

FiberQ Interchange Format v1

Status: stable. Version 1.0 is frozen. Anything added to it will be a 1.x minor version, which a 1.0 reader must still read — recognising what it knows and carrying the rest through (§9). A change that a 1.0 reader could not handle would be version 2.

A tool-neutral, open format for exchanging a complete fibre-access-network design — geometry, attributes, identity, and the relationships between elements — between the FiberQ QGIS plugin and any other conformant tool, without losing anything on the way.

- **Format identifier:** fiberq-interchange
- **This revision:** 1.0
- **Container:** GeoPackage (OGC GeoPackage 1.3), with a GeoJSON profile for the geometry-only subset
- **Licence:** this specification is published under CC-BY-4.0 so it can be implemented in any tool, open or closed, without licence conflict. The reference implementation inside the FiberQ plugin remains GPL-3.0-or-later.

Related: [field mapping](#) · [schema reference](#) · [validation rules](#) · [project versioning](#)

1. Why this exists

A fibre design is not a pile of lines and points. It is a set of **typed objects with relationships**: this cable enters that closure, fibre 7 of tube 2 is spliced to fibre 3 of tube 1, this slack loop belongs to that cable, this duct carries those cables.

Generic GIS formats carry the geometry and lose the rest. Exchanging designs through plain Shapefile or a hand-rolled GeoPackage means the receiving tool gets a drawing and has to guess the engineering. Every exchange loses a little more.

This format exists so that a design can move between tools and come back **unchanged** — including the parts the receiving tool does not understand.

2. The four rules

Everything below follows from these. A conformant implementation must honour all four.

1. **Preserve, don't discard.** A reader that encounters an element type, an attribute, or a relation it does not understand **must carry it through unchanged** on the next write. Silent loss is a conformance failure, not a limitation.
2. **Identity is permanent.** Every object carries a `fiberq_uuid`. It is assigned once, at creation, and **never regenerated** — not on import, not on export, not on reprojection. It is the only thing that makes a round trip verifiable.
3. **The format is the contract, not any one tool's schema.** Tools map to and from this canonical middle; they never map directly to each other. Adding a second tool must not require changing the first.
4. **Versioned and self-describing.** A bundle declares its format version and what wrote it. A reader that meets a newer minor version must still read what it recognises and pass through the rest.

3. Container layout

A bundle is a single `.gpkg` file containing three kinds of table.

design.gpkg		
— <feature layers>	one per element type, canonical names, OGC-standard	
— — Route		
— — Underground cables		
— — Poles		
— — ...		
— _fiberq_metadata	key/value; bundle-level facts	(\$7)
— fq_*	relational side-car tables	(\$6)
— — fq_splice_point		
— — fq_fiber_connection		
— — fq_relation		
— — fq_relation_member		
— — fq_container		
— — fq_container_slot		
— — fq_slot_assignment		
— fq_extension	the passthrough store	(\$8)

Feature layers are ordinary GeoPackage vector layers: any GIS tool can open the file and see the network. The `fq_*` tables are ordinary attribute tables: any SQLite client can read them. **Nothing in this format requires a FiberQ-aware tool to be useful** — that is deliberate.

GeoJSON profile

For exchanges that only need geometry and attributes (web maps, quick handover), a bundle may instead be a directory of GeoJSON files, one per element type, plus `_fiberq_metadata.json`.

Each file is named for the element type it holds — `pole.geojson`, `cable.underground.geojson`, and `<type>.<placement>.geojson` where placement is set, so `otb.indoor.geojson`. Every feature still carries `fq_type` and `placement`, so nothing depends on the filename.

Feature properties carry the canonical fields and nothing else. In particular a writer must **not** publish the container's row id (a `GeoPackage` `fid`, a shapefile record number) as a property: identity is `fiberq_uuid` and only `fiberq_uuid` (§4), and a second identifier-shaped field beside it invites a receiver to join on the one that does not survive a re-export.

```
bundle/  
├─ _fiberq_metadata.json  
├─ cable.underground.geojson  
├─ pole.geojson  
└─ otb.indoor.geojson
```

The relational side-car is **not** representable in the GeoJSON profile; a writer emitting GeoJSON while holding side-car data must say so (`profile = "geojson-lite"`) so the receiver knows the bundle is lossy by construction. A GeoJSON bundle with no side-car data to lose is complete, and omits the key — marking a complete bundle lossy misleads a reader as surely as not marking a lossy one.

4. Identity

`fiberq_uuid` — a string of at most 64 characters, unique across the whole bundle, present on every feature and on every side-car row that represents a real object.

Rules:

- assigned **once**, when the object is first created, by whichever tool created it;
- **never** regenerated — a tool that reassigns identities breaks every relation in the bundle and fails conformance;
- splitting one object into two produces **two new** identities, and the original is gone. Merging produces one new identity;
- relations reference objects **by fiberq_uuid only**, never by row id, feature id, or layer name — those are local to a file and do not survive a round trip.

Validation rule **B4** enforces presence and uniqueness inside the plugin.

5. Coordinates and measurement

- **Storage CRS is EPSG:4326.** All geometry in a bundle is WGS84 longitude/latitude. One storage CRS means a bundle is unambiguous without reading metadata first.
- **crs_epsg declares the restore target** — the CRS the design was authored in. A reader should reproject to it on import so the user gets their working CRS back, and a writer should record it.
- **Lengths are ellipsoidal metres**, measured on the ellipsoid, never in map units. A projected length is a property of a map, not of the ground, and does not survive reprojection. See the [validation rules](#) rule **D3**.
- Bearings, offsets and any other derived geometry values are likewise stored in real-world units, not map units.

6. Element types and the relational side-car

6.1 Element type codes

Every feature carries `fq_type` — a stable, untranslated code identifying what it is. The layer it happens to sit in is a presentation detail; `fq_type` is the contract.

Codes are lowercase, dot-free, and stable forever once published:

<code>fq_type</code>	Geometry	Meaning
<code>route</code>	LineString	Trench/span route the plant follows
<code>cable.aerial</code>	LineString	Aerial optical cable
<code>cable.underground</code>	LineString	Underground optical cable
<code>duct.pe</code>	LineString	PE duct
<code>duct.transition</code>	LineString	Transition duct
<code>pole</code>	Point	Pole
<code>manhole</code>	Point	Manhole / chamber
<code>closure.joint</code>	Point	Joint closure
<code>odf</code>	Point	Optical distribution frame
<code>otb</code>	Point	Optical termination box
<code>to</code>	Point	Termination outlet
<code>tb</code>	Point	Termination box
<code>patch_panel</code>	Point	Patch panel
<code>slack</code>	Point	Optical slack loop
<code>fiber_break</code>	Point	Recorded fibre break
<code>service_area</code>	Polygon	Service area
<code>building</code>	Polygon	Building / object

Placement is an attribute, not a type. Where an element is mounted — indoor, outdoor, on a pole, inside a joint closure — travels in a `placement` attribute (`indoor` | `outdoor` | `pole` | `closure` | `unset`), not in the type code. A tool that does not care about placement still understands the element; a tool that keeps separate layers per placement restores them exactly from `fq_type + placement`.

`otb` and `to` are **distinct types**. An optical termination box and a termination outlet are different pieces of equipment, and collapsing one into the other loses information that cannot be recovered.

A tool encountering an unknown `fq_type` **must not** reclassify it to something it does recognise. Mapping an unknown point to the nearest familiar type destroys information irreversibly and is the most common way an exchange goes wrong. Carry it through unchanged (§8).

6.2 Side-car tables

These carry what is not geometry. All references are `fiberq_uuid` strings.

`fq_splice_point` — an element's splice capability. `uuid`, `element_uuid`, `port_count`, `used_ports`, `config_json`, `notes`

`fq_fiber_connection` — one fibre spliced or connected to another. This is the table that makes the format worth having. `uuid`, `source_element_uuid`, `source_cable_uuid`, `source_tube`, `source_fiber`, `source_port`, `source_side`, `dest_element_uuid`, `dest_cable_uuid`, `dest_tube`, `dest_fiber`, `dest_port`, `dest_side`, `connection_type`, `loss_db`, `tray_ref`, `notes`

Tube and fibre numbers are **1-based**.

Fibre colour has one normative source and one optional cache. The colour is derived from the bundle's `color_standard` (§7) plus the tube/fibre index — that is the authority, and it is what makes a bundle internally consistent. A writer may additionally record `source_color_hex` / `dest_color_hex` as an **advisory** cache for readers that do not carry the standard's tables. Where the two disagree, the derived value wins, and a reader may report the mismatch. Never treat the cached hex as authoritative.

`fq_relation` + `fq_relation_member` — named groupings of objects: an optical path, a feeder, a named route. `fq_relation(uuid, name, category)` and `fq_relation_member(relation_uuid, member_uuid, order_index, role)`.

`fq_container`, `fq_container_slot`, `fq_slot_assignment` — the generic model for "a thing with numbered positions that other things occupy": duct holes in a manhole wall, ports on a panel, trays in a closure. `fq_container(uuid, host_uuid, host_kind, kind, name, geometry_json)`, `fq_container_slot(uuid, container_uuid, index, label, kind, spec_json, status)`, `fq_slot_assignment(slot_uuid, occupant_uuid, direction, notes, extra_json)`

Containers nest. `host_uuid` may reference either a feature or **another container**, with `host_kind` (`feature` | `container`) saying which. A splice tray inside a joint closure is a container hosted by a container; its positions are ordinary slots. This gives arbitrary nesting depth with no extra tables and no special cases.

Generic on purpose: modelling duct holes, tray positions and panel ports as three different table sets would mean a new table set for every tool's next idea.

6.3 Field names and values

Feature attributes use **canonical English field names and canonical values**, not any one tool's stored schema. A bundle whose columns were named in one tool's database language would be that tool's schema in a GeoPackage, and every other implementer would have to learn it — which is mapping directly to a tool rather than to the canonical middle, and is what rule 3 exists to prevent.

The full table — canonical field ↔ FiberQ's stored field, with type, units, value domains and the rule for each mismatch case — is published separately as the [field mapping](#). It is generated from the plugin's schema, so it cannot fall behind the implementation.

Two rules govern it:

- **fiberq_uuid is never renamed.** It is the join key every conformant tool already knows (§4).
- **A value outside a published domain is carried through unchanged.** Tidying a vocabulary is not worth discarding what the user recorded.

An attribute with no canonical column travels in the feature's own `fq_extra_json` rather than being dropped (§8).

6.4 Ordered traversal

`fq_path_stop` — the ordered list of elements a linear object passes through. `cable_uuid`, `element_uuid`, `order_index`, `is_latent`

A cable's geometry says where it goes; `fq_path_stop` says what it visits, in order, including elements it merely passes through without terminating at.

7. Bundle metadata

The `_fiberq_metadata` table is key TEXT PRIMARY KEY, value TEXT.

Key	Required	Meaning
<code>format</code>	yes	<code>fiberq-interchange</code>
<code>format_version</code>	yes	e.g. 1.0 — the version of this specification
<code>schema_version</code>	yes	the FiberQ data-schema version of the payload
<code>produced_by</code>	yes	free text naming the writing tool and its version
<code>produced_at</code>	yes	ISO-8601 UTC timestamp
<code>crs_epsg</code>	yes	the authoring CRS to restore on import (§5)
<code>color_standard</code>	yes	fibre colour standard, e.g. TIA-598-C
<code>color_catalog_json</code>	no	full colour catalogue if non-standard
<code>profile</code>	no	<code>geojson-lite</code> when the side-car is absent by construction (§3)

Do not put a producing tool's release number in a key that means the data-schema version. `format_version` and `schema_version` describe the payload; `produced_by` describes the writer. Conflating them makes a bundle unreadable by version checks.

A writer **must not** delete metadata keys it did not write. Rebuilding this table from scratch on every export silently discards whatever another tool recorded. Merge; do not replace.

8. The passthrough store

`fq_extension` is how rule 1 is actually honoured.

`uuid` · `owner_uuid` · `kind` · `namespace` · `payload_json` · `produced_by`

When a reader meets something it has no model for — an unknown `fq_type`, an attribute with no column, a side-car table it does not implement — it writes the whole thing here verbatim, keyed to the object it belongs to, and emits it again untouched on the next write.

Consequences worth stating plainly:

- A design can pass through a tool that understands **none** of the advanced data and come out the other side complete.
- When a tool later gains real support for something it used to pass through, the mapping is upgraded and those objects start importing properly — **older bundles still work**, because nothing was ever thrown away.
- Per-feature attributes with no canonical column travel in the feature's own `fq_extra_json` column rather than here; `fq_extension` is for whole objects and whole tables.

9. Conformance

A **conformant reader** must: 1. read every feature layer and the metadata table; 2. preserve `fiberq_uuid` exactly; 3. store anything it does not understand in the passthrough store; 4. refuse, with a clear error, a bundle whose `format_version` has a **major** number it does not implement — rather than importing it partially.

A **conformant writer** must: 1. use the canonical field names and values of the published [field mapping](#) (§6.3); 2. emit all required metadata keys (§7); 3. merge, never replace, metadata keys written by other tools; 4. emit the passthrough store contents unchanged; 5. never regenerate an identity it did not create; 6. never publish a container row id as a feature attribute (§3).

A **conformant round trip** is the real test: import a bundle, change nothing, export it, and every object, attribute and relation is still present — including everything the tool does not support. The FiberQ plugin asserts exactly this in CI.

10. What v1 deliberately does not cover

- **Styling and symbology.** Presentation is a tool concern. A bundle describes the network, not how to draw it.
- **Rendering geometry for schematic views.** Derived, not source.
- **Access control, users, projects-as-workspaces.** Out of scope for a file format.
- **Product catalogues as normative content.** A bundle may carry a catalogue reference on a feature, and the resolved values alongside it, so a receiver without the catalogue still has the numbers it needs.

11. Decisions taken

These were open in the first draft and are now settled. Recorded because the reasoning matters more than the outcome for anyone implementing against this.

1. **Placement is an attribute, not a type code.** Separate types per mounting position multiply the vocabulary without adding meaning, and a tool that does not model placement should still recognise the element. `otb` and `to` stay distinct.
2. **Colour is derived, with an optional advisory cache.** Deriving from standard + index keeps a bundle from contradicting itself; the optional hex lets a reader without the colour tables still display something. Precedence is explicit.
3. **Containers nest through `host_uuid` + `host_kind`.** One field covers tray-inside-closure and every future variant, rather than a new table set per idea.
4. **The GeoJSON profile stays in v1**, specified and explicitly lossy. It is the right tool for geometry-only handover, and marking it `geojson-lite` means nobody loses a side-car by accident.

12. Implementation status

The specification is ahead of the plugin, deliberately. The plugin does not model fibre splicing, trays or duct occupancy today, and this format carries all of them — so bundles from tools that do model them pass through the plugin intact, and the day the plugin gains those features, **bundles written today already contain the data.**

That is rule 1 doing its job: a format that only carried what today's implementation understands would have to be redesigned every time an implementation grew.

When the plugin later gains real support for something it passes through today — fibre splicing, trays, a splitter layer — the mapping is upgraded and those objects start importing properly. **Bundles written today still work**, and the data is already in them, because nothing was ever discarded. That is the only upgrade path that does not ask users to re-export everything they have.

What the round trip actually asserts

`tests/test_interchange_roundtrip.py` imports a bundle deliberately **richer than the plugin** and exports it again, then checks that all of the following survived:

Carried	How
An element type with no FiberQ layer	Kept whole in the passthrough store, re-emitted unchanged — never reclassified
An attribute with no FiberQ column	Kept against the feature's identity, re-emitted into its <code>fq_extra_json</code>
<code>fq_splice_point</code> , <code>fq_fiber_connection</code>	Kept verbatim and written back as rows , so they are still queryable
A passthrough row from a third tool	Untouched
A <code>_fiberq_metadata</code> key FiberQ did not write	Carried through the project into the next bundle
Relations and path stops	Out as <code>fiberq_uuid</code> references, back as project data, out again

It runs on **QGIS 3 / Qt5** and **QGIS 4 / Qt6** in CI. Import is also idempotent: importing the same bundle twice is a no-op, because identity is permanent and a feature with a `fiberq_uuid` already in the project is the same feature, not a second one.

Reference implementation

In the FiberQ QGIS plugin: **Plugins → FiberQ → Export interchange bundle...** and **Import interchange bundle...**

Part	Status
Feature layers, canonical names, <code>fq_type</code> + <code>placement</code>	written
Canonical field names and values (§6.3)	written
Storage CRS (§5), <code>crs_epsg</code> restore target	written
Identity preserved, never regenerated (§4)	written
Metadata merged, foreign keys preserved (§7)	written
<code>fq_relation</code> / <code>fq_relation_member</code> from project relations	written
<code>fq_path_stop</code> from recorded pass-through elements	written
<code>fq_extension</code> passthrough store (§8)	written
GeoJSON profile (§3)	written
All other side-car tables	created, populated by other tools
Reading a bundle back, adapting it into FiberQ's own layers	written
Carrying through types, attributes and side-car tables the plugin does not model	written
Round trip asserted in CI, on both Qt generations	written

Two notes on what "written" means for the relational tables. The plugin stores relations and pass-through elements against `(layer_id, feature_id)` — identifiers local to one QGIS project, which are meaningless in any other tool and do not survive a round trip. The writer resolves them to `fiberq_uuid` on the way out. A reference it cannot resolve is **reported and left out**, never mapped to the nearest plausible object.

The other side-car tables are created empty. That is intentional: a reader finds the table whether or not this particular writer had anything to put in it, and a tool that does model splicing can write into a bundle the plugin produced.

Part of the FiberQ QGIS plugin, developed with support from the [NLnet](#) NGI Zero Commons Fund. The specification is CC-BY-4.0; comments and implementations from other tools are welcome via [GitHub issues](#).