readme.md for @api.global/typedrequest-interfaces

interfaces for making typed requests

Install

To install `@api.global/typedrequest-interfaces`, you need to have Node.js installed on your system. You can then add this package to your project by running the following command in your terminal:

```
npm install @api.global/typedrequest-interfaces --save
```

This will add the package to your project's dependencies. Ensure you are in your project's directory or specify the path where your project is located.

Usage

To use the `@api.global/typedrequest-interfaces` package in your TypeScript project, follow the steps below. This package provides interfaces for making typed requests, which can help enforce a structured request and response pattern in your applications.

Setting Up Your Project

First, ensure your project is set up to use TypeScript and ESM (ECMAScript Modules). You will need a `tsconfig.json` file in your project root with at least the following settings:

```
{
  "compilerOptions": {
    "target": "ESNext",
```

```
    "module": "ESNext",
    "moduleResolution": "node",
    "outDir": "./dist",
    "declaration": true,
    "esModuleInterop": true,
    "experimentalDecorators": true,
    "forceConsistentCasingInFileNames": true
  },
  "include": ["src/**/*"]
}
```

Adjust the settings according to your project requirements. This configuration supports modern JavaScript features and TypeScript.

Basic Implementation

To use the interfaces provided by the package, start by importing them into your TypeScript file. Below is an example demonstrating how to define a typed request using the interfaces:

```
import { ITypedRequest, ITypedEvent, ITag, IVirtualStream, IStreamRequest } from
'@api.global/typedrequest-interfaces';

// Example: Defining a typed request
interface MyCustomRequest extends ITypedRequest {
  method: 'MyCustomMethod';
  authInfo?: {
    jwt: string;
  };
  request: {
    someData: string;
  };
  response: {
    resultData: number;
  };
}
```

This snippet shows how to create a custom interface `MyCustomRequest` that implements the `ITypedRequest` interface from the package. You can specify the types for the request and response data to match your application's needs.

Handling Typed Events

You can also define and use typed events similar to the following example:

```
// Example: Defining a typed event
interface MyCustomEvent extends ITypedEvent<{ message: string }> {
  name: 'MyCustomEvent';
  uniqueEventId: 'Event123';
  payload: {
    message: 'Hello World';
  };
}
```

Utilizing Tags

Tags can be useful for attaching metadata or categorization information to your requests or events. Implementing a tag looks like this:

```
// Example: Defining a tag
interface UserActionTag extends ITag {
  name: 'UserAction';
  payload: {
    userId: number;
    action: string;
  };
}
```

Virtual Streams

For more advanced use cases, like handling data streams, the `IVirtualStream` interface can be implemented as follows:

```
// Example: Implementing a virtual stream
class MyStream implements IVirtualStream<Uint8Array> {
  side = 'requesting';
  streamId = 'myStream123';
  sendMethod = async (data: Uint8Array) => { /* Implementation here */ };
  // Implement other required methods...
```

```
}
```

Working with Stream Requests

Stream requests allow for handling requests that involve streaming data. Here is how you can use the `IStreamRequest` interface:

```
// Example: Using an IStreamRequest
const myStreamRequest: IStreamRequest = {
  method: '##VirtualStream##',
  request: {
    streamId: 'myStream123',
    cycleId: 'cycle456',
    cycle: 'request',
    mainPurpose: 'start'
    // More properties here...
  },
  response: {
    // Response structure here...
  }
};
```

This setup demonstrates creating a request for initializing a virtual stream. Other `mainPurpose` values like 'chunk', 'read', 'end', and 'feedback' can also be used depending on the interaction with the stream.

The examples shown demonstrate the versatility of the `@api.global/typedrequest-interfaces` package in structuring your application's request-response mechanisms, event handling, tagging, and stream management. By following TypeScript and ESM conventions, you can leverage these interfaces to architect robust, typed APIs and services.

License and Legal Information

Trademarks

This project is owned and maintained by Task Venture Capital GmbH. The names and logos associated with Task Venture Capital GmbH and any related products or services are trademarks of Task Venture Capital GmbH and are not included within the scope of the MIT license granted herein. Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines, and any usage must be approved in writing by Task Venture Capital GmbH.

Company Information

Task Venture Capital GmbH
Registered at District court Bremen HRB 35230 HB, Germany

For any legal inquiries or if you require further information, please contact us via email at hello@task.vc.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.

Revision #7
Created 2026-03-28 10:48:10 UTC by foss.global Team
Updated 2026-03-28 12:13:18 UTC by foss.global Team