@api.global/typedelectron

Documentation for @api.global/typedelectron

- readme.md for @api.global/typedelectron

readme.md for @api.global/typedelectron

a package made for ipc communication in electron

Install

To start using @api.global/typedelectron in your Electron project, first, you need to add it to your project dependencies using npm:

```
npm install @api.global/typedelectron --save
```

This command will install the package and add it to your `package.json` file.

Usage

@api.global/typedelectron is a TypeScript package designed to facilitate IPC (Inter-Process Communication) in Electron applications, leveraging the power of typed requests for enhanced developer experience and code reliability. Below, we provide an exhaustive guide to integrating and utilizing this package in your Electron project, ensuring that your main process and renderer processes communicate seamlessly and efficiently.

Setting Up Your Project

Before diving into the usage of @api.global/typedelectron, ensure your Electron project is set up to support TypeScript and ESM (ECMAScript Modules) syntax. Your `tsconfig.json` should include settings that target ESNext modules, and your Electron version must meet the peer dependency requirement (>=28.0.0-beta.11).

Initialization in the Main Process

To leverage @api.global/typedelectron for handling IPC in the main process of your Electron application, start by importing and initializing the `TypedElectronBackend` class. This class provides

the functionalities needed to receive and respond to typed requests from renderer processes.

```
import { app, BrowserWindow } from 'electron';
import { TypedElectronBackend } from '@api.global/typedelectron';

app.on('ready', async () => {
  const mainWindow = new BrowserWindow({
    // BrowserWindow options here
    webPreferences: {
      preload: YOUR_PRELOAD_SCRIPT,
      contextIsolation: true,
    },
  });

  // Initialize TypedElectronBackend
  const typedElectronBackend = await TypedElectronBackend.createTypedElectronBackend();

  // Load your HTML file
  mainWindow.loadFile('path/to/your/index.html');
});
```

In the example above, replace `YOUR_PRELOAD_SCRIPT` with the path to your preload script, which should also import and use the `getPreloadScriptPath` provided by the package to ensure typed messages are properly set up.

Handling Typed Requests

The core advantage of @api.global/typedelectron is its ability to process typed requests, making IPC more robust and easier to manage. Let's define a typed request interface:

```
// Define in a shared location accessible by both main and renderer processes
export interface IMyTypedRequest {
  requestType: 'MY_REQUEST';
  payload: {
    someData: string;
  };
}
```

In the main process, use the `typedElectronBackend` instance to register handlers for specific request types:

```
typedElectronBackend.typedrouter.onTyped<IMyTypedRequest>('MY_REQUEST', async (request) => {
  console.log(request.payload.someData);
  return {
    response: 'Processed your request',
  };
});
```

This setup allows the main process to listen for `MY_REQUEST` types and respond accordingly.

Sending Requests from the Renderer Process

Now, let's send a typed request from the Renderer process. Ensure your preload script sets up the context bridge properly, and then use the provided `TypedRequest` class:

```
import { TypedRequest } from '@api.global/typedelectron';

const myTypedRequest = new TypedRequest<IMyTypedRequest>('MY_REQUEST');
myTypedRequest.send({
  someData: 'Hello from Renderer',
}).then((response) => {
  console.log(response); // Process response
});
```

In this example, the renderer process sends a request of type `MY_REQUEST` with some data, and then processes the response received from the main process.

Conclusion

By following the steps and patterns outlined above, you can leverage @api.global/typedelectron to enhance the communication layer of your Electron applications. Typed requests help ensure that the data exchanged between your main and renderer processes is predictable and well-structured, ultimately leading to more reliable and maintainable code.

License and Legal Information

This repository contains open-source code that is licensed under the MIT License. A copy of the MIT License can be found in the license file within this repository.

Please note: The MIT License does not grant permission to use the trade names, trademarks, service marks, or product names of the project, except as required for reasonable and customary use in describing the origin of the work and reproducing the content of the NOTICE file.

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.