UID-Explore · Grid Wave

Stand 01.10.2025 · Version 1.4.0

Architektur im Überblick

UID-Explore verfolgt eine lose gekoppelte **ESM-Architektur**. Die Schichten interagieren ausschließlich über den **Event-Bus** und spiegeln ausgewählte Signale zusätzlich in den DOM. Dadurch bleiben Module unabhängig, debuggbar und austauschbar. **Grid Wave** gehört zur Schicht **Presentation** und konsumiert ausschließlich Bus-Ereignisse; es publiziert standardmäßig keine System-Events. Die Bus-Konventionen (on/off/emit, DOM-Spiegelung, Event-Familien) sind identisch zur zentralen Bus-Doku.

bus_UID-E Explore Event-Bus

Schichten und Rollen (kontextualisiert für Grid Wave)

- **Input (`12-1_input/parameters`)** sendet Parameter-Events.
- **Base (`12-2_base`)** rechnet deterministisch und publiziert konsistente **SEIRVD-Zeitreihen**.
- **Presentation (`12-3_presentation`)** zeichnet Chart, KPI, Vektorrad **und Grid Wave**; alle reagieren auf Status und Pointer.
- **Bus (`12-2_base/bus.js`)** verbindet die Teile und spiegelt Events in den DOM.

  bus_UID-E Explore Event-Bus

Der Bus (Bezug & Konventionen)

Datei (System): `12-2_base/bus.js` – minimal, synchron.

```
on(type, handler)   // registriert Listener, liefert unsubscribe()
off(type, handler)  // entfernt Listener
emit(type, payload) // ruft Listener synchron auf, spiegelt zusätzlich als
DOM CustomEvent
```

Eigenschaften: Listener pro Typ in einer `Map`, **synchrones** `emit`, DOM-Spiegelung via `CustomEvent(type,{detail})`, kleiner Debug-Hook für ausgewählte Events. Grid Wave nutzt exakt diese API, sendet aber selbst nur **Widget-interne** UI-Aktionen (Header-Buttons) **nicht** über den System-Bus.

bus_UID-E Explore Event-Bus

DOM-Mithören (Debug-Beispiel)

```
window.addEventListener('uid:e:sim:data', ev => console.log('series',
ev.detail.series))
```

Die Möglichkeit, System-Events am `window` mitzulesen, bleibt vollständig erhalten.

bus_UID-E Explore Event-Bus

Dateien & Rollen (Quick-Map)

- `visual tool/grid wave/gridwave.js` – **Kern**: Metriken/Modi, Ranking, Diskretisierung, Rendering
- `visual tool/grid wave/index.js` – **Wiring**: Bus-Anbindung, Seed/Persistenz, Power-Gating, Mount/Unmount
- `visual tool/grid wave/gw.widget-actions.js` – **Header-Controls** (Modus, Dichte, Crisp, Dot-Ratio)
- `visual tool/grid wave/grid.i18n.json` – **i18n** (Modus-Texte, didaktischer Hinweis „keine Karte“)
- `visual tool/grid wave/styles.css` – **Layout/Theming** (Quadrat, Canvas-Container, CSS-Variablen)
- `visual tool/grid wave/demo.html` – **Demo-Harness** (lokale Tests)

(Nomenklatur und Ordnerstruktur sind konsistent mit den übrigen Präsentationsmodulen.)

Datenfluss

Boot

- Beim Mount liest `index.js` **Seed/Context** (z. B. `driverKey`) und Nutzerpräferenzen (Modus/Dichte/Crisp/Dot-Ratio) aus `localStorage`, registriert Listener und fordert keine Berechnung an.
- Sobald **`uid:e:sim:data`** erscheint, wird die **SEIRVD-Serie** gespeichert und ein erster Render ausgelöst. **`uid:e:model:update`** wird für Kontext/Tooltips mitgeführt. Die Reihenfolge und Semantik entsprechen dem System-Standard (erst `model:update`, dann `sim:data`).

  bus_UID-E Explore Event-Bus

Interaktion

- **Pointer**: `uid:e:sim:pointer` setzt den Frame-Index; Grid Wave rendert den passenden Zeitschritt **ohne** Neuberechnung der Engine. Das entspricht dem üblichen Playhead-Verhalten.

  bus_UID-E Explore Event-Bus

- **Serien-Sichtbarkeit** *(optional)*: `uid:e:viz:series:state` kann konsumiert werden, um Bänder temporär auszublenden (didaktische Kohärenz mit Chart-Legende). Default: off.

  bus_UID-E Explore Event-Bus

- **Engine-Status**: `uid:e:engine:status` wird zur Kontextanzeige (Integrator, Steps) genutzt, löst jedoch keine Layoutänderung aus.

  bus_UID-E Explore Event-Bus

Director & Engine (Bezug)

Der **Director** normalisiert Parameter, koppelt Größen algebraisch (z. B. $R_0 \leftrightarrow \beta, \gamma$; $D \leftrightarrow \gamma$; $L \leftrightarrow \sigma$) und publiziert **`uid:e:model:update`** sowie **`uid:e:sim:data`**. Die **Engine** liefert deterministische Zeitreihen (SIR, SEIR, SIRD, SIRV, optional SIS). Grid Wave vertraut **ausschließlich** auf diese Zeitreihen; es rechnet selbst **nichts** numerisch nach.

bus_UID-E Explore Event-Bus

Präsentation: Grid Wave (Rolle & Abgrenzung)

Was es ist: Ein pseudo-räumliches **Raster-Widget** für **SEIRVD**-Zeitreihen. Jeder der g^2Punkte steht für eine **diskrete Anteilseinheit** der Population. Die **Belegungsreihenfolge** der Zellen hängt vom **Modus** (Metrik) ab: *Proportional*, *Wave*, *Cluster*, *Hybrid*.

Was es nicht ist: Keine Landkarte, kein Agenten-/Netzwerk-Modell, kein Kontaktgraph. Positionen tragen **keine reale Geografie**; es werden **ausschließlich** Anteile visualisiert.

Event-Referenz (Grid-Wave-Sicht)

Konsumierte Familien & Wirkung

Event	Quelle	Wirkung im Widget
`uid:e:sim:data`	Director/Engine	SEIRVD-Reihen übernehmen → Diskretisierung → (Re)Render
`uid:e:sim:pointer`	Chart/Overlay	Frame-Index setzen (ohne neue Engine-Runs)
`uid:e:model:update`	Director	Seed/Context prüfen, Metrik ggf. neu (bei Seed-Wechsel)

Event	Quelle	Wirkung im Widget
`uid:e:engine:status`	Director	Integrator/Steps für Tooltips (rein informativ)
`uid:e:viz:series:state` *(optional)*	Legend	Bänder temporär aus-/einblenden (didaktische Kohärenz)

Diese Familie und Semantik entsprechen 1:1 der Event-Bus-Doku und nutzen deren DOM-Spiegelung für Debug.

bus_UID-E Explore Event-Bus

Payload-Gestalt (erwartet)

```
// sim:data
{
  series: { S:[], E:[], I:[], R:[], V:[], D:[] }, // Float64Array|number[]
  dt, T
}

// sim:pointer
{ idx }             // oder { idx: null }

// model:update
{ ...konsolidierte Parameter..., method, driverKey? }  // Context/Seed-Quelle

// viz:series:state (optional)
{ visible: { S:true, E:true, I:true, R:true, V:true, D:true } }
```

Die Formen sind konsistent zum System; fehlende `V`/`D` werden im Widget als 0 interpretiert (Abwärtskompatibilität).

bus_tech

Mathematik (SEIRVD) – *Bezug: Engine & Mapping: Grid Wave*

1) Dynamik (vom Rechenkern geliefert)

$$
\begin{aligned}
\dot{S} &= -\beta \frac{SI}{N_{\text{alive}}} - \nu S, \\
\dot{E} &= \beta \frac{SI}{N_{\text{alive}}} - \sigma E, \\
\dot{I} &= \sigma E - (\gamma + \mu) I, \\
\dot{R} &= \gamma I, \\
\dot{V} &= \nu S, \\
\dot{D} &= \mu I,
\end{aligned}
$$

mit $N_{\text{alive}} = S + E + I + R + V$, $N_{\text{tot}} = N_{\text{alive}} + D$. Kopplungen: $\sigma = 1/L_E$, $\gamma + \mu = 1/D_I$, $\mu = \rho/D_I$, $\gamma = (1-\rho)/D_I$, $\beta = R_0(\gamma + \mu)$, $\nu = -\ln(1-C)/T_v$. *Diese Beziehungen liegen im Director/Engine; Grid Wave konsumiert nur die Zeitreihen.*

bus_UID-E Explore Event-Bus

2) Diskretisierung (von Anteilen zu Zellen)

- **Rasterdichte** $g \Rightarrow N = g^2$ Zellen.
- **Runden/Kappen** je Frame t_k:

$$
\begin{aligned}
n_D &= \text{round}(DN),\ n_R = \text{round}(RN),\ n_I = \text{round}(IN), \\
n_E &= \text{round}(EN),\ n_V = \text{round}(VN),
\end{aligned}
$$

falls $n_D + n_R + n_I + n_E > N \Rightarrow$ proportional kappen;
$n_V \leftarrow \min$ [illegible]$(n_V, N - (n_D + n_R + n_I + n_E))$;
$n_S = N - (n_D + n_R + n_I + n_E + n_V)$.

- **Belegungsreihenfolge (innen→außen): D → R → I → E → S → V.**
  So bleibt **D** stabiler Kern, **V** ein äußerer Gürtel. (Reihenfolge = reine Visuallogik, **Anteile bleiben exakt**.)

3) Metriken & Modi (Ranking)

Sei Mdie Metrik pro Zelle, $\pi = \text{argsort}(M)$das Ranking.

- **Proportional:** $M(i) = i$(Zeilen-Scan) – neutrale Referenz.
- **Wave:** radial $r(x,y) = \frac{\sqrt{(x-c_x)^2 + (y-c_y)^2}}{r_{\max}}$– konzentrische „Welle“.
- **Cluster:** deterministisches **Value-Noise** $n(x,y)$(Seed via `hashStr`→`mulberry32`) + Mehrfach-Blur – fleckig.
- **Hybrid:** $M = (1-w)\, r + w\, n$, mit $w = \text{smoothstep}(d_0, 1, r)^p$– innen wave, außen cluster.

Determinismus: Seed aus `driverKey`/`simId` ⇒ Muster sind stabil simulationsweit; verschiedene Simulationen unterscheiden sich.

Rendering & Geometrie

- **Crisp-Dots**: integer-Zellgröße, DPR-Ausrichtung, Zentren auf Pixelraster; ein **`Path2D` pro Band** ⇒ je Band **ein `fill()`**.
- **Dot-Ratio**: Radius $r = \lfloor \text{cellsz} \cdot \text{ratio} \rfloor$, `ratio ∈ [0.15,0.48]` (Default 0.33).
- **Theming**: Farben über `--c-s, --c-e, --c-i, --c-r, --c-v, --c-d` (AA/AAA-taugliche Defaults).
- **Responsivität**: `aspect-ratio:1/1`, `ResizeObserver`; bei Dichtewechseln Rebuild von Zentren, Metrik, Ranking.
- **Zeichenreihenfolge**: **D→R→I→E→S→V**; so übermalt V keine inneren Bänder.

Performance

- **Sort** (Ranking) $O(N \log N)$ nur bei **Dichte/Modus/Seed-Änderung**.
- **Re-Draw** $O(N)$ pro Frame (Pointer).
- **Speicher**: Ranking/Metric/Zentren grob ≈ **16 · N Bytes** (z. B. $g = 128 \Rightarrow N = 16384 \Rightarrow \sim 256$ KB).
- **Praxis**: 60 fps bei $g \leq 128$ auf Standardgeräten; $g = 192$ bleibt flüssig dank Batching und integer-Geometrie.
  Die generellen Performance-Prinzipien (Batching, DPR-Korrektheit, getrennte Overlays) entsprechen der System-Präsentationsschicht.

  bus_UID-E Explore Event-Bus

Erweiterungsvorschläge

- **Legenden-Sync** über `uid:e:viz:series:state` aktivieren (S/E/I/R/V/D ein-/ausblenden).

  bus_UID-E Explore Event-Bus

- **Hybrid-Regler** (d_0, p, Blur-Pässe) im Header temporär freischalten (Lehr-Demos).
- **Snapshot-Tests** (Seeds × Dichten) – deterministische PNG-Vergleiche.
- **JSDoc-Typen** für Payloads und Widget-API (IDE-Support).

Typische Fehlerbilder & Gegenmittel

- **Fehlende DOM-Unsubscribe**: `on()` liefert Abmelder; `destroy()` ruft alle sauber ab – Konvention einhalten. (Systemregel)

bus_UID-E Explore Event-Bus

- **Zu kleine Anteile**: 0/1-Zellen-Quantisierung ist **erwartet**; didaktisch per Tooltip erklären.
- **Seed-Drift**: Bei fehlendem `driverKey` deterministisches Fallback verwenden; ansonsten Metrik rebuilden.
- **Überfüllung**: Summe $n_D + n_R + n_I + n_E > N$ – proportional **kappen** (bereits implementiert).

Mini-Rezepte (Debug & QA)

DOM-Events mitloggen (ohne Codeeingriff)

```
for (const ev of
['uid:e:model:update','uid:e:sim:data','uid:e:sim:pointer']) {
  window.addEventListener(ev, e => console.log(ev, e.detail))
}
```

QA-Invarianten (einfacher Check pro Frame)

```
const sum = nS+nE+nI+nR+nV+nD;  // sollte = g*g sein
```

Diese Debug-/QA-Muster entsprechen dem System-Vorgehen (DOM-Spiegelung, klare Invarianten).

bus_intro

Quick-Map Dateien und Rollen (konkret)

- `gridwave.js` – **Metriken/Modi**, Diskretisierung, **Rendering**
- `index.js` – **Bus-Wiring**, Seed/Persistenz, **Power-Gating**
- `gw.widget-actions.js` – **Header-Bedienelemente** (Modus 1–4, Dichte, Crisp, Dot-Ratio)
- `grid.i18n.json` – **Texte/Tooltips** (inkl. „didaktische Visualisierung, keine Karte“)
- `styles.css` – **Layout/Theming** (Quadrat, DPI-Aware Canvas)

(Die Aufzählungsform und Rollenbeschreibung spiegeln die Bus-Referenz „Quick-Map Dateien und Rollen“.)

bus_UID-E Explore Event-Bus

Kopplungen & Invarianten (Widget-Level)

1. **Summe**: $n_S + n_E + n_I + n_R + n_V + n_D = g^2$.
2. **Pro Band exakt** $n_{\ast}$Zellen.
3. **Innen→Außen**: **D→R→I→E→S→V**.
4. **Determinismus**: Bei gleichem Seed, Modus und Dichte bleibt das Muster identisch.

Diese Invarianten entsprechen der Systemidee „Konsistenz und klare Verantwortlichkeiten“.

bus_tech

Event-Familien (Referenz, wortgleich zum System – Grid-Wave-Sicht)

Parameter

`uid:e:params:ready` – Initiale Metadaten (`lang`, `mode`, `model`, `driverKey`) werden gelesen, aber Grid Wave **sendet** kein Gegen-Event.

bus_UID-E Explore Event-Bus

`uid:e:params:change` – löst **keinen** direkten Effekt im Widget aus (Engine-Rechnung folgt → `sim:data`).

bus_UID-E Explore Event-Bus

Modell

`uid:e:model:update` – Kontext/Methodenanzeige, Seed-Änderungen triggern **Metrik-Rebuild**.

bus_UID-E Explore Event-Bus

Simulation

`uid:e:sim:data` – zentrale Quelle: Zeitreihen (SEIRVD), Zeitschritt-Raster; sofortiger Render.
`uid:e:sim:pointer` – reines **Frame-Switching**; kein Engine-Run.

bus_UID-E Explore Event-Bus

Visualisierung

`uid:e:viz:series:state` – *(optional)* Sichtbarkeiten übernehmen; didaktisch nützlich.
`uid:e:viz:overlays:enable` – für Grid Wave **irrelevant** (keine Marker-Overlays).

bus_UID-E Explore Event-Bus

Engine

`uid:e:engine:status` – Integrator/Steps für Kontext-UI.
`uid:e:integrator:set` – **kann** extern gesendet werden; betrifft Grid Wave nur indirekt (neue Serien).

bus_UID-E Explore Event-Bus

Formel

`uid:e:formula:mark`, `uid:e:formula:pulse` – ohne direkte Wirkung auf Grid Wave.

bus_UID-E Explore Event-Bus

Fehler

`uid:e:error` – Anzeige optional (Toast/Console), keine Layoutänderung.

bus_UID-E Explore Event-Bus

Payload-Beispiele (konkret)

```
// typisches sim:data-Detail
{
  series: {
    S: Float64Array, E: Float64Array, I: Float64Array,
    R: Float64Array, V: Float64Array, D: Float64Array
  },
  dt: 0.25,
  T: 240
}
```

Die Form entspricht exakt der System-Event-Doku, mit der Erweiterung **D** (Deceased) gemäß SEIRVD.

bus_tech

Performance (System-Querbezüge)

- **RAF-Bündelung** und **DPR-Skalierung** sind System-Standard; Grid Wave bleibt integer-genau („crisp“) und nutzt Single-Pass-Fills pro Band.
- Präsentationsmodule profitieren von **ResizeObserver** und Canvas-Overlays; Grid Wave benötigt **nur** einen Basis-Canvas.
  Diese Leitlinien decken sich mit der zentralen Performance-Sektion.

  bus_UID-E Explore Event-Bus

Erweiterungsvorschläge (System-kompatibel)

- Optionaler `source`-Schlüssel in Payloads zur klaren Auslöser-Kennzeichnung (z. B. `playhead`).
- `model:update` kann `model` immer mitgeben (Kontext ohne Zusatzstatus).
- JSDoc-Typen für bessere IDE-Unterstützung.
  Diese Vorschläge sind deckungsgleich mit den System-Empfehlungen.

  bus_UID-E Explore Event-Bus

Typische Fehlerbilder & Gegenmittel (System-konform)

- **Import-Alias fehlt** (`@uid/presentation/...`) ⇒ Import-Map prüfen.
- **Unsubscribe vergessen** ⇒ Leaks. Immer `unsubscribe()` aufräumen.
- **Pointer-Null** nicht behandelt ⇒ bei `{idx:null}` Markierung löschen.
- **Bulk-Updates** nicht gebündelt ⇒ System über `requestAnimationFrame` bündelt; Widget bleibt reaktiv.

  bus_UID-E Explore Event-Bus

Mini-Rezepte (Copy/Paste)

Integrator von außen umschalten *(wirkt indirekt auf Grid Wave)*

```
import { bus } from '@uid/base/index.js'
bus.emit('uid:e:integrator:set', { method: 'rk4' })
```

Parameter gesammelt setzen

```
bus.emit('uid:e:params:change', { bulk: { R0: 2.2, D: 5.5, N: 1_000_000 }
})
```

Beides entspricht euren Standard-Beispielen in der Bus-Doku.

bus_UID-E Explore Event-Bus

Anhang A — Qualitäts-/Abnahmeplan

1. **Konsistenz je Frame**: $\sum n_{\ast} = g^2$; pro Band exakt $n_{\ast}$.
2. **Reihenfolge**: innen **D**, außen **V**; dazwischen **R→I→E→S**.

3. **Determinismus**: Seed + Modus + Dichte ⇒ identisches Muster nach Reload/Resize.
4. **Bus-Wiring**: `sim:data` → Render; `sim:pointer` → Frame-Switch; `model:update` → Rebuild bei Seed-Wechsel; DOM-Spiegelung sichtbar.

   bus_UID-E Explore Event-Bus

5. **Performance**: $g \in \{64,128,192\}$; Sort nur bei Layoutwechsel; framerate stabil.

Anhang B — Changelog v1.4.0

- **SEIRVD-Support**: **D (Deceased)** als eigenständiges Band; Visualreihenfolge **D→R→I→E→S→V**.
- **Mathe-Kopplungen** dokumentiert ($\gamma + \mu = 1/D_I$, $\mu = \rho/D_I$, $\beta = R_0(\gamma + \mu)$).
- **Abwärtskompatibilität**: fehlende `V/D` werden als 0 interpretiert.
- **i18n**: klarer Hinweis „didaktische Projektion, keine Karte“.
- **Theming**: `--c-d` eingeführt (Kontrast, AA/AAA je Theme).

Anhang C — Datei-Header (Quelle)

```
/*!
 * Project:  Understanding Infection Dynamics · Infektionsdynamiken
verstehen
 *           UID-Explore · Presentation Layer · Visual Tool · Grid Wave
Widget
 * File:     /visual tool/grid wave/index.js
 * Type:     Open Educational Resource (OER) · ESM
 * Authors:  B. D. Rausch · A. Heinz
 * Contact:  info@infectiondynamics.eu · info@infektionsdynamiken.de
 * License:  CC BY 4.0
 *
 * Created:  2025-10-01
 * Updated:  2025-10-01
 * Version:  1.4.0
 */
```