# HL7 Validation — The Three Tiers

MessageFoundry validates HL7 v2.x at three distinct tiers. Each catches a different class of problem; together they let a feed be both *tolerant* (real-world HL7 is frequently non-conformant) and *safe* (a safety-critical feed can reject anything off-spec). You pick the tiers a given feed needs — they compose.

| Tier | When | Checks | On failure |
|---|---|---|---|
| 1. Tolerant peek | Always (hot path) | Parse + fast field access; size/segment caps; MSH present | Unparseable / oversized → `ERROR` (NAK), never crashes the connection |
| 2. Strict structural | Opt-in per inbound ( `strict=True` ) | Version-aware **schema**: segment cardinality, datatypes, table values, lengths, MSH-12 version | Non-conformant → synchronous NAK (AR/AE) at the listener |
| 3. Business consistency | In a Router/Handler | **Cross-field** coherence the schema can't express | Handler decides: `FILTERED` or `ERROR` /dead-letter |

## Tier 1 — Tolerant Peek (Always On)

The hot path uses fast, forgiving field access ( `msg["PID-3"]` ), so routing never pays for full structural validation. It enforces only hard safety limits: maximum message bytes, maximum segments, and MSH presence. A message that can't be parsed, or that exceeds a limit, is routed to the error/dead-letter path and logged `ERROR` — it never crashes the connection.

This is the right default for most feeds: you accept what arrives, preserve the raw message, and route bad messages to where an operator sees them.

## Tier 2 — Strict Structural Validation (Opt-In)

For a feed where an off-spec message must be *rejected at the door*, enable strict validation on the **inbound** connection. It is version-aware — it checks the message against the official HL7 structure for its version — and it is the slow path, so it is kept off the routing hot path and is **opt-in per connection**:

```
from messagefoundry import MLLP, inbound

inbound("IB_ADT", MLLP(port=2575), router="adt_router", strict=True, hl7_version="2.5")
```

A non-conformant message is **NAK'd synchronously** (AR/AE) at the listener, before it is ever routed — the sender is told immediately. Be **explicit about** `hl7_version` for a strict feed; don't rely on silent

autodetection.

Enable strict validation when:

- the downstream system is intolerant of malformed structure,
- the feed is contractually conformant, or
- correctness outweighs throughput.

Leave it off for a tolerant archival or forwarding feed.

Strict validation surfaces a single conformance error rather than a full report — a full report of a message could leak field data. (See the PHI-safety note below.)

## Tier 3 — Cross-Field Business Consistency (Router/Handler)

Strict validation checks each item against the schema **independently**. It does **not** check that *combinations of related items are reasonable*: that a required identifier is present, that a value is echoed consistently across segments, or that admit ≤ discharge. That combined-item consistency is the **application's** job — in a code-first engine, the **Router/Handler**.

MessageFoundry provides small, **generic, composable** consistency primitives. Each takes the parsed `Message` plus field *paths* and returns a list of **PHI-safe** `Violation`s (rule + path, **never the field value**):

| Primitive | Checks |
|---|---|
| `required(msg, *paths)` | each path is present (non-empty) |
| `same_across(msg, *paths)` | the paths all hold the same value (catches differing values *and* present-vs-absent) |
| `valid_date(msg, path)` | the value is a well-formed HL7 date/time (when present) |
| `dates_in_order(msg, earlier, later)` | `earlier ≤ later` (when both present & valid) |
| `matches(msg, path, pattern)` | the value fully matches a regex (a field-format allow-list) |
| `check(*groups)` | flattens several results into one list for a single decision |

The library **detects; the Handler decides.** Keeping the primitives pure preserves the at-least-once *re-run* invariant: a Handler is a pure function of the message, so a replayed message yields the same outcome. Compose the generic primitives into your feed's message-type-specific rules:

```
from messagefoundry import Send, handler
from messagefoundry.parsing.consistency import (
    ConsistencyError, check, dates_in_order, required, same_across, valid_date,
)

@handler("validate_and_archive")
def validate_and_archive(msg):
    violations = check(
        required(msg, "PID-3", "PID-5", "MSH-10"),   # patient id, name, control id present
        same_across(msg, "MSH-9.2", "EVN-1"),         # trigger event echoed in EVN-1
        valid_date(msg, "PID-7"),                     # date of birth well-formed
        dates_in_order(msg, "PV1-44", "PV1-45"),      # admit ≤ discharge
    )
    if violations:
        raise ConsistencyError(violations)            # → ERROR / dead-letter (operator sees it)
        # ...or `return None` to drop it silently (logged FILTERED)
    return Send("FILE-OUT_ADT", msg)
```

Acting on a non-empty result is a deliberate choice:

- `raise ConsistencyError(violations)` → the message goes to the error/dead-letter path (logged `ERROR`, surfaced as a console Alert). Use this when a downstream system can't safely consume it. The exception's string is built from rule + paths only, so it is **safe to log and store** (no PHI).
- `return None` → the message is dropped and logged `FILTERED`. Use this for a benign coherence issue you'd rather not forward.

A full worked example — wired inbound → router → handler → file — ships in the engine samples:

```
python -m messagefoundry serve --config samples/consistency --db ./mf.db --env dev
```

#### PHI-Safety

A `Violation` records the **rule and the field path(s)**, never the field *value*. So violations can be logged, aggregated, or stored as a disposition without leaking PHI: a date violation reads *"PID-7 is not a valid HL7 date/time"*, never the offending value. This is what makes consistency results safe to surface in the console, in Alerts, and in Dead-Letters.

## Choosing Tiers

- **Most feeds:** Tier 1 only — tolerant, route bad messages to `ERROR`.
- **Add Tier 2** (`strict=True`) when off-spec *structure* must be rejected at the door.
- **Add Tier 3** when *business rules across fields* matter (required identifiers, sane and ordered dates, cross-segment coherence) — independent of whether Tier 2 is on.

## See Also

- PHI handling and redaction