# UID-Explore Director

Stand 27.09.2025

---

## Position im System

`uid.js` ist als **UID-Director** die Orchestrierungsschicht zwischen Eingabe, Schema, Engine und Bus. Er bündelt schnelle Änderungen, hält Konsistenz über Kopplungen und liefert reproduzierbare Ergebnisse an Chart, KPI und Vektorrad.

### Partnermodule

- **Bus** für Events
- **Schema** für Katalog, Normalisierung, Kopplungen
- **Engine** für die numerische Rechnung

---

## Öffentliche API

```
export function createUID(cfg = {})
// → Director-Instanz mit Getter und Steuerfunktionen
```

### Konfiguration

- `model` Name wie in der Engine (`'SIR'`, …). Fallback ist `document.documentElement.dataset.model` oder `SIR`.
- `mode` `'school'` oder `'university'`. Fallback über `dataset.mode`.
- `integrator` `'euler'`, `'heun'`, `'rk4'`. Fallback `'rk4'`.
- `params` Startparameter. Werden initial normalisiert.

### Rückgabe

```
{
  get model(),   // Großschreibung, z. B. 'SIR'
  get mode(),    // Klein, z. B. 'school'
  get params(),  // Kopie der aktuell gehaltenen Parameter
  setIntegrator(method), // Hot-Swap Integrator und Recalc
  recalc()       // manuell neu planen
}
```

---

## Boot-Ablauf

1. **Kontext lesen** Sprache und Datenattribute vom `<html>` werden genutzt.
2. **Katalog erzeugen** `makeCatalog(model, mode)` liefert Grenzen und Defaults.
3. **Parameter initial normalisieren** `normalizeParams(cfg.params, catalog)`.

4. **Ready-Signal** Der Director sendet `uid:e:params:ready` mit `{ state: { params, meta: { lang, mode, model, driverKey: 'init' } } }`.
5. **Initialer Status** `uid:e:engine:status` mit `{ model, method, steps }`.
6. **Erste Rechnung** über `requestAnimationFrame` geplant.

---

# Scheduler und Rechenzyklus

Der Director bündelt viele schnelle Änderungen über **rAF**.

```
let raf = 0, dirty = false;
function schedule(){ if (!raf) raf = requestAnimationFrame(recalc); dirty =
true; }
function recalc(){ raf = 0; dirty = false; /* publish → run → guard → data
*/ }
```

### Reihenfolge innerhalb `recalc`

1. **Modell-Update publizieren** `uid:e:model:update` mit dem konsolidierten Parametersatz und dem aktuellen Integrator.
2. **Engine ausführen** `run({ model, params, integrator })`.
3. **Drift-Guard** Toleranz `tol = max(1e-6·N, 1e-3)` und ggf. `uid:e:error` mit Kontext.
4. **Simulationsdaten publizieren** `uid:e:sim:data` mit `series`, `N`, `dt`, `T`.

---

# Eingänge verarbeiten

Der Director hört auf `uid:e:params:change`.

### Bulk-Änderungen

```
if (payload.bulk && typeof payload.bulk === 'object') {
  params = normalizeParams({ ...params, ...payload.bulk }, catalog);
}
```

- Robuste Übernahme vieler Werte in einem Schritt
- Normalisierung garantiert Clamping und Rasterung

### Einzel-Änderungen

```
if (typeof payload.key === 'string') {
  const { key } = payload; const v = Number(payload.value);
  if (Number.isFinite(v) && catalog[key]) {
    params[key] = clamp(v, catalog[key].min, catalog[key].max);
    applyCouplings(params, key);
  } else {
    emit('uid:e:error', { type: 'InvalidParam', context: { key, value:
payload.value } });
  }
```

```
}
```

- Schneller Pfad für UI-Slider
- Clamping schützt Grenzen
- **Kopplungen** halten abgeleitete Größen konsistent
- Hinweis für UI-Sync Rasterung bei Einzelwerten erfolgt über die UI

Nach jeder Änderung ruft der Director `schedule()`.

---

# Integrator Hot-Swap

Der Director hört auf `uid:e:integrator:set` und erlaubt einen **wechselnden Integrator ohne Remount** und ohne Reset der Parameter.

```
on('uid:e:integrator:set', (p) => {
  const m = String(p?.method || p?.mode || '').toLowerCase();
  if (INTEGRATORS.has(m) && m !== integrator) {
    integrator = m;
    emit('uid:e:engine:status', { model, method: integrator, steps:
Math.floor(params.T/params.dt) });
    schedule();
  }
});
```

- Status wird sofort publiziert
- Die nächste rAF-Runde rechnet mit identischen Parametern neu

---

# Veröffentliche Events im Überblick

- `uid:e:params:ready` signalisiert initiale Betriebsbereitschaft und liefert Meta-Kontext
- `uid:e:model:update` informiert Präsentations- und KPI-Schichten über den konsolidierten Parameterzustand
- `uid:e:engine:status` hält UI über Integrator und Step-Zahl informiert
- `uid:e:sim:data` liefert die Zeitreihen an Chart, KPI und Vektorrad
- `uid:e:error` meldet ungültige Eingaben oder Invariantenverletzungen

---

# Invarianten und Schutzmechanismen

- **Tolerierte Drift** `max(1e-6·N, 1e-3)` verhindert unnötige Fehlalarme bei großen Populationen
- **Input-Wächter** ungültige Einzel-Keys lösen `InvalidParam` aus
- **Kein Remount** Integratorwechsel ändert nur die Integrationsmethode, nicht den Zustand

---

# Performance und UX

- rAF-Bündelung reduziert Rechenlast bei schnellen Slider-Bewegungen
- `model:update` vor `sim:data` ermöglicht der UI, sofort Kontexte zu aktualisieren
- Pointer-Events (`uid:e:sim:pointer`) lösen **keine** Rechenzyklen aus und bleiben darum flüssig

---

# Erweiterbarkeit

- **Status-Payload** um zusätzliche Engine-Metriken erweitern, zum Beispiel `drift` oder `fps`
- **Interzeptoren** vor `run` einbauen, um z. B. Presets oder Szenarien zu injizieren
- **Konfigurierbare Toleranz** für Spezialfälle über `cfg.tolerance` oder Event-basierte Steuerung

---

# Typische Fehlerbilder und Gegenmittel

- **„Failed to resolve module specifier“** Import-Map für `@uid/base` und Slash-Alias prüfen
- **Keine Reaktion auf Slider** prüfen, ob Events als `payload.key` und `payload.value` ankommen oder als `bulk`
- **Zu viele Repaints** rAF-Kollisionen vermeiden, nur über `schedule()` neu berechnen
- **Drift-Fehler** bei extremen Parametern `dt` verkleinern oder Integrator `rk4` wählen, Kataloggrenzen prüfen

---

# Mini-Rezepte

### Minimalstart

```
import { createUID, bus } from '@uid/base';
createUID({ model: 'SEIR', mode: 'university', params: { I0: 5, R0: 2.5, D:
6 } });
```

### Programmatisch Parameter setzen

```
import { bus } from '@uid/base';
bus.emit('uid:e:params:change', { bulk: { R0: 2.0, D: 5.5, measures: 0.3 }
});
```

### Integrator wechseln

```
bus.emit('uid:e:integrator:set', { method: 'rk4' });
```

Damit ist klar, welche Verantwortung der Director trägt, wie seine Ereignisse strukturiert sind und wie man ihn sauber von außen steuert.