UID-Explore · Support Layer
Boot ESM · v1.0.0

Dokumentation für den Ordner `app/boot` und seine Unterordner

Stand 04.10.2025 · Version 1.0.0

---

1. Zweck und Einordnung

Der Boot-Layer startet UID-Explore. Er setzt die technische Bühne, verdrahtet die Infrastruktur und übergibt anschließend an die Darstellung. Er berührt keine Modell- oder Präsentationslogik und bleibt strikt von **Input**, **Base** und **Präsentation** getrennt.
Zentrale Prinzipien sind: **klare Reihenfolge**, **idempotenter Start**, **saubere Aliase** statt Root-absoluter Pfade.

---

2. Boot-Sequenz im Überblick

1. **Tooltips** – Styles und Layer sichern, Cursor-Tooltip aktivieren.
2. **Infra** – Sim-Bridge und Pointer-Bridge einmalig verdrahten.
3. **Engine** – Explore-Engine initialisieren, Seeds bereitstellen.
4. **Mount** – Widgets nacheinander montieren, Rehydrate anbinden.

Die Reihenfolge ist stabil und verhindert Seiteneffekte: erst Oberfläche absichern, dann Ereignisflüsse, dann Rechenkern, zuletzt die sichtbaren Module.

---

3. Dateien und Rollen (Quick-Map)

Einstieg

- `boot.js` – einziger `<script type="module">` in der Seite. Lädt `boot/bridge.js`.
- `boot/bridge.js` – Composition Root. Startet die Sequenz Tooltips → Infra → Engine → Mount.

Tooltips (`boot/tooltips/`)

- `index.js` – Styles und Layer sichern.
- `tooltip.js` – Cursor-Tooltip.
- `tips.js` – Binder für Header und Karten.
- `boot.js` – ruft Sicherung und Cursor-Tooltip auf.

Infra (`boot/infra/`)

- `sim-bridge.js` – Wiedergabe und Status normalisieren.
- `sim-bridge.boot.js` – ruft die Sim-Bridge genau einmal.
- `pointer-bridge.js` – vereinheitlicht Timeline- und Hover-Ereignisse zu einem Pointer.
- `pointer-bridge.boot.js` – ruft die Pointer-Bridge genau einmal.

Engine (`boot/engine/`)

- `index.js` – Engine-Shim.
- `boot-engine.js` – Explore-Engine starten, Bus mit Startwerten füllen.

Mount (`boot/mount/`)

- `index.js` – ruft die Widget-Mounts in fester Reihenfolge.
- `chart.js`, `kpi.live.js`, `kpi.key.js`, `gridwave.js`, `params.classic.js`, `params.formulas.js`, `state.visuals.js`, `leq.js`.

Rehydrate (`boot/rehydrate/`)

- `core.js` – kleine Helfer-API, die Widgets nach Daten- und Sichtbarkeitswechseln wieder synchronisiert.

Shims (`app/shims/`)

- `leq.index.js`, `leq.actions.js` – kapseln Importpfade für **living equation** ohne den Quellordner zu ändern.

---

4. Import-Map und Alias-Policy

Erlaubt sind **relative Pfade** und Aliase aus der Import-Map.
Nicht erlaubt sind Root-absolute Pfade wie `/uid/...` oder `/increments/...` im gesamten Boot-Baum.

Empfohlene Aliase:

- `@uid/app/` → `../12-4_support/app/`
- `@uid/support/` → `../12-4_support/`
- `@uid/base/` → `../12-2_base/`
- `@uid/input/` → `../12-1_input/`
- `@uid/pres/` → `../12-3_presentation/`
- `@uid/widgets` und `@uid/widgets/` → Widgets-Einstieg und -Ordner

Beispiel

```
// gut
```

```
import { initRehydrate } from '@uid/app/boot/rehydrate/core.js';

// vermeiden
import '/uid/12-4_support/app/boot/rehydrate/core.js';
```

---

5. Idempotenz und Guards

- Jede Boot-Stufe darf **ohne Nebenwirkung** mehrfach aufgerufen werden. Fenster-Guards (`window.__uid…Once`) sichern das ab.
- Infra-Boot ruft Brücken nur **einmal** auf und speichert den Teardown.
- Tooltips prüfen Head und Body und schalten sich erst zu, wenn der DOM soweit ist.
- Mount-Module geben klare Logs mit festem Wortlaut aus, damit Smoke-Tests zuverlässig sind.

---

6. Infra-Ereignisvertrag

Konsumierte Ereignisse

- `uid:e:timeline:set { t? , idx? }`
- `uid:e:sim:play`, `uid:e:sim:pause`, `uid:e:sim:reset`
- `uid:e:sim:speed:set { speed | label }`
- `uid:e:integrator:set { kind }`
- `uid:e:sim:data { series:{ t:number[] , … } }` (mit Replay)

Erzeugte Ereignisse

- `uid:e:sim:pointer { idx , t? , src:'pointer-bridge' }` – gemeinsame Zeitmarke
- `uid:e:sim:status { running , speed , idx }`
- `uid:e:integrator:status { kind }`

Ziel ist ein **einheitlicher Pointer** und ein **sichtbarer Simulationsstatus** für alle Widgets.

---

7. Mount-Kontrakte (Kurzfassung)

Chart
Host `#chart-widget` und `#chart-host`. Bindet Header-Werkzeuge, zeichnet die Simulation, Rehydrate-ID `chart`. Log: `[mount-widgets] Chart ready`.

Live KPIs
Host `#kpi-widget`. Deck „live“, Actions am Host, One-Header-Policy aktiv. Rehydrate-ID `kpi-live`. Log: `KPI ready`.

Key KPIs
Host `#kpi-static-widget`. Rendert statische Kennzahlen, Actions am Header. Rehydrate-ID `keykpi`. Log: `Key KPI ready`.

GridWave
Host `#gw-widget` und `#gw-host`. Lädt Actions alias-basiert mit Fallback, Rehydrate ruft `resize`. Rehydrate-ID `gridwave`. Log: `GridWave ready`.

Parameter klassisch
Host `#params-widget`. Bedienelemente für Parameter. Rehydrate-ID `params-classic`. Log: `Params Classic ready`.

Parameter Formeln
Host `#params-formulas-widget`. Wartet auf MathJax, dann rendert Formeln. Rehydrate-ID `params-formulas`. Log: `Params Formulas ready`.

State Visuals
Host `#sv-widget` und `#vt-host`. Wählt automatisch eine passende Mount-Variante, robustes Fallback, optionaler Adapter. Rehydrate-ID `state`. Log: `State Visuals ready`.

Kern-Gleichung (LEQ)
Host `#core-equation-widget` und `#core-equation`. Import **nur über App-Shims**, optional Actions, Adopt-Fallback für bereits gerenderten DOM. Rehydrate-ID `leq`. Log: `LEQ ready`.

---

8. Rehydrate-Verhalten

```
initRehydrate(widget, bus, { id , refresh })
```

- beim Mount,
- bei neuen Simulationsdaten (mit Replay),
- beim Sichtbarwerden und beim Umschalten
  sendet Rehydrate die aktuelle Zeit erneut an die Timeline, ruft optional `refresh()` und löst einen Fenster-Resize aus. Rückgabe ist ein **Cleanup**, der Observer beendet.

---

9. Kompat-Shims

Zweck: Pfade mit Leerzeichen oder abweichenden Strukturen kapseln, ohne Quellordner zu verändern.
Aktuell genutzt für **living equation**:

- `app/shims/leq.index.js` → re-exportiert das eigentliche Modul.
- `app/shims/leq.actions.js` → re-exportiert optionale Actions.
  Shims sind leicht wieder entfernbar, sobald der Quellpfad vereinheitlicht ist.

---

10. Performance und Robustheit

- Status-Emissionen per Microtask, Animationen per `requestAnimationFrame`.
- Tooltip-Layer mit `pointer-events: none`, geringe Layoutkosten.
- Keine DOM-Zugriffe vor `head`/`body`. Bei Bedarf warten die Module auf `DOMContentLoaded`.
- Alle dynamischen Imports nutzen Aliase oder getestete Fallbacks.
- Keine Blockierung des Event-Loops durch teure Initialisierungen im Boot.

---

11. QA-Checkliste

Konsole

- `[boot] sequence ok (tooltips → sim-bridge → pointer-bridge → engine → mount)`
- `[pointer-bridge] ready`
- `[mount-widgets] … ready` für alle Widgets

Netzwerk

- Keine `404` oder `ERR_ABORTED` aus dem Boot-Baum.

Ereignisse

- `EBUS.emit('uid:e:timeline:set', { t: 0 })` → `EBUS.getLast('uid:e:sim:pointer')` liefert Objekt.
- `EBUS.emit('uid:e:sim:play')` → `uid:e:sim:status.running === true`.

Pfadhygiene

- Grep im Boot-Baum: keine Imports, die mit `'/'` beginnen.

UI

- Keine doppelten Header bei KPI-Widgets.
- LEQ vorhanden oder sauberer Adopt-Fallback.

---

12. Fehlerbilder und Gegenmittel

Root-absolute Importe im Boot
Symptom: 404 im Network-Tab.
Fix: Aliase oder relative Pfade benutzen, Import-Map nachziehen.

Pointer kommt nicht an
Symptom: Widgets reagieren nicht auf Zeit.
Fix: `pointer-bridge.boot.js` in der Bridge-Reihenfolge prüfen, Guard entfernen, wenn blockierend.

LEQ Import-Fehler
Symptom: 404 auf `living-equation` oder Pfade mit Leerzeichen.
Fix: nur über `app/shims/leq.*` importieren.

Doppelte Header bei KPI
Symptom: zwei Kopfzeilen nach dem Mount.
Fix: One-Header-Policy im betreffenden Mount aktivieren.

GridWave Actions fehlen
Symptom: Aktionen nicht verfügbar.
Fix: Alias-Import `@uid/pres/grid/gw.widget-actions.js` prüfen, Fallback aktivieren, falls nötig.

Rehydrate triggert nicht
Symptom: Widget bleibt nach Datensprung stehen.
Fix: `initRehydrate` im Mount ergänzen, ID setzen, optional `refresh` übergeben.

Tooltips crashen früh
Symptom: Fehler bei fehlendem Body.
Fix: `tooltips/index.js` laden lassen, bis `DOMContentLoaded`. Guards aktiv.

---

13. Changelog

v1.0.0 · 2025-10-04
Erstfassung des Boot-ESM: Bridge-Sequenz, Sim-Bridge, Pointer-Bridge, Engine-Shim, Rehydrate-Core, eigenständige Mounts für Chart, KPIs, GridWave, Parameter, State Visuals und LEQ, App-Shims für LEQ, konsolidierte Alias-Policy und QA-Marker.

Hinweis zur Pflege
Neue Widgets erhalten ein eigenes Mount-Modul, werden in `boot/mount/index.js` eingetragen und geben einen konsistenten Ready-Log aus. Änderungen an Aliasen gehören in die Import-Map der HTML-Seite, nicht in den Boot-Baum.