EventProcessor

August 17, 2026

EventProcessor

Kind: Class

Source: `src/adapter/aws-lambda/handler.ts`

Part of: Adapter

`EventProcessor` adapts an AWS Lambda API Gateway event into a standard `Request` and converts the processed response into an `APIGatewayProxyResult`. It reads request path, method, query parameters, headers, cookies, and domain data while applying response headers and cookies to the Lambda result.

Methods

Method	Signature	Returns
`getPath`	`getPath(event: E)`	`string`
`getMethod`	`getMethod(event: E)`	`string`
`getQueryString`	`getQueryString(event: E)`	`string`
`getHeaders`	`getHeaders(event: E)`	`Headers`
`getCookies`	`getCookies(event: E, headers: Headers)`	`void`
`setCookiesToResult`	`setCookiesToResult(result: APIGatewayProxyResult, cookies: string[])`	`void`
`getHeaderValue`	`getHeaderValue(headers: E['headers'], key: string)`	`string
`getDomainName`	`getDomainName(event: E)`	`string
`createRequest`	`createRequest(event: E)`	`Request`
`createResult`	`createResult(event: E, res: Response, options:`	`Promise<APIGatewayProxyResult>`

Method	Signature	Returns
	`Pick<HandleOptions, 'isContentTypeBinary'>)`	
`setCookies`	`setCookies(_event: E, res: Response, result: APIGatewayProxyResult)`	`void`

Where it refuses work

- `EventProcessor` stops the work with an early return when `event.requestContext && 'domainName' in event.requestContext`.
- `EventProcessor` stops the work with an early return when `hostFromHeaders`.

Diagram

```
graph LR
  Event[API Gateway event] --> Processor[EventProcessor]
  Processor --> Request[Request]
  Request --> Handler[Application handler]
  Handler --> Response[Response]
  Response --> Processor
  Processor --> Result[APIGatewayProxyResult]
```

Usage

```
import { EventProcessor } from "./adapter/aws-lambda/handler";

async function handleInvocation(processor: EventProcessor) {
  const request = processor.createRequest();

  const response = await app.fetch(request);

  response.headers.forEach((value, name) => {
    // Apply response headers through the processor integration.
    processor.getHeaders().set(name, value);
  });

  return processor.createResult();
}
```

AI Coding Instructions

- Keep AWS event parsing inside `EventProcessor`; pass `createRequest()` output to application code that expects a standard `Request`.
- Read request metadata through `getPath()`, `getMethod()`, `getQueryString()`, `getHeaders()`, and `getDomainName()` instead of accessing event fields in downstream handlers.
- Preserve cookie handling through `getCookies()` and `setCookiesToResult()` so Lambda response cookies are emitted in the expected result format.
- Return the value from `createResult()` at the Lambda boundary, where an `APIGatewayProxyResult` is required.

How it works

`EventProcessor` is an exported abstract base class for translating one of the supported AWS Lambda event shapes into a Web `Request`, and translating a Web `Response` into an `APIGatewayProxyResult`. Its generic event type `E` must extend the `LambdaEvent` union of API Gateway v1, API Gateway v2, ALB, and Lattice v2 events. src/adapter/aws-lambda/handler.ts:23-27 src/adapter/aws-lambda/handler.ts:278-278

Concrete subclasses must define how to extract a path, HTTP method, query string, headers, cookies, and response cookies for their event shape. src/adapter/aws-lambda/handler.ts:279-289