде (балансировка)
    "sub_id": str,           # id запроса в бэкэнде (распознавание)
    "sub_no": int,           # [только в /chunked_stream] соответствие номеру отправленного пакета данных

    "speech": bool,          # была ли найдена речь во фрагменте
    "error": bool,           # [опционально] ставится если при распознавании данного фрагмента произошла ошибка
    "br": bool,              # [только в /chunked_stream] окончание фразы, финальная гипотеза
    "last": bool,            # [только в /chunked_stream и /phrases] последний пакет данных

    "data": str,             # [опционально] звуковые данные, если в запросе include_data:true

    "timestamp": int,        # [кроме /chunked_stream] начало фрагмента (фреймы)
    "duration": int,         # [кроме /chunked_stream] длительность фрагмента (фреймы)

    "speec_info": {          # содержит разметку речи, разметку VAD:
        "name": str,         # имя типа разметки; например: speech, __noise, online
        "vad": {             # [опционально] описание разметки VAD, если в запросе vad_results:true
            "duration": int, # длительность фрагмента (фреймы)
            "marking": [     # разметка VAD по моделям:
                [ int,       # начало (фреймы)
                  int,       # конец (фреймы)
                  str        # имя модели
```

```
                ], ...
            ]
        },
        "nbest": {           # [опционально] 2-й и последующие варианты распознавания фрагмента
            "score": float,  # confidence данного варианта распознавания (мера уверенности распознавания)
            "duration": int, # длительность речи (фреймы)
            "marking": [     # разметка фрагмента по словам (может быть пустым списком []):
                [ int,       # начало (фреймы)
                  int,       # конец (фреймы)
                  str,       # слово
                  float      # confidence данного слова
                ], ...
            ],
            "text": str      # текст данного варианта распознавания
        },
        "speech": {          # [опционально] описание разметки фрагмента
            "score": float,  # confidence данного фрагмента (мера уверенности распознавания)
            "duration": int, # длительность речи (фреймы)
            "marking": [     # разметка фрагмента по словам (может быть пустым списком []):
                [ int,       # начало (фреймы)
                  int,       # конец (фреймы)
                  str,       # слово
                  float      # confidence данного слова
                ], ...
            ]
        },
        "text": str         # [опционально] распознанный текст во фрагменте
    }
}
```

4.2 Распознавание речи в файле (метод /file)

4.2.1 Синхронный или асинхронный запрос

По умолчанию запрос является синхронным, блокируется до полного распознавания файла и возвращает весь результат.
Асинхронный запрос сразу возвращает идентификатор созданной задачи, по которому надо получать статус и результат.
Для асинхронного запроса необходимо указать **'async': True**.

запрос:

```
url = 'http://{}/file'.format(HOST)
data = {
    'apikey': str,          # Персональный ключ
    'model':  str,          # Имя модели распознавания
    'wav' :   bytes,        # Бинарный контент аудиофайла
    # далее опционально
    'vad_model': str,       # Имя модели VAD, (без указания — данные нарезаются по 1 минуте)
    'async':  bool,         # Асинхронный вариант метода, (по умолчанию False)
    'parent_id': str,       # id запроса от клиента, фигурирует в логах и результатах
    'priority': int,        # Приоритет запроса во входной очереди 2..10 (по умолчанию 10 (низкий))
    'post_callback': str,   # Вызывается на каждый фрагмент; пример: 'http://127.0.0.1/callback'
    'include_data': bool,   # Включать в результат поле data (по умолчанию False)
    'postprocessing': bool, # Использовать ли постпроцессинг (по умолчанию True для русского, False для остальных
    'nbest': int,           # Сколько nbest вариантов фраз выводить (по умолчанию 1)
    'vad_results': bool,    # Возвращать ли результат VAD (по умолчанию False)
    'make_utterance': bool, # Ставить ли заглавную и точку при отключенном постпроцессинге (по умолчанию False)
}
packed_data = msgpack.packb(data, use_bin_type=True)

r = requests.post(url, data=packed_data)
if r.status_code == 200:
    result = msgpack.unpackb(r.content, raw=False)
```

результат:

Результатом синхронного варианта является список фрагментов обработанных данных.
Результатом асинхронного варианта является идентификатор задачи (task_id), для последующих вызовов.

4.2.2 Асинхронное получение результата

Каждый запрос на статус или результат должен содержать идентификатор задачи (task_id).

запрос статуса задачи (/task/status)

```
url = 'http://{}/task/status'.format(HOST)
data = {
    'apikey': str,  # Персональный ключ
    'task_id': str, # Идентификатор, полученный из запроса /file
}
packed_data = msgpack.packb(data, use_bin_type=True)

r = requests.post(url, data=packed_data)
if r.status_code == 200:
    result = msgpack.unpackb(r.content, raw=False)
```

результат:

Результатом является статус задачи в данный момент.

```
{
    'registered': bool, # True – задача зарегистрирована в системе
    'queue': bool,      # True – задача в очереди на исполнение
    'done': bool,       # True – задача выполнена
}
```

запрос результата задачи (/task/result)

```
url = 'http://{}/task/result'.format(HOST)
data = {
    'apikey': str,  # Персональный ключ
    'task_id': str, # Идентификатор, полученный из запроса /file
}
packed_data = msgpack.packb(data, use_bin_type=True)

r = requests.post(url, data=packed_data)
if r.status_code == 200:
    result = msgpack.unpackb(r.content, raw=False)
```

результат:

Результат аналогичен синхронному запросу (список фрагментов обработанных данных).

4.2.3 Получение результата через POST-callback

Результат также может быть получен, по мере выполнения, через POST-callback на другом сервере/порту.

ожидание результата:

```
from http.server import BaseHTTPRequestHandler, HTTPServer
import msgpack

class Handler(BaseHTTPRequestHandler):
    def do_POST(self):
        content_length = int(self.headers['Content-Length'])
        raw_data = self.rfile.read(content_length)
        result = msgpack.unpackb(raw_data, raw=False)
        return

HTTPServer(('', PORT), Handler).serve_forever()
```

результат:

Результатом каждого POST-callback является фрагмент обработанных данных.

4.3 Распознавание фраз в потоке (метод /phrases)

Позволяет распознавать поток аудиоданных и получать результаты по фрагментам обработанных данных (фразам) с минимальной задержкой.

> ❗ Timeout бездействия фразового декодера - 20сек, после которых он автоматически освобождается с удалением результатов

4.3.1 Резервирование декодера (/phrases)

запрос:

```
url = 'http://{}/phrases'.format(HOST)
data = {
    'apikey': str,          # Персональный ключ
    'model':  str,          # Имя модели распознавания
    # далее опционально
    'vad_model': str,       # Имя модели VAD, (по умолчанию данные нарезаются по 1 минуте)
    'parent_id': str,       # id запроса от клиента, фигурирует в логах и результатах
    'post_callback': str,   # Вызывается на каждый фрагмент; пример: 'http://127.0.0.1/callback'
    'include_data': bool,   # Включать в результат поле data (по умолчанию False)
    'postprocessing': bool, # Использовать ли постпроцессинг (по умолчанию True для русского, False для остальных
    'nbest': int,           # Сколько nbest вариантов фраз выводить (по умолчанию 1)
    'make_utterance': bool, # Ставить ли заглавную и точку при отключненном постпроцессинге (по умолчанию False)
}

r = requests.post(url, json=data)
if r.status_code == 200:
    request_id = r.json()
```

результат:

Результатом резервирования является идентификатор декодера (request_id) для последующих вызовов.

4.3.2 Передача данных (/phrases/feed)

Каждый запрос должен содержать:

- apikey
- request_id идентификатор декодера
- wav аудиоданные без заголовка, в формате **моно 8кГц 16бит**
- packet_id порядковый номер пакета данных (нумерация пакетов начинается с 0)

запрос:

```
url = 'http://{}/phrases/feed'.format(HOST)
data = {
    'apikey': str,                  # Персональный ключ
    'request_id': str,              # Идентификатор, полученный при инициализации декодера
}

packet_id = 0
while True:
    frames = getframes()
    if not frames:
        break

    data['wav'] = base64.b64encode(frames).decode()
    data['packet_id'] = packet_id

    r = requests.post(url, json=data)
    assert(r.status_code == 200)
    packet_id += 1
```

4.3.3 Получение результатов потокового декодера (/phrases/results)

Результаты должны периодически запрашиваться вручную.

```
url = 'http://{}/phrases/results'.format(HOST)
data = {
    'apikey': str,      # Персональный ключ
    'request_id': str,  # Идентификатор, полученный при инициализации декодера
}

r = requests.post(url, json=data)
if r.status_code == 200:
    result = r.json()
```

результат:

Возвращается список фрагментов, декодированный с момента последнего их получения.
Формат результата аналогичен результату метода /file.
Последним фрагментом в результатах возвращается **{"last": True}**.

4.3.4 Получение результата через POST-callback

Результат так же может быть получен по мере выполнения, через POST-callback на другом сервере/порту.
Использование идентично методу /file.

4.3.5 Освобождение декодера (/phrases/stop)

запрос:

```
url = 'http://{}/phrases/stop'.format(HOST)
data = {
    'apikey': str,      # Персональный ключ
    'request_id': str,  # Идентификатор, полученный при инициализации декодера
}

r = requests.post(url, json=data)
assert(r.status_code == 200)
```

4.4 Онлайн распознавание фраз в http-потоке (метод /chunked_stream)

Позволяет распознавать поток аудиоданных и получать результаты по фрагментам обработанных данных (фразам) с минимальной задержкой.
Для открытия http-потока клиент должен поддерживать Chunked transfer encoding (RFC 7230 section 4.1).

> ❗ Timeout бездействия онлайн-декодера - 3сек, после которых он автоматически освобождается с удалением результатов

4.4.1 Резервирование декодера (/chunked_stream)

запрос:

```
url = 'http://{}/chunked_stream'.format(HOST)
data = {
    'apikey': APIKEY,
    'model': MODEL,
    'vad_model': VAD_MODEL,
    # далее опционально
    'parent_id': str,             # id запроса от клиента, фигурирует в логах и результатах
    'nbest': int,                 # Сколько nbest вариантов фраз выводить (по умолчанию 1)
    'online_return_phrase': bool, # Отдавать результат стабильными фразами (по умолчанию False)
}

r = requests.post(url, json=data)
assert r.status_code == 200
request_id = r.json()
```

результат:

Результатом резервирования является идентификатор декодера (request_id) для последующих вызовов.

4.4.2 Передача данных (/chunked_stream/feed)

> ❗ Параметры apikey и request_id передаются в URI

Все данные должны быть отправлены в пределах одного соединения с использованием chunked transfer encoding:

Схематическое представление: `<длина_блока_в_HEX>\0x0D\0x0A<содержание_блока>\0x0D\0x0A`
Последний блок строится по той же схеме: `0\0x0D\0x0A\0x0D\0x0A`
Содержание блока - аудиоданные в формате **моно 8кГц 16бит**, закодированные в **base64**.

запрос (с использованием фреймворка requests):

```
url = 'http://{}/chunked_stream/feed?apikey={}&request_id={}'.format(HOST, APIKEY, request_id)
data_generator = (b64encode(chunk) for chunk in split_wav(WAV))

r = requests.post(url, data=data_generator, stream=True)
assert r.status_code == 200
```

запрос (реализация без использования фреймворка)

```
from http.client import HTTPConnection
conn = HTTPConnection(HOST)
conn.connect()
conn.putrequest('POST', '/chunked_stream/feed?apikey={}&request_id={}'.format(APIKEY, request_id))
conn.putheader('Transfer-Encoding', 'chunked')
conn.endheaders()
last_chunk = len(wav_chunked) - 1
for i, chunk in enumerate(wav_chunked):
    chunk_encoded = base64.b64encode(chunk)
    chunk_body = hex(len(chunk_encoded))[2:].encode() + b'\r\n' + chunk_encoded + b'\r\n'
    conn.send(chunk_body)
    if i == last_chunk:
        break
    sleep(SPLIT_SEC)
conn.send(b'0\r\n\r\n')
r = conn.getresponse()
print(r.status, r.reason, r.read())
```

4.4.3 Получение результатов потокового декодера (/chunked_stream/results)

> ❗ Запрос GET, параметры apikey и request_id передаются в URI

```
url = 'http://{}/chunked_stream/results?apikey={}&request_id={}'.format(HOST, APIKEY, request_id)

r = requests.get(url, stream=True)
for res in r.iter_content(None):
    result = json.loads(res)
```

результат:

В chunked потоке возвращаются гипотезы фрагментов фраз.
Формат результата аналогичен результату метода /file:

Получение **'br': True** означает:

- завершение фразы, финальная гипотеза
- заполнено поле ['speec_info']['text']
- заполнены confidence коэффициенты

Получение **'last' : True** означает:

- последний пакет
- результаты распознавания пустые
- декодер освобождён

4.4.4 Получение результата через POST-callback

Результат так же может быть получен по мере выполнения, через POST-callback на другом сервере/порту.
Использование идентично методу /file.

4.4.5 Освобождение декодера (/chunked_stream/stop)

запрос:

```
url = 'http://{}/chunked_stream/stop'.format(HOST)
data = {
    'apikey': APIKEY,
    'request_id': request_id,
}
r = requests.post(url, json=data)
assert r.status_code == 200
```

5. Методы обработки речевого сигнала

5.1 Определение языка речи (метод /lis)

запрос:

```
url = 'http://{}/lis'.format(HOST)
data = {
    'apikey':   str,       # Персональный ключ
    'wav' :     bytes,     # Бинарный контент аудиофайла
    # далее опционально
    'model':    str,       # Имя модели распознавания (по умолчанию lis_common)
    'async':    bool,      # Асинхронный вариант, (по умолчанию False)
    'langs':    list(str), # Фильтр по языкам, (по умолчанию все языки)
}
packed_data = msgpack.packb(data, use_bin_type=True)

r = requests.post(url, data=packed_data)
if r.status_code == 200:
    result = msgpack.unpackb(r.content, raw=False)
```

результат:

Результатом асинхронного варианта является task_id для последующих вызовов.
Использование идентично методу /file.

Результатом синхронного является список списков вида:

```
[ [ str,    # имя языка в формате ISO 639-3
    float   # степень соответствия
  ], ...
]
```

5.2 Разделение на дикторов (метод /ldiariz)

> ❗ deprecated

запрос:

```
url = 'http://{}/ldiariz'.format(HOST)
data = {
    'apikey':    str,   # Персональный ключ
    'wav' :      bytes, # Бинарный контент аудиофайла
    # далее опционально
    'async':     bool,  # Асинхронный вариант, (по умолчанию False)
}
packed_data = msgpack.packb(data, use_bin_type=True)

r = requests.post(url, data=packed_data)
if r.status_code == 200:
    result = msgpack.unpackb(r.content, raw=False)
```

результат:

Результатом асинхронного варианта является task_id для последующих вызовов (использование идентично методу /file).
Результатом синхронного является список списков вида:

```
[ [ int,    # начало (фреймы)
    int,    # конец (фреймы)
    str     # номер диктора
  ], ...
]
```

5.3 Разделение на дикторов (метод /mdiariz)

запрос:

```
url = 'http://{}/mdiariz'.format(HOST)
data = {
    'apikey':    str,   # Персональный ключ
    'wav' :      bytes, # Бинарный контент аудиофайла
    # далее опционально
    'model':     str,   # Имя модели распознавания (по умолчанию mdiariz_common)
    'async':     bool,  # Асинхронный вариант, (по умолчанию False)
    'threshold': float, # Порог разделения дикторов, (по умолчанию 0.0)
}
packed_data = msgpack.packb(data, use_bin_type=True)

r = requests.post(url, data=packed_data)
if r.status_code == 200:
    result = msgpack.unpackb(r.content, raw=False)
```

результат:

Результатом асинхронного варианта является task_id для последующих вызовов.
Использование идентично методу /file.

Результатом синхронного является список списков вида:

```
[ [ int,    # начало (фреймы)
    int,    # конец (фреймы)
    str     # номер диктора
  ], ...
]
```

6. Методы биометрической идентификации (метод /bio)

6.1 Формат фрагмента обработанных данных

```
{
    'speaker_id': int,      # [опционально] индекс модели диктора из списка speakers
    'data': bytes,          # [опционально] бинарная модель диктора
    'win_score': float,     # [опционально] степень схожести, с которой выиграл эталон
    'win_etalon_id': int,   # [опционально] индекс выигравшего эталона в списке etalons
    'scores': [float, ...], # [опционально] список степеней схожести по всем эталонам
}
```

6.2 Получение модели диктора из аудиофайлов

запрос:

```
url = 'http://{}/bio'.format(HOST)
data = {
    'apikey': str,          # Персональный ключ
    'model':  str,          # Модель аудио канала
    # данные
    'wavs' : [bytes, ],     # Список бинарных контентов аудиофайлов
    # опционально
    'async':  bool,         # Асинхронный вариант, (по умолчанию False)
    'post_callback': str,   # Пример: 'http://127.0.0.1/callback'
}
packed_data = msgpack.packb(data, use_bin_type=True)

r = requests.post(url, data=packed_data)
if r.status_code == 200:
    result = msgpack.unpackb(r.content, raw=False)
```

результат:

При отправлении любого количества файлов возвращается бинарная модель диктора:

```
[
```

```
  {'data': bytes},
]
```

6.3 Сравнение аудиофайлов с эталонами

запрос:

```
url = 'http://{}/bio'.format(HOST)
data = {
    'apikey': str,          # Персональный ключ
    'model':  str,          # Модель аудио канала
    # данные
    'wavs' :    [bytes, ],  # Список бинарных контентов аудиофайлов
    'etalons' : [bytes, ],  # Список бинарных контентов эталонов
    # опционально
    'async':  bool,         # Асинхронный вариант, (по умолчанию False)
    'post_callback': str,   # Пример: 'http://127.0.0.1/callback'
}
packed_data = msgpack.packb(data, use_bin_type=True)

r = requests.post(url, data=packed_data)
if r.status_code == 200:
    result = msgpack.unpackb(r.content, raw=False)
```

результат:

Возвращается бинарная модель диктора и результат сравнения этой модели со всеми эталонами.

Пример: аудиофайлы, 3 эталона:

```
[
  {'data': bytes, 'win_score': 0.86, 'win_etalon_id': 2, 'scores': [0.80, 0.83, 0.86]},
]
```

6.4 Сравнение моделей дикторов с эталонами

запрос:

```
url = 'http://{}/bio'.format(HOST)
data = {
    'apikey': str,          # Персональный ключ
    'model':  str,          # Модель аудио канала
    # Данные
    'speakers' : [bytes, ], # Список бинарных контентов моделей дикторов
    'etalons' :  [bytes, ], # Список бинарных контентов эталонов
    # далее опционально
    'async':  bool,         # Асинхронный вариант, (по умолчанию False)
    'post_callback': str,   # Пример: 'http://127.0.0.1/callback'
}
packed_data = msgpack.packb(data, use_bin_type=True)

r = requests.post(url, data=packed_data)
if r.status_code == 200:
    result = msgpack.unpackb(r.content, raw=False)
```

результат:

Возвращается список результатов сравнений моделей дикторов со всеми эталонами.

Пример: 2 модели дикторов, 3 эталона:

```
[
  {'speaker_id': 0, 'win_score': 0.99, 'win_etalon_id': 1, 'scores': [0.87, 0.99, 0.32]},
  {'speaker_id': 1, 'win_score': 0.68, 'win_etalon_id': 0, 'scores': [0.68, 0.39, 0.05]},
]
```

6.5 Сравнение аудиофайлов и моделей дикторов с эталонами

запрос:

```
url = 'http://{}/bio'.format(HOST)
data = {
    'apikey': str,          # Персональный ключ
    'model':  str,          # Модель аудио канала
    # данные
    'wavs' :     [bytes, ], # Список бинарных контентов аудиофайлов
    'speakers' : [bytes, ], # Список бинарных контентов моделей дикторов
    'etalons' :  [bytes, ], # Список бинарных контентов эталонов
    # опционально
    'async':  bool,         # Асинхронный вариант, (по умолчанию False)
    'post_callback': str,   # Пример: 'http://127.0.0.1/callback'
}
packed_data = msgpack.packb(data, use_bin_type=True)

r = requests.post(url, data=packed_data)
if r.status_code == 200:
    result = msgpack.unpackb(r.content, raw=False)
```

результат:

Возвращается список результатов сравнений моделей дикторов со всеми эталонами.
Последним в списке будет результат сравнения модели дикторов из аудиофайлов со всеми эталонами.

Пример: аудиофайлы, 2 модели дикторов, 3 эталона:

```
[
  {'speaker_id': 0, 'win_score': 0.99, 'win_etalon_id': 1, 'scores': [0.87, 0.99, 0.32]},
  {'speaker_id': 1, 'win_score': 0.68, 'win_etalon_id': 0, 'scores': [0.68, 0.39, 0.05]},
  {'data': bytes, 'win_score': 0.86, 'win_etalon_id': 2, 'scores': [0.80, 0.83, 0.86]},
]
```