

SPEC-2026-007 — SynthIQ Health Conformance Contract for Third-Party Backends

Prose designation: SPEC-2026-007 **QMS controlled-doc number:** DOC-2026-392 (*status: draft — pending operator review/approval*) **Version:** v1 **Status:** Draft → for QMS registration + portal publication **Product:** SynthIQ (DICOM load-balancing director) **Roadmap:** RI-2026-134 (vendor-neutral pooling) — Phase 2b **Implements / supersedes:** the wire shape consumed by SynthIQ `BackendHeartbeatService` as of SynthIQ 1.0.0.105 (DCR-2026-770 liveness tiering, DCR-2026-771 weighted load, DCR-2026-772 CERTIFIED parse + load_score promotion) **Conformance tool:** `tools/validate_synthiq_health.py` **Classification:** **NON-DEVICE.** This contract carries operational load telemetry only — never PHI, never clinical content. Per FD&C Act §520(o)(1)(D).

1. Purpose & scope

SynthIQ is a **vendor-neutral DICOM load-balancing director**: it sits in front of a *pool* of homogeneous imaging backends and routes each inbound study to the least-busy healthy member. SynthIQ does not interpret images and is not a diagnostic device.

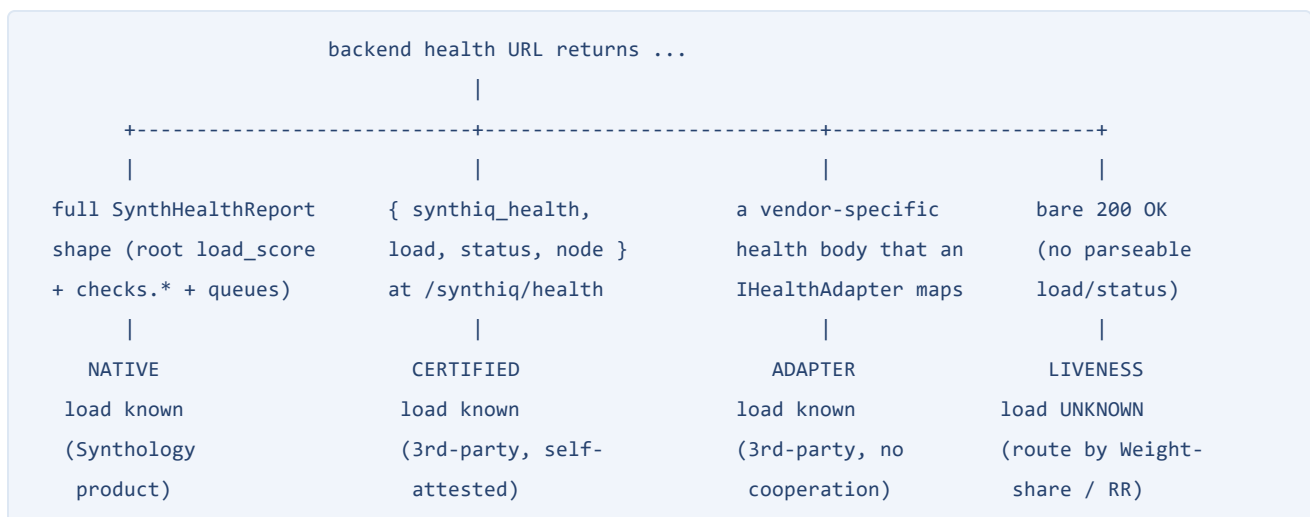
A pool may contain Synthology products (SR Engine, Router, ...) **and** third-party imaging backends (an AI inference vendor, a competitor's storage SCP, a homegrown receiver). For SynthIQ to balance load *fairly* across a mixed pool, every member must tell SynthIQ **how busy it is** in a common, comparable unit.

This specification defines that common unit — the **SynthIQ Health Contract** — and the four conformance **tiers** a backend can occupy. It is the contract a third-party vendor implements to earn a **CERTIFIED** node badge ("SynthIQ-Ready"). It is normative for the CERTIFIED tier and informative for the others.

Out of scope: the DICOM transport itself (C-STORE association handling is unchanged and vendor-native), authentication to the backend, and the *clinical* behaviour of the backend. SynthIQ only reads health and routes associations.

2. The four backend tiers

SynthIQ classifies each pool member into exactly one tier, **by the shape of what its health URL returns** — not by configuration. Classification is per-heartbeat ($\approx$ every 5 s) and can change as a backend degrades.



Tier	Who	Health surface	LoadKnown	How SynthIQ routes it
NATIVE	A Synthology product	Full SynthHealthReport (root load_score , checks.* , queues) — see §6	true	least-busy by load_score
CERTIFIED	A third party that implemented this spec	GET /synthiq/health → the §3 contract	true	least-busy by load , ranked <i>with</i> NATIVE nodes in one comparable scale
ADAPTER	A third party that did not implement the spec, but whose native health can be translated	vendor-native; a SynthIQ-side IHealthAdapter maps it to the contract (Phase 3)	true (via adapter)	least-busy by the adapter’s computed load
LIVENESS	A backend that only proves it’s <i>alive</i>	any URL that returns 200 but no parseable load/status	false	weighted round-robin by Weight ; excluded from least-busy

Detection precedence (first match wins, per heartbeat):

1. Body parses as JSON **and** contains the magic field `synthiq_health` → **CERTIFIED**.
2. Else body parses as the NATIVE `SynthHealthReport` shape (root `load_score` numeric) → **NATIVE**.
3. Else a configured `IHealthAdapter` claims the body → **ADAPTER**.
4. Else HTTP 200 → **LIVENESS** (`LoadKnown=false`).
5. Else (non-200 / unreachable / timeout) → **unhealthy**, drained from rotation.

Why tiers, not a flag: a CERTIFIED backend that starts returning malformed health (bad deploy, version regression) automatically **degrades to LIVENESS** rather than poisoning least-busy with a stale/garbage `load` . The pool keeps routing (by Weight-share) instead of stalling. Fail-safe over fail-shut.

3. The CERTIFIED contract (normative)

A CERTIFIED backend exposes `GET /synthiq/health` returning `application/json` with HTTP 200. The body:

```
{
  "synthiq_health": "1.0",           // REQUIRED. Magic + contract version. Its PRESENCE is how SynthIQ
                                     // detects a CERTIFIED node. Major version gates compatibility
  ($7).
  "load": 37,                       // REQUIRED. Normalized weighted busyness, number in [0,100].
                                     // 0 = idle, 100 = saturated. THE primary least-busy routing key.
                                     // MUST reflect queue *content cost*, not raw object count ($4).
  "status": "ready",                // REQUIRED. Enum: ready | busy | draining | down.
                                     // ready/busy => routable. draining/down => drained (no new
  studies).
  "node": "acme-ai-inference-01",    // REQUIRED. Stable per-instance identifier. Drives study->node
                                     // affinity and console/labelling. Stable across restarts.

  "queues": { "inference": 3, "egress": 1 }, // OPTIONAL. {name: depth>=0}. Rendered in Queue
  Health.
  "disk_free_pct": 64,               // OPTIONAL. number [0,100]. Surfaced in the pool DISK column.
  "version": "acme-2.4.1"           // OPTIONAL. Informational build string.
}
```

3.1 Field rules

Field	Req	Type	Rule	If violated
<code>synthiq_health</code>	✓	string	"<major>.<minor>", major ≥ 1	absent → not detected as CERTIFIED (falls to LIVENESS)
<code>load</code>	✓	number	$0 \leq \text{load} \leq 100$	absent/non-numeric → CERTIFIED parse fails → LIVENESS. Out of range → clamped to [0,100]
<code>status</code>	✓	string	one of <code>ready, busy, drainin g, down</code>	absent or outside enum → node drained (fail-safe)
<code>node</code>	✓	string	non-empty, stable	absent → CERTIFIED parse fails → LIVENESS
<code>queues</code>	✗	object	<code>{string: int≥0}</code>	malformed → ignored
<code>disk_free_pct</code>	✗	number	<code>[0,100]</code>	malformed → ignored
<code>version</code>	✗	string	—	—

3.2 Status → routing semantics

status	Routable?	Meaning
ready	✓	accepting work, healthy
busy	✓	accepting work but heavily loaded (still ranked by <code>load</code> ; a busy node loses to a ready one with lower <code>load</code>)
draining	X	finishing in-flight work; send no new studies (graceful shutdown / maintenance)
down	X	not serving

`busy` is **not** a drain signal — it is informational. Routing is by `load` ; `status` only gates routable-vs-drained. This lets a backend say “I’m slammed (load 95, busy) but still take work if I’m the least-bad option” — which is correct for a pool with no idle members.

3.3 PHI prohibition (normative)

The body MUST NOT contain any patient, study, or person-identifying content. `/synthiq/health` is operational telemetry. The conformance tool fails any payload whose field names look like PHI (`patient` , `mrn` , `accession` , `studyInstanceUID` , ...). This preserves SynthIQ’s and the backend’s non-device posture.

4. Load semantics — why content, not count

`load` is the heart of the contract. **A naïve `load` = queue-depth is a lie** when queue items have wildly different costs.

Worked example (the bug this spec exists to prevent — DCR-2026-771): SR Engine’s outbound queue holds *both* HL7 messages (cheap, sub-millisecond forward) and DICOM image forwards (expensive STOW of multi-MB objects). A depth of “10” means something completely different in each case:

Outbound queue	Naïve depth-load	Honest weighted-load
10 HL7 messages	100 (saturated!)	31
10 DICOM image forwards	100	100

If SynthIQ believed the naïve number it would route *away* from a node holding 10 trivial HL7 messages as if it were as busy as one chewing 10 big image transfers. The HL7 node is actually nearly idle.

4.1 Recommended normalization

SynthIQ’s NATIVE products compute `load` via the shared `WeightedLoad.Score` helper:

```
load = round( 100 * min(1, weighted_work / (slots * k)) )

weighted_work =  $\sum$  (queue_depth_i * cost_weight_i)    # cost-weight each queue by its real unit cost
slots          = the backend's concurrency (max parallel jobs / associations)
k              = a per-product tuning constant: "how many weighted units per slot before we call it
full"
```

Reference calibrations (NATIVE):

Product	weighted_work	slots	k
SR Engine	forwarding_stow*8 + forwarding_hl7*1 + received*1	MaxConcurrentJobs	8
Migration Engine	$\sum$ num_instances over migrating projects	MaxParallelCmove	2000

A CERTIFIED vendor is **free to compute load** however best models their backend — GPU utilization, model queue depth $\times$ per-model latency, memory pressure, whatever — as long as the result is a sane, monotonic, **varying** [0,100] that rises with real busyness. The soak test (§8.2) checks that it actually varies.

5. How SynthIQ routes (informative)

For a pool, per inbound study:

1. **Affinity** — if the study's `StudyInstanceUID` was already routed to a member that is still routable, keep it there (study cohesion: all instances of one study land together).
2. **Least-busy among load-known members** (NATIVE + CERTIFIED + ADAPTER): order by **load ASC** → **queue_depth ASC** → **disk_free_pct DESC** → **round-robin cursor**. `load` is primary (DCR-2026-772).
3. **Weight-share among liveness-only members** if no load-known member is routable: weighted round-robin by `Weight` (positional cursor). `Weight = 0` ⇒ **cold standby** (never routed unless every other member is down). A liveness node is otherwise **excluded** from the least-busy comparison (its load is unknown — it must not be treated as load 0 and flooded; DCR-2026-770 fixed exactly that flood).
4. **Fallback** — if nothing is routable, the study queues / errors per the pool's overflow policy.

CERTIFIED and NATIVE nodes compete in **one** comparable `load` scale — a CERTIFIED node reporting `load=15` beats a NATIVE node at `load=45` for the next study, and vice-versa. That is the whole point of normalization.

6. NATIVE shape (informative cross-reference)

NATIVE Synthology products emit the richer `Synthology.Shared.Web.SynthHealthReport` shape (root `load_score`, `checks.associations`, `checks.cstore_audit`, `queues`). SynthIQ reads `load_score` as the load-known signal. Third

parties do **not** implement the NATIVE shape — they implement §3 (CERTIFIED), which is the minimal comparable subset. The two are kept interconvertible by the shared `WeightedLoad` helper so the scales match.

7. Versioning & forward compatibility

`synthiq_health` is "`<major>.<minor>`".

- **Minor bump** (`1.0` → `1.1`): additive only — new OPTIONAL fields. Old SynthIQ ignores unknown fields and keeps consuming the required ones. A CERTIFIED node may emit `1.1` to any SynthIQ ≥ 1.0 .
- **Major bump** (`1.x` → `2.0`): may change required-field meaning. A SynthIQ that does not recognize the major version **degrades the node to LIVENESS** (does not trust the load) rather than mis-reading it. Forward progress without poisoning routing.

SynthIQ 1.0.0.105 understands `synthiq_health` major **1**.

8. Conformance & self-attestation

A vendor earns the CERTIFIED ("SynthIQ-Ready") badge by **self-attesting**, then Synthology **counter-signs**. This is deliberately lightweight — it is a load-balancing contract, not a clinical claim.

8.1 Static conformance

```
python validate_synthiq_health.py https://your-backend.example/synthiq/health
```

Must exit **0** (no MAJOR findings). The tool checks every §3.1 rule plus the §3.3 PHI prohibition.

8.2 Soak attestation (proves the load is real, not a constant)

```
python validate_synthiq_health.py https://your-backend.example/synthiq/health --soak 30 --json >
soak.json
```

Run under representative traffic for ≥ 30 min. The soak log records the load distribution (min/mean/p50/p95/max), the status histogram, and **varies** — `load` MUST take ≥ 2 distinct values across the soak. A flat-line `load` reads as un-wired or synthetic and is rejected at counter-sign.

8.3 Submission

Submit to Synthology: (a) the soak `soak.json`, (b) the backend's `node / version` and a one-paragraph description of how `load` is computed (§4), (c) the backend's transport details for a joint pool test. Synthology re-runs the validator + a live pool routing test, then **counter-signs** the attestation and issues the CERTIFIED listing. No source disclosure is required — only the health surface is in scope.

8.4 Re-attestation

Re-attest on any change to the `load` computation or a `synthiq_health` major bump. Minor/additive changes do not require re-attestation.

9. Minimal reference implementation (informative)

```
GET /synthiq/health HTTP/1.1
Host: your-backend
```

```
{
  "synthiq_health": "1.0",
  "load": 0,
  "status": "ready",
  "node": "vendor-prod-01"
}
```

That four-field body is a **complete, conformant** CERTIFIED node (load pinned to a constant fails the *soak* but passes static validation — wire real load before attesting). Add `queues / disk_free_pct` to light up the SynthIQ Queue Health and DISK columns.

10. Security & non-device posture

- `/synthiq/health` is **operational telemetry only** — no PHI, no clinical content (§3.3, enforced).
 - SynthIQ never sends a study to a `draining / down / unknown-status` node (fail-safe drain).
 - A malformed CERTIFIED body degrades to LIVENESS, never crashes the director (DCR-2026-772 unknown-major path).
 - The contract makes no diagnostic claim; it balances transport load. SynthIQ and conformant backends both retain their non-device classification.
-

11. Change history

Ver	Date	Change
v1	2026-06-22	Initial spec — extracted from the SynthIQ 1.0.0.105 implementation (DCR-2026-770/771/772).

Conformance tool: `tools/validate_synthiq_health.py` . SynthIQ implementation: `BackendHeartbeatService` (`TryParseCertifiedContract`), `SynthIqProxyProvider` (`PickLeastBusyAsync`), `WeightedLoad.Score` . Companion partner-facing brief: SB-2026-002 (DOC-2026-393).

Regulatory classification. Non-device software under the FD&C Act §520(o)(1)(D) software exclusion. This statement is published by Synthology Healthcare Solutions Group, LLC as a controlled document; the authoritative version is the approved record in the Synthology quality-management system.