StreamingApi

August 17, 2026

StreamingApi

Kind: Class

Source: `src/utils/stream.ts`

Part of: Utils

`StreamingApi` coordinates streamed output, delayed work, piping, and cancellation in `src/utils/stream.ts`. It exposes async write methods alongside abort handling so callers can stop work when the stream is no longer active.

Methods

Method	Signature	Returns
`write`	`write(input: Uint8Array	string)`
`writeln`	`writeln(input: string)`	`Promise<StreamingApi>`
`sleep`	`sleep(ms: number)`	`Promise<unknown>`
`close`	`close()`	`void`
`pipe`	`pipe(body: ReadableStream)`	`void`
`onAbort`	`onAbort(listener: () => void	Promise)`
`abort`	`abort()`	`void`

Properties

Property	Type
`responseReadable`	`ReadableStream`
`aborted`	`boolean`
`closed`	`boolean`

When something fails

- `StreamingApi` handles failure in 2 places: it discards it silently in all 2.

Diagram

```
graph LR
  Caller --> StreamingApi
  StreamingApi --> Write[write / writeln]
  StreamingApi --> Delay[sleep]
  StreamingApi --> Pipe[pipe]
  StreamingApi --> Abort[onAbort / abort]
  StreamingApi --> Close[close]
```

Usage

```
import type { StreamingApi } from "./utils/stream";

async function handleStream(stream: StreamingApi) {
  stream.onAbort(() => {
    console.log("Stream aborted");
  });

  await stream.write();
  await stream.writeln();

  await stream.sleep();

  stream.close();
}
```

AI Coding Instructions

- Await `write()` and `writeln()` before dependent streamed work.
- Register `onAbort()` handlers before starting long-running work.
- Stop pending work when an abort handler runs; do not continue writing after cancellation.
- Call `close()` when streaming is complete.
- Keep stream lifecycle handling in the caller that owns the request or response.

Used by

2 references from 2 files. Each is a place in this repository where the symbol is actually used — go read one rather than trusting an example.

Imported by (2)

- `SSEMessage` — `src/helper/streaming/sse.ts`:6
- `stream` — `src/helper/streaming/stream.ts`:7