@api.global/typedrequest

Documentation for @api.global/typedrequest

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

readme.md for @api.global/typedrequest

A TypeScript library for making **fully typed request/response cycles** across any transport — HTTP, WebSockets, broadcast channels, or custom protocols. Define your API contract once as a TypeScript interface, then use it on both client and server with compile-time safety, automatic routing, middleware chains, virtual streams for real-time data, and built-in traffic monitoring hooks.

Issue Reporting and Security

For reporting bugs, issues, or security vulnerabilities, please visit community.foss.global/. This is the central community hub for all issue reporting. Developers who sign and comply with our contribution agreement and go through identification can also get a code.foss.global/ account to submit Pull Requests directly.

Install

```
pnpm install @api.global/typedrequest
```

You'll also want the interfaces package for defining stream types:

```
pnpm install @api.global/typedrequest-interfaces
```

Usage

All examples use ESM imports and TypeScript.

```
import {
  TypedRequest,
  TypedHandler,
```

```
  TypedRouter,
  TypedTarget,
  VirtualStream,
  TypedResponseError,
} from '@api.global/typedrequest';
```

□□ Define Your API Contract

Every request/response pair is described by a simple interface extending `ITypedRequest`:

```
interface IGetUser {
  method: 'getUser';
  request: { userId: string };
  response: { username: string; email: string };
}

interface IAddNumbers {
  method: 'add';
  request: { a: number; b: number };
  response: { result: number };
}
```

The `method` field acts as a discriminator — it's what the router uses to match requests to handlers.

□□ Making Typed Requests (Client Side)

`TypedRequest` fires a typed request against an HTTP endpoint or a `TypedTarget`:

```
// Against an HTTP endpoint
const getUser = new TypedRequest<IGetUser>('https://api.example.com/rpc', 'getUser');
const user = await getUser.fire({ userId: 'user-123' });
console.log(user.username); // fully typed!

// With response caching
const cachedUser = await getUser.fire({ userId: 'user-123' }, true);
```

□□ Handling Requests (Server Side)

`TypedHandler` processes a specific method and returns a typed response:

```
const addHandler = new TypedHandler<IAddNumbers>('add', async (req) => {
  return { result: req.a + req.b };
});
```

The second argument to the handler function is an optional `TypedTools` instance that gives access to guard validation and transport-layer context:

```
const secureHandler = new TypedHandler<IGetUser>('getUser', async (req, tools) => {
  // Access transport-layer context (e.g., authenticated user info)
  const peer = tools.localData.peer;

  // Validate with guards
  await tools.passGuards([myAuthGuard], req);

  return { username: 'Alice', email: 'alice@example.com' };
});
```

□□ Routing Requests

`TypedRouter` dispatches incoming requests to the correct handler based on the `method` field:

```
const router = new TypedRouter();
router.addTypedHandler(addHandler);
router.addTypedHandler(secureHandler);

// Route an incoming request object
const response = await router.routeAndAddResponse(incomingTypedRequest);
```

Routers are **composable** — you can nest them to build modular API architectures:

```
const coreRouter = new TypedRouter();
const authRouter = new TypedRouter();

// Each sub-router manages its own handlers
coreRouter.addTypedHandler(addHandler);
authRouter.addTypedHandler(secureHandler);

// Link them together — requests flow through the entire chain
```

```
const mainRouter = new TypedRouter();
mainRouter.addTypedRouter(coreRouter);
mainRouter.addTypedRouter(authRouter);
```

□□ Middleware

Add middleware functions to a `TypedRouter` that run **before** any handler on that router executes. Middleware is great for authentication, logging, rate limiting, or input validation:

```
const router = new TypedRouter();

// Add authentication middleware
router.addMiddleware(async (typedRequest) => {
  const token = typedRequest.localData?.authToken;
  if (!token || !isValidToken(token)) {
    throw new TypedResponseError('Unauthorized', { reason: 'invalid_token' });
  }
});

// Add logging middleware
router.addMiddleware(async (typedRequest) => {
  console.log(`Processing ${typedRequest.method}`);
});

// Handlers are only reached if all middleware passes
router.addTypedHandler(secureHandler);
```

Middleware functions receive the full `ITypedRequest` object and run in the order they were added. Throw a `TypedResponseError` from any middleware to reject the request before it reaches the handler.

□□ Custom Targets with TypedTarget

For non-HTTP transports (WebSockets, broadcast channels, IPC), use `TypedTarget` with a custom post function:

```
// Synchronous target (post returns the response directly)
const target = new TypedTarget({
  postMethod: async (payload) => {
```

```
    // Send via your custom transport and return the response
    return await myWebSocket.sendAndWait(payload);
  },
});

const request = new TypedRequest<IGetUser>(target, 'getUser');
const user = await request.fire({ userId: 'user-123' });
```

For **async targets** where the response arrives separately (e.g., WebSocket push), pair a `TypedTarget` with a `TypedRouter`:

```
const router = new TypedRouter();

const asyncTarget = new TypedTarget({
  postMethodWithTypedRouter: async (payload) => {
    // Fire-and-forget — response will arrive via router
    mySocket.send(JSON.stringify(payload));
  },
  typedRouterRef: router,
});

// When the response arrives later, route it back:
mySocket.onMessage((data) => {
  router.routeAndAddResponse(JSON.parse(data));
});
```

□□ Virtual Streams

`VirtualStream` enables **bidirectional binary streaming** over any transport that supports typed requests. Data is automatically chunked with backpressure control:

```
import type { IVirtualStream } from '@api.global/typedrequest-interfaces';

// Define a streaming endpoint
interface IFileUpload {
  method: 'uploadFile';
  request: { filename: string; dataStream: IVirtualStream };
  response: { bytesReceived: number };
}
```

```
// Client side: create and send a stream
const uploadStream = new VirtualStream<Uint8Array>();
const upload = new TypedRequest<IFileUpload>('https://api.example.com/rpc', 'uploadFile');

const responsePromise = upload.fire({
  filename: 'data.bin',
  dataStream: uploadStream,
});

// Send data through the stream
await uploadStream.sendData(new TextEncoder().encode('Hello, World!'));
await uploadStream.close();

const result = await responsePromise;
console.log(result.bytesReceived);
```

```
// Server side: receive and process the stream
const uploadHandler = new TypedHandler<IFileUpload>('uploadFile', async (req) => {
  let total = 0;
  // Read chunks from the stream
  const chunk = await req.dataStream.fetchData();
  total += chunk.byteLength;

  return { bytesReceived: total };
});
```

VirtualStreams also integrate with the Web Streams API:

```
// Pipe a ReadableStream into a VirtualStream
await virtualStream.readFromWebstream(readableStream);

// Pipe a VirtualStream into a WritableStream
await virtualStream.writeToWebstream(writableStream);
```

⚠ Error Handling

Throw `TypedResponseError` inside handlers to send structured errors back to the caller:

```
const handler = new TypedHandler<IGetUser>('getUser', async (req) => {
  const user = await db.findUser(req.userId);
  if (!user) {
    throw new TypedResponseError('User not found', { userId: req.userId });
  }
  return { username: user.name, email: user.email };
});
```

On the client side, `TypedResponseError` is thrown when the server responds with an error:

```
try {
  await getUser.fire({ userId: 'nonexistent' });
} catch (err) {
  if (err instanceof TypedResponseError) {
    console.error(err.errorText); // 'User not found'
    console.error(err.errorData); // { userId: 'nonexistent' }
  }
}
```

□□ Traffic Monitoring Hooks

Monitor all TypedRequest traffic with global or per-router hooks:

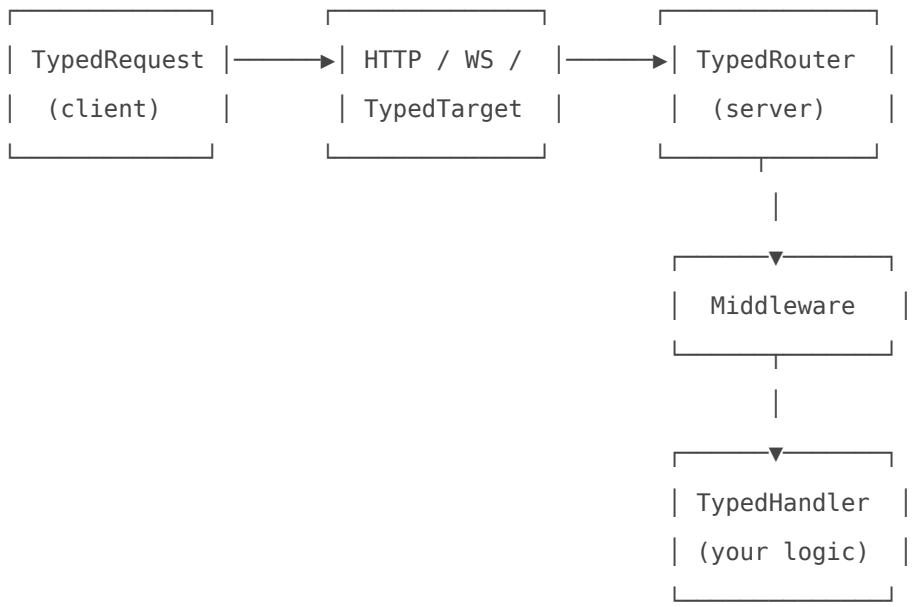
```
// Global hooks — apply to ALL routers and requests across bundles
TypedRouter.setGlobalHooks({
  onOutgoingRequest: (entry) => {
    console.log(`→ ${entry.method} [${entry.correlationId}]`);
  },
  onIncomingResponse: (entry) => {
    console.log(`← ${entry.method} took ${entry.durationMs}ms`);
  },
  onIncomingRequest: (entry) => {
    console.log(`⇒ handling ${entry.method}`);
  },
  onOutgoingResponse: (entry) => {
    if (entry.error) console.error(`⇐ ${entry.method} error: ${entry.error}`);
  },
});
```

```
// Per-router hooks
router.setHooks({
  onIncomingRequest: (entry) => metrics.trackRequest(entry),
});

// Skip hooks for internal requests (e.g., health checks)
const internalReq = new TypedRequest<IHealthCheck>(target, 'healthCheck');
internalReq.skipHooks = true;
```

Global hooks are shared across all bundles via `globalThis`, making them ideal for application-wide observability.

□□ Architecture Overview



Component	Role
TypedRequest	Fires typed requests against a URL or TypedTarget
TypedTarget	Abstracts the transport layer (HTTP, WebSocket, custom)
TypedRouter	Routes incoming requests to the correct handler; composable via `addTypedRouter()`
TypedHandler	Processes a single method and returns a typed response
Middleware	Pre-handler functions for auth, validation, logging — throw `TypedResponseError` to reject
VirtualStream	Bidirectional binary streaming with backpressure over any supported transport

Component	Role
TypedResponseError	Structured error propagation across the wire
TypedTools	Guard validation and transport-layer context available inside handlers

License and Legal Information

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

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 or third parties, 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 or the guidelines of the respective third-party owners, and any usage must be approved in writing. Third-party trademarks used herein are the property of their respective owners and used only in a descriptive manner, e.g. for an implementation of an API or similar.

Company Information

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

For any legal inquiries or 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.

changelog.md for @api.global/typedrequest

2026-03-03 - 3.3.0 - feat(typedrouter)

add middleware support to TypedRouter and export middleware type

- Introduces TMiddlewareFunction and TypedRouter.addMiddleware() to run pre-handler middleware (throw TypedResponseError to reject).
- Adds getTypedHandlerAndRouter() to determine owning router so middleware runs on the correct router.
- Middleware errors are converted into response errors, encoded for network, and outgoing hooks are called; request short-circuits if middleware rejects.
- TypedRouter is now generic (TReqConstraint) and several internal references updated to TypedRouter for compatibility.
- Exports TMiddlewareFunction from index.ts so consumers may reference middleware types.
- Documentation (readme) updated with middleware usage and guidance.
- Tests updated: browser test replaced/renamed to chromium, server tests updated to use TypedServer. Data handling in test adjusted to support Buffer/Uint8Array and serialized Buffer shapes.
- package.json: devDependencies and some deps bumped; build script simplified (removed legacy flags).

2026-03-01 - 3.2.7 - fix(virtualstream)

reconstitute JSON-serialized binary data in VirtualStream; update docs, build config, and dependency bumps

- Add VirtualStream.reconstituteBinaryData to restore Buffer/Uint8Array shapes lost to JSON serialization and use it when handling chunk data and response chunkData.
- Remove .gitlab-ci.yml and add a tsbundle bundle config in npmextra.json; update package.json build script to invoke tsbundle without the previous extra argument.
- Bump multiple devDependencies and dependencies to newer patch/minor versions.
- Significantly expand README with usage examples for TypedRequest, TypedHandler, TypedRouter, TypedTarget, VirtualStream, error handling, hooks, and architecture overview.
- Minor README/legal wording clarifications.

2026-02-11 - 3.2.6 - fix(deps)

upgrade @push.rocks/webrequest to ^4.0.1 and adapt TypedRequest to new API

- Bump dependency @push.rocks/webrequest from ^3.0.37 to ^4.0.1 and update code to match new client/API
- ts/classes.typedrequest.ts: replace plugins.webrequest.WebRequest with plugins.webrequest.WebrequestClient and change postJson call to pass options ({ cacheStrategy: 'cache-first' }) instead of a boolean cache flag
- npmextra.json: reorganize config keys (gitzone -> @git.zone/cli, tsdoc -> @git.zone/tsdoc), add release registries and accessLevel, and add @ship.zone/szci entry

2025-12-04 - 4.0.0 - BREAKING CHANGE(typedrouter)

Introduce options object for TypedRouter.routeAndAddResponse (localRequest, skipHooks); add defaultRouteOptions and make hook calls respect skipHooks; bump package version to 3.2.3

- Changed TypedRouter.routeAndAddResponse signature to accept an options object ({ localRequest?: boolean; skipHooks?: boolean }) instead of a boolean second argument — this is a breaking API change.
- Added TypedRouter.defaultRouteOptions with defaults { localRequest: false, skipHooks: false }.
- Routing now respects options.localRequest and options.skipHooks; hook calls (onIncomingRequest, onOutgoingResponse, onIncomingResponse) are skipped when skipHooks is true to avoid hook recursion or duplicate handling (useful for broadcast-received messages).
- Bumped package.json version to 3.2.3.

2025-12-04 - 3.2.2 - fix(typedrequest)

Add skipHooks flag to TypedRequest to optionally suppress global hooks for internal requests

- Introduce public skipHooks boolean on TypedRequest (default false) with documentation comment explaining it should be used for internal/logging requests to prevent infinite loops.
- Guard calls to global hooks (onOutgoingRequest and onIncomingResponse) in TypedRequest.fire() so hooks are not invoked when skipHooks is true.

2025-12-04 - 3.2.1 - fix(typedrouter)

Use globalThis-backed globalHooks for TypedRouter to enable cross-bundle sharing; fix merging and clearing of global hooks.

- Replace static globalHooks field with getter/setter that stores hooks on globalThis so hooks are shared across bundles.
- Fix setGlobalHooks to merge new hooks with existing ones (avoiding accidental overwrite).
- Update clearGlobalHooks to clear the globalThis storage used for hooks.

2025-12-04 - 3.2.0 - feat(typedrouter)

Add request/response hooks and monitoring to TypedRouter; emit hooks from TypedRequest; improve VirtualStream encoding/decoding; re-export hook types

- Introduce ITypedRequestLogEntry and ITypedRouterHooks interfaces to represent structured traffic log entries and hook callbacks.
- Add static globalHooks on TypedRouter with helper APIs TypedRouter.setGlobalHooks and TypedRouter.clearGlobalHooks for global traffic monitoring.

- Add instance-level hooks on TypedRouter (setHooks) and a unified callHook() that invokes both global and instance hooks safely (errors are caught and logged).
- TypedRequest now emits onOutgoingRequest before sending and onIncomingResponse after receiving, including timestamps, duration and payload/error details.
- TypedRouter now emits lifecycle hooks while routing: onIncomingRequest when a request arrives, onOutgoingResponse for responses (both success and handler-missing error cases), and onIncomingResponse when responses arrive to be fulfilled.
- VirtualStream.encodePayloadForNetwork and decodePayloadFromNetwork were enhanced to recurse into arrays and nested objects (preserving special built-ins) to correctly handle embedded virtual streams.
- Re-export ITypedRequestLogEntry and ITypedRouterHooks from the package index for external consumption.

2025-12-03 - 3.1.11 - fix(virtualstream)

Expose transport localData to handlers via TypedTools; improve VirtualStream payload encode/decode to preserve built-ins and handle nested arrays/objects

- TypedHandler: pass typedRequest.localData into a TypedTools instance so handlers can access transport-layer context (e.g. websocket peer).
- TypedTools: add a public localData property to store transport-specific context available to handlers.
- VirtualStream.decodePayloadFromNetwork: preserve built-in objects (Set, Map, Date, RegExp, Error, Promise or thenable) to avoid incorrect transformation.
- VirtualStream.encodePayloadForNetwork / decodePayloadFromNetwork: added proper recursion for arrays and objects to correctly handle nested payloads and virtual streams, with path tracking to support deduplication logic.

2024-10-16 - 3.1.10 - fix(VirtualStream)

Fix stream closure logic in `writeToWebstream` method

- Added `writer.releaseLock()` call before closing WritableStream when `closingBit` is received in `writeToWebstream` method.

2024-10-16 - 3.1.9 - fix(VirtualStream)

Ensure writable streams are correctly closed asynchronously to prevent potential sync issues.

- Updated VirtualStream to use 'await' when closing writable streams, ensuring proper asynchronous handling.

2024-10-16 - 3.1.8 - fix(VirtualStream)

Fix stream closing behavior to correctly handle closing bits

- Introduced a 'closingBit' constant to properly signal the end of stream data.
- Updated the 'readFromWebstream' function to send a closing bit upon completion if 'closeAfterReading' is true.
- Modified the 'close' method to optionally send a closing bit when terminating the stream.

2024-10-16 - 3.1.7 - fix(VirtualStream)

Fix issue in VirtualStream to handle null values during data writing.

- Ensured writableStream closes gracefully when null values are encountered.
- Added a null check before writing data to the writableStream to prevent errors.

2024-10-16 - 3.1.6 - fix(VirtualStream)

Fix backpressure handling in VirtualStream workOnQueue method

- Resolved an issue in the workOnQueue method of VirtualStream where concurrent execution was not properly managed.
- Introduced a workingDeferred promise to ensure proper queue handling and resolve potential race conditions.

2024-10-16 - 3.1.5 - fix(virtualstream)

Add console log for debugging backpressure feedback loop

- Inserted a console log message to provide insight when waiting due to backpressure in the workOnQueue method.

2024-10-16 - 3.1.4 - fix(VirtualStream)

Corrected the logic for backpressure handling in response

- Fixed backpressure flag assignment in the response handling logic of VirtualStream.
- Ensured correct negation logic for checking receive backpressure status.

2024-10-14 - 3.1.3 - fix(VirtualStream)

Fix keepAlive flag handling in VirtualStream and added stream closure in tests

- Ensure that the keepAlive status is correctly maintained in the keepAlive trigger method.
- Added closure of VirtualStreams in the test suite for proper resource cleanup.

2024-10-14 - 3.1.2 - fix(core)

Fix incorrect backpressure logic in VirtualStream class

- Corrected the logic for determining backpressure status by checking the available space in the receiveBackpressuredArray.
- Introduced a looping mechanism to wait when the other side is backpressured before sending more data.

2024-10-14 - 3.1.1 - fix(virtualstream)

Fix handling of virtual streams for proper shutdown

- Ensured that writeToWebstream method checks for remaining items in receiveBackpressuredArray before closing.
- Corrected package.json dependency for @push.rocks/tapbundle.
- Updated @types/node to version 22.7.5.

2024-10-11 - 3.1.0 - feat(virtualstream)

Enhance VirtualStream with optional closure when reading from webstream

- Added an optional parameter `closeAfterReading` to the `readFromWebstream` method.
- The stream will close automatically after reading if `closeAfterReading` is set to true.

2024-10-11 - 3.0.33 - fix(test)

Increase delay duration before stopping the server in test suite.

- Adjusted the delay time from 1000 ms to 10000 ms before stopping the server to ensure tests complete smoothly.

2024-09-06 - 3.0.32 - fix(virtualstream)

Fix keep-alive loop handling and test cleanup

- Prevent unnecessary keep-alive loop from starting on the responding side
- Add logging for keep-alive loop initiation in VirtualStream
- Temporarily comment out stream close and tap forceful stop in test to avoid abrupt termination

2024-09-06 - 3.0.31 - fix(core)

Updated dependencies and added close method to VirtualStream

- Updated dependencies in package.json for better compatibility
- Added close method to VirtualStream class in ts/classes.virtualstream.ts for more graceful stream termination

2024-05-31 - 3.0.28 - Error Handling

Enhancement to error handling mechanisms.

- Logs now include the method to which an error was given.

2023-08-04 - 3.0.0 - Core

Introduced a breaking change.

- Major update to core functionalities.