MODIVX STANDARD DEFINITION
JSON SCHEMA

Protocol version	1.0
Document version:	2.0
Status:	Release Candidate
Category:	Standards Track
Date:	2026-01
Author:	© 2026 Ruben Jaybird Institute

The Seven-Layer MODIVX Specification Stack

1. MLD Mathematical Core (MC)
2. Binding Specification
3. Wire Protocol
4. Wire Output Envelope
5. **JSON Schema**
6. Implementation Profile (Runtime Rules)
7. Interop Standard (Tests and Certification)

Status of This Document (Formal Validation)

The JSON Schema represents the machine-readable validation rule used to verify whether a message complies with formal structural requirements ('formal well-formedness'). Rationale for the requirement: A schema can only validate properties that have been normatively established beforehand; it functions as a control mechanism, not as the source of semantic definitions.

The MODIVX **Base Standard** defines the diagnostic framework and semantics, while the **DCO Profile** specifies additional deterministic and canonicalization requirements for Interoperability. All requirements with the [PROFILE:DCO] label are excluded from the Base Standard.

ENVELOPE JSON SCHEMA

1. wire.schema.json

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "mld://std-0.1/wire.schema.json",
  "title": "MLD Wire Output Envelope (STD-0.1) — L1–L4",
  "type": "object",
  "additionalProperties": false,
  "required": ["wire_version", "type", "request_id", "payload", "errors"],
  "properties": {
    "wire_version": {
      "type": "string",
      "const": "MLD-WIRE-2",
      "description": "Wire protocol identifier. Must be deterministic and fixed for this schema."
[PROFILE:DCO]
    },
    "type": { "$ref": "#/$defs/WireType" },
    "request_id": {
      "type": "string",
      "minLength": 1,
      "description": "Deterministic request identifier (implementation-defined derivation)." [PROFILE:DCO]
    },
    "timestamp_utc": {
      "type": "string",
      "format": "date-time",
      "description": "Optional. If present, must be derived deterministically (avoid non-deterministic wall-clock
in conformance runs)." [PROFILE:DCO]
    },
    "payload": {
      "type": "object",
      "description": "Type-specific payload. Validated via conditional schemas below."
    },
    "errors": {
      "type": "array",
      "items": { "$ref": "#/$defs/WireError" },
      "description": "Transport-level errors. Must be present (may be empty). Must be canonical-ordered by
implementation (ordering not enforceable by JSON Schema)." [PROFILE:DCO]
    }
  },
  "allOf": [
    {
      "if": { "properties": { "type": { "const": "DIAG_RESULT" } }, "required": ["type"] },
      "then": { "properties": { "payload": { "$ref": "#/$defs/DiagResultPayload" } }, "required": ["payload"] }
    },
    {
      "if": { "properties": { "type": { "const": "ERROR" } }, "required": ["type"] },
      "then": { "not": { "required": ["payload"] }}
    },

    {
      "if": { "properties": { "type": { "const": "ADAPT_RESULT" } }, "required": ["type"] },
      "then": { "properties": { "payload": { "$ref": "#/$defs/AdaptResultPayload" } }, "required": ["payload"] }
    },
    {
      "if": { "properties": { "type": { "const": "META_RESULT" } }, "required": ["type"] },
      "then": { "properties": { "payload": { "$ref": "#/$defs/MetaResultPayload" } }, "required": ["payload"] }
    },
```

```
    {
      "if": { "properties": { "type": { "const": "CAPABILITIES_RESULT" } }, "required": ["type"] },
      "then": { "properties": { "payload": { "$ref": "#/$defs/CapabilitiesPayload" } }, "required": ["payload"] }
    },
    {
      "if": { "properties": { "type": { "enum": ["DIAG_REQUEST", "ADAPT_REQUEST", "META_REQUEST",
"CAPABILITIES_REQUEST"] } }, "required": ["type"] },
      "then": { "properties": { "payload": { "$ref": "#/$defs/GenericRequestPayload" } }, "required": ["payload"]
}
    }
  ],
  "$defs": {
    "WireType": {
      "type": "string",
      "enum": [
        "DIAG_REQUEST",
        "DIAG_RESULT",
        "ADAPT_REQUEST",
        "ADAPT_RESULT",
        "META_REQUEST",
        "META_RESULT",
        "CAPABILITIES_REQUEST",
        "CAPABILITIES_RESULT"
      ]
    },

    "WireErrorCode": {
      "type": "string",
      "minLenght: 1,
      “description”: “Error code from the MLD-Wire Protokoll Errorcode Registry (e.g. E1000, E1005, E9000,
E9001).”
    },

    "WireErrorScope": {
      "type": "string",
      "enum": ["ENGINE", "ANALYZER", "WIRE"]
    },

    "WireError": {
      "type": "object",
      "additionalProperties": false,
      "required": ["class", "code", "state", "message"],
      "properties": {
        "class": {
          "type": "string",
          "enum": ["E1", "E2", "E3", "E4", "E5", "E6"],
          "description": "Semantic error class per MLD-Wire Protokoll v2.0 §9.2."
        },
        "code": { "$ref": "#/$defs/WireErrorCode" },
        "state": {
          "type": "string",
          "enum": ["S1", "S2", "S3", "S4", "S4A"],
          "description": "Error-State per MLD-Wire Protokoll v2.0 §9.3."
        },
        "message": { "type": "string" },
        "details": {
          "type": "object",
          "additionalProperties": true
        }
      }
```

```
  },

  "EqClassRef": {
    "type": "object",
    "additionalProperties": false,
    "required": ["eqclass_key", "format"],
    "properties": {
      "eqclass_key": {
        "type": "string",
        "minLength": 1,
        "description": "CanonKey of the equivalence class (string encoding of Canon bytes)."
      },
      "format": {
        "type": "string",
        "enum": ["canon-bytes-b64", "canon-hex", "canon-utf8"],
        "description": "Encoding format for eqclass_key."
      },
      "label": {
        "type": "string",
        "description": "Optional human-readable label; must not affect semantics."
      }
    }
  },

  "ZRef": {
    "type": "object",
    "additionalProperties": false,
    "required": ["z_id"],
    "properties": {
      "z_id": {
        "type": "string",
        "minLength": 1,
        "description": "Deterministic identifier of context."
      },
      "z_payload": {
        "type": "object",
        "additionalProperties": true,
        "description": "Optional context payload; must be deterministic and serializable if present."
      }
    }
  },

  "Dim": {
    "description": "Dimension index. Implementations typically use Int, but string is allowed if binding specifies so.",
    "oneOf": [
      { "type": "integer" },
      { "type": "string", "minLength": 1 }
    ]
  },

  "DimOrder": {
    "type": "array",
    "items": { "$ref": "#/$defs/Dim" },
    "minItems": 1,
    "description": "Deterministic dimension iteration order. Must contain each dimension exactly once (uniqueness not enforceable by JSON Schema)."
  },

  "Horizon": {
```

```
    "type": "object",
    "additionalProperties": false,
    "required": ["Nmax", "CandidateBudget", "TimeBudget"],
    "properties": {
      "Nmax": { "type": "integer", "minimum": 0 },
      "CandidateBudget": { "type": "integer", "minimum": 0 },
      "TimeBudget": { "type": "integer", "minimum": 0 }
    }
  },

  "BindParams": {
    "type": "object",
    "additionalProperties": false,
    "required": [“registry_snapshot_id”, "dim_order", "horizon"],
    "properties": {
      "dim_order": { "$ref": "#/$defs/DimOrder" },
      "horizon": { "$ref": "#/$defs/Horizon" },

      "approx_policy": { "type": "string" },
      "tie_break_policy": { "type": "string" },
      "run_mode": { "type": "string" },

      "max_steps": {
        "type": "integer",
        "minimum": 0,
        "description": "Required for L3/L4 result types (ADAPT_RESULT/META_RESULT)."
      },
      "cycle_policy": {
        "type": "string",
        "description": "Required for L3 result types. Example: EQCLASS_KEY_REPEAT."
      },

      "registry_snapshot_id": { "type": "string" },
      "config_fingerprint": { "type": "string" }
    },
  },

  "ConfLabel": {
    "type": "string",
    "enum": ["LOW", "MEDIUM", "HIGH"]
  },

  "StateLabel": {
    "description": "Ordinal state. Either label strings (recommended) or integer code with disclosed
mapping.",
    "oneOf": [
      {
        "type": "string",
        "minLength": 1
      },
      {
        "type": "integer"
      }
    ]
  },

  "Evidence": {
    "type": "object",
    "additionalProperties": false,
    "required": ["items"],
```

```
      "properties": {
        "items": {
          "type": "array",
          "items": { "type": "string" },
          "description": "Evidence items. Must be canonical-ordered if order is meaningful (ordering not
enforceable by JSON Schema)."
        }
      }
    },

    "DiagnosisResult": {
      "type": "object",
      "additionalProperties": false,
      "required": ["state", "confidence", "evidence", "errors"],
      "properties": {
        "state": { "$ref": "#/$defs/StateLabel" },
        "confidence": { "$ref": "#/$defs/ConfLabel" },
        "evidence": { "$ref": "#/$defs/Evidence" },
        "errors": {
          "type": "array",
          "items": { "$ref": "#/$defs/WireErrorCode" },
          "description": "Diagnostic-local errors. Must not be encoded via state downgrade."
        }
      }
    },

    "CandidateSetDisclosure": {
      "description": "Candidate set disclosure. Mode FULL preferred; HASHED permitted if deterministic.",
      "oneOf": [
        {
          "type": "object",
          "additionalProperties": false,
          "required": ["mode", "eqclasses"],
          "properties": {
            "mode": { "const": "FULL" },
            "eqclasses": {
              "type": "array",
              "items": { "$ref": "#/$defs/EqClassRef" },
              "description": "Must be CanonKey-sorted."
            }
          }
        },
        {
          "type": "object",
          "additionalProperties": false,
          "required": ["mode", "count", "fingerprints"],
          "properties": {
            "mode": { "const": "HASHED" },
            "count": { "type": "integer", "minimum": 0 },
            "fingerprints": {
              "type": "array",
              "items": { "type": "string" },
              "description": "Deterministic fingerprints; must be stable and canonical-ordered."
            }
          }
        }
      ]
    },

    "DimState": {
```

```
    "type": "object",
    "additionalProperties": false,
    "required": ["dim", "state"],
    "properties": {
      "dim": { "$ref": "#/$defs/Dim" },
      "state": { "$ref": "#/$defs/StateLabel" }
    }
  },

  "Profile": {
    "type": "object",
    "additionalProperties": false,
    "required": ["dims"],
    "properties": {
      "dims": {
        "type": "array",
        "items": { "$ref": "#/$defs/DimState" },
        "minItems": 1,
        "description": "Must be in DimOrder."
      }
    }
  },

  "CanonContractDisclosure": {
    "type": "object",
    "additionalProperties": false,
    "required": ["canon_format"],
    "properties": {
      "canon_format": { "type": "string", “enum”: [“FULL”, “HASHED”] },
      "transform_manifest_id": { "type": "string" },
      "transform_fingerprint": { "type": "string", “minLenght”:1},
      "max_collision_rate": { "type": "number", "minimum": 0 },
      "witness_set_id": { "type": "string" }
    }
    "allOf": [
      {
        "if": {
          "properties": { "max_collision_rate": { "exclusiveMinimum": 0 } },
          "required": ["max_collision_rate"]
        },
        "then": { "required": ["witness_set_id"] }
      }
    ]
  },

  "Disclosures": {
    "type": "object",
    "additionalProperties": false,
    "required": ["horizon", "canon_contract", "config_fingerprint", “context_fingerprint”,
"bind_params_fingerprint", “transform_fingerprint”, “rank_theta_direction”],
    "properties": {
      "registry_snapshot_id": { "type": "string" },
      "config_fingerprint": { "type": "string" },
      "context_fingerprint": { "type": "string" },
      "bind_params_fingerprint": { "type": "string" },

      "horizon": { "$ref": "#/$defs/Horizon" },
      "approx_policy": { "type": "string" },
      "tie_break_policy": { "type": "string" },
      "canon_contract": { "$ref": "#/$defs/CanonContractDisclosure" }
```

```
    "rank_theta_direction": { "type": "string", "enum": ["NON_INCREASING", "NON_DECREASING"] },
  },
  "allOf": [
    {
      "anyOf": [
        { "required": ["registry_snapshot_id"] },
        { "required": ["config_fingerprint"] }
      ]
    }
  ]
},

"ThetaRef": {
  "type": "object",
  "additionalProperties": false,
  "required": ["theta_id"],
  "properties": {
    "theta_id": { "type": "string", "minLength": 1 }
  }
},

"StopReason": {
  "type": "string",
  "enum": ["FIXPOINT", "MAX_STEPS", "CYCLE_DETECTED"]
},

"SelectionTrace": {
  "type": "object",
  "additionalProperties": false,
  "required": ["stage", "pareto_max_count", "editcost_min", "canonkey_tiebreak_used"],
  "properties": {
    "stage": {
      "type": "string",
      "enum": ["PARETO", "EDITCOST", "CANONKEY"]
    },
    "pareto_max_count": { "type": "integer", "minimum": 0 },
    "editcost_min": {
      "description": "Minimum edit cost; use large sentinel for +INF if needed (implementation disclosed).",
      "oneOf": [
        { "type": "integer", "minimum": 0 },
        { "type": "string", "const": "+INF" }
      ]
    },
    "canonkey_tiebreak_used": { "type": "boolean" }
  }
},

"StepLog": {
  "type": "object",
  "additionalProperties": false,
  "required": ["n", "phi_n", "i_n", "phi_n1", "theta", "horizon", "admissible_count", "selection_trace"],
  "properties": {
    "n": { "type": "integer", "minimum": 0 },
    "phi_n": { "$ref": "#/$defs/EqClassRef" },
    "i_n": { "$ref": "#/$defs/Dim" },
    "phi_n1": { "$ref": "#/$defs/EqClassRef" },
    "theta": { "$ref": "#/$defs/ThetaRef" },
    "horizon": { "$ref": "#/$defs/Horizon" },
    "admissible_count": { "type": "integer", "minimum": 0 },
    "selection_trace": { "$ref": "#/$defs/SelectionTrace" }
```

```
    }
  },

  "SelectionDisclosure": {
    "type": "object",
    "additionalProperties": false,
    "required": ["neutral_selection_rule", "editcost_definition", "canonkey_only_final_tiebreak"],
    "properties": {
      "neutral_selection_rule": { "type": "boolean" },
      "editcost_definition": { "type": "string", "minLength": 1 },
      "canonkey_only_final_tiebreak": { "type": "boolean" }
    }
  },

  "DimDiagnosisBlock": {
    "type": "object",
    "additionalProperties": false,
    "required": ["dim", "delta", "candidates"],
    "properties": {
      "dim": { "$ref": "#/$defs/Dim" },
      "delta": { "$ref": "#/$defs/DiagnosisResult" },
      "candidates": { "$ref": "#/$defs/CandidateSetDisclosure" }
    }
  },

  "DiagResultPayload": {
    "type": "object",
    "additionalProperties": false,
    "required": ["input", "bind_params", "dimension_results", "global_profile", "reactive_global",
"stable_global", "disclosures"],
    "properties": {
      "input": {
        "type": "object",
        "additionalProperties": false,
        "required": ["phi", "z"],
        "properties": {
          "phi": { "$ref": "#/$defs/EqClassRef" },
          "z": { "$ref": "#/$defs/ZRef" }
        }
      },
      "bind_params": { "$ref": "#/$defs/BindParams" },
      "dimension_results": {
        "type": "array",
        "items": { "$ref": "#/$defs/DimDiagnosisBlock" },
        "minItems": 1,
        "description": "Must follow DimOrder."
      },
      "global_profile": { "$ref": "#/$defs/Profile" },
      "reactive_global": { "type": "boolean" },
      "stable_global": { "type": "boolean" },
      "disclosures": { "$ref": "#/$defs/Disclosures" }
    }
  },

  "AdaptResultPayload": {
    "type": "object",
    "additionalProperties": false,
    "required": [
      "input",
      "bind_params",
```

```
    "trajectory",
    "step_logs",
    "stop_reason",
    "final_profile",
    "stable_global_final",
    "reactive_global_final",
    "selection_disclosure",
    "disclosures"
   ],
   "properties": {
    "input": {
     "type": "object",
     "additionalProperties": false,
     "required": ["phi0", "z", "theta"],
     "properties": {
      "phi0": { "$ref": "#/$defs/EqClassRef" },
      "z": { "$ref": "#/$defs/ZRef" },
      "theta": { "$ref": "#/$defs/ThetaRef" }
     }
    },
    "bind_params": {
     "$ref": "#/$defs/BindParams",
     "allOf": [
      { "required": ["max_steps", "cycle_policy"] }
     ]
    },
    "trajectory": {
     "type": "array",
     "items": { "$ref": "#/$defs/EqClassRef" },
     "minItems": 1,
     "description": "Step order trajectory (phi0..phiT)."
    },
    "step_logs": {
     "type": "array",
     "items": { "$ref": "#/$defs/StepLog" },
     "description": "Must be ordered by increasing n."
    },
    "stop_reason": { "$ref": "#/$defs/StopReason" },
    "final_profile": { "$ref": "#/$defs/Profile" },
    "stable_global_final": { "type": "boolean" },
    "reactive_global_final": { "type": "boolean" },
    "selection_disclosure": { "$ref": "#/$defs/SelectionDisclosure" },
    "disclosures": { "$ref": "#/$defs/Disclosures" }
   }
  },

  "ThetaManifest": {
   "type": "object",
   "additionalProperties": false,
   "required": ["thetas", "rank_policy"],
   "properties": {
    "thetas": {
     "type": "array",
     "items": { "$ref": "#/$defs/ThetaRef" },
     "minItems": 1,
     "description": "Deterministic ordering of implemented thetas."
    },
    "rank_policy": {
     "type": "string",
     "minLength": 1,
```

```
        "description": "Disclosed description of RankTheta and monotonicity direction."
      }
    }
  },

  "AdaptResultSummary": {
    "type": "object",
    "additionalProperties": false,
    "required": ["theta", "stop_reason", "steps", "fixpoint_reached", "cycle_detected"],
    "properties": {
      "theta": { "$ref": "#/$defs/ThetaRef" },
      "stop_reason": { "$ref": "#/$defs/StopReason" },
      "steps": { "type": "integer", "minimum": 0 },
      "fixpoint_reached": { "type": "boolean" },
      "cycle_detected": { "type": "boolean" }
    }
  },

  "MetaObs": {
    "type": "object",
    "additionalProperties": false,
    "required": ["obs_id", "data"],
    "properties": {
      "obs_id": {
       "type": "string",
       "minLength": 1,
       “description”: “Stable observation code from an Engine-declared observation set. MUST NOT be a
per-run UUID. [PROFILE:DCO]
      },

      "data": {
        "type": "object",
        "additionalProperties": true,
        "description": "Deterministic observation payload."
      }
    }
  },

  "MetaLog": {
    "type": "object",
    "additionalProperties": false,
    "required": ["t", "theta_in", "obs", "rank_in", "theta_out", "rank_out", "switch_accepted"],
    "properties": {
      "t": { "type": "integer", "minimum": 0 },
      "theta_in": { "$ref": "#/$defs/ThetaRef" },
      "obs": { "$ref": "#/$defs/MetaObs" },
      "rank_in": { "type": "integer" },
      "theta_out": { "$ref": "#/$defs/ThetaRef" },
      "rank_out": { "type": "integer" },
      "switch_accepted": { "type": "boolean" }
    }
  },

  "MetaStopReason": {
    "type": "string",
    "enum": ["LOCKED", "FIXPOINT_REACHED", "THETA_EXHAUSTED"]
  },

  "MetaResultPayload": {
    "type": "object",
```

```
  "additionalProperties": false,
  "required": [
    "input",
    "bind_params",
    "theta_manifest",
    "runs",
    "final_theta",
    "meta_observations",
    "meta_logs",
    "meta_stop",
    "disclosures"
  ],
  "properties": {
    "input": {
      "type": "object",
      "additionalProperties": false,
      "required": ["phi0", "z", "theta0"],
      "properties": {
        "phi0": { "$ref": "#/$defs/EqClassRef" },
        "z": { "$ref": "#/$defs/ZRef" },
        "theta0": { "$ref": "#/$defs/ThetaRef" }
      }
    },
    "bind_params": {
      "$ref": "#/$defs/BindParams",
      "allOf": [
        { "required": ["max_steps", "cycle_policy"] }
      ]
    },
    "theta_manifest": { "$ref": "#/$defs/ThetaManifest" },
    "runs": {
      "type": "array",
      "items": { "$ref": "#/$defs/AdaptResultSummary" },
      "description": "Deterministic order; typically follows theta_manifest.thetas order."
    },
    "final_theta": { "$ref": "#/$defs/ThetaRef" },
    "meta_observations": {
      "type": "array",
      "items": { "$ref": "#/$defs/MetaObs" },
      "description": "Deterministic sequence of observations."
    },
    "meta_logs": {
      "type": "array",
      "items": { "$ref": "#/$defs/MetaLog" },
      "description": "Deterministic meta-iteration logs."
    },
    "meta_stop": { "$ref": "#/$defs/MetaStopReason" },
    "disclosures": { "$ref": "#/$defs/Disclosures" }
  }
},

"CapabilitiesPayload": {
  "type": "object",
  "additionalProperties": false,
  "required": ["wire_version", "supported_levels", "state_encoding", "error_codes", "disclosures"],
  "properties": {
    "wire_version": { "type": "string" },
    "supported_levels": {
      "type": "array",
      "items": { "type": "string", "enum": ["L1", "L2", "L3", "L4"] },
```

```
        "minItems": 1,
        "description": "Must include L1–L4 for a fully compliant system claim."
      },
      "theta_manifest": { "$ref": "#/$defs/ThetaManifest" },
      "state_encoding": {
        "type": "object",
        "additionalProperties": true,
        "description": "Disclosed mapping if StateLabel uses integer codes."
      },
      "error_codes": {
        "type": "array",
        "items": { "$ref": "#/$defs/WireErrorCode" }
      },
      "disclosures": { "$ref": "#/$defs/Disclosures" }
    }
  },

  "GenericRequestPayload": {
    "type": "object",
    "additionalProperties": true,
    "description": "Requests are not the focus of STD-0.1 conformance; included as a generic
placeholder."
  }
 }
}
```

2. Purpose and Scope [PROFILE:DCO]

The goal is a uniform, deterministic, interoperable output format for:

- Diagnosis (L2)
- Adaptation / Trajectory (L3)
- Meta-Adaptation (L4)
- Capability disclosure (L1–L4 support)

Scope: Every wire message is a WireEnvelope with a type-dependent payload.

3. Normative Core Principles [PROFILE:DCO]

Determinism and Conformance (MUST)

For identical inputs (EqClass, context z, identical registry snapshot / fingerprint, same Horizon/BindParams), output MUST be deterministic.

Non-deterministic sources (wall-clock, etc.) are to be avoided in conformance runs. In particular, wire-visible identifiers (e.g., obs_id) MUST NOT be derived from wall-clock time or random sources. They MUST originate from normative, deterministic sources (e.g., registry / observation set).

Conformance MUST be demonstrated by validating every wire message against the normative JSON Schema (including all required and conditional (if/then) rules).

Implementations MAY accept alias identifiers on the transport level, but MUST internally map them to the canonical WireType values of the schema; canonical conformance checking is performed against the types defined in the schema.

Canonical Ordering (MUST)

All sequences are deterministically ordered:

- Dimensions: DimOrder
- EqClass lists: sorted by CanonKey
- WireEnvelope.errors: since per run exactly one run-level failure is transported, len(errors) ∈ {0,1}; therefore, an ordering rule is not applicable.

JSON Schema does not enforce ordering; the normative requirement is checked via tests / normalizer.

Error Handling (MUST)

Errors are not encoded via “worse states”, but exclusively via:

- WireEnvelope.errors (transport/system)
- DiagnosisResult.errors (diagnosis-local)
- confidence / evidence

For run-level failures in the sense of the Wire Protocol (FAILED/S6), WireEnvelope.errors MUST contain exactly one entry (len(errors)=1); for non-failing results, WireEnvelope.errors MUST be empty (len(errors)=0).

Disclosure (MUST)

Every response must disclose the execution frame:

- bind_params (DimOrder, Horizon, and if applicable max_steps/cycle_policy)
- disclosures (registry_snapshot_id; config_fingerprint optional)
- context_fingerprint
- bind_params_fingerprint

4. Top-Level Message: WireEnvelope (L1)

Every wire message has the form:

- wire_version (MUST): "MLD-WIRE-2"
- type (MUST): message type
- request_id (MUST): deterministic [PROFILE:DCO]
- timestamp_utc (SHOULD): deterministic only [PROFILE:DCO]
- payload (MUST): type-dependent
- errors (MUST): list of WireError (empty list allowed)

Message types

Permitted type values:

- DIAG_RESULT (L2)
- ADAPT_RESULT (L3)
- META_RESULT (L4)
- CAPABILITIES_RESULT (L1–L4 support indicator)

(Optional request types are included, but are not the core of conformance.)

5. Shared Types

EqClassRef [PROFILE:DCO]

Represents an equivalence class as CanonKey:

- eqclass_key (MUST): CanonKey string representation
- format (MUST): "canon-bytes-b64" | "canon-hex" | "canon-utf8"
- label (OPTIONAL): purely informative

ZRef (Kontext) [PROFILE:DCO]

- z_id (MUST): deterministic context identifier
- z_payload (OPTIONAL): serializable and deterministic

Dim, DimOrder

- Dim: integer or string (binding-stable)
- DimOrder: list of Dim (each Dim exactly once; test obligation)

Horizon

- Nmax (MUST)
- CandidateBudget (MUST)
- TimeBudget (MUST, 0 allowed)

BindParams (Disclosure requirement)

- dim_order (MUST)
- horizon (MUST)
- registry_snapshot_id (MUST); config_fingerprint (OPTIONAL)
- optional: approx_policy, tie_break_policy, run_mode
- additionally for L3/L4 Outputs: max_steps and cycle_policy

WireError

- code (MUST): normalized ErrorCode strings
- message (MUST)
- scope (MUST): ENGINE|ANALYZER|WIRE
- details (OPTIONAL): deterministic, serializable [PROFILE:DCO]
- Permitted code: INVALID_INPUT, UNSUPPORTED, TIMEOUT, EMPTY_SET, NOT_CANONICAL, NON_TERMINATING_GENERATOR, INTERNAL_INCONSISTENCY, INTERNAL_ERROR

6. Diagnose Ergebnis: DIAG_RESULT (L2)

Payload

DiagResultPayload contains:

- input: { phi: EqClassRef, z: ZRef } (MUST)
- bind_params: BindParams (MUST)
- dimension_results: list of DimDiagnosisBlock in DimOrder (MUST) [PROFILE:DCO]
- global_profile: Profile (MUST)
- stable_global: Bool (MUST)
- reactive_global: Bool (MUST)
- disclosures: Disclosures (MUST)

DimDiagnosisBlock

- dim (MUST)

- delta (MUST): DiagnosisResult
- candidates (MUST): CandidateSetDisclosure

DiagnosisResult

- state (MUST): ordinal (String oder Int-Code)
- confidence (MUST): LOW|MEDIUM|HIGH
- evidence (MUST): { items: [String] }
- errors (MUST): [ErrorCode]

CandidateSetDisclosure

Two modes:

- FULL: eqclasses: [EqClassRef] (CanonKey-sorted)
- HASHED: count + fingerprints[] deterministically sorted [PROFILE:DCO]

Profile

- dims: list {dim, state} in DimOrder

7. Adaptation Ergebnis: ADAPT_RESULT (L3)

Payload

AdaptResultPayload contains:

- input: {phi0, z, theta} (MUST)
- bind_params: BindParams (MUST; additionally max_steps, cycle_policy)
- trajectory: [EqClassRef] (MUST; in step order)
- step_logs: [StepLog] (MUST; increasing by n)
- stop_reason: FIXPOINT|MAX_STEPS|CYCLE_DETECTED (MUST)
- final_profile: Profile (MUST)
- stable_global_final, reactive_global_final: Bool (MUST)
- selection_disclosure: SelectionDisclosure (MUST)
- disclosures: Disclosures (MUST)

StepLog (minimal normativ)

- n (MUST)
- phi_n, i_n, phi_n1 (MUST)
- theta (MUST)
- horizon (MUST)
- admissible_count (MUST)
- selection_trace (MUST)

SelectionTrace

- stage: PARETO|EDITCOST|CANONKEY
- pareto_max_count
- editcost_min: Int or "+INF"
- canonkey_tiebreak_used: Bool

SelectionDisclosure

- neutral_selection_rule (MUST)
- editcost_definition (MUST)
- canonkey_only_final_tiebreak (MUST)

8. Meta-Adaptation Result: META_RESULT (L4)

Payload

MetaResultPayload contains:

- input: {phi0, z, theta0} (MUST)
- bind_params: BindParams (MUST; additionally max_steps, cycle_policy)
- theta_manifest: ThetaManifest (MUST)
- runs: [AdaptResultSummary] (MUST)
- final_theta (MUST)
- meta_observations: [MetaObs] (MUST) [PROFILE:DCO]
- meta_logs: [MetaLog] (MUST) [PROFILE:DCO]
- meta_stop: LOCKED|FIXPOINT_REACHED|THETA_EXHAUSTED (MUST) [PROFILE:DCO]
- disclosures: Disclosures (MUST)

ThetaManifest

- thetas: list of thetas in deterministic order [PROFILE:DCO]
- rank_policy: description of rank direction/rule

AdaptResultSummary

1. theta
2. stop_reason
3. steps
4. fixpoint_reached
5. cycle_detected

MetaObs

- obs_id: string (stable observation code from an engine-declared observation set; corresponds to Binding-Spec MetaObservation.obs_code; must not be a per-run UUID)
- data: object (deterministic) [PROFILE:DCO]

MetaLog

1. t
2. theta_in, obs
3. rank_in
4. theta_out
5. rank_out
6. switch_accepted

Rank monotonicity is a test obligation (not fully enforceable via schema).

9. Capabilities: CAPABILITIES_RESULT (Support Indicator, L1–L4)

CapabilitiesPayload contains:

- wire_version
- supported_levels: list from {L1,L2,L3,L4}
- theta_manifest (optional)
- state_encoding: mapping if states are int-encoded
- error_codes: supported ErrorCodes
- disclosures

Full standard claim: supported_levels contains all L1–L4.

10. Disclosures [PROFILE:DCO]

Disclosures contains:

- registry_snapshot_id (MUST); config_fingerprint (OPTIONAL)
- horizon
- approx_policy (optional)
- tie_break_policy (optional)
- canon_contract: CanonContractDisclosure (MUST)
- context_fingerprint
- bind_params_fingerprint
- rank_theta_direction ∈ {NON_INCREASING, NON_DECREASING}

CanonContractDisclosure contains at minimum:

- canon_format ∈ (FULL, HASHED)

Optional:

- transform_manifest_id
- max_collision_rate
- witness_set_id