UID-Explore index.js

Stand 27.09.2025

Position im System

`index.js` ist ein **Barrel**. Es bündelt ausgewählte Kern-APIs der Base-Schicht und stellt sie unter einem stabilen Pfad bereit. Konsumenten greifen damit auf **Director-Fabrik** und **Event-Bus** zu, ohne interne Dateipfade kennen zu müssen.

- Ein Import auf Paket-Ebene lädt **keinen** unnötigen Code nach. `index.js` enthält **nur Re-Exports** und hat **keine Seiteneffekte**.
- Das macht `index.js` zum **sichtbaren Vertrags-Layer** zwischen Base und allen aufrufenden Modulen in Input- und Presentation-Schicht.

Ist-Zustand der Exporte

```
// UID-E · base/index.js (Barrel)
export { createUID } from './uid.js';
export * as bus from './bus.js';
```

- **`createUID`** kommt aus `uid.js` (Director-Fabrik). Sie initialisiert den UID-Explore-Kern und verbindet Engine, Schema und Bus.
- **`bus`** wird als **Namespace-Export** bereitgestellt und enthält die minimalen Operationen `on`, `off`, `emit` sowie die DOM-Spiegelung für CustomEvents.

Öffentliche API über **`index.js`**

Empfohlene Import-Formen

```
// 1) Präziser Paket-Import (stabil, tree-shaking-freundlich)
import { createUID, bus } from '@uid/base';

// 2) Explizit (falls gewünscht):
import { createUID } from '@uid/base/index.js';
import { on, off, emit } from '@uid/base/bus.js';

// 3) Namespace (wenn beides gemeinsam geführt werden soll)
import * as UID from '@uid/base';
// UID.createUID(...), UID.bus.on(...)
```

Kein Default-Export. Das ist gewollt und hält den Vertrag explizit.

Import-Map und Pfadkontrakt

Für den stabilen Zugriff muss die Import-Map beide Varianten abdecken: den **Barrel** selbst und die **Ordner-Basis**.

```
<script type="importmap">
{
  "imports": {
    "@uid/base": "/12-2_base/index.js",
    "@uid/base/": "/12-2_base/"
  }
}
</script>
```

- **Ohne** Slash-Variante (`"@uid/base/": …`) schlagen tiefe Importe wie `@uid/base/bus.js` fehl.
- JSON muss **valide** sein. Kommentare oder hängende Kommata verursachen `Failed to parse import map`.

Zusammenspiel mit Director, Engine und Bus

- `createUID` kapselt die Orchestrierung des Directors und bindet Engine, Schema und Präsentations-Wiring zusammen.
- `bus` ist zentraler Transportweg für Parameter-, Modell- und Simulations-Events. Präsentationsmodule (Chart, KPI, Vektorrad) hören darüber zu.
- `index.js` sorgt dafür, dass beides unter **einem** stabilen Import-Namen erreichbar ist und sich Refactorings hinter dem Barrel verbergen können.

Garantien und Designprinzipien

- **Barrel only:** keine Berechnung, keine Nebenwirkungen, nur Re-Exports.
- **Stabilität:** Änderungen der internen Dateistruktur bleiben ohne Auswirkungen auf Konsumenten.
- **Explizitheit:** Keine Wildcard-Re-Exports außer für den `bus`-Namespace. Das verhindert unkontrolliertes API-Wachstum.

Typische Fehlerbilder und Gegenmittel

- **Invalid Import-Map** führt zu Syntaxfehlern beim Laden von `index.js`.
  Gegenmittel: JSON strikt validieren, beide `@uid/base`-Keys setzen.
- **Auflösungsfehler bei Aliassen** wie „Failed to resolve module specifier …“.
  Gegenmittel: Slash-Alias ergänzen, korrekte Root-Pfade prüfen.

- **Falscher MIME-Type** beim lokalen Dev-Server. ESM-Module benötigen `text/javascript`. Gegenmittel: Server richtig konfigurieren (z. B. Vite, esbuild-serve, http-server mit passenden Headern).
- **Hardcodierte Tiefenimporte** in Fremdcode. Gegenmittel: immer `@uid/base` nutzen, nicht `../../12-2_base/...`.

Erweiterbarkeit

- **Weitere Kern-APIs bündeln**: z. B. kuratierte Re-Exports aus `schema.js` oder `engine.js` anbieten, ohne die „reine Barrel"-Eigenschaft zu verlieren.
- **Versionierung kommunizieren**: Semver-Regel – nur neue Exporte sind **minor**, Brüche an bestehenden Named-Exports sind **major**.
- **Typ-Definitionen**: JSDoc-Typen im Barrel re-exportieren, damit IDE-Hints bei `import { createUID } …` direkt greifen.

Mini-Rezepte

Probe-Import im Dev-Tool

```
// Direkt im Browser-DevTool der laufenden Seite
const m = await import('@uid/base');
console.log(Object.keys(m)); // ['createUID', 'bus']
```

UID starten und Bus nutzen

```
import { createUID, bus } from '@uid/base';

const app = createUID({ mount: document.getElementById('app') });
const off = bus.on('uid:e:sim:data', e => console.log(e.series));

// … später
off();
```

Quick-Map Dateien

- `12-2_base/index.js` Barrel für Base-APIs
- `12-2_base/uid.js` Director-Fabrik `createUID`
- `12-2_base/bus.js` Event-Bus mit DOM-Spiegel

Damit ist klar, welchen Zweck `index.js` erfüllt, wie du es importierst, welche Verträge es garantiert und wo du Erweiterungen sauber andocken kannst.