UID-Explore schema.js

Stand 27.09.2025

Position im System

`schema.js` vermittelt zwischen Eingabe-Welt (Slider, Textfelder, Presets) und Rechenkern. Es liefert

- den **Parameter-Katalog** mit Grenzen, Schrittweiten und Defaults,
- die **Normalisierung** für eingehende Werte (Clamping + Step-Rastern),
- die **Kopplungslogik**, damit abgeleitete Größen konsistent bleiben.

Konsumenten sind vor allem der **Director (`uid.js`)** und die **Parameter-UI**. Die Engine verlässt sich darauf, dass Parameter nach diesem Schema gültig sind.

Öffentliche API

```
export function makeCatalog(model='SIR', mode='school')
// → { [key]: { min, max, step, def } }

export function normalizeParams(p, cat)
// → { ...normalisierte Parameter gemäß cat }

export function applyCouplings(p, changedKey)
// → p (mutiert) mit konsistenter Ableitung verknüpfter Größen

// Hilfen
export function toStep(val, step)
export function clamp(v, min, max)
```

Designentscheidungen

- `normalizeParams` ist **whitelist-basiert**: Nur Keys aus dem Katalog werden übernommen. Unbekannte Eingaben haben keine Wirkung.
- Nicht-finite Eingaben (NaN, ±∞) werden auf den **Default** gesetzt, dann geclamped und auf die Schrittweite gerastert.
- `applyCouplings` **mutiert** das Eingabeobjekt `p` und gibt es zurück. Das erleichtert **chaining** im Director.

Der Parameter-Katalog (Ist-Stand)

Die Default-Spezifikation ist aktuell **modell-agnostisch** und deckt alle Modelle ab; UI-Schichten können nicht benötigte Regler ausblenden.

Key	Einheit	min	max	step	def	Bedeutung
`N`	Personen	1	1e9	1	1 000 000	Gesamtbevölkerung
`I0`	Personen	0	1e7	1	10	Anfangs-Infizierte
`T`	Tage	1	3650	1	180	Simulationsdauer
`dt`	Tage/Step	0.01	10	0.01	0.5	Zeitschritt
`R0`	—	0.1	10	0.01	3.0	Basisreproduktionszahl
`beta`	1/Tag	0	5	0.0001	0.6	Ansteckungsrate
`gamma`	1/Tag	1e-4	5	0.0001	0.2	Genesungsrate, `D = 1/γ`
`D`	Tage	0.2	365	0.01	5.0	Infektiöse Dauer
`measures`	Anteil [0..1]	0	1	0.01	0.0	Maßnahmenstärke
`sigma`	1/Tag	1e-4	5	0.0001	0.25	Übergang E→I, `L = 1/σ` (SEIR)
`mu`	1/Tag	0	1	0.0001	0.0	Mortalitätsrate (SIRD)
`nu`	1/Tag	0	1	0.0001	0.0	Impfrate (SIRV)
`L`	Tage	1	14	1	4	Latenzzeit (SEIR)

Hinweis Die Schrittweiten sind so gewählt, dass UI-Slider sauber rasten und Text-Eingaben exakt auf zulässige Gitterpunkte fallen.

Normalisierungspipeline

```
function normalizeParams(p, cat) {
  const out = {};
  for (const k of Object.keys(cat)) {
    const def = cat[k].def;
    let v = (p && Number.isFinite(p[k])) ? Number(p[k]) : def;
    v = clamp(v, cat[k].min, cat[k].max);
    v = toStep(v, cat[k].step);
    out[k] = v;
  }
  return out;
}
```

Ablauf in Worten

1. **Default** ziehen, falls kein gültiger Wert vorliegt.
2. **Clamping** auf zulässige Grenzen.
3. **Step-Rastern** auf das Gitter der UI.

Vorteile

- Robust gegen fehlerhafte Eingaben.
- Keine Nebenwirkungen auf nicht katalogisierte Felder.
- Identische Ergebnisse bei Slider-Drag und direkter Texteingabe.

Algebraische Kopplungen

Die Kopplungen stellen Konsistenz zwischen abgeleiteten Größen her. Der treibende Parameter wird über `changedKey` angegeben.

```
// gamma ↔ D
if (changedKey==='gamma') D = 1/gamma;
else if (changedKey==='D') gamma = 1/D;

// R0 ↔ beta ↔ gamma
if (changedKey==='R0') beta = R0 * gamma;
else if (changedKey==='beta') R0 = beta / gamma;
else if (changedKey==='gamma') R0 = beta / gamma;

// sigma ↔ L
if (changedKey==='sigma') L = 1/sigma;
else if (changedKey==='L') sigma = 1/L;
```

Eigenschaften

- **Einseitiger Treiber:** `changedKey` verhindert Ping-Pong. Wer ändert, „führt“ die Ableitung.
- **Reihenfolge:** Bei `changedKey = 'gamma'` werden **zwei** Beziehungen bedient: zuerst `D`, dann `R0`.
- **Klemmen:** Gekoppelte Werte werden auf **sinnvolle Grenzen** geclamped (z. B. `gamma ∈ [1e-6, 5]`, `L ∈ [1e-6, 14]`).
- **Schrittweiten:** Kopplungen runden **nicht** auf Schrittweiten; das kann nachgelagert (UI-Sync) erfolgen.

Praxisempfehlung

- Bei **Bulk-Updates** eine **Priorität** festlegen oder Kopplungen in definierter Reihenfolge anwenden, z. B. zuletzt veränderte Größe treibt.

Zusammenspiel mit Director, UI und Engine

- **UI → Director:** UI sendet Änderungsereignisse mit Schlüssel und Wert.
- **Director:** ruft `normalizeParams`, dann `applyCouplings` mit dem treibenden Schlüssel und übergibt das Ergebnis an die Engine.
- **Engine:** rechnet auf Basis der konsistenten Parameter und liefert Zeitreihen zurück.

Dadurch bleiben UI-Darstellung, numerisches Modell und Ableitungen synchron.

Invarianten und Einheiten

- Raten (`beta`, `gamma`, `sigma`, `mu`, `nu`) sind **pro Tag** definiert.
- `D` und `L` sind **Tage** und die **Gegenwerte** zu `gamma` bzw. `sigma`.
- `measures ∈ [0,1]` wirkt als **Anteil** und reduziert später `β_eff = β·(1−measures)` in der Engine.

Typische Fehlerbilder und Gegenmittel

- **Driftende UI-Werte nach Kopplung:** Da Kopplungen nicht auf Schrittweiten runden, sollten UI-Felder beim Sync das **Gitter** (`toStep`) anwenden.
- **Unbekannte Parameter „verschwinden“:** Gewollt. `normalizeParams` ist whitelist-basiert. Bei Bedarf Katalog erweitern.
- **Ping-Pong bei Bulk-Änderungen:** `changedKey` klar definieren oder Kopplungen sequentiell aufrufen.

Erweiterbarkeit

- **Modell-spezifische Kataloge:** `makeCatalog(model, mode)` kann pro Modell Felder aktivieren/deaktivieren oder Ranges anpassen.
- **Modusabhängige Schrittweiten:** Für „School“ größere Steps, für „University“ feinere Raster; UI kann denselben Katalog nutzen.
- **Neue Kopplungen:** Als reine Ableitungen implementieren, die nur auf **einen** Treiber hören. Beispiel: saisonale Modulation von `beta`.
- **Rückgabeform anreichern:** Optional `applyCouplings` so erweitern, dass eine **Liste geänderter Keys** zurückkommt.

Mini-Rezepte

Robuste Übernahme eines Textfeld-Inputs

```
const cat = makeCatalog('SEIR','university');
const p1  = normalizeParams({ R0: '2.73', D: 6.2 }, cat);
const p2  = applyCouplings(p1, 'D'); // γ wird konsistent gesetzt, R0
bleibt Treiber falls später γ geändert wird
```

Gekoppelte Änderung über Slider „Infektiöse Dauer“

```
state.params.D = 4.5;
normalizeParams(state.params, cat); // optional für UI-Raster
applyCouplings(state.params, 'D');  // setzt γ = 1/D
```

Quick-Map Dateien

- `12-2_base/schema.js` Katalog, Normalisierung, Kopplungen
- `12-2_base/uid.js` Director, ruft Normalisierung + Kopplungen auf und schiebt zur Engine
- `12-2_base/engine.js` interpretiert Parameterwerte numerisch
- `12-2_base/bus.js` transportiert die Änderungsereignisse

Damit ist klar, wie `schema.js` als stabiles Fundament für gültige, gekoppelte Parameter fungiert und wie UI, Director und Engine sich darauf verlassen.