ParamProperties

August 19, 2026

ParamProperties

Kind: Interface

Source: `packages/core/helpers/context-utils.ts`

Part of: Core

`ParamProperties` describes the metadata required to resolve a handler or controller parameter at runtime. It identifies the parameter position, its declared type, extraction configuration, optional transformation pipes, and the extractor responsible for obtaining the raw value.

Properties

Property	Type
`index`	`number`
`type`	`` `T ``
`data`	`ParamData`
`pipes`	`PipeTransform[]`
`extractValue`	`IExtractor`

Diagram

```
graph LR
  A[Incoming request/context] --> B[IExtractor]
  B --> C[Extracted value]
  C --> D[PipeTransform[]]
  D --> E[Resolved parameter value]
  E --> F[Handler argument at index]

  G[ParamProperties] --> H[index: number]
  G --> I[type: T | string]
  G --> J[data: ParamData]
```

```
  G --> K[pipes: PipeTransform[]]
  G --> L[extractValue: IExtractor]
```

Usage

```
import type { ParamProperties } from './context-utils';

const userIdParam: ParamProperties<string> = {
  index: 0,
  type: String,
  data: {
    name: 'id',
    source: 'params',
  },
  pipes: [parseIntPipe],
  extractValue: {
    extract(context, data) {
      return context.params[data.name];
    },
  },
};

// Runtime parameter resolution flow:
const rawValue = userIdParam.extractValue.extract(context, userIdParam.data);
const value = userIdParam.pipes.reduce(
  (result, pipe) => pipe.transform(result),
  rawValue,
);

handlerArguments[userIdParam.index] = value;
```

AI Coding Instructions

- Use `index` as the authoritative position when assigning the resolved value into a handler argument array.
- Keep `extractValue` focused on reading raw values from the current execution context; apply validation and conversion through `pipes`.
- Preserve pipe order, since each `PipeTransform` should receive the output of the previous pipe.
- Use `data` to pass extractor-specific configuration such as parameter names, headers, query keys, or request-body paths.
- Support both constructor types and string-based types in `type`, especially when runtime type metadata may be unavailable.

Relationships

- IMPORTS → `ParamData`
- IMPORTS → `PARAMTYPES_METADATA`
- IMPORTS → `RESPONSE_PASSTHROUGH_METADATA`
- IMPORTS → `ContextType`
- IMPORTS → `Controller`
- IMPORTS → `PipeTransform`
- IMPORTS → `Type`
- IMPORTS → `isFunction`