Orpheus TTS — Library User Guide (Production)

Audience: **App developers** using the `orpheus_tts` Python SDK to synthesize speech over HTTP.

This guide covers: install, quickstart, capabilities, streaming patterns, performance/concurrency, error handling, audio handling, and integration recipes.

---

1) Install & Requirements

```
pip install orpheus-tts   # if published
# or: add the package/repo to your requirements.txt / pyproject.toml
```

Runtime

- Python 3.10+
- Async event loop (e.g., `asyncio`)
- Network access to the Orpheus **Gateway** (recommended) or directly to **Workers**

Audio format

- SDK yields **raw PCM** chunks: **mono**, **16-bit little-endian**, **24 kHz**.

---

2) Quickstart

2.1 Minimal streaming example (save to WAV)

```
import asyncio, wave
from orpheus_tts.sdk import OrpheusTTS
```

```
SAMPLE_RATE = 24000
SAMPLE_WIDTH = 2  # bytes
CHANNELS = 1

async def main():
    tts = OrpheusTTS(
        base_urls=["https://tts.example.com"],  # one or many
gateway/worker URLs
        api_key="YOUR_API_KEY",                 # or None if not
required
        hedge_ttfb_ms=400                       # set None to disable
hedging
    )
    pcm_chunks = []
    async for chunk in tts.stream(
        text="Hello world from Orpheus.",
        voice="alloy",                          # optional;
service-defined
        temperature=0.4, top_p=0.9, top_k=0,    # optional tuning
passed through
        max_tokens=1536
    ):
        pcm_chunks.append(chunk)

    with wave.open("hello.wav", "wb") as wf:
        wf.setnchannels(CHANNELS)
        wf.setsampwidth(SAMPLE_WIDTH)
        wf.setframerate(SAMPLE_RATE)
        wf.writeframes(b"".join(pcm_chunks))

    await tts.close()       # IMPORTANT: release HTTP connections

asyncio.run(main())
```

2.2 Realtime playback (optional)

```
# pip install sounddevice numpy
import asyncio, numpy as np, sounddevice as sd
```

```
from orpheus_tts.sdk import OrpheusTTS

SAMPLE_RATE = 24000

async def speak(text):
    tts = OrpheusTTS(["https://tts.example.com"], api_key="KEY")
    async for chunk in tts.stream(text=text, temperature=0.4):
        sd.play(np.frombuffer(chunk, dtype=np.int16), SAMPLE_RATE,
blocking=False)
        # (Optional) sd.wait() at end if you want to block until done
    await tts.close()

asyncio.run(speak("Streaming playback demo."))
```

---

3) Public API Overview

3.1 `OrpheusTTS`

Facade for single endpoint **or** multi-endpoint cluster routing.

```
OrpheusTTS(
    base_urls: list[str],            # gateway URL or list of worker
URLs
    api_key: str | None = None,      # Bearer key sent to the service
    hedge_ttfb_ms: float | None = 400
)
```

- **`hedge_ttfb_ms`**: if the first byte is late, a backup request is launched to another endpoint; the **first** stream to yield audio "wins."
  Set **`None`** (or `0`) to disable hedging.

`stream(...) -> AsyncIterator[bytes]`

```
async def stream(
    text: str,
    voice: str | None = None,
    **params
```

```
) -> AsyncIterator[bytes]:
    ...
```

- Yields **raw PCM** audio bytes in small chunks.
- `voice` is optional (service-defined).
- `**params` are passed through to the service (e.g., `temperature`, `top_p`, `top_k`, `max_tokens`, etc.).

`close()`

```
await tts.close()
```

Closes underlying HTTP clients. **Always call this** (or manage the instance for app lifetime).

> **Tip:** Reuse a single `OrpheusTTS` instance for your whole service/process to avoid reconnect overhead.

---

4) Capabilities & Parameters

The SDK forwards parameters to the service. Common knobs (subject to your deployment):

- `voice: str | None` — voice preset or custom voice id
- `temperature: float` — 0.0–1.0 (stylistic variance)
- `top_p: float` — nucleus sampling
- `top_k: int` — top-K sampling (0 = disabled)
- `max_tokens: int` — generation budget
- other **engine-specific** keys as exposed by your server

> **Server timeout**: a default request deadline is enforced **server-side** (typically 60s). The current SDK forwards your params but does **not** expose a top-level

`timeout_s` override yet. If you need a custom deadline, request it from the service team or pin to a gateway version that supports overriding via parameters.

---

5) Patterns & Recipes

5.1 Concurrency (multiple synths at once)

```
import asyncio
from orpheus_tts.sdk import OrpheusTTS

texts = ["One", "Two", "Three", "Four"]

async def one(tts, text, idx):
    chunks = []
    async for c in tts.stream(text=text, temperature=0.3):
        chunks.append(c)
    return idx, b"".join(chunks)

async def main():
    tts = OrpheusTTS(["https://tts.example.com"], api_key="KEY")
    results = await asyncio.gather(*(one(tts, t, i) for i, t in
enumerate(texts)))
    await tts.close()
    # results = list[(idx, pcm_bytes)]
asyncio.run(main())
```

Best practices

- **Create 1 shared `OrpheusTTS` per event loop** and reuse it.
- Use `asyncio.gather` to fire multiple `stream(...)` concurrently.
- Avoid creating/closing a client per request in high-QPS apps.

5.2 Streaming over HTTP (server → client)

If you're building your own API, stream chunks as they arrive:

FastAPI example

```
from fastapi import FastAPI
from fastapi.responses import StreamingResponse
from orpheus_tts.sdk import OrpheusTTS

app = FastAPI()
tts = OrpheusTTS(["https://tts.example.com"], api_key="KEY")

@app.get("/speak")
async def speak(q: str):
    async def gen():
        async for chunk in tts.stream(text=q, temperature=0.4):
            yield chunk
    return StreamingResponse(gen(),
media_type="application/octet-stream")

@app.on_event("shutdown")
async def _shutdown():
    await tts.close()
```

5.3 Save as WAV/OGG/MP3

- **WAV**: trivial (header + PCM). See Quickstart 2.1.
- **OGG/MP3**: use an encoder (e.g., `pydub`, `ffmpeg-python`). You'll need to buffer or pipe PCM; some encoders accept streaming via stdin.

---

6) Performance Tuning

- **Hedging (`hedge_ttfb_ms`)**: Lower values reduce TTFB tail at the cost of occasional duplicate work. For very stable networks, you can raise or disable it.
- **Chunking text**: Very long text => longer latency & memory usage. Prefer chunking by sentence/paragraph and stream sequentially for responsiveness.

- **Co-location**: Run your app and the gateway in the same region to minimize RTT.
- **Backpressure**: The iterator yields as network chunks arrive. If you write to a file/socket, flush promptly to avoid buffering delays.
- **Reuse**: Keep one `OrpheusTTS` instance per process; avoid creating temp clients.

---

7) Error Handling

The iterator may raise before or during streaming:

- `httpx.ConnectError`, `httpx.ReadTimeout`, `httpx.RemoteProtocolError` — network/HTTP issues
- `asyncio.TimeoutError` — if the upstream gateway enforces a hard deadline (server-side)
- `HTTPStatusError` — non-2xx HTTP response on stream start (invalid auth, bad request)
- `RuntimeError("Circuit open for endpoint")` — local breaker temporarily blocking a sick endpoint
- Generic `Exception` — unexpected internal failures

Recommended pattern

```
async def safe_stream(tts, text):
    try:
        async for chunk in tts.stream(text=text, temperature=0.4):
            yield chunk
    except httpx.HTTPError as e:
        # log & map to your app's error type
        raise
    except Exception as e:
        # fallback or retry policy, depending on your UX
        raise
```

Retries?
The SDK **does not** automatically retry once bytes have been received (to prevent duplicate audio). If connect fails **before** the first chunk, your app can retry the **entire** request safely (idempotent).

---

8) Authentication

Pass an API key when constructing `OrpheusTTS`:

```
tts = OrpheusTTS(["https://tts.example.com"], api_key="YOUR_KEY")
```

The SDK will send:

- `Authorization: Bearer YOUR_KEY`

If your deployment uses `X-API-Key` instead, the gateway will still accept Bearer (or is configured to accept both). Consult your service ops guide if customized.

---

9) Cluster Awareness

If you supply **multiple** `base_urls`, the SDK routes per request:

```
tts = OrpheusTTS(
    base_urls=[
        "https://gw-a.example.com",
        "https://gw-b.example.com",
    ],
    api_key="KEY",
    hedge_ttfb_ms=300
)
```

- Endpoints are scored with latency & queue hints; lower score is preferred.

- Hedging (if enabled) fires a backup after `hedge_ttfb_ms` unless the first endpoint has already produced audio.
- If an endpoint becomes sick, a **client-side circuit breaker** prevents immediate reuse for a cooldown window.

---

10) Audio Handling Details

- **PCM format**: mono, 16-bit LE, 24,000 Hz

To NumPy:

```
import numpy as np
arr = np.frombuffer(chunk, dtype=np.int16)  # each chunk independently
```

-
- **Combine safely**: Concatenate in memory (`b"".join`), or write chunks directly to a file or socket as they arrive.
- **Silence/trim**: The service may or may not add leading/trailing silence; do it client-side if you need strict timing.

---

11) Integration Tips

- **Web apps**: Use a background task or WebSocket to push chunks to the browser; the browser can append to a `MediaSource` or use WebAudio with a PCM decoder (or transcode server-side to WAV/OGG).
- **Mobile**: Buffer a few chunks (e.g., 150–300 ms) before playback to absorb jitter.
- **Batch/offline**: For many short prompts, use `asyncio.Semaphore` to cap in-process concurrency; start ~CPU_count×2 and tune.

---

12) Lifecycle & Resource Cleanup

- Prefer **one** long-lived `OrpheusTTS` per process / per event loop.
- Call `await tts.close()` on shutdown (FastAPI `shutdown` event, atexit hook for scripts).
- Do **not** share the same instance across **multiple event loops** (e.g., different threads). If you must, create one instance per loop.

---

13) Troubleshooting Checklist (for Library Users)

- **401 Unauthorized**: Check API key; verify the gateway expects `Bearer`.
- **High latency**: Lower `hedge_ttfb_ms`, verify network path (same region), chunk your input.
- **Broken audio**: Ensure you treat bytes as **16-bit LE PCM @ 24 kHz**; wrong sample rate during playback will sound fast/slow.
- **Intermittent failures**: Add app-level retry on **connect** errors (no bytes received), not mid-stream.
- **No audio**: Make sure the server voice/engine params are valid; try minimal params first.

---

14) Versioning & Compatibility

- **SDK ↔ Service**: The wire format is NDJSON; this SDK targets the `"v1"` contract exposed by the gateway.
- Backwards compatibility: Unknown fields are ignored; you can upgrade the service independently as long as `chunk`, `metrics`, `eos`, and `error` frame types remain.

---

15) Example: Full App Skeleton (FastAPI)

```
from fastapi import FastAPI, HTTPException
from fastapi.responses import StreamingResponse
from orpheus_tts.sdk import OrpheusTTS
import os, asyncio

API_KEY = os.getenv("ORPHEUS_KEY", "")
BASE_URLS = os.getenv("ORPHEUS_URLS",
"https://tts.example.com").split(",")

app = FastAPI()
tts = OrpheusTTS(BASE_URLS, api_key=API_KEY, hedge_ttfb_ms=400)

@app.get("/v1/tts.wav")
async def tts_wav(q: str):
    import wave, io
    buf = io.BytesIO()
    with wave.open(buf, "wb") as wf:
        wf.setnchannels(1); wf.setsampwidth(2); wf.setframerate(24000)
        async for chunk in tts.stream(text=q, temperature=0.4):
            wf.writeframes(chunk)
    buf.seek(0)
    return StreamingResponse(buf, media_type="audio/wav")

@app.on_event("shutdown")
async def _shutdown():
    await tts.close()
```

---

16) FAQ

Q: Can I set a per-request timeout?
A: The current SDK uses the server's default deadline. If your deployment supports a custom `timeout_s`, expose it via the SDK version you're using (ask your service owner).

Q: How do I pick `hedge_ttfb_ms`?
A: Start at 300–400 ms. If your users are sensitive to first-audio latency and you have spare worker capacity, decrease to 200–250 ms.

Q: Can I get metrics from the SDK?
A: For advanced use, you can introspect endpoint snapshots via lower-level clients, but that API is not considered stable. Prefer service-side `/metrics` and `/v1/system`.

Q: Is streaming deterministic?
A: No—sampling params (`temperature`, `top_p`, `top_k`) introduce variability. Set `temperature=0.0` for more deterministic output, subject to the engine.
