Barrier

August 18, 2026

Barrier

Kind: Class

Source: `packages/core/helpers/barrier.ts`

Part of: Core

A simple barrier to synchronize flow of multiple async operations.

`Barrier` synchronizes multiple asynchronous operations at a shared checkpoint. Call `signal()` when an operation reaches the barrier, and use `wait()` to pause until the required signals have been received; `signalAndWait()` performs both steps atomically for participating operations.

Methods

Method	Signature	Returns
`signal`	`signal()`	`void`
`wait`	`wait()`	`Promise<void>`
`signalAndWait`	`signalAndWait()`	`Promise<void>`

Diagram

```
graph LR
  A[Async operation A] -->|signal / signalAndWait| B[Barrier]
  C[Async operation B] -->|signal / signalAndWait| B
  D[Async operation C] -->|signal / signalAndWait| B
  B -->|all participants signaled| E[Release waiting operations]
```

Usage

```ts
import { Barrier } from '@your-package/core/helpers/barrier';

const barrier = new Barrier(2);

async function worker(name: string) {
  console.log(`${name} reached the checkpoint`);

  await barrier.signalAndWait();

  console.log(`${name} continues after the checkpoint`);
}

await Promise.all([
  worker('worker-a'),
  worker('worker-b'),
]);
```

AI Coding Instructions

- Create a `Barrier` with the number of async participants expected to reach the synchronization point.
- Prefer `signalAndWait()` when a participant should both register its arrival and wait for all other participants.
- Use `signal()` only for operations that should notify the barrier without blocking themselves.
- Ensure every expected participant signals the barrier; otherwise, calls to `wait()` can remain pending indefinitely.
- Use barriers for deterministic coordination between concurrent tasks, not as a replacement for error handling or cancellation logic.