```
/**
 * PlayNAC-KERNEL drop-in — Aura Resonance Index (ARI) v2.3
 * One-page coded description for devs. Convertible to JSON or TS.
 * License: CARE Commons Attribution License v2.1 (CCAL)
 * Date: 2025-09-15
 */

// ==== Types
================================================================
export type HueCode =
  | "5R" | "7.5R" | "5YR" | "7.5YR" | "5Y" | "10Y" | "5GY" | "7.5GY"
  | "5G" | "5BG" | "7.5BG" | "5B" | "7.5B" | "5PB" | "5P" | "5RP";

export interface KBlock { // Kirlian / Bio-electric presence (0..1 score + traces)
  score: number; // 0..1 precomputed from morphology/biometrics
  coronaIntensity?: number; fractalDimension?: number; symmetry?: number; textureEntropy?:
number;
  hrv?: { sdnn?: number; rmssd?: number; lf_hf?: number };
}

export interface FBlock { // Fourier / Coherence & timing (0..1 score + traces)
  score: number; // 0..1 precomputed from spectral/phase relations
  alphaPeakHz?: number; coherenceHB?: number; plv?: number; pacThetaGamma?: number;
drift?: number;
}

export interface MBlock { // Munsell / Empowerment (numeric base + hue semantics)
  base: number; // 0..1 base before hue adjustment (e.g., f(value, chroma))
  hue: HueCode; value: number; chroma: number; // V:0..10, C:0..12+
  clarity?: number; // 0..1
  perfunctoriness?: number; // 0..1 (higher = more fleeting)
  SPI?: number; // 0..1 (Splash Persistence Index = 1 - perfunctoriness)
  scentPrimary?: string; scentSecondary?: string; scentConfidence?: number; // 0..1
}

export interface EBlock { // Environment & context (optional covariates)
  cctK?: number; lux?: number; kpIndex?: number; schumannCoh?: number;
}

export interface Qualifiers { // Resonance qualifiers used in multiplier S
  coherence?: number; stability?: number; resilience?: number; entrainment?: number;
  collectiveCoupling?: number; dissonance?: number; saturation?: number; hysteresis?: number;
  qFactor?: number; polarity?: number; // −1..+1
  selfHarm?: number; otherHarm?: number; // 0..1 ethical guards
```

```
}

export interface ARIInputV23 { K: KBlock; F: FBlock; M: MBlock; E?: EBlock; q?: Qualifiers }

export interface ARIResultV23 {
  Mprime: number; // hue-adjusted M
  S: number; // scale multiplier
  F_eff: number; // F after antiphase guard (if applied)
  ARI: number; // final index
  equilibria: { L1: boolean; L2: boolean; L3: boolean; L4: boolean; L5: boolean };
}

// ==== Constants (weights and guards) ====================================
const W = { clarity: 0.15, perfunct: 0.10, spiS: 0.04 } as const; // M' and S weights

// ==== Core functions ====================================================
export function adjustM(M: MBlock): number {
  const clarity = clamp01(M.clarity ?? 0);
  const perf = clamp01(M.perfunctoriness ?? 0);
  return clamp01(M.base * (1 + W.clarity * clarity - W.perfunct * perf));
}

export function applyAntiphaseGuard(F_score: number, q?: Qualifiers): number {
  const polarity = q?.polarity ?? 0; // −1..+1
  const dissonance = q?.dissonance ?? 0; // 0..1
  const antiphase = polarity <= -0.4 && dissonance >= 0.4;
  return antiphase ? clamp01(F_score * 0.85) : clamp01(F_score);
}

export function computeS(q?: Qualifiers, SPI?: number): number {
  const v = (k: keyof Qualifiers, d=0) => clamp01(q?.[k] as number ?? d);
  let S = 1
    + 0.10 * v("coherence")
    + 0.08 * v("stability")
    + 0.06 * v("resilience")
    + 0.06 * v("entrainment")
    + 0.05 * v("collectiveCoupling")
    + W.spiS * clamp01(SPI ?? 0)
    - 0.12 * v("dissonance")
    - 0.10 * v("saturation")
    - 0.08 * v("hysteresis");
  const selfHarm = v("selfHarm");
  const otherHarm = v("otherHarm");
  S -= 0.15 * Math.max(selfHarm, otherHarm);
```

```
  if (otherHarm >= 0.4) S = Math.min(S, 0.85); // hard cap
  return clamp01(S);
}

export function computeARI(input: ARIInputV23): ARIResultV23 {
  const { K, F, M, q } = input;
  const Mprime = adjustM(M);
  const F_eff = applyAntiphaseGuard(F.score, q);
  const S = computeS(q, M.SPI ?? (1 - clamp01(M.perfunctoriness ?? 0)));
  const core = clamp01(K.score) * clamp01(F_eff) + clamp01(Mprime);
  const ARI = clamp01(core * S);
  return { Mprime, S, F_eff, ARI, equilibria: evaluateEquilibria({ M, q }) };
}

// ==== Equilibria (LaGrange-style nodes) ===================================
export function evaluateEquilibria({ M, q }: { M: MBlock; q?: Qualifiers }) {
  const v = (k: keyof Qualifiers, d=0) => clamp01(q?.[k] as number ?? d);
  const SPI = clamp01(M.SPI ?? (1 - clamp01(M.perfunctoriness ?? 0)));
  return {
    L1: v("coherence") >= 0.65 && v("collectiveCoupling") >= 0.55 && (M.clarity ?? 0) >= 0.50 &&
Math.max(v("selfHarm"), v("otherHarm")) <= 0.20,
    L2: v("resilience") >= 0.60 && (v("saturation") <= 0.50) && (v("polarity", 0) >= 0) && SPI >=
0.40,
    L3: v("stability") >= 0.60 && (v("hysteresis") <= 0.25) && v("collectiveCoupling") >= 0.60 &&
SPI >= 0.45,
    L4: (q?.qFactor ?? 0) >= 0.55 && v("entrainment") >= 0.60 && v("dissonance") <= 0.35 &&
(M.clarity ?? 0) >= 0.55,
    L5: v("stability") >= 0.60 && v("resilience") >= 0.60 && SPI >= 0.60 && (M.clarity ?? 0) >=
0.60,
  } as const;
}

// ==== Utilities ===========================================================
export const AriV23Schema = {
  $schema: "https://json-schema.org/draft/2020-12/schema",
  $id: "https://playnac.kernel/schemas/ari.v2.3.json",
  type: "object",
  required: ["K", "F", "M"],
  properties: {
    K: { type: "object", required: ["score"], properties: { score: { type: "number", minimum: 0,
maximum: 1 } } },
    F: { type: "object", required: ["score"], properties: { score: { type: "number", minimum: 0,
maximum: 1 } } },
    M: { type: "object", required: ["base", "hue", "value", "chroma"], properties: {
```

```
      base: { type: "number", minimum: 0, maximum: 1 },
      hue: { type: "string" }, value: { type: "number" }, chroma: { type: "number" },
      clarity: { type: "number", minimum: 0, maximum: 1 },
      perfunctoriness: { type: "number", minimum: 0, maximum: 1 },
      SPI: { type: "number", minimum: 0, maximum: 1 },
      scentPrimary: { type: "string" }, scentSecondary: { type: "string" }, scentConfidence: { type:
"number", minimum: 0, maximum: 1 }
    } },
    E: { type: "object" },
    q: { type: "object", properties: {
      coherence: n01(), stability: n01(), resilience: n01(), entrainment: n01(), collectiveCoupling:
n01(),
      dissonance: n01(), saturation: n01(), hysteresis: n01(), qFactor: n01(), polarity: { type:
"number", minimum: -1, maximum: 1 },
      selfHarm: n01(), otherHarm: n01()
    } }
  }
} as const;

function n01(){ return { type: "number", minimum: 0, maximum: 1 } as const }
function clamp01(x: number){ return Math.max(0, Math.min(1, x)) }

// ==== Example
================================================================
export const exampleAriInput: ARIInputV23 = {
  K: { score: 0.72, fractalDimension: 1.46, symmetry: 0.68 },
  F: { score: 0.69, alphaPeakHz: 10.3, coherenceHB: 0.74, plv: 0.69 },
  M: { base: 0.64, hue: "5BG", value: 6, chroma: 8, clarity: 0.78, perfunctoriness: 0.32, SPI: 0.68,
scentPrimary: "Aquatic" },
  q: { coherence: 0.74, stability: 0.66, resilience: 0.62, entrainment: 0.61, collectiveCoupling:
0.59, dissonance: 0.18, saturation: 0.41, selfHarm: 0.06, otherHarm: 0.04, polarity: 0.22 }
};

// Usage:
// import { computeARI, exampleAriInput } from "./ari.v2.3";
// const result = computeARI(exampleAriInput);
// console.log(result); // { Mprime, S, F_eff, ARI, equilibria }

/**
 * PlayNAC-KERNEL drop-in — Aura Resonance Index (ARI) v2.3
 * One-page coded description for devs. Convertible to JSON or TS.
 * License: CARE Commons Attribution License v2.1 (CCAL)
 * Date: 2025-09-15
 */
```

```
// ==== Types
================================================================
export type HueCode =
  | "5R" | "7.5R" | "5YR" | "7.5YR" | "5Y" | "10Y" | "5GY" | "7.5GY"
  | "5G" | "5BG" | "7.5BG" | "5B" | "7.5B" | "5PB" | "5P" | "5RP";

export interface KBlock { // Kirlian / Bio-electric presence (0..1 score + traces)
  score: number; // 0..1 precomputed from morphology/biometrics
  coronaIntensity?: number; fractalDimension?: number; symmetry?: number; textureEntropy?:
number;
  hrv?: { sdnn?: number; rmssd?: number; lf_hf?: number };
}

export interface FBlock { // Fourier / Coherence & timing (0..1 score + traces)
  score: number; // 0..1 precomputed from spectral/phase relations
  alphaPeakHz?: number; coherenceHB?: number; plv?: number; pacThetaGamma?: number;
drift?: number;
}

export interface MBlock { // Munsell / Empowerment (numeric base + hue semantics)
  base: number; // 0..1 base before hue adjustment (e.g., f(value, chroma))
  hue: HueCode; value: number; chroma: number; // V:0..10, C:0..12+
  clarity?: number; // 0..1
  perfunctoriness?: number; // 0..1 (higher = more fleeting)
  SPI?: number; // 0..1 (Splash Persistence Index = 1 - perfunctoriness)
  scentPrimary?: string; scentSecondary?: string; scentConfidence?: number; // 0..1
}

export interface EBlock { // Environment & context (optional covariates)
  cctK?: number; lux?: number; kpIndex?: number; schumannCoh?: number;
}

export interface Qualifiers { // Resonance qualifiers used in multiplier S
  coherence?: number; stability?: number; resilience?: number; entrainment?: number;
  collectiveCoupling?: number; dissonance?: number; saturation?: number; hysteresis?: number;
  qFactor?: number; polarity?: number; // −1..+1
  selfHarm?: number; otherHarm?: number; // 0..1 ethical guards
}

export interface ARIInputV23 { K: KBlock; F: FBlock; M: MBlock; E?: EBlock; q?: Qualifiers }

export interface ARIResultV23 {
  Mprime: number; // hue-adjusted M
```

```
  S: number; // scale multiplier
  F_eff: number; // F after antiphase guard (if applied)
  ARI: number; // final index
  equilibria: { L1: boolean; L2: boolean; L3: boolean; L4: boolean; L5: boolean };
}

// ==== Constants (weights and guards) ====================================
const W = { clarity: 0.15, perfunct: 0.10, spiS: 0.04 } as const; // M' and S weights

// ==== Core functions ====================================================
export function adjustM(M: MBlock): number {
  const clarity = clamp01(M.clarity ?? 0);
  const perf = clamp01(M.perfunctoriness ?? 0);
  return clamp01(M.base * (1 + W.clarity * clarity - W.perfunct * perf));
}

export function applyAntiphaseGuard(F_score: number, q?: Qualifiers): number {
  const polarity = q?.polarity ?? 0; // −1..+1
  const dissonance = q?.dissonance ?? 0; // 0..1
  const antiphase = polarity <= -0.4 && dissonance >= 0.4;
  return antiphase ? clamp01(F_score * 0.85) : clamp01(F_score);
}

export function computeS(q?: Qualifiers, SPI?: number): number {
  const v = (k: keyof Qualifiers, d=0) => clamp01(q?.[k] as number ?? d);
  let S = 1
    + 0.10 * v("coherence")
    + 0.08 * v("stability")
    + 0.06 * v("resilience")
    + 0.06 * v("entrainment")
    + 0.05 * v("collectiveCoupling")
    + W.spiS * clamp01(SPI ?? 0)
    - 0.12 * v("dissonance")
    - 0.10 * v("saturation")
    - 0.08 * v("hysteresis");
  const selfHarm = v("selfHarm");
  const otherHarm = v("otherHarm");
  S -= 0.15 * Math.max(selfHarm, otherHarm);
  if (otherHarm >= 0.4) S = Math.min(S, 0.85); // hard cap
  return clamp01(S);
}

export function computeARI(input: ARIInputV23): ARIResultV23 {
  const { K, F, M, q } = input;
```

```
  const Mprime = adjustM(M);
  const F_eff = applyAntiphaseGuard(F.score, q);
  const S = computeS(q, M.SPI ?? (1 - clamp01(M.perfunctoriness ?? 0)));
  const core = clamp01(K.score) * clamp01(F_eff) + clamp01(Mprime);
  const ARI = clamp01(core * S);
  return { Mprime, S, F_eff, ARI, equilibria: evaluateEquilibria({ M, q }) };
}

// ==== Equilibria (LaGrange-style nodes) ==================================
export function evaluateEquilibria({ M, q }: { M: MBlock; q?: Qualifiers }) {
  const v = (k: keyof Qualifiers, d=0) => clamp01(q?.[k] as number ?? d);
  const SPI = clamp01(M.SPI ?? (1 - clamp01(M.perfunctoriness ?? 0)));
  return {
    L1: v("coherence") >= 0.65 && v("collectiveCoupling") >= 0.55 && (M.clarity ?? 0) >= 0.50 &&
Math.max(v("selfHarm"), v("otherHarm")) <= 0.20,
    L2: v("resilience") >= 0.60 && (v("saturation") <= 0.50) && (v("polarity", 0) >= 0) && SPI >=
0.40,
    L3: v("stability") >= 0.60 && (v("hysteresis") <= 0.25) && v("collectiveCoupling") >= 0.60 &&
SPI >= 0.45,
    L4: (q?.qFactor ?? 0) >= 0.55 && v("entrainment") >= 0.60 && v("dissonance") <= 0.35 &&
(M.clarity ?? 0) >= 0.55,
    L5: v("stability") >= 0.60 && v("resilience") >= 0.60 && SPI >= 0.60 && (M.clarity ?? 0) >=
0.60,
  } as const;
}

// ==== Utilities ==========================================================
export const AriV23Schema = {
  $schema: "https://json-schema.org/draft/2020-12/schema",
  $id: "https://playnac.kernel/schemas/ari.v2.3.json",
  type: "object",
  required: ["K", "F", "M"],
  properties: {
    K: { type: "object", required: ["score"], properties: { score: { type: "number", minimum: 0,
maximum: 1 } } },
    F: { type: "object", required: ["score"], properties: { score: { type: "number", minimum: 0,
maximum: 1 } } },
    M: { type: "object", required: ["base", "hue", "value", "chroma"], properties: {
      base: { type: "number", minimum: 0, maximum: 1 },
      hue: { type: "string" }, value: { type: "number" }, chroma: { type: "number" },
      clarity: { type: "number", minimum: 0, maximum: 1 },
      perfunctoriness: { type: "number", minimum: 0, maximum: 1 },
      SPI: { type: "number", minimum: 0, maximum: 1 },
```

```
    scentPrimary: { type: "string" }, scentSecondary: { type: "string" }, scentConfidence: { type:
"number", minimum: 0, maximum: 1 }
   } },
   E: { type: "object" },
   q: { type: "object", properties: {
    coherence: n01(), stability: n01(), resilience: n01(), entrainment: n01(), collectiveCoupling:
n01(),
    dissonance: n01(), saturation: n01(), hysteresis: n01(), qFactor: n01(), polarity: { type:
"number", minimum: -1, maximum: 1 },
    selfHarm: n01(), otherHarm: n01()
   } }
  }
} as const;

function n01(){ return { type: "number", minimum: 0, maximum: 1 } as const }
function clamp01(x: number){ return Math.max(0, Math.min(1, x)) }

// ==== Example
=============================================================
export const exampleAriInput: ARIInputV23 = {
  K: { score: 0.72, fractalDimension: 1.46, symmetry: 0.68 },
  F: { score: 0.69, alphaPeakHz: 10.3, coherenceHB: 0.74, plv: 0.69 },
  M: { base: 0.64, hue: "5BG", value: 6, chroma: 8, clarity: 0.78, perfunctoriness: 0.32, SPI: 0.68,
scentPrimary: "Aquatic" },
  q: { coherence: 0.74, stability: 0.66, resilience: 0.62, entrainment: 0.61, collectiveCoupling:
0.59, dissonance: 0.18, saturation: 0.41, selfHarm: 0.06, otherHarm: 0.04, polarity: 0.22 }
};

// Usage:
// import { computeARI, exampleAriInput } from "./ari.v2.3";
// const result = computeARI(exampleAriInput);
// console.log(result); // { Mprime, S, F_eff, ARI, equilibria }

/**
 * PlayNAC-KERNEL drop-in — Aura Resonance Index (ARI) v2.3
 * One-page coded description for devs. Convertible to JSON or TS.
 * License: CARE Commons Attribution License v2.1 (CCAL)
 * Date: 2025-09-15
 */

// ==== Types
=============================================================
export type HueCode =
  | "5R" | "7.5R" | "5YR" | "7.5YR" | "5Y" | "10Y" | "5GY" | "7.5GY"
```

```
  | "5G" | "5BG" | "7.5BG" | "5B" | "7.5B" | "5PB" | "5P" | "5RP";

export interface KBlock { // Kirlian / Bio-electric presence (0..1 score + traces)
  score: number; // 0..1 precomputed from morphology/biometrics
  coronaIntensity?: number; fractalDimension?: number; symmetry?: number; textureEntropy?:
number;
  hrv?: { sdnn?: number; rmssd?: number; lf_hf?: number };
}

export interface FBlock { // Fourier / Coherence & timing (0..1 score + traces)
  score: number; // 0..1 precomputed from spectral/phase relations
  alphaPeakHz?: number; coherenceHB?: number; plv?: number; pacThetaGamma?: number;
drift?: number;
}

export interface MBlock { // Munsell / Empowerment (numeric base + hue semantics)
  base: number; // 0..1 base before hue adjustment (e.g., f(value, chroma))
  hue: HueCode; value: number; chroma: number; // V:0..10, C:0..12+
  clarity?: number; // 0..1
  perfunctoriness?: number; // 0..1 (higher = more fleeting)
  SPI?: number; // 0..1 (Splash Persistence Index = 1 - perfunctoriness)
  scentPrimary?: string; scentSecondary?: string; scentConfidence?: number; // 0..1
}

export interface EBlock { // Environment & context (optional covariates)
  cctK?: number; lux?: number; kpIndex?: number; schumannCoh?: number;
}

export interface Qualifiers { // Resonance qualifiers used in multiplier S
  coherence?: number; stability?: number; resilience?: number; entrainment?: number;
  collectiveCoupling?: number; dissonance?: number; saturation?: number; hysteresis?: number;
  qFactor?: number; polarity?: number; // −1..+1
  selfHarm?: number; otherHarm?: number; // 0..1 ethical guards
}

export interface ARIInputV23 { K: KBlock; F: FBlock; M: MBlock; E?: EBlock; q?: Qualifiers }

export interface ARIResultV23 {
  Mprime: number; // hue-adjusted M
  S: number; // scale multiplier
  F_eff: number; // F after antiphase guard (if applied)
  ARI: number; // final index
  equilibria: { L1: boolean; L2: boolean; L3: boolean; L4: boolean; L5: boolean };
}
```

```
// ==== Constants (weights and guards) ====================================
const W = { clarity: 0.15, perfunct: 0.10, spiS: 0.04 } as const; // M' and S weights

// ==== Core functions ====================================================
export function adjustM(M: MBlock): number {
  const clarity = clamp01(M.clarity ?? 0);
  const perf = clamp01(M.perfunctoriness ?? 0);
  return clamp01(M.base * (1 + W.clarity * clarity - W.perfunct * perf));
}

export function applyAntiphaseGuard(F_score: number, q?: Qualifiers): number {
  const polarity = q?.polarity ?? 0; // −1..+1
  const dissonance = q?.dissonance ?? 0; // 0..1
  const antiphase = polarity <= -0.4 && dissonance >= 0.4;
  return antiphase ? clamp01(F_score * 0.85) : clamp01(F_score);
}

export function computeS(q?: Qualifiers, SPI?: number): number {
  const v = (k: keyof Qualifiers, d=0) => clamp01(q?.[k] as number ?? d);
  let S = 1
    + 0.10 * v("coherence")
    + 0.08 * v("stability")
    + 0.06 * v("resilience")
    + 0.06 * v("entrainment")
    + 0.05 * v("collectiveCoupling")
    + W.spiS * clamp01(SPI ?? 0)
    - 0.12 * v("dissonance")
    - 0.10 * v("saturation")
    - 0.08 * v("hysteresis");
  const selfHarm = v("selfHarm");
  const otherHarm = v("otherHarm");
  S -= 0.15 * Math.max(selfHarm, otherHarm);
  if (otherHarm >= 0.4) S = Math.min(S, 0.85); // hard cap
  return clamp01(S);
}

export function computeARI(input: ARIInputV23): ARIResultV23 {
  const { K, F, M, q } = input;
  const Mprime = adjustM(M);
  const F_eff = applyAntiphaseGuard(F.score, q);
  const S = computeS(q, M.SPI ?? (1 - clamp01(M.perfunctoriness ?? 0)));
  const core = clamp01(K.score) * clamp01(F_eff) + clamp01(Mprime);
  const ARI = clamp01(core * S);
```

```
  return { Mprime, S, F_eff, ARI, equilibria: evaluateEquilibria({ M, q }) };
}

// ==== Equilibria (LaGrange-style nodes) ==================================
export function evaluateEquilibria({ M, q }: { M: MBlock; q?: Qualifiers }) {
  const v = (k: keyof Qualifiers, d=0) => clamp01(q?.[k] as number ?? d);
  const SPI = clamp01(M.SPI ?? (1 - clamp01(M.perfunctoriness ?? 0)));
  return {
    L1: v("coherence") >= 0.65 && v("collectiveCoupling") >= 0.55 && (M.clarity ?? 0) >= 0.50 &&
Math.max(v("selfHarm"), v("otherHarm")) <= 0.20,
    L2: v("resilience") >= 0.60 && (v("saturation") <= 0.50) && (v("polarity", 0) >= 0) && SPI >=
0.40,
    L3: v("stability") >= 0.60 && (v("hysteresis") <= 0.25) && v("collectiveCoupling") >= 0.60 &&
SPI >= 0.45,
    L4: (q?.qFactor ?? 0) >= 0.55 && v("entrainment") >= 0.60 && v("dissonance") <= 0.35 &&
(M.clarity ?? 0) >= 0.55,
    L5: v("stability") >= 0.60 && v("resilience") >= 0.60 && SPI >= 0.60 && (M.clarity ?? 0) >=
0.60,
  } as const;
}

// ==== Utilities ==========================================================
export const AriV23Schema = {
  $schema: "https://json-schema.org/draft/2020-12/schema",
  $id: "https://playnac.kernel/schemas/ari.v2.3.json",
  type: "object",
  required: ["K", "F", "M"],
  properties: {
    K: { type: "object", required: ["score"], properties: { score: { type: "number", minimum: 0,
maximum: 1 } } },
    F: { type: "object", required: ["score"], properties: { score: { type: "number", minimum: 0,
maximum: 1 } } },
    M: { type: "object", required: ["base", "hue", "value", "chroma"], properties: {
      base: { type: "number", minimum: 0, maximum: 1 },
      hue: { type: "string" }, value: { type: "number" }, chroma: { type: "number" },
      clarity: { type: "number", minimum: 0, maximum: 1 },
      perfunctoriness: { type: "number", minimum: 0, maximum: 1 },
      SPI: { type: "number", minimum: 0, maximum: 1 },
      scentPrimary: { type: "string" }, scentSecondary: { type: "string" }, scentConfidence: { type:
"number", minimum: 0, maximum: 1 }
    } },
    E: { type: "object" },
    q: { type: "object", properties: {
```

```
    coherence: n01(), stability: n01(), resilience: n01(), entrainment: n01(), collectiveCoupling:
n01(),
    dissonance: n01(), saturation: n01(), hysteresis: n01(), qFactor: n01(), polarity: { type:
"number", minimum: -1, maximum: 1 },
    selfHarm: n01(), otherHarm: n01()
   } }
  }
} as const;

function n01(){ return { type: "number", minimum: 0, maximum: 1 } as const }
function clamp01(x: number){ return Math.max(0, Math.min(1, x)) }

// ==== Example
=============================================================
export const exampleAriInput: ARIInputV23 = {
  K: { score: 0.72, fractalDimension: 1.46, symmetry: 0.68 },
  F: { score: 0.69, alphaPeakHz: 10.3, coherenceHB: 0.74, plv: 0.69 },
  M: { base: 0.64, hue: "5BG", value: 6, chroma: 8, clarity: 0.78, perfunctoriness: 0.32, SPI: 0.68,
scentPrimary: "Aquatic" },
  q: { coherence: 0.74, stability: 0.66, resilience: 0.62, entrainment: 0.61, collectiveCoupling:
0.59, dissonance: 0.18, saturation: 0.41, selfHarm: 0.06, otherHarm: 0.04, polarity: 0.22 }
};

// Usage:
// import { computeARI, exampleAriInput } from "./ari.v2.3";
// const result = computeARI(exampleAriInput);
// console.log(result); // { Mprime, S, F_eff, ARI, equilibria }
```