Format Specification

UOS: Unified Oral Scene

Version 0.2

An open container for a complete dental case:
geometry, volume, appearance, imaging, clinical findings
and the *relations between them*, in one file.

Status	Draft; reference implementation complete and shipping
Container media type	`application/vnd.historia.uos` (proposed, unregistered)
File extension	`.uos`
Manifest schema	`schemas/uos-manifest-0.2.schema.json` (JSON Schema 2020-12; see §1.5 for its `$id`)
Dependencies	glTF 2.0 and `KHR_gaussian_splatting` (Release Candidate)
References	DICOM PS3.10; FHIR R4; ISO 3950 (FDI)
Audience	Implementers of readers, writers, viewers and agents

Editor	Luis Garbayo Fernández

Produced under	Version 0.2 was written during the Becas de Verano ANFAIA 2026, on a clinical case contributed by Historia.
Version	0.2 — last revised August 29, 2026

Every figure in this document was measured on a real clinical case contributed by Historia. This is a versioned, living specification, not the deliverable of a fixed term. Later versions revise it under the process of §16; §1.5 names where the schema for each version lives.

This document is the contract. An implementation that follows it can open, render, verify, extend and re-emit a `.uos` without access to the reference implementation and without asking its authors anything.

Contents

1 Introduction

1.1 What a `.uos` is

A dental case today is three unrelated things: one or two STL files from the intraoral scanner, a directory of several hundred DICOM slices from the CBCT, and a PDF report. Each lives in its own coordinate system, each opens in a different program, and the *relations* between them, where this CBCT sits with respect to this scan, which tooth this finding refers to, live in someone's head or inside proprietary software. Those relations do not travel with the data.

A `.uos` is those three things **plus the relations between them**, in one file, addressable over HTTP, verifiable byte by byte, and openable in a browser.

The one-sentence definition. A `.uos` is an **uncompressed ZIP** whose first entry is a `manifest.json` that declares every payload it carries by content hash, the named coordinate frames those payloads live in, and the measured transforms that relate those frames.

1.2 What a `.uos` is not

- **It is not a pixel format.** UOS never re-encodes source data. DICOM travels byte-identical; the STL travels byte-identical. What UOS adds is composition.
- **It is not a replacement for DICOM, glTF or FHIR.** It references them. A reader that already has a DICOM parser, a glTF loader and a FHIR client needs almost no new code, see §14.
- **It is not a clinical validation.** Everything produced by a model is isolated under `derived/` and marked `investigational` (§9).
- **It is not yet a standard.** It is a container proposal with a schema and a validator, both distributed with the reference implementation. Until a second implementer opens one, it is nothing more.

1.3 How to read this document

This document is long because it is exhaustive, not because all of it is for everybody. Find your task below and read only that path; each one is self-contained and ends in a worked recipe.

- **To show a `.uos` on screen** — read §2 (the container), §4 (the manifest), §5 (scenes, *in full*) and §10 (views), then the recipe *Render it* (§14.2). A minimal viewer is about thirty lines.
- **To open one and check it is valid** — read §2 (the container), §3 (the data model), §4 (the manifest) and §13 (the validation algorithm), then the recipe *Open a case and verify it* (§14.1).
- **To add data and re-emit it** — read §3 (the data model), §4 (the manifest), §11 (versioning: a `.uos` is never edited in place) and, if the base format has no place for what you carry, §12 (extensions). The recipes are in §14.
- **To act on one as an autonomous agent** — §13 and Appendix A are enough to operate: what to verify, what to refuse, what may be added and where. Everything else is justification for decisions you do not need to re-derive.
- **To judge the design rather than implement it** — read §1 for the problem this format exists to solve, the *Rationale* paragraphs and the highlighted boxes for why each decision went the way it did, and §16 for what v0.2 deliberately does not do.

1.4 Conformance language

The key words **MUST**, **MUST NOT**, **SHOULD**, **SHOULD NOT** and **MAY** are to be interpreted as described in RFC 2119. Sections and paragraphs marked *Rationale* are informative; everything else is normative.

Two failure classes are used throughout and they are not interchangeable:

Class	Meaning for a reader
Error	The container is invalid. A reader **MUST** refuse to present it as a valid case. Errors are always about internal contradiction — a declared hash that does not match, an asset that is referenced and absent.
Warning	The container is valid but carries a weaker guarantee than it could, and the reader **MUST** surface that to the user rather than swallow it. An unverified registration and an externalised original are the two canonical examples.

The design rule behind every warning in this document. The format's central commitment is that **two things that are not the same must never look the same**. An automatic alignment nobody reviewed and one signed by a clinician are different facts. A DICOM series carried inside the container and one merely referenced by hash are different guarantees. Where the format cannot enforce the difference, it **MUST** at minimum name it.

1.5 Normative artifacts

Artifact	Location and role
Manifest JSON Schema	JSON Schema 2020-12, at `schemas/uos-manifest-0.2.schema.json` in the reference repository and retrievable at its `$id`, `https://raw.githubusercontent.com/ANFAIA/Agentic-Smart-Health/uos-spec-v0.2/schemas/uos-manifest-0.2.schema.json` — pinned to a tag, so the identifier always names the same document. It is *generated* from the reference implementation's types, never hand-written. **What it does and does not check.** It is the only artifact a third party needs to check a manifest's *shape*. It cannot check truth: not that a `sha256` matches its asset, that the frame graph reaches the canonical frame, or that `derived/` carries its sidecar. That is why §13 requires both the schema and the validation algorithm, and why passing the schema alone is not conformance.
Reference implementation	Python package `uos` (v0.2.0). Modules map one-to-one onto the sections of this document: `manifiesto.py` (§4), `contenedor.py` (§2), `escena.py` (§5), `volumen.py` (§6), `clinico.py` (§8), `derivados.py` (§9), `vistas.py` (§10), `procedencia.py` (§11), `validador.py` (§13).
The measured container	`another_patient.uos` — 12 entries, 18 views, 3 declared extensions, a provenance chain of 3 versions. Every real value quoted in this document is read from it. **It is not distributed with this specification**: it holds one patient's case, pseudonymised but real, and it is held by the project rather than published. A reader should not go looking for a file that does not accompany the document. A distributable conformance fixture is listed as pending in §16.

Where this document and the reference implementation disagree, **this document is wrong and MUST be corrected**; but a reader encountering the disagreement **SHOULD** follow the implementation, because that is what the containers in the wild were written by.

2 The container

2.1 Physical format

A `.uos` file **MUST** be a ZIP archive (PKZIP, ISO/IEC 21320-1 profile) in which:

1. Every entry **MUST** use compression method `STORE` (0). A reader **MUST** report an error on any deflated entry.
2. The **first physical entry** **MUST** be named `manifest.json`.
3. Entry names **MUST** be relative, use `/` as separator, be pure ASCII, and **MUST NOT** contain a `..` path segment or a leading `/`.
4. An entry name ending in `/` denotes a *directory asset* (§3.4.2).

Rationale: why STORE. With STORE and the ZIP central directory at the end, an HTTP client issuing *range requests* reads the index and pulls a single asset without downloading the case. On the example container that is the difference between fetching 17.9 MB of scene and fetching 183 MB of case. Deflating the archive would take that away: a compressed entry has to be inflated from its start, so there is no useful byte range to ask for, and the container stops being addressable and becomes a package you open whole before touching anything.

And what STORE costs, measured rather than assumed. It is not free, and the usual justification — "the payloads are already compressed" — is only half true here. On the example container the entries total 183.2 MB stored against 37.3 MB deflated: the `.glb` is near its limit at 81 %, but the `.ply` layers, which are raw `float32`, fall to 13–15 %. The container is therefore about five times the size a compressed archive would be, and a writer should know that before quoting a transfer time. The trade is still taken, because **the right place to compress is inside the asset, not in the container**. A Gaussian layer stored as SPZ, or a mesh carrying Draco, shrinks by the same order *and* stays addressable by range; deflating the ZIP buys the same bytes and gives up the one property the container exists to provide. Compression belongs to the payload's own format, where a reader that understands the payload can undo it, rather than to the envelope, where undoing it is a precondition for reading anything at all.

The precedent, and how far it goes. `.usdz` made the same trade: it is a ZIP whose entries are stored uncompressed, so a runtime can read a texture in place instead of unpacking the archive first. It goes one step further and *aligns* each entry, which is what makes the bytes mappable into memory without a copy. UOS does not align: an HTTP range request has no such requirement, and the property cannot be granted retroactively — alignment moves offsets, so containers written today would not acquire it when a later version asked for it. A future revision that wants zero-copy local reads has to add the rule and accept that it splits the format.

Rationale: why the manifest goes first. It is the format's positive identification. A reader opens the first bytes, sees the entry and its `uos_version` field, and knows what it is holding without trusting the file extension or a magic number we would have had to invent.

2.2 Media type and identification

- **Media type:** `application/vnd.histora.uos` — **proposed**. The `vnd.` tree is to be registered with IANA when this specification is published. Until then a reader **MUST NOT** rely on it and **MUST** identify by rule 2 above.
- **Extension:** `.uos`
- **Magic:** None of its own. `PK\x03\x04` followed by the local file header of `manifest.json`.
- **Fragment identifier:** `caso.uos#view=<view.id>` selects a stored view (§10). Readers that do not implement views **MUST** ignore the fragment.

2.3 Reserved paths

The following paths are reserved by this specification. A writer **MUST NOT** place unrelated content at them; a reader **MAY** rely on their meaning.

Path	Required	Contents
`manifest.json`	**MUST**, first entry	§4
`scene/`	—	Mesh scene, Gaussian layers, their descriptors (§5)
`scene/scene.glb`	by convention	The glTF binary scene
`scene/*.gs.json`	—	Gaussian layer descriptors, extension `ash_gs_measured`
`volume/<id>.volume.json`	—	The volume sidecar — dimensions, spacing, orientation, encoding (§6). The DICOM series it describes is referenced, not carried, so this file is normally the only thing under `volume/`
`clinical/observations.json`	—	Per-tooth clinical layer, extension `ash_clinical` (§8)
`derived/`	—	**Everything** produced by inference (§9)
`views.json`	—	Stored presentation states (§10)
`provenance/chain.json`	—	Version chain (§11)
`provenance/signatures/`	—	Reserved for Ed25519 signatures; **not specified in v0.2**

A referenced original occupies no path at all. Acquired originals are named by content address and never by location (§3.4.3), so no reserved path corresponds to one. There is no `images/` holding the photographs and no `volume/ct_001/` holding the slices: those assets are in the manifest, addressed as `sha256:<hex>`, and nowhere in the ZIP. That is the property worth protecting. A path would have to be either the local one — and the local path of a clinical case carries the patient's own directory name — or an invented logical one, which would be a promise about a location inside an archive the file is not in. A hash is neither: it says which file it is and nothing about where anyone keeps it.

```
manifest.json                <- first entry, always
scene/scene.glb              <- glTF scene: mesh + appearance splats
scene/field.ply              <- Gaussian field of the CBCT (measured density)
scene/field.gs.json          <- its column descriptor
scene/composite.ply          <- scanner + CBCT in one frame
scene/composite.gs.json
scene/appearance.gs.json     <- descriptor of the splats embedded in scene.glb
views.json                   <- 18 stored camera states
clinical/observations.json   <- per-tooth findings, keyed by FDI
derived/seg_teeth.bin        <- int16 FDI label per vertex  (layer 3)
derived/seg_teeth.meta.json  <- which model produced it     (layer 3)
provenance/chain.json        <- 3 versions of this case
```

Listing 1: Directory layout of the example container

3 Data model

3.1 The three planes

Everything in a `.uos` belongs to exactly one of three planes, and the separation is deliberate:

Plane	Where	What it is, and what may be done to it
Data	`assets[]`	Acquired measurements, and the representations UOS composes from them — each declaring its source in `derived_from` and, where the conversion loses something, saying so (§5). Regulatory layer 1. Immutable: a writer never edits an asset in place (§11).
Presentation	`views.json`	Camera states, layer visibility, clipping. Layer 1, disposable, versionable, cheap to rewrite.
Inference	`derived/`	Anything a model produced. Regulatory layer 3. **Removable**: deleting the directory and its manifest entries leaves a valid, complete `.uos`.

Rationale. The third plane is what makes the format shippable. Distributing the same case in a jurisdiction where the AI module is not cleared requires deleting a directory, not re-exporting anything. That property only holds if it is enforced — which is why §13 makes "layer 3 outside `derived/`" an error in both directions.

3.2 Frames and registrations

A **frame** is a named coordinate system. Every asset declares exactly one. One frame is the `canonical_frame`: the geometric hub of the case, conventionally the intraoral scanner's frame. A **registration** is a first-class object carrying the 4×4 homogeneous transform that maps points from `source_frame` into `target_frame`, together with *how it was obtained and how well it fits.*

Normative: frame connectivity. Treat `registrations[]` as undirected edges over the set of frames. Every frame named by any asset **MUST** be reachable from `canonical_frame`. A reader **MUST** report an error otherwise — an asset in a frame that cannot be related to the hub would be placed somewhere arbitrary, and nothing downstream could detect it.

Conventions within a container.

- Units are **millimetres**, and this one is checked: `canonical_frame.units` **MUST** be `"mm"`.
- Handedness is **right**. It is the default every frame carries and no emitter varies it, but **no validator enforces it** — a frame declaring otherwise is accepted and would mirror everything placed through it. Treat it as a convention a reader **SHOULD** check for itself until a later version makes it normative.
- `transform_4x4_row_major` is exactly 16 numbers in **row-major** order. glTF stores matrices column-major; a writer emitting this transform into a glTF node **MUST** transpose (§5.5). Getting this wrong does not crash — it places the cloud rotated and mirrored, which is worse.

3.3 Visits

`visits[]` names the temporal axis. Every asset and every view declares the visit it belongs to. Longitudinal comparison, the same case with a baseline and a follow-up in the same frame, is a matter of two visits and one shared registration, not a second study.

3.4 Identity: how anything in a `.uos` is named and verified

3.4.1 File assets

`sha256` is the SHA-256 of the exact bytes stored in the ZIP entry, lowercase hex, 64 characters. `bytes` is their length. A reader **MUST** verify both.

3.4.2 Directory assets

A **directory asset** is one whole multi-file series — in practice a DICOM study. What identifies it is that it carries a `parts[]` array, one entry per file, with a relative `name`, a `sha256` and `bytes`. Its `uri` follows the same rule as any other asset: a path ending in `/` if the series travels, a content address if it is referenced (§3.4.3), which for an acquired series is always the case.

`parts[]` travels even when the series does not. That is the point of splitting it out: the container cannot verify a series it does not hold, but whoever holds it can — file by file, against these hashes. Referencing gives up custody, not traceability.

The asset's own `sha256` is a **digest over the part names and hashes**, defined here because the format needs writer and validator to compute it identically:

```
def digest_of_parts(parts):
    h = hashlib.sha256()
    for p in sorted(parts, key=lambda x: x.name):     # sort by name, byte order
        h.update(f"{p.name}\0{p.sha256}\n".encode())  # NAME, NUL, HEX HASH, LF
    return h.hexdigest()
```

Rationale. It runs over names and hashes rather than concatenated bytes for two reasons: renaming a slice changes the digest — correct, because the ordering of a DICOM series is clinical data — and verifying it costs nothing beyond the manifest, without re-reading the series itself. Per-file hashes are the difference between reporting "this series does not match" and reporting "slice 214 is corrupt".

3.4.3 Referenced originals, and content addressing

Normative: acquired originals are referenced, not carried. No acquired original **MUST** travel inside a `.uos` — not the DICOM series, not the scanner STL, not the clinical photographs, not the reports. They are declared with `external: true` and named by content address. What a container carries is what UOS composes: the scene, the Gaussian layers, the views, the clinical layer, and anything under `derived/`. **This is the format, not a variant of it.** There is no "full" packaging to opt into. A container found to carry an acquired original is a writer defect, not a richer case: it is duplicated source data whose custody the format never claimed. The photographs are worth naming explicitly because they are the ones that slipped — eighteen megabytes of patient images once travelled inside a container that declared it carried no originals, because they were tied to a different flag from the STL and the DICOM.

An asset that is declared but not carried sets `external` to `true`, and its `uri` **MUST** be a **content address**, not a path:

```
{ "id": "asset.ios", "external": true,
  "uri":    "sha256:abca87e971dab2d6a49d60edfe7f73f95ce62410603df86e72373a8c874197cb",
  "sha256":        "abca87e971dab2d6a49d60edfe7f73f95ce62410603df86e72373a8c874197cb",
  "bytes": 11004334, "media_type": "model/stl", "frame": "frame.ios_master" }
```

Normative rules, checked in both directions:

- `external: true` ⇒ `uri` matches `^sha256:[0-9a-f]{64}$`.

- `uri` is a content address ⇒ `external` is `true` (otherwise the asset would be unlocatable inside the ZIP).
- The hash inside the `uri` **MUST** equal the `sha256` field.
- A reader **MUST** make plain, once per container, which originals are referenced rather than carried: the container declares what each file is and cannot prove it holds it. This is a statement about custody, not a defect to be flagged per asset — since every acquired original is referenced, a per-asset warning would fire on every container and distinguish nothing.

What is lost, and what replaces it. A referenced original is not in the container, so "the DICOM that comes out is byte-identical to the one that went in" is a claim this format does not make about a file it does not hold. What it does hold is enough to make two weaker, useful claims: **identity** — whoever has the file can prove it is the one this case was built from, slice by slice (§3.4.2) — and **reconstruction**: the `ash_reversible` extension regenerates an arch mesh from the scene itself (§12). Reversibility here means producing the mesh again from what travels, not handing back the file that was ingested. And what comes back is not the file that went in: the regenerated mesh carries per-vertex colour sampled from the patient's own photographs and an FDI code per vertex, neither of which the ingested STL ever had. It is exported as `.ply`, `.stl` or `.3mf` — the colour and the codes surviving in the two formats able to carry them — so the practical answer to "why not just keep the STL" is that the container hands back an improved one.

Rationale: why not a path. A file that does not travel has no location inside the container, so a path would be a promise about a ZIP it is not in. What remains true of it is *which file it is*, and that is exactly what a content address states. It also has a property a path does not: the local path of a clinical case contains the patient's directory name; a hash contains nothing. Referencing takes files out of the container, never identities.

4 manifest.json

The manifest is the contract. What is not declared here cannot be in a `.uos`, the point being that a reader can *refuse* rather than guess. The schema sets `additionalProperties: false` on every object: an unknown field is an error, and new information is added through the extension mechanism (§12), not by sprinkling keys.

4.1 Top level

Field	Type	Req.	Description
`uos_version`	string	default `"0.2"`	Format version. A reader **MUST** check it before anything else.
`case_id`	string	**MUST**	Stable case identifier. Conventionally `urn:uuid:<v5>`; **MUST NOT** encode patient identity.
`created`	date-time	—	RFC 3339, UTC.
`generator`	map str→str	**MUST**	Who wrote this container, e.g. `{"name": "...", "version": "0.4.0"}`.
`phi_state`	enum	**MUST**	§4.4.
`subject`	`Subject`	**MUST**	§4.4.

Field	Type	Req.	Description
`canonical_frame`	`Frame`	**MUST**	The geometric hub. `units` **MUST** be `"mm"`.
`frames`	`Frame[]`	—	Every *other* frame. The canonical one is not repeated here.
`visits`	`Visit[]`	—	Temporal axis.
`assets`	`Asset[]`	—	The data plane.
`registrations`	`Registration[]`	—	The relations. §3.2.
`fhir_map`	map str→`FhirResource`	—	Keyed by asset `id`, plus the reserved key `"case"` for the container itself. §4.5.
`extensions`	map str→`Extension`	—	§12.
`extensions_used`	string[]	—	§12.
`extensions_required`	string[]	—	§12.
`provenance`	`Provenance`	—	§11.

4.2 Asset

Field	Type	Req.	Description
`id`	string	**MUST**	Unique within the manifest. Convention `asset.<name>`; this is the key used by `fhir_map`, `derived_from` and sidecars.
`kind`	enum	**MUST**	`volume`, `mesh_gs_scene`, `image2d`, `signal`, `derived_seg`, `document`.
`visit`	string	**MUST**	A `visits[].id`.
`uri`	string	**MUST**	Internal path, or `/`-terminated directory, or a content address — which is what every acquired original carries (§3.4.3).
`media_type`	string	**MUST**	IANA type of the payload, e.g. `model/gltf-binary`, `application/dicom`, `model/stl`, `image/jpeg`. Declared for referenced assets too: it says what the file *is*, which stays true whether or not the container holds it.
`sha256`	hex(64)	**MUST**	§3.4.
`bytes`	int ≥ 0	**MUST**	For a directory asset, the sum over `parts`.
`frame`	string	**MUST**	Must be reachable from the canonical frame.
`acquisition`	object	—	`{time: date-time|null, device: map str->str}`.
`load_priority`	int	default 30	Progressive-loading hint; lower loads first. See table below.

Field	Type	Req.	Description
`regulatory`	`Regulatory`	—	`{layer: 1..3, status: str|null, jurisdictions: str[]}`. §9.
`projection`	`Projection` | null	—	2D imaging only. §7.
`parts`	`Part[]`	—	Directory assets only, carried or referenced. §3.4.2.
`sidecar_uri`	string | null	—	A description of the asset that spares the reader from parsing it. If present it **MUST** exist in the container — including when the asset it describes does not (§6).
`derived_from`	string[]	—	Asset ids this one was produced from. Purely additive: a reader that ignores it still opens the case.
`external`	bool	default `false`	§3.4.3. The default is `false` because it is a property of the envelope, not a recommendation: every acquired original sets it to `true`.

Default load priorities. `mesh_gs_scene` 10, `image2d` 20, `volume` 30, `signal` 30, `derived_seg` 40, `document` 50. The mesh scene comes first because it is what lets a viewer show something useful having fetched only the manifest and one asset. A writer **MAY** override a single asset's value — the example container gives its Gaussian field 25, ahead of the volume it was fitted to.

On `derived_from`, and why the field exists. `scene/scene.glb` is the scanner mesh re-indexed into glTF: the same 220,085 triangles as the STL. Without this field the manifest declared it as a free-standing `mesh_gs_scene` with no link to the `asset.ios` it converts, so a recipient had no way to know it was a conversion rather than an independent source. The information existed — inside the GLB, as `extras.uos_source_asset` — but recovering it meant parsing eighteen megabytes of glTF to learn something the manifest states in one line.

4.3 Registration

Field	Type	Req.	Description
`id`	string	**MUST**	e.g. `reg.ct_to_ios`.
`source_frame`	string	**MUST**	Points are mapped *from* here.
`target_frame`	string	**MUST**	. . . *to* here.
`transform_4x4_row_major`	number[16]	**MUST**	Row-major, mm, right-handed.
`method`	string	**MUST**	How it was computed. Free-form; observed values are `icp_surface`, `auto_dl`, `manual`, `fiducial`. It describes the algorithm, **not** whether a human stood behind it.
`rms_error_mm`	number | null	—	Residual of the fit. `null` means *not reported*, never *zero*.
`computed`	date-time | null	—	When.

Field	Type	Req.	Description
`operator`	string | null	—	Who or what produced it. A machine **MUST** write `auto:<agent>@<version>`; this prefix is what marks the registration as unattended.
`verified_by`	string | null	—	The human who reviewed it. `null` means **nobody has**.
`regulatory`	`Regulatory`	—	As for assets.

Normative: provisional registrations. A registration is **provisional** when it was produced without a human, `operator` beginning `auto:`, and `verified_by` is `null`. A validator **MUST** warn, and a viewer **MUST** present it as unverified. An automatic alignment nobody has looked at and one signed by a clinician are not the same claim, and the format refuses to let them look alike.

The rule keys on the operator, and it must. The reference implementation originally keyed it on `method == "auto_dl"`, and the consequence is instructive: the only registration it emits declares `method: "icp_surface"`, so the case in the example container, automatic, unverified, 0.666 mm of residual, *did not trip the rule it exists for*. A safeguard written against one algorithm's name stops working the moment someone uses a different algorithm, which is every time. Who computed it is the question; how, is a separate field.

```
{ "id": "reg.ct_to_ios", "source_frame": "frame.ct_001", "target_frame": "frame.ios_master",
  "transform_4x4_row_major": [0.99786, 0.02798, 0.05902, -89.3628,
                              -0.05533,-0.11815, 0.99145, -22.6927,
                               0.03471,-0.99260,-0.11635,  38.7773,
                               0.0,     0.0,     0.0,       1.0],
  "method": "icp_surface", "rms_error_mm": 0.666,
  "operator": "auto:geometric-fusion-agent@0.2.0", "verified_by": null }
```

4.4 PHI and subject

`phi_state` is an **explicit state**, one of `identified`, `pseudonymized`, `anonymized`, `quarantined`. Without the field, "I do not know whether this carries identifiers" and "it does not" would be indistinguishable, which is the worst way to handle PHI. Viewers **MUST** honour it in banners and export policy.

`subject` carries `pseudonym` (an opaque token; the reference implementation uses a truncated hash) and an optional `fhir_patient` reference. A patient name **MUST NOT** appear anywhere in a `.uos`.

The obligation this puts on a writer. `phi_state` is a claim about the *case*, and a case includes originals the container does not hold (§3.4.3). DICOM headers carry `PatientName`, `PatientBirthDate`, `InstitutionName` and friends; a writer declaring `pseudonymized` **MUST** have inspected those tags at ingestion rather than assumed someone else's work, and it is the writer's last chance to, because once the series is referenced the container cannot be re-inspected. Note that `PatientID` is deliberately *not* on that list: a correctly pseudonymised export populates it with an opaque token, and rejecting it would reject perfectly anonymous series.

Referencing does not weaken this and does not strengthen it either. It keeps identifiers out of *this file* — a content address carries no patient directory, no filename, no tags — but the tags still exist in the series someone else holds, and `phi_state` still speaks for them.

4.5 FHIR mapping

`fhir_map` tells a practice-management connector what to create for each asset.

Field	Type	Req.	Description
`resource_type`	string	**MUST**	An R4 resource *type*: `ImagingStudy`, `DocumentReference`, `Media`, `Observation`.
`resource`	string | null	—	A *reference* to a resource that exists on a server, e.g. `ImagingStudy/is-9911`. `null` until the case actually lives in a PMS.
`note`	string	—	Why this mapping and not another.

Rationale, type versus reference. They are not the same thing and conflating them is an active hazard: a connector reading a fabricated `resource` would try to resolve it. So a writer declares what it actually knows, the type, and leaves the reference null.

The honest gap. FHIR R4 has **no resource for dental geometry**, and `Media` is defined for photo, video and audio. Meshes and Gaussian layers therefore map to `DocumentReference`, the generic clinical-binary envelope, and the `note` field says so out loud rather than pretending the mapping is exact.

5 Scenes

This is the section a viewer implementer needs in full. "Scene" in UOS means three distinct, separable things, and conflating them is the most common way to misread a container.

First, what a Gaussian layer is here. A `.uos` carries Gaussian layers of **two kinds that must never be confused**, and the format spends an entire extension keeping them apart:

- A **measured** layer is a physical measurement expressed as Gaussians. The CBCT density field is one primitive per occupied voxel carrying the attenuation the scanner actually measured — `density` is normalised sigma, Beer–Lambert, in the patient's frame and in linear millimetres. It is clinical data that happens to be rendered by a splat rasteriser, not a rendering that happens to look like data.
- A **reconstructed** layer is appearance fitted by an optimiser: the trained colour of the arch. The colour **is the patient's own**, sampled from their photographs; what `measured: false` denies is narrower and specific — the optimiser moved, split and pruned the primitives, so no Gaussian corresponds one-to-one with the pixels its colour came from. Provenance of the colour and traceability of each primitive are different claims, and only the second is lost.

De-facto 3DGS knows only the second kind, and its column names — `density`, `scale`, `opacity` — carry that meaning by default. Reading a measured layer with those assumptions produces an image that looks right and means nothing. That is why every layer travels with a descriptor stating which kind it is (§5.6), and why `ash_gs_measured` exists at all: it is the part of this format the base draft has no vocabulary for.

And what `mesh_gs_scene` means. It is the asset `kind` (§4) for anything that is part of the renderable scene — the glTF binary and every Gaussian layer alike. One `kind` covers both because a viewer treats them as one thing to compose; what separates a mesh from a splat layer, and a measured layer from a reconstructed one, is said elsewhere and more precisely.

With that established, the three things:

Thing	Where	What it is
The scene graph	`scene/scene.glb`	A glTF 2.0 binary asset. The node hierarchy *is* the spatial composition: node 0 is the canonical frame and every other layer hangs off it with its registration baked into the node transform.
Embedded appearance	a primitive inside that GLB	A `KHR_gaussian_splatting` point primitive. Any conformant glTF loader renders it without knowing anything about UOS.
External Gaussian layers	`scene/*.ply` + `*.gs.json`	Layers too large or too semantically specific to embed — notably the *measured* CBCT density field. Referenced from a glTF node via `extras.uos_gs_uri`.

5.1 Why a glTF conversion exists at all

An STL is a triangle soup: no indices, no attributes, not one metadata field. Semantic picking (§5.3) is defined over per-sub-mesh metadata, and an STL cannot carry any. glTF can.

Normative: the conversion is lossy, so the original stays declared. glTF requires `float32` positions and the ingested mesh is `float64`. At ±100 mm that leaves roughly 10 nm of resolution, irrelevant for looking, not irrelevant for claiming reversibility. Therefore a container carrying `scene/scene.glb` **MUST** declare the source asset it was converted from — by content address, since acquired originals are referenced and not carried (§3.4.3) — and the GLB **SHOULD** name it in `extras.uos_source_asset` *and* in the manifest's `derived_from`. The converted scene is **presentation**; the original is the record, and the record is identified even where it is not held.

Measured reversibility on the reference case: the mesh regenerates from the container with 4.59×10^{-6} mm of error against a 0.1 mm budget — the residual is the `float32` of the STL itself.

5.2 The scene graph

```
scene 0
‘-- node 0  "scan"                        mesh 0   <- THE CANONICAL FRAME
    |-- node 1  "apariencia real..."      mesh 1   <- KHR_gaussian_splatting primitive
    |                                               (no matrix: trained in the canonical
    |                                                frame, so transform = identity)
    |-- node 2  "campo gaussiano..."      no mesh  <- external layer, extras.uos_gs_uri
    |                                               matrix = reg.ct_to_ios, COLUMN-major
    ‘-- node 3  "compuesto CBCT+escaner" no mesh  <- external layer
```

Normative rules for the graph:

1. `scenes[0].nodes` **MUST** contain exactly the root node, and that root node **MUST** carry the mesh geometry expressed in the canonical frame.
2. Every Gaussian layer, embedded or external, **MUST** be a **child** of the root node. Its node transform **MUST** be the registration that carries it into the canonical frame. The identity transform **MAY** be omitted.
3. Node `matrix` is **column-major** (glTF rule) while `transform_4x4_row_major` in the manifest is row-major. A writer **MUST** transpose; a reader converting the other way **MUST** transpose back.

4. A node transform on a splat node **MUST** decompose into translation, rotation and *positive* scale (a `KHR_gaussian_splatting` requirement). Registrations from rigid alignment satisfy this; a mirrored or degenerate matrix does not, and its rendering is undefined.
5. The manifest remains the auditable statement of the same relation. A reader **MAY** place layers using only the GLB — which is the point: an off-the-shelf `GLTFLoader` composes the case correctly with no UOS knowledge — but an auditor **MUST** read `registrations[]`, which is where the error and the verification status live.

5.2.1 Scene-level extras

```
"extras": { "uos_frame": "frame.ios_master", "uos_units": "mm",
            "uos_source_asset": "asset.ios",
            "uos_note": "presentation: float32 from float64. The original is asset.ios,
                         referenced by content address and not carried here; the mesh is
                         regenerated from this scene" }
```

5.3 Mesh primitives and semantic picking

The mesh is split into **one primitive per tooth**, plus one primitive holding everything else. Each tooth primitive carries its FDI code:

```
{ "attributes": {"POSITION": 0, "NORMAL": 1}, "indices": 16, "mode": 4,
  "extras": {"uos_fdi": "27"} }
```

Rule	Statement
Vertex order is invariant	All primitives **MUST** share *the same* `POSITION` and `NORMAL` accessors. Only the index buffer is partitioned. This is what allows `derived/seg_teeth.bin` to be a flat array indexed by vertex, joined by position and nothing else. A writer that re-orders vertices here breaks that join silently.
Face assignment	A face belongs to a tooth when **all three** of its vertices carry that label. Two would suffice to keep the border, but would drag in triangles crossing to the neighbouring tooth and the crown would come out with lateral flash.
The remainder	Faces belonging to no tooth — gingiva, and faces crossing between teeth — go into a primitive with *no* `uos_fdi`. They are neither redistributed (which would invent a boundary) nor discarded (which would leave holes a viewer would draw as perforations of the scan).
Code type	`uos_fdi` is a **string** (`"27"`, not `27`). An FDI code is not a quantity, and writing it as a number invites arithmetic on it.

The compromise this section encodes, stated plainly. The partition into primitives comes from a segmentation model — i.e. from layer 3 — and it is baked into a layer-1 scene. So deleting `derived/` no longer erases *every* trace of inference: it leaves the mesh split into fourteen pieces carrying their codes. What it does erase is the **per-vertex** label, which is the one with enough resolution to paint and to measure; what survives is primitive granularity, which is only enough to select. This is a trade, not a clean rule, and it is documented as one. It is made because picking is defined over `extras.uos_fdi`: without it a foreign viewer opens the container and cannot select a tooth, however faithfully the labels travel in `derived/`. A writer that needs the clean rule instead **MAY** emit the mesh unpartitioned and declare in the manifest that picking requires `derived/`.

5.4 Appearance: the `KHR_gaussian_splatting` primitive

Upstream status. `KHR_gaussian_splatting` is a Khronos *Release Candidate* at the time of writing, written against the glTF 2.0 spec. UOS v0.2 emits it and additionally keeps the external-layer fallback of §5.5. When the extension is ratified, UOS v1.0 retires that fallback.

The primitive **MUST** use `mode: 0` (POINTS) — each Gaussian is a point, and the ellipse is raised by the rasteriser from scale and rotation. The extension object carries two required properties:

```
"extensions": { "KHR_gaussian_splatting": { "kernel": "ellipse",
                                            "colorSpace": "srgb_rec709_display" } }
```

Property	Type	Req.	Description
`kernel`	string	**MUST**	`"ellipse"` is the only defined value.
`colorSpace`	string	**MUST**	`"srgb_rec709_display"` or `"lin_rec709_display"` (ASWF CIF naming). **No sensible default exists**; getting it wrong shifts the hue of the whole arch. UOS appearance layers trained against sRGB renders declare the former.
`projection`	string	default `"perspective"`	Splat rendering assumes a perspective camera.
`sortingMethod`	string	default `"cameraDistance"`	Back-to-front sorting.

5.4.1 Attributes

Semantic	Type	Req.	Meaning / units
`POSITION`	VEC3 f32	**MUST**	Gaussian centre, mm, in this node's frame. `min`/`max` are mandatory in glTF for this accessor.
`KHR_gaussian_splatting:ROTATION`	VEC4	**MUST**	Unit quaternion in **glTF order** (x, y, z, w).
`KHR_gaussian_splatting:SCALE`	VEC3	**MUST**	**Linear and non-negative**, mm.
`KHR_gaussian_splatting:OPACITY`	SCALAR	**MUST**	**Linear**, in $[0, 1]$.
`KHR_gaussian_splatting:SH_DEGREE_0_COEF_0`	VEC3 f32	**MUST**	DC term of the spherical-harmonic colour.
`KHR_gaussian_splatting:SH_DEGREE_`l`_COEF_`k	VEC3 f32	**MAY**	Degrees 1–3. If a degree is present, **all lower degrees MUST be present and complete** (degree 1 is exactly three VEC3 coefficients). Partial degrees are prohibited.

Semantic	Type	Req.	Meaning / units
`COLOR_0`	VEC3 /VEC4	**SHOULD**	Fallback diffuse colour for renderers without the extension. See below.
`NORMAL`	VEC3 f32	**MAY**	Standard glTF semantic; a point primitive may carry it. UOS uses the nearest mesh vertex's normal.
`_AO`	SCALAR f32	**MAY**	**Application attribute** (UOS). Ambient occlusion in $[0, 1]$, to be multiplied into the colour at draw time.
`_REGION_ID`	SCALAR i16	**MAY**	**Application attribute** (UOS). FDI code per Gaussian.

Reconstructing colour from SH. The zeroth-order coefficient is not a colour; it becomes one through the SH basis constant and the training bias:

$$\mathrm{RGB}_{\mathrm{diffuse}} \;=\; \mathrm{SH}_{0,0} \times 0.2820947917738781 \;+\; 0.5$$

A writer emitting the `COLOR_0` fallback **MUST** compute it as that expression clamped to $[0, 1]$; for an sRGB `colorSpace` it must be converted from sRGB to linear before storage. **UOS v0.2 does not emit `COLOR_0`**, which means a glTF viewer without the extension draws the appearance layer as an uncoloured point cloud. Writers **SHOULD** emit it; this is expected to become a **MUST** in v1.0.

Extent. Splats assume a 3σ cut-off (Mahalanobis distance 3). This is the same convention the `ash_reversible` extension uses when sampling colour per vertex, and it is cited rather than redefined.

Underscore attributes. `_AO` and `_REGION_ID` use the underscore prefix glTF reserves for what its specification does not define. A conformant viewer ignores them and draws correctly; the UOS viewer uses them for shading and for tooth selection. That they are non-standard is precisely why the `ash_gs_measured` descriptor still exists.

Normative: unit conversions happen *before* this primitive. An INRIA-convention 3DGS `.ply` stores opacity as a logit, scales as logarithms and the quaternion as (w, x, y, z). This extension requires linear opacity in $[0, 1]$, non-negative linear scale, and glTF quaternion order (x, y, z, w). A writer **MUST** apply sigmoid, exponential and reordering before emitting, and **SHOULD** validate:

- any negative `SCALE` means a logarithmic scale was never exponentiated;
- any `OPACITY` outside $[0, 1]$ means a logit never went through the sigmoid.

The reference implementation raises on both, because a container that ships either renders plausibly and wrongly.

5.5 External Gaussian layers

Layers that do not embed — the measured CBCT density field is 1,341,990 primitives and 78 MB — travel as separate entries inside the container and are referenced from a child node:

```
{ "name": "campo gaussiano del twin",
  "extras": { "uos_gs_uri": "scene/field.ply",
              "uos_descriptor_uri": "scene/field.gs.json",
              "uos_measured": true },
  "matrix": [ /* 16 numbers, COLUMN-major: the transpose of reg.ct_to_ios */ ] }
```

A reader that does not understand `uos_gs_uri` sees an empty node and renders the rest of the scene correctly. A reader that does **MUST** read the descriptor (§5.6) before interpreting any column, because the column names collide with de-facto 3DGS and *mean something else*.

5.6 The `.gs.json` layer descriptor

Declared by extension `ash_gs_measured`. One descriptor per Gaussian layer, column by column.

Field	Type	Description
`role`	string	Human-readable role of the layer.
`measured`	bool	**The load-bearing field.** `true` = the values are a physical measurement; `false` = they are a reconstruction.
`profile`	string	e.g. `ash-twin/1.0`, `ash-gs-apariencia/1.0`.
`frame`	string	The frame the coordinates are in.
`units`	string	`"mm"`.
`n_primitives`	int	Gaussian count.
`columns`	object[]	Per column: `name`, `unit`, `scale` (`lineal` | `log`), `measured`, `derived_from`, `meaning`, `vocabulary`.
`note`	string	What a consumer would otherwise get wrong.

A profile **MAY** add keys of its own — `ash-twin/1.0` carries `subsampling` with the voxel step and the counts before and after. A reader **MUST** ignore keys it does not know; the fields above are the ones every descriptor has.

Why this descriptor is not optional in practice. The CBCT field's `density` column is **normalised sigma — Beer–Lambert attenuation — and not opacity**, and its scales are in *linear millimetres, not logarithms*. Those column names are identical to de-facto 3DGS and mean something else. A consumer that assumes the usual convention will exponentiate a value that is already linear and render a physically meaningless image that looks fine.

Two real descriptors, abridged:

```
// scene/field.gs.json -- the CBCT density field
{ "role": "campo gaussiano del twin", "measured": true, "profile": "ash-twin/1.0",
  "frame": "frame.ct_001", "units": "mm", "n_primitives": 1341990,
  "note": "density MEASURED by the CBCT: ‘density‘ is normalised sigma, not opacity, and
           the scales are in linear millimetres, NOT logarithms",
  "subsampling": {"step_voxels": [3,3,1], "from": 12072785, "to": 1341990},
  "columns": [ {"name":"x","unit":"mm","scale":"lineal","measured":true,  ...},
               {"name":"scale_0","unit":"mm","scale":"lineal","measured":true, ...},
               {"name":"density","unit":"sigma_normalizada","scale":"lineal",
                "measured":true, ...},
               {"name":"region_id","unit":"","scale":"lineal","measured":false, ...} ] }

// scene/appearance.gs.json -- the trained appearance embedded in scene.glb
{ "role": "apariencia real entrenada con gsplat", "measured": false,
  "profile": "ash-gs-apariencia/1.0", "frame": "frame.ios_master", "n_primitives": 113827,
  "note": "f_dc_* = colour MEASURED PER TOOTH ... NOT measured=true: the optimiser moved,
           split and pruned the Gaussians, so there is no 1:1 correspondence with what was
           projected" }
```

5.7 What a viewer must do, in order

1. Read `manifest.json`; check `uos_version`.
2. Sort `assets` by `load_priority` and fetch in that order. Several assets share `kind: mesh_gs_scene`; the one to fetch first is the glTF binary, which the manifest gives the lowest priority and which alone is enough to show the case.
3. Load `scene/scene.glb` with any glTF 2.0 loader. Node 0 is already in the canonical frame; children are already positioned.
4. For each child node carrying `extras.uos_gs_uri`: fetch the descriptor, *then* the payload, *then* apply the node matrix. Skip the layer if the descriptor is missing — do not guess the column semantics.
5. Read `views.json`, apply the default or fragment-selected view (§10).
6. Read `registrations[]` and mark every provisional one in the UI.
7. If `derived/` is present and the deployment is permitted to show inference, load it *as a separate toggleable layer* labelled with its regulatory status.

6 Volume

A CBCT series is a **directory asset** (§3.4.2): `kind: volume`, one `parts[]` entry per slice with its own hash, and, being an acquired original, a content address for its `uri` (§3.4.3). The slices do not travel; `parts[]` and the sidecar do.

What `parts[]` guarantees, and what a validator can check. Per-slice hashes let **whoever holds the series** prove it is the one this case was built from, and prove it slice by slice, which is the difference between “this series does not match” and “slice 214 is corrupt”. The container itself cannot check a series it does not hold, and **MUST NOT** claim to. Where a series *is* carried (a container from another emitter, since ours never does) a validator **MUST** check it in both directions: a slice present in the ZIP that the manifest does not declare is as fatal as a declared slice that is missing. Either means the series coming out is not the one claimed to have gone in. The reference test suite builds such a container itself and removes a slice, adds a spurious one and alters one, expecting three distinct errors.

Slice names travel verbatim in `parts[]`. Renaming them to `IM0001.dcm` would change the digest and describe a series nobody has, and the ordering of a series is clinical data. Names produced by CBCT exports are equipment-generated (`3DSlice100.dcm`); a vendor emitting patient identifiers in filenames is an ingestion problem, not a container problem, by then the DICOM carries them in its tags anyway.

6.1 The volume sidecar

`sidecar_uri` points at `volume/<id>.volume.json`. It exists so that a web viewer does not need a complete DICOM parser (hundreds of files, transfer syntaxes, per-slice rescale) merely to learn what the volume is. It matters more, not less, now that the series is referenced: the sidecar is all a reader has until it resolves the series elsewhere. It **MUST** be read from the ingested bytes themselves, never from a description computed somewhere else, and it **MUST** travel even though the series does not.

Field	Type	Description
`frame`	string	The frame this volume is in. **Nothing else about alignment appears here** — the transform lives in `registrations[]`, and a system with two homes for one transform ends up with two different transforms.
`dimensions`	int[3]	Columns, rows, slice count.
`spacing_mm`	number[3]	(x, y, z); z from the measured inter-slice distance when available, which overrides the declared `SliceThickness`.
`orientation`	number[3][3]	Rows from `ImageOrientationPatient`, third row their cross product. **Read, never assumed** — a volume with an assumed orientation renders just as well, mirrored.
`origin_mm`	number[3] | null	`ImagePositionPatient` of the first slice; `null` with a warning if absent.
`rescale`	object	`{slope, intercept}`.
`value_range`	number[2] | null	Rescaled [min, max], from `Smallest/LargestImagePixelValue` when the series declares them. `null` **otherwise**, with a warning: sweeping every pixel of the series on each export is too expensive, and inventing a plausible CBCT range would feed a fabricated number straight into a viewer's windowing.
`pixel_encoding`	string	e.g. `int16-le`. Byte order comes from the transfer syntax, not from a convention — "almost all modern DICOM is little-endian" is not "all", and a viewer that assumes reads good-looking garbage.
`modality`	string	DICOM `Modality`.
`transfer_function_presets`	string[]	Named presets a viewer may offer: `cbct_bone`, `cbct_soft`, `cbct_metal_suppress`. These are names, not data.

7 2D imaging and documents

`kind: image2d` assets are clinical photographs and 2D radiographs. They are acquired originals, so they are referenced by content address and not carried (§3.4.3); what the manifest keeps is their identity, their `media_type` and the optional `projection` object, which says what kind of image it is and which teeth it targets:

```
"projection": { "type": "intraoral_frontal", "fdi_targets": ["11","12","13"] }
```

Normative: `fdi_targets` empty means *not recorded*. It does not mean *none*. A loose photograph from a clinic folder does not say which tooth it points at, and inferring it from the pixels is exactly the work the format declines to assume has been done. A consumer **MUST NOT** read the empty array as a negative assertion.

Camera `pose` is explicitly optional in v0.2 and the reference implementation does not emit it: computing it requires photo↔mesh fusion, which is measurable and does not converge cheaply without calibration.

`kind: document` covers PDFs, reports and any other file the case declares whose type has no better `kind` — including the referenced scanner STL, which is a mesh but not part of the renderable scene.

8 The clinical layer

`clinical/observations.json`, declared by extension `ash_clinical`.

Why it exists. v0.2 gives per-tooth clinical attributes no place: the FHIR mapping (§4.5) sends them to an `Observation` on an external server. The practical consequence is that a standalone `.uos` cannot answer "what does the report say about tooth 24", which is the question a clinician asks while looking at the model. So it is declared as an extension, carried as `kind: document`, mapped to `Observation` in `fhir_map`, and written down as *ours* so nobody mistakes it for ratified.

Why layer 1 and not `derived/`. What travels is the **transcription of a report a person signed**. That is clinical record. Putting it under `derived/` would make it removable, and removing `derived/` would then strip the case of what the report says, which is not what that operation means.

Normative: extraction provenance per value. Transcription is layer 1, but the *extraction* may well be inference, and every value declares which. `derivation` is `deterministic` (a pattern found it written), `inferred` (a model proposed it), or `null`. **`null` means not declared, not deterministic**, and a consumer **MUST NOT** treat an undeclared value as reproducible. Silence never gets promoted to an assertion.

```
{ "schema": "ash-clinical/1.0",
  "vocabulary": "ISO-3950 (FDI) for teeth; the finding vocabulary is closed",
  "regulatory": {"layer": 1, "note": "transcription of a report signed by a person"},
  "teeth": [
    { "fdi": "11", "findings": ["aparato_ortodoncico"], "confidence": 0.745,
      "agent": "report-agent@0.1.0", "derivation": "deterministic",
      "observed": "2026-08-17T15:39:38.921947+00:00",
      "n_roots": 1, "n_canals": 1,
      "color": { "space": "CIELAB",
                 "cervical": [74.89, 9.83, 24.06],
                 "middle":   [74.83, 6.70, 20.93],
                 "incisal":  [71.56, 6.77, 19.14],
                 "from_photo": "sha256:baeb31fa...", "n_pixels": 33508, "measured": true,
                 "illumination_slope": [0.649, 0.577, 0.677],
                 "note": "measured per tooth; NOT a certified shade-guide value; flash
                          falloff removed using the patient's own gingiva as reference, so
                          teeth ARE comparable to each other; the absolute level is not,
                          because the photograph carries no grey reference" } } ] }
```


Note the shape of that `note`: it states what the measurement supports (relative comparison between teeth) and what it does not (absolute shade), instead of shipping a number that looks like a shade match.

9 derived/ — the inference plane

The three hard rules

1. Everything a model produced lives **only** under `derived/`.
2. Every such asset declares `regulatory.layer: 3`, and *nothing outside* `derived/` may declare layer 3. A validator **MUST** report an error in both directions.
3. Every such asset has a sidecar declaring which model produced it, at which version, with which weights hash, over which source assets.

And the consequence that makes them worth enforcing: **a `.uos` with no `derived/` is still valid and complete**. Deleting the directory and its manifest entries is a supported operation, requiring no re-export.

9.1 Tooth segmentation

`derived/seg_teeth.bin`	FDI codes as `int16` little-endian, one per vertex of the source asset, in the order the scene declares them. `0` = unassigned.
`derived/seg_teeth.meta.json`	The sidecar below.


```
{ "model": {"name": "ash-seg-teeth", "version": null, "weights_sha256": "ea0060d5..."},
  "source_assets": ["asset.scene"],
  "regulatory": {"layer": 3, "status": "investigational", "jurisdictions": []},
  "generated": "2026-08-28T02:07:17.244507+00:00",
  "encoding": { "dtype": "int16-le", "count": 112067,
                "indexes": "one code per vertex of ‘asset.scene‘, in the same order the
                            scene declares them. 0 = unassigned.",
                "vocabulary": "ISO-3950 (FDI)" } }
```


Why not DICOM SEG or NRRD. Both are designed for segmentation over a grid. This one is not on a grid: it is a label **per primitive**, which is a third legitimate shape and the only one that loses nothing. Inventing a grid to fit a known format would resample the data to suit the wrapper.

`weights_sha256: null` is normative. It is `null` when unknown and **MUST NOT** be filled with the hash of something else. The field exists so the inference can be reproduced; a hash that is not the weights' turns "I don't know" into "I do" and is indistinguishable from outside.

10 views.json

A view is a stored presentation state: "what the clinician was looking at". Views are layer 1, cheap, versionable, and they are the natural mechanism for deep links (`caso.uos#view=view.pieza_24` — a **proposed** fragment syntax, since the media type that would define it is not registered and no viewer implements it yet) and for longitudinal comparison: the same view applied to `v1` and `v2`, side by side. The example container stores 18.

```
{ "views": [
  { "id": "view.oclusal", "label": "Oclusal", "visit": "v1",
    "camera": { "position": [-11.852,-179.860, 31.622],
                "target":   [ -3.013,   0.415,   3.980],
                "up":       [ -0.256,   0.159,   0.953], "fov": 35.0 },
    "layers": { "mesh": {"visible": true, "opacity": 1.0} },
    "mpr": null, "clip_planes": null } ] }
```

Field	Type	Req.	Description
`id`	string	**MUST**	Unique within the file. Conventions: `view.oclusal`, `view.frontal`, `view.vestibular_derecha`, `view.vestibular_izquierda`, `view.pieza_<FDI>`.
`label`	string	**MUST**	Display name.
`visit`	string	**MUST**	**Must** name a declared visit; a validator errors otherwise.
`camera`	object	**MUST**	`position`, `target`, `up` (3 numbers each, mm, canonical frame) and `fov` in degrees, default 35.
`layers`	map	—	Per layer `{visible: bool, opacity: 0..1}`. Known keys: `mesh`, `gs`, `volume`.
`mpr`	object | null	—	`{enabled, plane, window}`.
`clip_planes`	array | null	—	Clipping planes.

Normative: `mpr` and `clip_planes` are null when there is no volume. They are volume controls. In a `.uos` that carries no volume, emitting them — even empty — implies there is a plane to cut and a window to adjust. When a volume *is* present, omitting them is the opposite error: a viewer expects them.

Four conventions worth copying, each with its reason:

- The appearance layer defaults to `opacity: 0.85`, below 1.0, because it is reconstructed rather than measured and the viewer must let the mesh show through.
- The volume layer defaults to **off**. Composing translucent Gaussians with a translucent volume has no exact solution in two independent passes, so enabling it by default would store the view in the one state the format itself calls ill-defined.
- `fov` of 35°: narrower flattens the arch, wider distorts its edges. It is the range in which a molar does not read as a cylinder.

- `time_cursor_s` is defined by the format for signal timelines and is not emitted by a writer that produces no signals — a temporal cursor over a case with no time series points at nothing.

11 Provenance, versioning and PHI

11.1 A `.uos` is logically append-only

Modifying a case is **not** editing its `.uos`. It is writing a new version of the manifest that points at the hash of the previous one. Assets are never touched in place. That turns "this case has changed" into something the recipient can verify without trusting the sender and without holding the intermediate versions.

`provenance.prev_manifest_sha256`	**Where the authority lives.** The SHA-256 of the bytes of the previous version's `manifest.json`.
`provenance.chain`	Path to the materialised chain; **MUST** be `"provenance/chain.json"` when present.

```
{ "case_id": "urn:uuid:6a91c924-de59-5abc-94ec-8432c9f446ed",
  "links": [
    { "version": 1, "manifest_sha256": "f25b5a65...", "prev_manifest_sha256": null,
      "created": "2026-08-27T21:20:52.280181Z",
      "generator": {"name": "agentic-smart-health", "version": "0.4.0"},
      "assets": 19, "note": "18 view(s), 1 registration(s)" },
    { "version": 2, "manifest_sha256": "5f2e4a35...",
      "prev_manifest_sha256": "f25b5a65...", ... } ] }
```

What the chain does and does not guarantee. `chain.json` is a *materialisation*: it makes the history legible at a glance and walkable without opening every version. It adds no cryptographic guarantee — it is not covered by any hash in the manifest, and whoever rewrites the file breaks nothing cryptographic. What a validator does detect, and what actually matters, is the chain and the manifests **telling different stories**.

Hashing the manifest. The hash is over the **exact bytes written into the ZIP entry**. A writer therefore **MUST** serialise once and hash those bytes; re-serialising to compute the hash leaves it pointing at a second copy that merely resembles the first. The reference implementation emits canonical JSON (stable key order, `indent=1`, nulls retained, UTF-8).

Signatures. `provenance/signatures/` is reserved. **v0.2 does not specify signing.** What is missing is not code but a decision — which key signs (the issuing clinic, the platform, or both) and where it lives. Signing with an invented key would produce a `.uos` that *looks* signed, which is worse than one that declares it is not. A validator that encounters the directory **MUST** warn that the signatures are present and unverified rather than ignore them silently.

12 Extensions

UOS builds on glTF, which has had `extensionsUsed` / `extensionsRequired` since glTF 1.0 and carries them unchanged into 2.0: a reader opens the file, sees which extensions it carries, and knows whether it can read it fully, partly, or not at all. v0.2 of the draft did not inherit that mechanism at container level, and the practical consequence showed up immediately: our extensions lived in their own directories and **a foreign reader ignored them without noticing that it was**

ignoring them. That turns an open format into one only its author reads completely — exactly what UOS exists to avoid.

And that gap is what the rest of this section closes. With the mechanism below, a reader that does not implement `ash_clinical` still opens the case, still renders it, and can say precisely what it left unread. Being open was never the problem; being silent about what a reader was missing was.

Field	Type	Req.	Description
`name`	string	**MUST**	Registry key; also the map key in `extensions`.
`version`	string	**MUST**	Extension version, independent of `uos_version`.
`uri`	string | null	—	The asset that carries it, if any. If non-null it **MUST** be a declared asset `uri`.
`schema_id`	string | null	—	Identifier of its schema, e.g. `ash-clinical/1.0`.
`description`	string	—	What it adds and why the base format needed it.

Normative: `used` versus `required` is the whole mechanism.

- `extensions_used` — "this file contains this; ignore it if you must". A reader that does not know it **MUST** warn and **MUST** still open the case.
- `extensions_required` — "**do not open this file without understanding this**". A reader that does not implement it **MUST** refuse.
- Everything in `extensions_used` **MUST** be declared in `extensions`; everything in `extensions_required` **MUST** also appear in `extensions_used`. Violations are errors.
- An extension that only *adds* information belongs in `used` and **never** in `required` — otherwise a conformant viewer refuses a case whose geometry it could have shown perfectly.

Every extension the reference implementation emits is `used`; `required` is empty, deliberately.

12.1 Registered extensions

Name	Version	What it adds
`ash_clinical`	1.0	Per-tooth clinical attributes (pH, roots, canals, findings, measured colour) with per-value extraction provenance. §8. Carried at `clinical/observations.json`.
`ash_gs_measured`	1.0	The `.gs.json` descriptor per Gaussian layer: whether it is *measured* or reconstructed, and the schema of its columns. §5.6. The base format treats 3DGS as appearance in a reconstruction frame; here there are fields fitted to CBCT density, in the patient frame, with error in HU.
`ash_reversible`	1.0	Declares that an arch mesh with per-vertex colour, FDI code and a `measured` column can be *regenerated* from `asset.scene`: geometry and FDI from the mesh primitives (`extras.uos_fdi`), colour from the `KHR_gaussian_splatting` primitive, taking Gaussians that cover each vertex within 3σ and weighting by alpha. **The mesh does not travel: it is reconstructed.**

12.2 Defining your own

1. Pick a prefixed name (`<vendor>_<thing>`); do not squat on `ash_` or `KHR_`.
2. Put its payload at a path of your own, declared as an asset like any other, with its hash. Extension payloads are not exempt from verification.
3. Add an `Extension` entry and list it in `extensions_used`.
4. Put it in `extensions_required` **only** if a reader that ignores it would show the user something *wrong* — not merely something incomplete.
5. If it applies to inference output, it still obeys §9: under `derived/`, layer 3, with a model sidecar.

13 Conformance and validation

13.1 Conformance levels

Levels are derived from what a container *actually holds*, not from what it claims. A writer does not choose them.

Level	Requires asset kinds	Meaning
`UOS-Core`	`mesh_gs_scene`	Manifest + a renderable scene. The floor.
`UOS-Vol`	`+ volume`	Plus a DICOM volume.
`UOS-Sig`	`+ signal`	Plus time-series signals. *Not implemented in v0.2.*
`UOS-Full`	`mesh_gs_scene + volume + signal`	All of the above.

A container reports every level whose required set is a subset of the kinds present.

Levels count declared kinds, not carried bytes. An acquired original is referenced rather than carried (§3.4.3), and it is still a declared asset of its `kind`. A container whose CBCT is referenced therefore reports `UOS-Vol` exactly like one that somehow held the slices: the level says the case *has* a volume and that this container knows how it relates to everything else, which is what an implementer needs in order to decide whether it can open it. What a level does **MUST NOT** be read as is a custody claim. Custody is stated separately, per asset, by `external` — and a reader that conflates the two will promise a file it does not have.

13.2 The validation algorithm

A conforming validator performs these checks in this order. **E** = error, **W** = warning.

#	Cls	Check
1	E	First ZIP entry is `manifest.json`. If not, stop.
2	E	No entry uses a compression method other than STORE.
3	W	The manifest validates against the distributed JSON Schema. A failure here is a warning, not an error: what it detects is that our distributed schema has fallen behind the contract, which is our publication problem, not the file's.
4	E	The manifest parses against the contract (types, required fields, no unknown keys).

#	Cls	Check
5	E	For each non-external asset: the `uri` exists in the ZIP; its SHA-256 and byte length match.
6	W	Referenced originals: **one** line per container naming how many there are and which, not one warning per asset. Report the count separately so a caller can act on it without parsing prose.
7	E	For each directory asset: `parts` is non-empty; every declared part exists and matches; **no undeclared file exists under the prefix**; the parts digest (§3.4.2) equals the asset `sha256`; the byte sum matches.
8	E	Every `sidecar_uri` exists in the container.
9	E	Provenance: if `chain` is declared it equals `provenance/chain.json`, the entry exists, parses, its `case_id` matches, and its links are consistent with `prev_manifest_sha256` and with the hash of this manifest.
10	W	`prev_manifest_sha256` present but no `chain`: history exists but cannot be walked.
11	E	`provenance/chain.json` present in the ZIP but not declared: a chain nobody references cannot be verified.
12	W	`provenance/signatures/` present: n signatures that this validator does not check.
13	E	Views: each `visit` names a declared visit; ids are unique.
14	W	No `views.json`: no stored views.
15	E	Frame connectivity (§3.2).
16	W	Each provisional registration (§4).
17	E	Regulatory: every asset under `derived/` has layer 3, and every layer-3 asset is under `derived/`.
18	E	Extensions: `used` $\subseteq$ `extensions`; `required` $\subseteq$ `used`; every non-null extension `uri` is a declared asset.
19	W	Extensions declared but unused; and one warning per required extension, since a reader that does not implement it must not open the container.
20	E	`canonical_frame.units == "mm"`.
21	—	Report the conformance levels satisfied, the chain length, and the view count.

Why check 3 is only a warning. Validating against the distributed schema is not redundant with validating against the implementation's types. The implementation checks that a manifest fits *our* contract; the schema shipped alongside it is the only artifact a third party can check *theirs* against. Running both is what guarantees the schema we publish is the one we actually accept — and if they drift, this check says so before an outside implementer discovers it.

Reporting. A validator **SHOULD** report the chain length as `version` and 0 when no chain is declared. Zero means "there is no history to walk", not "this is the first version"; they are different facts.

14 Recipes

Python examples use the reference package. Every one of them has a language-agnostic equivalent stated alongside, because none of this requires that package.

14.1 Open a case and verify it

```
from pathlib import Path
from uos import lee_manifiesto, valida

m = lee_manifiesto(Path("case.uos"))       # errors unless manifest.json is entry #0
print(m.case_id, m.uos_version, m.phi_state)

report = valida(Path("case.uos"))          # the algorithm of section 12.3
print(report.valido, report.niveles)       # e.g. True [UOS-Core, UOS-Vol]
for e in report.errores: print("ERROR␣␣", e)
for w in report.avisos:  print("WARNING", w)
```

Without the package: open the ZIP, assert entry 0 is `manifest.json`, parse it, validate against the distributed JSON Schema, then walk the checks in §13.2. No step needs anything but a ZIP reader, a JSON parser and SHA-256.

14.2 Render it

Follow §5.7 exactly. The minimum viable viewer is: read the manifest, extract `scene/scene.glb`, hand it to a stock glTF loader, apply `views.json`[0]. That already gives correct geometry, correct appearance and the clinician's framing, and it is roughly thirty lines. Everything after that — volume, MPR, provisional badges, derived toggles — is additive.

14.3 Add an asset (e.g. a clinical photograph)

```
from uos.contenedor import asset_de, escribe_uos
from uos.manifiesto import Clase

nuevo = asset_de(                          # hashes and measures the real file
    Path("/data/photo_09.jpg"), "images/img_009.jpg",
    id_="asset.img_009", kind=Clase.IMAGE2D, visit="v1",
    frame="frame.ios_master", media_type="image/jpeg",
)
m.assets.append(nuevo)
m.fhir_map["asset.img_009"] = RecursoFHIR(
    resource_type="Media", note="clinical␣photograph;␣Media␣is␣the␣non-DICOM␣image␣resource")
```

Checklist for *any* new asset, in order:

1. Choose a `kind` and a path consistent with §2.3.
2. Hash the exact bytes you will store; set `sha256` and `bytes`.
3. Name a `frame` that is reachable from the canonical one — adding a new frame means also adding the `registrations[]` edge that connects it.
4. Name a declared `visit`.
5. If it came from a model, it goes under `derived/` with layer 3 and a sidecar, no exceptions (§9).
6. If it came from another asset, populate `derived_from`.
7. Map it in `fhir_map` if a connector should know what to create.
8. Then write a **new version**, per §14.6.

14.4 Add a view

Append to `views.json` with a unique `id`, a declared `visit`, camera vectors in millimetres in the canonical frame, and `mpr`/`clip_planes` null unless the container carries a volume. Views are cheap and rewriting them wholesale is fine.

14.5 Add a derived layer

1. Write the payload at `derived/<name>.<ext>` and a sidecar at `derived/<name>.meta.json` carrying `model` (name, version, `weights_sha256` — `null` if unknown, never a substitute hash), `source_assets`, `regulatory` (layer 3, `status`, `jurisdictions`), `generated`, and an `encoding` block that says **what the bytes index**.
2. Declare both as assets with `regulatory.layer: 3`.
3. Verify the removability invariant: delete `derived/` and its manifest entries, re-run the validator, and confirm the container is still valid. If it is not, you have put inference in the wrong plane.

14.6 Write version $N{+}1$

```
# 1. serialise ONCE; the hash must be of the bytes that go into the ZIP
texto = manifiesto.json_canonico()
sha   = sha256_texto(texto)

# 2. link it to the previous version
manifiesto.provenance.prev_manifest_sha256 = sha_anterior
manifiesto.provenance.chain = "provenance/chain.json"
cadena = cadena_anterior.continua(Eslabon(
    version=len(cadena_anterior.links) + 1, manifest_sha256=sha,
    prev_manifest_sha256=sha_anterior, created=datetime.now(UTC),
    generator=manifiesto.generator, assets=len(manifiesto.assets), note="added␣img_009"))

# 3. write atomically; pass the SAME serialisation you hashed
escribe_uos(Path("case-v4.uos"), manifiesto, ficheros,
            directorios=dirs, extras={"provenance/chain.json": cadena.json_canonico()},
            json_manifiesto=texto)
```

Rules a writer **MUST** honour here: assets are never edited in place; the file is written to a temporary path and renamed, so a half-written container never sits where someone could mistake it for the good one; and every `uri` in the manifest must have a file and every supplied file must be declared — a dangling reference is an error, not a gap.

14.7 Declare an acquired original

Every acquired original is declared this way; none is packed into the container (§3.4.3). Set `external: true`, put the content address `sha256:<hash>` in `uri`, and keep `sha256`, `bytes`, `media_type` and — for a series — the whole `parts[]` array. Referencing gives up *custody*, not traceability: whoever holds the series can still prove that no slice is missing.

MUST NOT: a local filesystem path in `uri`. That is what would leak exactly what the pseudonym exists to prevent.

14.8 Regenerate the improved mesh

This is what `ash_reversible` declares, and the practical answer to "why keep the container if I already have the STL". What comes back is not the file that went in: it is that geometry with a per-vertex colour measured from the patient's own photographs and a per-vertex FDI code, neither of which the ingested STL ever had.

1. Read `scene/scene.glb`. Take `POSITION` and the index buffers of the *mesh* primitives; the `extras.uos_fdi` of each primitive gives the code for the vertices its faces touch.
2. From the `KHR_gaussian_splatting` primitive take `POSITION`, `SH_DEGREE_0_COEF_0`, `OPACITY` and `SCALE`.
3. For each mesh vertex, take the Gaussians within 3σ — the same cut-off the extension defines (§5.4) — and average their colour weighted by *opacity* $\times$ *Gaussian falloff*. That is the colour *seen* at that point, which is not the colour of the nearest centre.

4. Record, per vertex, whether it received colour at all. Write it out as a `measured` column.

```
from export_agents.malla_mejorada import (
    color_desde_gaussianas, descriptor, escribe_ply, escribe_3mf, escribe_stl_viscam)

rgb, medido = color_desde_gaussianas(pos, centros, f_dc, opacidad, escalas)
meta = descriptor(pos, caras, rgb, medido, fdi,
                  origen="asset.scene", sha256_malla=sha, piezas_con_color=piezas)
escribe_ply(ruta.with_suffix(".ply"), pos, caras, rgb, fdi, meta["comentarios"], medido)
```

Normative: the colour MUST come from the Gaussian layer. Not from any intermediate painted mesh the producing pipeline happened to hold. That distinction is the whole point: a recipient with the container and nothing else — no repository, a year later — has to be able to obtain this. If the colour came from a file that does not travel, the Gaussian layer would be contributing nothing and the chain would not close.

The `measured` column is not decoration. Without it, "I have no colour here" and "I have the wrong colour here" look identical. On the reference case 108,922 of 112,067 vertices take colour from their own tooth; of the remainder, 97 inherit it from the nearest measured vertex and 3,041 are painted with a fallback gradient that is **not the patient's colour**. One byte per vertex turns that from a doubt into a question anyone can answer without opening the `.uos`.

Ambient occlusion is deliberately not applied. `_AO` is a visualisation factor. It has no place in a file somebody may print or take measurements on: darkening a crown because there is a fissure beside it would put a shadow of ours inside the patient's data.

Three formats, because none alone suffices — and each says what it loses. `.ply` carries per-vertex colour, the FDI code and the `measured` flag, and any mesh viewer opens it. `.3mf` is what slicers accept with colour and units inside. `.stl` with per-face colour is a non-standard convention almost nothing reads, emitted because there are laboratory workflows that accept nothing else — it is the most widely opened and the one that asserts least, and the descriptor says so rather than letting the loss pass for absence.

Whole arch, no root, no per-tooth files. Composing a scanner crown with a CBCT root joins two surfaces measured by different machines across a 0.666 mm registration, and the seam shows. The scanner STL has no root, so giving it one makes it a different object: reversibility asks for the same piece back, improved. Per-tooth files additionally depend on a segmentation that does not currently separate crowns reliably (§16).

14.9 Strip inference before distribution

Delete every entry under `derived/`, delete the corresponding `assets[]` entries, re-serialise, and write a new version with a note saying what was removed. Re-validate. The result is a valid, complete `.uos` at the same conformance levels.

15 Versioning and compatibility

`uos_version` is the first field a reader checks, and this section says what checking it buys. The format is versioned so that it can keep being developed without stranding the implementations already written against it.

15.1 What each kind of version change promises

Change	Guarantee
Minor (0.2 → 0.3)	A reader written for the earlier version **MUST** still be able to open the container. A minor **MAY** add optional fields, add extensions, and add values to an open vocabulary. It **MUST NOT** add a required field, change what an existing field means, or remove one.
Major (0.x → 1.0)	No compatibility is promised. A reader **MUST** check `uos_version` and refuse what it does not implement rather than guess.

15.2 How that survives a strict schema

The rule that makes minor compatibility real Each version publishes **its own** schema, and every schema sets `additionalProperties: false`. That strictness is deliberate and is *about that version*: within a version a reader must refuse rather than guess, which is the premise of §4. It is not a claim about later versions. So:

- Reading a container whose `uos_version` is **the reader's own**, an unknown field is an error.
- Reading a container whose `uos_version` is a **higher minor**, an unknown field **MUST** be ignored, and the reader **MUST** report which version it read as which. Validating it against the older schema is not a conformance failure of the container.

This is why check 3 of §13.2 is a warning and not an error: the contract decides validity, and a schema left behind is the publisher's problem, not the file's.

Reading a higher minor, in practice. A reader **MUST** branch on the declared version *before* validating anything: a container it will refuse is not one it should have parsed. The reference implementation does this in `uos.version`, and the lenient path asks the validator which fields it rejected rather than keeping a second, permissive copy of the model — a duplicated model drifts from the real one at the first field somebody adds, which is the same failure that makes this document's schema generated rather than hand-written.

Normative: reading a higher minor and re-emitting it are mutually exclusive. A reader that ignores fields it does not understand **MUST NOT** then write version $N{+}1$ of that case. It would drop those fields, and the provenance chain would still verify: the new version would be a cryptographically legitimate successor of the old one, with content silently missing. That is worse than refusing, because nothing downstream can detect it. Preserving unknown fields and emitting them back is the other way out, and it is more work than is needed while no later version exists. So the rule is the narrow one: read it, do not succeed it.

15.3 Deprecation

A field being withdrawn **MUST** spend one full version marked deprecated before it goes:

1. In the version that deprecates it, the field is still emitted, still valid, and the validator **MUST** warn that it is on its way out and say what replaces it.
2. In the next version it is removed from the schema.

One version of notice is what lets an implementer migrate on their own schedule instead of discovering the change when their reader breaks. The warning is also the written record of *why* the field went, which a diff of the schema does not carry.

15.4 What is versioned alongside the format

- The manifest schema: `uos-manifest-<version>.schema.json`, whose `$id` is pinned to a release tag so the identifier always names the same document (§1.5). A published tag **MUST NOT** be moved; a correction is a new one.
- Extensions, independently. `ash_clinical` at 1.0 does not become 1.1 because UOS did (§12).
- This document, whose version is on its title page.

16 What v0.2 deliberately does not do

- **It validates nothing clinically.** Everything inferred ships as `investigational`, on purpose.
- **Signals (`UOS-Sig`) are unimplemented.** The `kind` exists; no writer emits it and `time_cursor_s` in views is therefore never populated.
- **Signatures are unspecified.** §11.
- **`value_range` may be null.** Filling it with an invented number would be worse than leaving it empty — a viewer would window on it.
- **Camera pose for 2D imaging is not emitted.** §7.
- **`COLOR_0` is not emitted** on splat primitives, so non-extension viewers see an uncoloured point cloud. §5.4.
- **There is no distributable example container.** Every figure here was measured on a real patient's case, which the project holds and does not publish. A conformance fixture — synthetic, or built from an openly licensed dataset — is what would let an implementer check their reader against something other than their own output, and it does not exist yet.
- **The media type is unregistered.** `application/vnd.histora.uos` awaits an IANA registration in the `vnd.` tree, and `histora.dev` — the namespace it names — is not a domain anybody holds. The schema's `$id` therefore resolves through the reference repository, not through that namespace.
- **It is not a standard.** It is a container proposal with a schema and a validator; the schema's `$id` does not yet resolve to a published document.

A Agent cheat sheet

Enough to act on a `.uos` without reading the rest of this document.

Question	Answer
Is this file a `.uos`?	ZIP whose first entry is `manifest.json` containing `uos_version`.
Where is everything?	`manifest.json` → `assets[]`. Nothing is discovered by walking the ZIP; the manifest is exhaustive.
Can I trust a payload?	Re-hash it. `sha256` + `bytes` on every asset; `parts[]` for directories.
Where is the geometry?	`scene/scene.glb`, stock glTF 2.0. Node 0 = canonical frame.
Where is the colour?	The `KHR_gaussian_splatting` primitive in that GLB. RGB = $SH_0 \times 0.282094791 + 0.5$.
Which tooth is this?	Mesh primitive `extras.uos_fdi` (string, FDI/ISO-3950), or `_REGION_ID` per Gaussian, or `derived/seg_teeth.bin` per vertex.
Where is the CBCT?	Directory asset under `volume/`, `kind: volume`, with a `.volume.json` sidecar so you need no DICOM parser to inspect it.
How do two frames relate?	`registrations[]`, row-major, mm. Transpose for glTF.

Question	Answer
Is this alignment trustworthy?	`rms_error_mm` + `verified_by`. `null` means nobody checked.
What did a model produce?	Everything under `derived/`, and only that. Layer 3.
What does the report say about tooth 24?	`clinical/observations.json`, `teeth[].fdi == "24"`.
Has this case changed?	`provenance.prev_manifest_sha256` and `provenance/chain.json`.
Can I add data?	Yes: new asset + hash + frame + visit, then a **new version**. Never edit in place.
Can I remove the AI output?	Yes: delete `derived/` and its manifest entries. The container stays valid.
What must I never do?	Invent a hash; put a local path in an external `uri`; put a patient name anywhere; re-order scene vertices; treat an empty `fdi_targets` as "no teeth"; treat `derivation: null` as deterministic.

B Measured figures

Every number in this document comes from a real clinical case — a maxillary arch, an intraoral scan, a CBCT, nine photographs and three reports — not from an example built to illustrate it. They are grouped by what each one is evidence *for*.

What the example container holds

These are the figures a reader will meet on opening `another_patient.uos`, and they describe a container emitted exactly as §3.4.3 requires: the acquired originals are referenced, everything UOS composes is inside.

Item	Value
ZIP entries	12
Stored views	18 (four anatomical, the rest one per annotated tooth)
Declared extensions	3, all in `extensions_used`, none required
Provenance chain	3 versions
Mesh	112,067 vertices, 220,085 triangles, 15 primitives (14 teeth + remainder)
Appearance layer	113,827 Gaussians, SH degree 1, `srgb_rec709_display`
CBCT density field	1,341,990 Gaussians, subsampled $3 \times 3 \times 1$ from 12,072,785

What has been measured, and what it supports

Measurement	Value	The claim it backs
CBCT→scanner registration	`icp_surface`, RMS 0.666 mm, `verified_by` null	That a registration carries its own error and its own verification status (§4) — this one is provisional and a viewer must say so.
Mesh reversibility	4.59×10^{-6} mm against a 0.1 mm budget	That reversibility means regenerating the mesh from what travels, not returning the ingested file (§3.4.3). The residual is the `float32` of the STL itself.
Per-slice verification at scale	397 slices, 419 entries, 0 errors, 0 warnings	That §3.4.2 works on a full series: a missing, altered or extra slice is caught individually. Measured on a container written before originals stopped travelling (§3.4.3); the machinery it exercised is the same one that now emits `parts[]` for a referenced series, so that whoever holds the CBCT can prove no slice is missing.
Closed arch export	watertight, 12,025 closing faces, 11,936 mm^3, 3.2 s	That the composed scene yields something the scan alone does not: a printable solid, and a per-tooth file with a scanner-measured crown and a CBCT-reconstructed root.

C References

Every document this specification depends on or cites normatively. Web resources were consulted on **2026-08**; where a link points at a branch rather than a tag, its content may have moved since.

Normative dependencies

A reader cannot implement UOS without these.

- Khronos Group, *glTF 2.0 Specification.* `https://registry.khronos.org/glTF/specs/2.0/glTF-2.0.html`
- Khronos Group, *`KHR_gaussian_splatting`* — **Release Candidate**, not yet ratified; ratification was announced for Q2 2026. Cited here as it stood on 2026-08-28. `https://github.com/KhronosGroup/glTF/tree/main/extensions/2.0/Khronos/KHR_gaussian_splatting`
- ISO/IEC 21320-1:2015, *Document Container File — Part 1: Core* — the ZIP profile §2 requires.
- *JSON Schema*, draft 2020-12 — the manifest schema's dialect (§1.5). `https://json-schema.org/draft/2020-12/schema`
- RFC 2119, *Key words for use in RFCs to Indicate Requirement Levels*, Bradner, March 1997 — the conformance language of §1.

Referenced vocabularies and formats

Needed to interpret what a container carries, not to open one.

- NEMA, *DICOM Standard*: PS3.10 (media storage and file format) and PS3.15 Annex E.1 (attribute confidentiality profiles) — §6, §4.4.
- HL7, *FHIR Release 4* (4.0.1), 2019: `ImagingStudy`, `DocumentReference`, `Media`, `Observation` — §4.5.
- ISO 3950:2016, *Dentistry — Designation system for teeth and areas of the oral cavity* — the FDI two-digit notation used throughout.
- RFC 3339, *Date and Time on the Internet: Timestamps*, Klyne and Newman, July 2002 — the format of every timestamp in the manifest.
- RFC 6838, *Media Type Specifications and Registration Procedures*, Freed, Klensin and Hansen, January 2013 — the registration template §2 follows.
- ASWF Color Interoperability Forum — the naming behind `srgb_rec709_display` and `lin_rec709_display` (§5.4).

Precedents and recommended payload formats

Cited as prior art or as the compression this specification points a writer towards (§2); neither is required.

- Apple, *USDZ File Format Specification* — the uncompressed-ZIP precedent, including the alignment rule UOS does *not* adopt. `https://openusd.org/release/spec_usdz.html`
- Niantic Labs, *SPZ* — a compressed interchange format for Gaussian splats. `https://github.com/nianticlabs/spz`
- Khronos Group, *`KHR_draco_mesh_compression`*. `https://github.com/KhronosGroup/glTF/tree/main/extensions/2.0/Khronos/KHR_draco_mesh_compression`
- Kerbl, Kopanas, Leimkühler and Drettakis, *3D Gaussian Splatting for Real-Time Radiance Field Rendering*, ACM Transactions on Graphics 42(4), SIGGRAPH 2023 — the INRIA `.ply` conventions this format converts away from (§5.4).