MODIVX STANDARD DEFINITION
WIRE PROTOCOL

Protocol version	1.0
Document version:	2.4
Status:	Released
Category:	Standards Track
Date:	2026-05
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
7. Interop Standard

Status of This Document (Message Meaning and Errors)

The Wire Protocol defines the interaction and communication rules between the participating components, including the unambiguous interpretation of message types, status updates, and error signals. Rationale for the requirement: Even when meanings are clarified and mandatory information is provided, interoperability is only guaranteed if every message is unequivocally classified and understood as a result, a notification, or an error.

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.

1. Introduction

This document specifies a normative application protocol in the MODIVX Standard stack. It defines roles, message semantics, state transitions, error classes, comparability criteria, and conformance requirements for interoperable, deterministic, and reproducible diagnostic execution.

The protocol standardizes the interaction between a requesting instance (Client) and a diagnosing instance (Analyzer) in order to produce a diagnostic finding (Diagnostic Finding) that is comparable and reproducible under an explicit diagnostic context and a specified set of permissible transformations.

The protocol serves to formalize Metalogical Diastagraphy (MLD) such that structural diagnoses of formal systems, under explicitly specified conditions, deterministically complete, are unambiguously classified, and are reliably comparable across different implementations.

2. Requirements and Conventions

The key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL are to be interpreted as normative.

- MUST / MUST NOT: mandatory requirements
- SHALL / SHALL NOT: normative stipulation
- SHOULD / SHOULD NOT: recommended practice
- MAY / OPTIONAL: permissible extension without conformance impact

3. Terminology

3.1 Roles

- Client: Initiates diagnoses by sending a DIAGNOSE_REQUEST and receives results.
- Analyzer: Executes diagnoses, processes requests, and produces DIAGNOSTIC_FINDING, META_DIAGNOSTIC_FINDING, or ERROR.

3.2 Objects

1. Structural Carrier
   Representation of the system to be diagnosed, considered up to structural equivalence.

2. Equivalence Scheme (equiv_scheme_id)
   Defines structural equivalence relations on Structural Carriers.
3. Context
   Fixed specification of the diagnostic purpose and the permissible structural variation.
4. Transformation (transform_id)
   Permissible local structural variation of a carrier.
5. Dimension Set (dimension_set_id)
   Referenced set of diagnostic dimensions.
6. Progress Scheme (progress_scheme_id)
   Identifier of a well-founded progress relation on carrier equivalence classes.
7. Determination Profile (determination_profile_id)
   Identifier of the dimension-specific determination logic.
8. Adaptation Scheme (adaptation_scheme_id)
   Structure for adapting diagnostic schemata (permissible families of update policies and update operators).
9. Diagnostic Finding
   Protocol-conformant outcome of a diagnostic execution.
10. Meta Diagnostic Finding
    Outcome of a structural diagnosis of a diagnostic schema.
11. Update-Policy
    Selection rule for choosing update operators during adaptive executions.
12. Update-Operator
    Concrete operator for structural modification during adaptation.

3.3 Fields

- Context Fingerprint
  Canonical hash of all context-relevant parameters.
- Carrier Equivalence ID
  Pair (equiv_scheme_id, class_representative_id) for referencing an equivalence class.

4. Architecture Overview

MODIVX-WP consists of:

- Request/response interaction between Client and Analyzer.
- A normative, finite state machine on the Analyzer side.
- Normative rules for exploration, evaluation, and comparability.
- Formal message syntax (ABNF).

5. Transport

This document does not define a transport protocol.

MODIVX-WP messages MUST be transmitted as complete and unmodified messages. Wire encoding MUST be UTF-8. Line endings MUST be terminated with CRLF. Message boundaries MUST be adhered to exactly as specified by ABNF.

6. Message Model

The following message types exist:

- DIAGNOSE_REQUEST
- DIAGNOSTIC_FINDING
- META_DIAGNOSTIC_FINDING
- ERROR

The Analyzer MUST terminate a run with either DIAGNOSTIC_FINDING (alias DIAG_RESULT) or ERROR.

Therefore MAY perform zero or more ADAPT_REQUEST/ADAPT_RESULT rounds.

In addition, a META_DIAGNOSTIC_FINDING (alias META_RESULT) MAY be delivered.

7. State Machine

7.1 Purpose of the State Machine

The state machine defines:

- when a structural diagnosis begins,
- under what conditions it continues,
- when it validly completes,
- when it formally fails.

All conformant implementations MUST support these states and transitions.

7.2 State Set

The protocol defines the following finite set of states:

State	Label	Meaning
S0	IDLE	No active diagnosis
S1	REQUESTED	Diagnostic request received
S2	INITIALIZED	Carrier and context checked
S3	EXPLORING	Structural exploration in progress
S4	CLASSIFYING	Classification is computed
S4A	ADAPTING	Application of an update operator
S5	COMPLETED	Diagnosis successfully completed
S6	FAILED	Diagnosis aborted

7.3 Permitted Transitions

The following transitions are the only permitted ones:

From → To	Condition
S0 → S1	Receipt of a valid DIAGNOSE_REQUEST
S1 → S2	Carrier and context syntactically valid
S2 → S3	At least one permissible transformation exists
S3 → S3	Exploration not yet completed
S3 → S4	Exploration completed
S4 → S5	Classification successful
S1 → S6	Request invalid
S2 → S6	Carrier or context invalid
S3 → S6	Exploration not feasible
S4 → S6	Classification not well-defined
S4 → S4A	After successful classification, if adaptive execution is active
S4A → S5	After successful update execution and fixpoint check
Any state → S6	On error

7.4 Abort Conditions

A transition to FAILED (S6) MUST occur if:

- no permissible structural variation exists,
- exploration cannot terminate,
- classification is not well-defined,
- internal consistency is violated.

A FAILED state MAY NOT transition to COMPLETED.

See Section 20.1 for normative ERROR generation upon FAILED.

7.5 Completion Condition

A transition to COMPLETED (S5) is only permitted if:

- exploration is completed,
- all relevant dimensions are classified,
- a consistent DIAGNOSTIC_FINDING has been produced.

COMPLETED is terminal and also covers the adaptive stage if S4A was traversed.

META_DIAGNOSTIC_FINDING is not part of the operational state machine and MAY only be generated after reaching a terminal state (COMPLETED or FAILED).

7.6 Normative Role Binding to States

Client (diagnosis requester)

- SHOULD only initiate states S0 → S1
- SHOULD NOT influence the flow
- MAY abort a diagnosis, which leads to S6

Analyzer (diagnosis executor)

- MUST correctly implement all state transitions
- MUST deliver the normative result format

7.7 Normative Determinism Rule [PROFILE:DCO]

Two implementations are considered protocol-conformant if:

- for an identical request,
- under an identical context,
- they produce an equivalent Diagnostic Finding,

where equivalence is defined at the ordinal classification level, not at the level of internal exploration paths.

Adaptive execution (S4A) is considered deterministic under this rule if, for an identical request, context, and update policy, it always reaches the same ordinal fixpoint.

8. Message Semantics

This section defines the only permissible message objects of MODIVX-WP. All implementations MUST adhere to this semantics exactly.

The following message types exist:

1. DIAGNOSE_REQUEST

2. DIAGNOSTIC_FINDING
3. META_DIAGNOSTIC_FINDING
4. ERROR

For serialization in the WireEnvelope, the following aliases apply:

- DIAGNOSE_REQUEST ≙ DIAG_REQUEST
- DIAGNOSTIC_FINDING ≙ DIAG_RESULT
- META_DIAGNOSTIC_FINDING ≙ META_RESULT

These aliases are purely representational and do not change the semantics.

8.1 General Conventions

Messages are content-based, not transport-dependent.

- (No JSON/XML/HTTP prescribed.)
- All terms are semantically standardized, not syntactically.
- Required fields are MUST, optional fields are MAY.

8.2 DIAGNOSE_REQUEST und ADAPT_REQUEST

8.2.1 Purpose

A DIAGNOSE_REQUEST initiates a structural diagnosis according to MODIVX-WP. It specifies:

- what is being diagnosed,
- under what structural conditions,
- with what formal constraints.

8.2.2 Normative Structure

A DIAGNOSE_REQUEST MUST contain the following components:

(1) Structural Carrier

Representative of a formal system, consolidated up to structural equivalence.

- MUST be uniquely referencable
- MUST not contain implementation details
- SHALL be abstract or concrete

(2) Structural Context

Description of the permissible diagnostic context.

- defines permissible local transformations
- defines exclusions (e.g., no extension of the state space)

- MUST be fixed prior to exploration

(3) Intervention Budget

Finite bound for structural exploration.

- MUST be finite
- MAY be specified as depth, count, or class restriction
- SHALL be interpreted internally by the Analyzer

(4) Diagnostic Scope

Set of structural dimensions to be considered.

- MUST be explicitly specified
- MUST be fixed prior to classification

(5) Update-Policy

- SHALL reference a selection rule for choosing update operators.
- OPTIONAL: it may degenerate to exactly one fixed update operator.

8.2.3 Validity Conditions

A DIAGNOSE_REQUEST is invalid if:

- no permissible carrier is specified,
- the context is empty or contradictory,
- no structural variation is permissible.

Invalid requests MUST lead to state FAILED (S6).

8.2.4 ADAPT_REQUEST

An ADAPT_REQUEST initiates an adaptation round within an ongoing diagnosis process. A protocol-conformant Analyzer MUST support ADAPT_REQUEST if the diagnosis requires iterative adaptation.

8.3 DIAGNOSTIC_FINDING und ADAPT_RESULT

8.3.1 Purpose

A DIAGNOSTIC_FINDING is the only permissible outcome of a successful diagnosis.

It contains structural classifications.

8.3.2 Normative Structure

A DIAGNOSTIC_FINDING MUST contain:

(1) Carrier Reference

- reference to the diagnosed carrier
- MUST be identical to the request

(2) Dimensionwise Classification

Ordinal classification per dimension. For each considered dimension:

- stable
- reactive
- unstable

The short terms are informative. The ordering is ordinal, not metric. For each dimension, at minimum the fields per Section 17.2 MUST be provided:

- ordinal_state
- reactivity_class
- sensitivity_class
- stability_class

(3) Validity Marker

- MUST indicate whether all classifications are valid
- MUST indicate exploration completion

(4) Diagnostic Metadata

- exploration completed / aborted
- indications of edge cases

(5) Update-Operator

- SHALL indicate the concrete operator used

(6) Adaptive-Fixpoint

- MUST provide true/false as the outcome of the fixpoint criterion

8.3.3 Prohibitions

A DIAGNOSTIC_FINDING:

- MUST NOT contain scores
- MUST NOT contain probabilities
- MUST NOT contain optimization recommendations
- MUST NOT contain temporal predictions

8.4.4 ADAPT_RESULT

An ADAPT_RESULT is the outcome of an adaptation round.

An ADAPT_RESULT MUST NOT be interpreted as the terminal overall result; only DIAGNOSTIC_FINDING (i.e., DIAG_RESULT) or ERROR are terminal.

8.4 META_DIAGNOSTIC_FINDING

8.4.1 Purpose

META_DIAGNOSTIC_FINDING is a standardized result object for the structural diagnosis of a diagnostic schema. It describes the state of the diagnostic architecture itself, not the state of an individual diagnosed carrier.

8.4.2 Content

A META_DIAGNOSTIC_FINDING unambiguously states the adaptation scheme used, the associated meta progress order, and the current meta stability status. All statements are context-bound and uniquely identifiable.

8.4.3 Validity

A META_DIAGNOSTIC_FINDING is considered complete once the meta stability status is determined. The result is comparable if the adaptation scheme and the meta progress order match.

8.5 Semantic Consistency Rules

Two DIAGNOSTIC_FINDINGS are considered equivalent if:

- they concern the same dimensions,
- each dimension is classified identically at the ordinal level,
- the validity markers match.

Internal explorations MUST NOT be identical.

9. Error Conditions and Error Codes

This section defines all permissible error states of MODIVX-WP. Each conformant implementation MUST distinguish these error classes and report them correctly.

An error is not a diagnostic result, but a protocol event.

9.1 Basic Principle

- Each error MUST be uniquely classified.
- Each error MUST deterministically lead to a transition into FAILED (S6). [PROFILE:DCO]
- Errors MUST NOT be partially compensated or obscured.

- A FAILED state MAY NOT produce a finding. There are no exceptions. Partial findings are only permissible upon successful completion (COMPLETED).

9.2 Error Classes

E1 — INVALID_REQUEST

Meaning:

The DIAGNOSE_REQUEST is structurally invalid.

Causes (non-exhaustive):

- missing carrier
- contradictory context
- empty or infinite intervention budget
  Mandatory reaction:
- immediate transition to FAILED (S6)
- no exploration
- no finding

E2 — INVALID_CARRIER

Meaning:

The specified Structural Carrier is not diagnosable.

Causes:

- cannot be consolidated up to structural equivalence
- violates formal prerequisites of the protocol
  Mandatory reaction:
- transition to FAILED
- error code MUST be returned

E3 — INVALID_CONTEXT

Meaning:

The diagnostic context is inconsistent or not applicable.

Causes:

- no permissible local transformations
- context contradicts the carrier structure
  Mandatory reaction:
- transition to FAILED

- no exploration

E4 — EXPLORATION_FAILURE

Meaning:

The structural exploration could not be performed or completed.

Causes:

- exploration does not terminate
- no reachable variants
- budget exhausted without valid exploration
  Mandatory reaction:
- transition to FAILED
- exploration MUST be aborted

E5 — CLASSIFICATION_UNDEFINED

Meaning:

The classification is not well-defined.

Causes:

- contradictory exploration results
- violation of ordinal comparability
- lack of decidability of a dimension
  Mandatory reaction:
- transition to FAILED
- no partial finding is permissible

E6 — INTERNAL_INCONSISTENCY

Meaning:

Internal consistency conditions of the protocol were violated.

Causes:

- contradictory state transitions
- violation of normative invariants
  Mandatory reaction:
- transition to FAILED
- error MUST be explicitly marked

9.3 Error Report Format

Each error MUST contain at least:

- semantic error class (E1–E6)
- specific error code (registry)
- error state (S1–S4A)
- short description (human-readable)

The error report MUST be encoded in WireEnvelope.errors, with field mapping:

class = error class (E1–E6), code = error code (registry), state = error state (S1–S4A), message = short description. In this case, WireEnvelope.errors MUST contain exactly one entry (len(errors)=1).

Error reports:

- MUST NOT contain diagnostic statements
- MUST NOT contain recommendations

9.4 Prohibited Error Practices

An implementation:

- MUST NOT encode errors in findings
- MUST NOT interpret errors as a valid result
- MUST NOT ignore or bypass errors

Errors MAY be transported exclusively via WireEnvelope.errors. A wire result (DIAGNOSTIC_FINDING / Envelope Result) MUST NOT simultaneously carry a run-level error in WireEnvelope.errors.

9.5 Mapping of Error Classes to the Error-Code Registry

Semantic error class	Meaning	Error codes (registry)	Explanation
E1. INVALID_REQUEST	Request structurally or syntactically invalid	E1000 MALFORMED_REQUEST; E1001 MISSING_REQUIRED_FIELD; E1002 UNSUPPORTED_VERSION; E1003 UNKNOWN_CONTEXT_REFERENCE; E1004 INVALID_TRANSFORM_SPEC	Errors arise before the start of diagnosis due to impermissible or incomplete requests
E2. INVALID_CARRIER	Carrier not diagnosable	E2001 VARIANT_NOT_IN_CARRIER_DOMAIN; E2002 EQUIVALENCE_INVARIANCE_VIOLATION	Carrier or derived variants violate formal carrier prerequisites
E3. INVALID_CONTEXT	Context inconsistent or not applicable	E1005 CONTEXT_INCONSISTENT	Context definition contradicts the carrier or the permissible transformations
E4. EXPLORATION_FAILURE	Exploration not feasible or non-terminating	E2000 NON_TERMINATING_EXPLORATION; E2003 TRANSFORM_APPLY_ERROR; E2004 EXPLORATION_BUDGET_EXCEEDED	Exploration fails despite a formally valid request
E5. CLASSIFICATION_UNDEFINED	Classification not well-defined	E3000 CLASSIFICATION_UNDEFINED; E3001 COMPARABILITY_VIOLATION; E3002 DIMENSION_INVALID; E3003 INTERNAL_EVALUATION_ERROR	Evaluation yields no consistent ordinal classification
E6. INTERNAL_INCONSISTENCY	Violation of normative invariants	E4000 UPDATE_UNDEFINED; E4001 UPDATE_NOT_REALIZABLE; E4002	Protocol-internal violation of invariants or

	/ internal inconsistency	PROGRESS_VIOLATION; E9000 INTERNAL_ERROR; E9001 TIMEOUT	implementation inconsistency

10. Conformance and Interoperability

This section defines mandatory minimum requirements for implementations and criteria for comparability between different executions of the protocol.

10.1 Conformance Classes

The protocol distinguishes two logical roles that can be implemented independently:

- Client
- Analyzer

An implementation SHOULD combine both roles but MUST logically separate them.

10.2 Client Conformance

A conformant Client implementation:

- MUST generate valid DIAGNOSE_REQUESTs per Section 8
- MUST set all required fields
- MUST NOT interfere with the diagnostic flow
- MUST correctly interpret error codes
- MAY abort diagnoses, which leads to FAILED

A Client:

- SHOULD NOT generate classifications
- SHOULD NOT perform exploration

10.3 Analyzer Conformance

A conformant Analyzer implementation:

- MUST implement the complete state machine (Section 7)
- MUST distinguish all error classes (Section 9)
- MUST, for each valid request, deliver either:
    - a complete DIAGNOSTIC_FINDING, OR
    - a standardized error
- MUST terminate (no infinite run)

An Analyzer:

- MAY use internal heuristics
- SHOULD NOT output non-normative results
- SHOULD NOT declare partial results as a finding

10.4 Interoperability

Two implementations are considered interoperable if:

- they accept identical DIAGNOSE_REQUESTs,
- they produce equivalent DIAGNOSTIC_FINDINGS,
- they deliver identical error codes for identical error cases.

Interoperability relates exclusively to:

- ordinal classifications
- validity markers
- error codes

Internal explorations MUST NOT be identical.

10.5 Equivalence of Findings

Two DIAGNOSTIC_FINDINGS are equivalent if:

- the same set of dimensions is classified,
- each dimension has the same ordinal class,
- the validity markers match.

Metadata MAY differ as long as it is not normative.

10.6 Minimum Requirements for Implementations

A minimum implementation MUST:

- be able to successfully perform at least one diagnosis,
- correctly handle at least one error case,
- deterministically terminate,
- produce a complete finding.

An implementation that does not meet these conditions is not conformant.

11. Security Considerations (Protocol Level)

This protocol defines no transport mechanisms, no authentication, and no access control. Security aspects therefore do not arise at the protocol level, but from embedding into specific environments.

Implementations MUST ensure:

- that Structural Carriers are not disclosed without authorization,
- that diagnostic contexts are not manipulated,
- that Diagnostic Findings are not altered or selectively suppressed.

The protocol itself does not generate active interventions, but describes structural explorations. Risks arise exclusively from the selection of permissible interventions and are the responsibility of the Client.

12. Operational Considerations (Protocol Level)

MODIVX-WP is not a real-time protocol.

Implementations:

- MUST guarantee finite exploration,
- MUST implement abort conditions,
- MAY apply resource-dependent heuristics.

Protocol conformance does not depend on:

- runtime,
- computational effort,
- memory consumption,
  but exclusively on the semantic correctness of the results.

13. References

13.1 Normative References

RFC 2119 – Key words for use in RFCs to Indicate Requirement Levels

RFC 8174 – Ambiguity of Uppercase vs Lowercase in RFCs

13.2 Informative References

- MLD Mathematical Core
- MODIVX Binding Specifications
- MODIVX Wire Output Envelope
- MODIVX JASON Schema

14. Normative Protocol Example (Non-Executable)

14.1 Example DIAGNOSE_REQUEST

Structural Carrier:

Formal representation of an adaptive decision system, consolidated to structural equivalence

Structural Context:

Local internal reweighting permitted; no extension of state space

Intervention Budget:

Maximum exploration depth = 2

Diagnostic Scope:

Structural dimensions D_1–D_n

14.2 Example DIAGNOSTIC_FINDING

Carrier Reference:

As specified in request

Classifications:

D_1 → stable

D_2 → reactive

D_3 → unstable

Validity Marker:

Exploration completed, classification well-defined

15. Validation

Upon receipt of a DIAGNOSE_REQUEST, the Analyzer MUST check:

- syntax / ABNF conformance,
- completeness of all REQUIRED fields,
- version compatibility,
- finiteness and validity of the transformation list,
- canonicity / verifiability of the context_fingerprint [PROFILE:DCO].

In case of violation, an ERROR with phase VALIDATING MUST be sent. [PROFILE:DCO]

16. Exploration

16.1 Exploration-Plan

The Analyzer MUST determine an exploration plan consisting of:

- depth_limit (Default: 1)
- max_variants (Default: |Transforms|)
- coverage_mode ∈ {TRANSFORM, FRONTIER} (Default: TRANSFORM)
- allow_partial (Default: false)

If coverage_mode = FRONTIER, the Analyzer MAY internally increase max_variants. This MUST be reflected in the Context Fingerprint. [PROFILE:DCO]

16.2 Minimum Requirements

16.2.1 TRANSFORM Coverage (Default)

For each transformation p in Transforms, the Analyzer MUST attempt at least once to apply p to the initial carrier (or to suitable frontier elements, if the implementation model provides for this), and either:

- produce a defined variant, or
- record not_applicable.

If this cannot be ensured, E2003 or E2004 MUST be triggered.

16.2.2 FRONTIER Coverage (Optional)

With coverage_mode = FRONTIER, at least one frontier round MUST be executed in which—if possible—at least one new variant is produced.

16.3 Completion Criterion

Exploration ends if at least one condition holds:

- fixpoint (no new variants)
- depth_limit reached and no frontier
- max_variants reached
- error case

Non-termination under the given limits MUST trigger E2000.

17. Evaluation

17.1 Input

Evaluation MUST be performed on the frozen set of variants.

17.2 Ordinal Classification

For each dimension d in dimension_set_id, the Analyzer MUST determine results:

- ordinal_state(d)
- reactivity_class(d)
- sensitivity_class(d)
- stability_class(d)
- determination_profile_id

The protocol specification does not define semantics of dimensions; semantics are part of instantiation and are referenced via dimension_set_id.

17.3 Comparability

For each dimension, the Analyzer MUST check whether comparability is satisfied. In case of violation, Section 18 or 19 applies.

progress_scheme_id is part of the evaluation parameters and becomes a comparability attribute once shown in the finding.

18. Dimension Invalidity and Partial Results

“Exploration Completed” always refers to the configured exploration plan, taking allow_partial into account.

If comparability is violated for a dimension:

- the dimension MUST be marked as valid=false.

If allow_partial=false (Default):

- the Analyzer MUST abort with E3001/E3002 and send an ERROR.

If allow_partial=true:

- the Analyzer MAY send a DIAGNOSTIC_FINDING in which individual dimension(s) are valid=false, provided that at least one dimension has been classified as valid=true and no terminal error has occurred.

Normative note: valid=false does not mean that the mathematical diagnostic function is undefined. It means that, under this protocol run, the result for the dimension is not interoperable (e.g., due to a registry, version, or scope mismatch) and MUST NOT be used in the sense of the comparability and equivalence rules defined in Section 19.

- Terminal errors are: E3000 as well as error class E5. In these cases, the Analyzer MUST abort and send exactly one ERROR message.
- In partial findings, all dimensions with valid=true MUST be fully and normally reported.

19. Comparability and Equivalence of Findings

19.1 Comparability

Two findings are comparable if:

- version identical
- context fingerprint identical
- dimension set ID identical
- carrier equivalence ID equivalent
- progress scheme identical (if present in the finding)

19.2 Carrier Equivalence ID

carrier_equivalence_id MUST be provided as the pair (equiv_scheme_id, class_representative_id).

19.3 Context Fingerprint [PROFILE:DCO]

context_fingerprint is a canonical hash over:

- context description
- canonically sorted transformation list
- semantically relevant parameters: coverage_mode, depth_limit, max_variants, saturation mode (if referenced)

The Analyzer MUST state hash_alg_id in the finding and use canonical serialization.

19.4 Equivalence

Findings are equivalent if all dimensions with valid=true have identical values in both findings for:

- ordinal_state
- reactivity_class
- sensitivity_class
- stability_class

Differences in metadata are allowed.

20. Error Handling

20.1 ERROR-Message

Each transition to FAILED (S6) MUST produce exactly one ERROR message.

ERROR contains:

- error_code (registry)
- error_phase
- error_state
- error_detail
- job_id

20.2 Error-Code Registry

VALIDATING

- E1000 MALFORMED_REQUEST
- E1001 MISSING_REQUIRED_FIELD
- E1002 UNSUPPORTED_VERSION
- E1003 UNKNOWN_CONTEXT_REFERENCE
- E1004 INVALID_TRANSFORM_SPEC
- E1005 CONTEXT_INCONSISTENT

EXPLORING

- E2000 NON_TERMINATING_EXPLORATION
- E2001 VARIANT_NOT_IN_CARRIER_DOMAIN
- E2002 EQUIVALENCE_INVARIANCE_VIOLATION
- E2003 TRANSFORM_APPLY_ERROR
- E2004 EXPLORATION_BUDGET_EXCEEDED
- E2005 EXPLORATION_EMPTY
- E2006 EXPLORATION_NOT_FEASIBLE

EVALUATING

- E3000 CLASSIFICATION_UNDEFINED
- E3001 COMPARABILITY_VIOLATION
- E3002 DIMENSION_INVALID
- E3003 INTERNAL_EVALUATION_ERROR

ADAPTING

- E4000 UPDATE_UNDEFINED
- E4001 UPDATE_NOT_REALIZABLE
- E4002 PROGRESS_VIOLATION
- E4003 ADAPTATION_BUDGET_EXCEEDED

GENERAL

- E9000 INTERNAL_ERROR
- E9001 TIMEOUT

21. Conformance

21.1 Analyzer Conformance

An Analyzer is conformant if it:

- implements the state machine per Section 7,
- validates requests per Section 8,
- performs exploration per Section 16,
- performs evaluation per Section 17,
- reports comparability per Section 19,
- handles errors per Section 20,
- supports ABNF syntax per Section 22.

21.2 Client Conformance

A Client is conformant if it:

- generates requests per Section 8,
- correctly applies defaults when fields are missing,
- can parse findings and errors per ABNF.

22. Message Syntax (ABNF)

The normative serialization of messages and results MUST conform to the WireEnvelope data model as specified in “MODIVX WIRE OUTPUT ENVELOPE”. The JSON schema “MODIVX JSON SCHEMA” serves as a machine-checkable derivation; in case of conflicts, the textual envelope specification takes precedence.

Message = Request / Finding / Meta_Finding / Error

state = "S0" / "S1" / "S2" / "S3" / "S4" / "S4A" / "S5" / "S6"

Request = "DIAGNOSE_REQUEST" CRLF

"Version:" SP ver CRLF

"Job-Nonce:" SP token CRLF

"Carrier-ID:" SP token CRLF

"Equiv-Scheme:" SP token CRLF

"Context-Fingerprint:" SP token CRLF

"Dimension-Set-ID:" SP token CRLF

"Transforms:" SP tlist CRLF

["Depth-Limit:" SP 1*DIGIT CRLF]

["Max-Variants:" SP 1*DIGIT CRLF]

["Coverage-Mode:" SP ("TRANSFORM" / "FRONTIER") CRLF]

["Allow-Partial:" SP ("true" / "false") CRLF]

["Update-Policy:" SP token CRLF]

CRLF

Finding = "DIAGNOSTIC_FINDING" CRLF

"Version:" SP ver CRLF

"Job-ID:" SP token CRLF

"Carrier-Equivalence-ID:" SP token CRLF

"Context-Fingerprint:" SP token CRLF

"Dimension-Set-ID:" SP token CRLF

"Hash-Alg:" SP token CRLF

["Progress-Scheme:" SP token CRLF]

["Update-Operator:" SP token CRLF]

["Adaptive-Fixpoint:" SP ("true" / "false") CRLF]

1*(Dimblock)

[metadata]

CRLF

Meta_Finding = "META_DIAGNOSTIC_FINDING" CRLF

"Version:" SP ver CRLF

"Job-ID:" SP token CRLF

"Adaptation-Scheme:" SP token CRLF

"Meta-Progress:" SP token CRLF

"Meta-Stability:" SP ("progressing" / "stabilized") CRLF

CRLF

Dimblock = "DIM" SP token CRLF

"Valid:" SP ("true" / "false") CRLF

["Determination-Profile:" SP token CRLF]

["Ordinal-State:" SP token CRLF]

["Reactivity:" SP token CRLF]

["Sensitivity:" SP token CRLF]

["Stability:" SP token CRLF]

[metadata]

CRLF

Error = "ERROR" CRLF

"Version:" SP ver CRLF

"Job-ID:" SP token CRLF

"Error-Code:" SP token CRLF

"Error-Phase:" SP ("VALIDATING" / "EXPLORING" / "EVALUATING" / "ADAPTING" / "GENERAL") CRLF

["Error-State:" SP state CRLF]

["Error-Detail:" SP 1*(VCHAR / SP) CRLF]

[metadata]

CRLF

metadata = "Metadata:" SP 1*(VCHAR / SP) CRLF

tlist = token *("," token)

ver = 1*DIGIT "." 1*DIGIT

token = 1*(ALPHA / DIGIT / "-" / "_" / ".")

CRLF = %x0D.0A

SP = %x20

VCHAR = %x21-7E

ALPHA = %x41-5A / %x61-7A

DIGIT = %x30-39

Defaults (normativ):

- Depth-Limit default = 1
- Max-Variants default = |Transforms|
- Coverage-Mode default = TRANSFORM
- Allow-Partial default = false

23. Examples

23.1 Example Request

DIAGNOSE_REQUEST

Version: 1.0

Job-Nonce: n-001

Carrier-ID: sysA.rep1

Equiv-Scheme: eqv.v1

Context-Fingerprint: ctxhash123

Dimension-Set-ID: dims.v1

Transforms: p1,p2,p3

Depth-Limit: 2

Max-Variants: 50

Coverage-Mode: TRANSFORM

Allow-Partial: false

23.2 Example Finding

DIAGNOSTIC_FINDING

Version: 1.0

Job-ID: job-777

Carrier-Equivalence-ID: eqv.v1:classA

Context-Fingerprint: ctxhash123

Dimension-Set-ID: dims.v1

Hash-Alg: sha256

DIM D1

Valid: true

Ordinal-State: S1 ; per dimension_set dims.v1

Reactivity: reactive

Sensitivity: sensitive

Stability: stable

DIM D2

Valid: true

Ordinal-State: S0

Reactivity: nonreactive

Sensitivity: nonsensitive

Stability: stable

23.3 Example Error

ERROR

Version: 1.0

Job-ID: job-777

Error-Code: E3001

Error-Phase: EVALUATING

Error-Detail: comparability violated for dimension D3

24. Registries

The following registries are defined. An implementation MUST maintain local registries or configure them statically:

- Transformation Registry: transform_id → specification/implementation
- Dimension Set Registry: dimension_set_id → list of dimensions + semantic reference
- Equivalence Scheme Registry: equiv_scheme_id → equivalence check
- Hash Algorithm Registry: hash_alg_id

- Progress Scheme Registry: progress_scheme_id → implementation reference of the well-founded relation
- Determination Profile Registry: determination_profile_id → {invariant_set_id, perturbation_set_id, classification_rule_id}; the three sub-IDs may be maintained as internal registry sub-structures; the protocol value remains determination_profile_id.
- Update-Policy Registry: update_policy_id → {dimension_selector_id, update_operation_family_id}
- Update-Operator Registry: update_operator_id → operator reference
- Adaptation Scheme Registry: adaptation_scheme_id → {diagnostic_maps_family_id, update_operators_family_id}
- Meta Progress Registry: meta_progress_relation_id → well-founded meta order

25. Security Considerations (Operational / Deployment Level)

This section supplements Section 11 with operational aspects.

- Analyzer MUST enforce resource limits (max_variants, depth_limit).
- Analyzer MUST validate transformation IDs (registry).
- For network transport, authenticity/integrity SHOULD be ensured (e.g., TLS).
- Logs SHOULD not contain sensitive representation details unless necessary.

26. Operational Considerations (Implementation)

- Analyzer SHOULD maintain auditable logs per job_id: validation, exploration plan, error codes, invalid dimensions.
- In case of stochastic exploration, a seed SHOULD be documented in metadata; otherwise reproducibility is limited.

27. Conclusion

MODIVX-WP specifies an interoperable protocol layer for MLD structural diagnoses. Dimension semantics remain an instantiation matter and are referenced via registered IDs. The protocol ensures comparability and minimum coverage and provides clear error semantics.

Appendix: Process Map incl. Mathematical Core (MC) references

S1 — REQUESTED

Action: S1: diagnostic request received

Error: S1 → S6: request invalid

Success: S1 → S2: carrier & context syntactically valid

Forwarding: to S2 INITIALIZED

MC references: Definition 8: (Purpose Contexts P), Axiom 9 (context parameterization exactly one $p \in P$), Definition 10 (dimension index set D)

Process map:

(1) The request is accepted as a candidate for a diagnostic run.

(2) The run is prepared such that it is interpretable as a “diagnostic construction”:

- assignment to exactly one context parameter $p \in P$ (Ax. 9)
- fixation of a finite dimension index set D (Def. 10)

(3) Decision:

- if this assignment/basic structure is not possible → S6.
- if syntactically valid → hand-off to initialization S2.

S2 — INITIALIZED

Action: S2: carrier and context checked

Error: S2 → S6: carrier or context invalid

Success: S2 → S3: at least one permissible transformation exists

Forwarding: to S3 EXPLORING

MC references: Definition 4 (structural equivalence ≡), Proposition 1 (≡ is an equivalence relation)

Process map:

1. The run establishes/checks the structural basis on which, in the Core, carrier/variants become comparable via ≡.
2. For this, ≡ must be viable as an equivalence relation (Def. 4, Prop. 1).
3. Additionally it is checked that exploration does not run empty: at least one permissible transformation line exists.

4. Decision:

    - if carrier/context invalid → S6.
    - if no permissible transformation exists → exploration would be meaningless → no transition to S3 (protocol condition violated).
    - if at least one permissible transformation exists → S3.

S3 — EXPLORING

Action: S3 → S3: exploration not yet completed (loop)

Error: S3 → S6: exploration not feasible

Success: S3 → S4: exploration completed

Forwarding: to S4 CLASSIFYING

MC references: Definition 4 / Proposition 1 (equivalence relation for formation of the class basis $[x]$), Definition 12 (diagnostic assignment system δi), Axiom 13 (totality δi([x], p))

Process map

Loop:

1. S3 is explicitly an iteration: “not yet completed” ⇒ S3 again.

Individual steps within the loop (typical pattern):

1. Generate/check another structural variation (carrier/transformation sequence).
2. Assign variation(s) consistently via ≡ → formation/validation of candidates for equivalence classes $[x]$.
3. Check whether exploration is already “completed” (completion criterion in the sense of the protocol).

Abort conditions (error):

2. If exploration becomes “not feasible” (e.g., no more valid variants, inconsistent structure, non-terminating) → S6.

Success/exit of the loop:

3. If exploration is “completed” → a completed argument basis exists such that δi evaluation can be applied next in S4.

S4 — CLASSIFYING

Action: S4: classification is computed

Error: S4 → S6: classification not well-defined

Success: S4 → S5: classification successful

S4 → S4A: if adaptive execution is active

Forwarding: to S5 COMPLETED or S4A ADAPTING

MC references: Definition 12 (diagnostic assignment system δi), Axiom 13 (totality of diagnosis), Definition 13 (diagnostic state vector)

Process map:

Individual steps (see Section 7.3 for S4 itself):

1. Classification is computed for the argument basis established in S3.
2. Evaluate the δi family and build the diagnostic state vector (Def. 12, Def. 13); totality guarantees definedness in this model framework (Ax. 13).

Decision points:

4. if “not well-defined” → S6.
5. if “successful”:
   (1) either direct completion S5, or
   (2) transition to S4A if “adaptive execution active” (protocol branch).

S4A — ADAPTING

Action: S4A: adaptation phase (update) (state is explicitly S4A ADAPTING, entry from S4 when “adaptive execution active”)

Error: S4A → S6: error in adaptation/update phase leads to FAILED (in the state model, globally: “any state → S6 on error”; moreover update/progress error codes are provided as separate classes)

Success: S4A → S5: diagnosis completed (COMPLETED) after successful adaptation phase

Forwarding: to S5 COMPLETED

MC references:

→ Definition 36: Adaptive Update Operator (update rule/family)

→ Definition 37: Adaptive Progress Condition

→ Axiom 22: Dimensionwise Progress (update satisfies progress condition)

→ Definition 38: Adaptive Trajectory (sequence of iterated updates)

→ Axiom 10: Well-Foundedness (as termination basis)

→ Definition 40: Adaptive Stabilization (eventually constant)

→ Proposition 10: Stabilization of Adaptive Trajectories (stabilization guaranteed)

Process map:

I. Entry (single step)

1. S4 → S4A occurs exactly when, after successful classification, “adaptive execution is active”.
2. Mathematically: the run transitions from a computed diagnostic state into update dynamics (Def. 36, Def. 38).

II. Core process (loop structure / iteration)

S4A is processually the place where iterated application takes place:

1. Apply update via update operator (Def. 36).
2. Result is part of an adaptive trajectory (Def. 38).
3. After each update, check the adaptive progress condition (Def. 37).
4. The Core requires that the update operator satisfies the progress condition (Axiom 22).

Thus S4A is conceptually a loop/iteration, even if the protocol does not necessarily externalize inner iteration steps as separate messages/states.

III. Abort conditions (error)

A protocol ERROR (S4A → S6) is mandatory if the adaptation phase cannot be executed consistently, in particular if:

→ the progress condition cannot be satisfied (Def. 37 / Ax. 22)

→ well-foundedness/termination basis is not given (Ax. 10)

At the protocol level, this is handled as an error in S4A, and the global abort rule “any state → S6 on error” applies.

IV. Exit / success (completion)

S4A ends successfully if the adaptive dynamics can be carried into completion:

→ mathematically: adaptive stabilization (Def. 40)

→ and the associated guarantee: stabilization of adaptive trajectories (Prop. 10)

At the protocol level this corresponds to the transition: S4A → S5 (COMPLETED).

S5 — COMPLETED

Action: S5: diagnosis successfully completed (terminal)

(No S5 → S6 is specified as a normal transition; errors lead into the error path per global abort rule, but S5 itself is terminal.)

S5 is the success/terminal state (“COMPLETED”).

Forwarding: production of a DIAGNOSTIC_FINDING as the completion artifact of the run (in the sense of the completion condition “a consistent DIAGNOSTIC_FINDING produced”).

In addition, a META_DIAGNOSTIC_FINDING MAY be produced after a terminal state.

MC references: Definition 40 (Adaptive Stabilization), Proposition 10 (Stabilization of Adaptive Trajectories)

Process map:

1) Entry into S5

- S5 is reached via the success transitions permitted in the protocol:
- S4 → S5: “classification successful”, or
- S4A → S5: successful adaptation phase

2) Completion conditions

Transition to COMPLETED only permitted if (explicitly stated):

- “exploration is completed”
- “all relevant dimensions are classified”
- “a consistent DIAGNOSTIC_FINDING has been produced”

3) Messaging

- completion is materialized outward via a DIAGNOSTIC_FINDING.
- optional: META_DIAGNOSTIC_FINDING may be produced after the terminal state (not part of the operational state machine).

4) Link to MC

If the run passed through S4A, then completion is compatible with:
adaptive stabilization (Def. 40) and the stabilization guarantee (Prop. 10). This explains mathematically why a terminal completion state exists.

S6 — FAILED

Action: S6: diagnosis aborted (terminal)

Error: S6 is reached as soon as an error transition occurs, in particular:

- S1 → S6: “request invalid”
- S2 → S6: “carrier or context invalid”
- S3 → S6: “exploration not feasible”
- S4 → S6: “classification not well-defined”
  as well as globally: “any state → S6 on error”

Success: no success: S6 is the terminal error state.

Forwarding: upon transition to FAILED, it holds: MUST produce exactly one ERROR message.

In addition, a META_DIAGNOSTIC_FINDING MAY be produced after a terminal state.

MC references: no direct “FAILED” definition in the Core (the Core does not formalize protocol terminal states).

Mathematical error causes that may lead protocol-wise to FAILED typically correspond to violations of:

- Axiom 13 (Totality)
- Axiom 22 (Dimensionwise Progress)
- Axiom 10 (Well-Foundedness)
  (these are not protocol transitions, but mathematical impossibility/inconsistency as a cause).

Process map:

1. Entry into S6 (single step)
   S6 is reached by an error transition from any state:
   - explicitly from S1/S2/S3/S4 (above),
   - plus the global rule “any state → S6 on error”.
2. Terminality
   FAILED is terminal. After entry, no further operational states are traversed.
3. ERROR message handling
   For each transition to FAILED, exactly one ERROR message MUST be produced.
   Thus the ERROR message is not bound to “after validating”, but to the transition into S6.
4. Optional meta outcome
   After a terminal state (including FAILED), a META_DIAGNOSTIC_FINDING may additionally be produced.


RUBEN JAYBIRD
INSTITUTE

PROCESS DIAGRAM