ExternalErrorProxy

August 17, 2026

ExternalErrorProxy

Kind: Class

Source: `packages/core/helpers/external-proxy.ts`

Part of: Core

`ExternalErrorProxy` creates a proxy around external dependencies so errors thrown by third-party clients can be handled consistently. It centralizes external error interception and translation, allowing the rest of the application to interact with integrations through a safer, uniform interface.

Methods

Method	Signature	Returns
`createProxy`	`createProxy(targetCallback: (...args: any[]) => any, exceptionsHandler: ExternalExceptionsHandler, type: TContext)`	`void`

When something fails

- `ExternalErrorProxy` handles failure in 1 place: it turns it into a return value in all 1.

Diagram

```
graph LR
  A[Application Service] --> B[ExternalErrorProxy]
  B -->|createProxy| C[Proxied External Client]
  C --> D[External API / SDK]
  D -->|Success| C
  C -->|Result| A
  D -->|Throws external error| B
  B -->|Normalized application error| A
```

Usage

```
import { ExternalErrorProxy } from '@your-package/core/helpers/external-proxy';

const externalClient = {
  async getCustomer(customerId: string) {
    const response = await fetch(
      `https://api.example.com/customers/${customerId}`,
    );

    if (!response.ok) {
      throw new Error(`External API request failed: ${response.status}`);
    }

    return response.json();
  },
};

const errorProxy = new ExternalErrorProxy();
const customerClient = errorProxy.createProxy(externalClient);

const customer = await customerClient.getCustomer('customer_123');
console.log(customer);
```

AI Coding Instructions

- Wrap external SDKs, HTTP clients, and integration adapters with `createProxy()` instead of duplicating error-handling logic in every caller.
- Preserve the original external client method signatures when adding new proxied dependencies.
- Ensure error normalization retains useful context, such as the failing operation or upstream response details, without exposing sensitive data.
- Do not bypass the proxy for production integration calls; doing so can create inconsistent error behavior across the application.
- Add tests for both successful method delegation and failures thrown by proxied external methods.

How it works

`ExternalErrorProxy` is a class whose `createProxy` method returns an asynchronous callback wrapper around a target callback. The wrapper accepts any arguments and invokes the target with those same arguments. [packages/core/helpers/external-proxy.ts:5-13]

- `createProxy` takes a callable `targetCallback`, an `ExternalExceptionsHandler`, and an optional context-type value constrained to `string` (defaulting generically to `ContextType`). It performs no runtime validation of these inputs. [packages/core/helpers/external-proxy.ts:6-10]
- On normal completion, the wrapper resolves to the value produced by `targetCallback`, including when that value is a promise, because it awaits the invocation. [packages/core/helpers/external-proxy.ts:11-13]
- It catches both synchronous throws and promise rejections from `targetCallback`. [packages/core/helpers/external-proxy.ts:11-17] The repository test exercises both forms and expects the exception handler's `next` method to be called. [packages/core/test/helpers/external-proxy.spec.ts:24-41]
- On a caught value, it constructs an `ExecutionContextHost` from the wrapper's received arguments, sets its context type, and returns `exceptionsHandler.next(e, host)`. [packages/core/helpers/external-proxy.ts:14-18] `ExecutionContextHost` retains the supplied argument array; its initial type is `'http'`, and `setType` changes that type only when its argument is truthy. [packages/core/helpers/execution-context-host.ts:10-21,35-40]
- Consequently, when `type` is omitted or otherwise falsy at runtime, the host passed to the exception handler retains the default `'http'` type. [packages/core/helpers/external-proxy.ts:15-17] [packages/core/helpers/execution-context-host.ts:11,19-21]
- `ExternalExceptionsHandler.next` first invokes a selected custom filter when one matches; otherwise it calls the inherited `catch` method. [packages/core/exceptions/external-exceptions-handler.ts:11-17,26-35] The inherited path logs non-intrinsic `Error` instances and then throws the caught exception. [packages/core/exceptions/external-exception-filter.ts:6-15]
- `ExternalContextCreator` wraps its generated external handler with this proxy only when `options.filters` is truthy; otherwise it returns the target handler unwrapped. [packages/core/helpers/external-context-creator.ts:164-185]

Relationships

- IMPORTS → `ContextType`