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` (the text was corrected in FiberQ 1.6.0 and the format is unchanged, see §13)
- **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_path_stop        ordered traversal              (§6.4)
    └── 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, plus any column of the writer's own that has no canonical name, under that column's own name (§6.3). 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.

Inside the plugin, validation rule **B4** reports a feature whose `fiberq_uuid` is missing or shared with another. It does not block an export, so a bundle written from a project B4 flags carries the same fault.

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, may contain a dot or an underscore (`cable.aerial`, `patch_panel`), and are stable forever once published:

fq_type	Geometry	Meaning
route	LineString	Trench/span route the plant follows
cable.aerial	LineString	Aerial optical cable
cable.underground	LineString	Underground optical cable
duct.pe	LineString	PE duct
duct.transition	LineString	Transition duct
pole	Point	Pole
manhole	Point	Manhole / chamber
closure.joint	Point	Joint closure
odf	Point	Optical distribution frame
otb	Point	Optical termination box
to	Point	Termination outlet
tb	Point	Termination box
patch_panel	Point	Patch panel
slack	Point	Optical slack loop
fiber_break	Point	Recorded fibre break
service_area	Polygon	Service area
building	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`, `source_color_hex`, `dest_element_uuid`, `dest_cable_uuid`, `dest_tube`, `dest_fiber`, `dest_port`, `dest_side`, `dest_color_hex`, `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, slot_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.

A writer's own column with no canonical name is written as a column under its own name (the field mapping states the rule), so a reader must expect columns outside the canonical set. An attribute a reader has no column for travels in the feature's own `fq_extra_json` on its next write 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
`format`	yes	`fiberq-interchange`
`format_version`	yes	e.g. `1.0` — the version of **this specification**
`schema_version`	yes	the FiberQ data-schema version of the payload
`produced_by`	yes	free text naming the writing tool **and its version**
`produced_at`	yes	ISO-8601 UTC timestamp
`crs_epsg`	yes	the authoring CRS to restore on import (§5)
`color_standard`	yes	fibre colour standard, e.g. `TIA-598-C`
`color_catalog_json`	no	full colour catalogue if non-standard
`profile`	no	`geojson-lite` 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. A key outside the table above belongs to the writer that recorded it, and every other writer carries it unchanged; the FiberQ plugin's own is described in §12.

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, it keeps it verbatim, keyed to the object it belongs to, and emits it again untouched on the next write: an unknown `fq_type` as a whole object here, a side-car table it does not implement as rows of that same table, and an attribute with no column in the feature's own `fq_extra_json` (below).

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 a reader has no column for travel in the feature's own `fq_extra_json` column rather than here; `fq_extension` is for whole objects, and for the rows other tools have already put in it.

9. Conformance

A **conformant reader** must: 1. read every feature layer and the metadata table; 2. preserve `fiberq_uuid` exactly; 3. carry anything it does not understand through to its next write, as §8 describes; 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 runs this round trip in CI for the data listed in §12.

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 sidecar 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 and re-emitted unchanged as an `fq_extension` row, not as a layer of its own — never reclassified
An attribute with no FiberQ column	Kept against the feature's identity, re-emitted into its `fq_extra_json`
`fq_splice_point`, `fq_fiber_connection`	Kept verbatim and written back **as rows**, so they are still queryable
A passthrough row from a third tool	Untouched
A `_fiberq_metadata` key FiberQ did not write	Carried through the project into the next bundle
Relations	Out under their own uuid with their category, member roles and order, back as project data, out again unchanged
Path stops	Out with `is_latent` 1 or 0 in the cable's order, back into the project, out again with the same order and marks
A side-car table this version does not define, and a column a defined table lacks	Re-created as they arrived and written back as rows

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, `fq_type` + `placement`	written
Canonical field names and values (§6.3)	written
Storage CRS (§5), `crs_epsg` restore target	written
Identity preserved, never regenerated (§4)	written; a relation that a FiberQ 1.5.0 import brought in keeps an identity derived from the project file until its bundle is imported again
Metadata merged, foreign keys preserved (§7)	written
`fq_relation` / `fq_relation_member` from project relations	written, with the category, member roles and order they came with
`fq_path_stop` from recorded pass-through elements	written
`fq_extension` 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, including tables and columns a later 1.x version adds	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 project-local reference it cannot resolve is **reported and left out**, never mapped to the nearest plausible object. A `fiberq_uuid` that arrived in a bundle is different: a path stop naming a cable or an element the plugin does not draw is kept as it came and written out again.

An export over a bundle the plugin wrote before replaces the `fq_relation`, `fq_relation_member` and `fq_path_stop` rows of the relations and cables it writes, whichever tool added them, and removes the rows of a relation or a cable it wrote there last time that the project no longer has, and the feature table it wrote for a layer the project no longer has. While a FiberQ layer of the project cannot be read (QGIS cannot open its data, or its coordinates do not convert to EPSG:4326), or the project's relations or pass-through elements cannot be read, it removes none of those rows, and that layer's table stays. In the GeoJSON profile it removes the files it wrote last time in the same way. It records what it wrote in the metadata key `fiberq_written`, its own (§7), tied to that export's `produced_at`: a copy of the key in another file is not trusted. Every other row, table and file stays, and a bundle written before FiberQ 1.6.0, which has no such record, loses nothing -- except the copy of a relation FiberQ wrote there under the identity derived from the project file, which goes once the relation is written under its own.

`color_standard` is the name of the project's own colour catalogue when it has one, and an arrived `color_catalog_json` is then removed, since it defines another standard. While the project has none, the standard and catalogue the bundle arrived with are written again, or TIA-598-C when it arrived with none. The two are taken together from one place, never one from each: what the last import naming another standard than TIA-598-C brought, else the file being written over, else the GeoJSON folder. A bundle that names only TIA-598-C, with no catalogue, does not replace a pair an earlier import kept. Writing over a file also removes keys that describe only an earlier write to it: `profile`, which only the GeoJSON profile sets, and the five FiberQ's project export to GeoPackage writes (`relations_json`, `latent_elements_json`, `plugin_settings_json`, `export_timestamp`, `project_version`).

In FiberQ, `is_latent` 0 means an element the pass-through dialog lists along the cable but the user has not marked as passed through, which is every element's default there. It does not record that the cable terminates at that element.

The other side-car tables are always created, even though the plugin never writes rows of its own into them: what they hold is what it carried through from a bundle it imported and, when an export replaces an existing bundle, the rows already in that file, which are kept. 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.

A side-car table this version does not define, and a column a defined table lacks, are carried too, as rule 4 asks of a 1.0 reader: re-created as they arrived, with their columns, their declared types (an unusual type is written as the plain type with the same SQLite affinity) and their primary key. Rows that several imported bundles brought for the same table are kept side by side; for one key the row an earlier import brought is kept, and a later bundle's different one is reported. Text that is not UTF-8 is kept byte for byte. Views, indexes and triggers are not carried.

The plugin has separate layers per placement only for `otb` and `to` mounted indoors, outdoors and on a pole, and for `to` in a joint closure (§6.1). Any other placement it reads is kept in the feature's `fq_extra_json`, so the feature's `placement` column is empty when the plugin writes it again.

A column of the plugin's own whose name, compared regardless of letter case as SQLite compares column names, is one the writer fills on that layer cannot be a column of the same table. The plugin writes its values into the feature's `fq_extra_json` under the column's own name instead; the field mapping lists the cases.

JSON has no date type, so a date or time the plugin carries in `fq_extra_json` or in an `fq_extension` payload travels as text. From FiberQ 1.6.0, one that arrived in a bundle's `DATE` or `DATETIME` column travels as the text that column stored, character for character, in whatever form the writer chose. One the plugin formats itself, from a field of a project layer, is ISO 8601: a date-time with milliseconds and, where QGIS reports one, its offset from UTC (`2026-01-02T03:04:05.678Z`); a time with milliseconds when it has some; a date as `YYYY-MM-DD`. For a GeoPackage `DATETIME`, QGIS 3.22 reports no offset.

13. Corrections to this text

The format has not changed since `1.0` was published. Until FiberQ 1.6.0, this document described it wrongly in six ways. They are corrected above:

- the §3 tree left out `fq_path_stop`, which §6.4 specifies, so it showed eight `fq_*` tables where a bundle has nine;
- §3 said a feature carries the canonical fields and nothing else, and §6.3 and §8 that an attribute with no canonical column travels in `fq_extra_json`. A writer's own column with no canonical name is written under its own name; `fq_extra_json` is where a reader keeps an attribute it has no column for;
- §8 and §9 said a reader stores everything it does not understand in `fq_extension`. Only a whole object of an unknown type goes there: an attribute with no column travels in the feature's `fq_extra_json`, as §8's own last point already said, and a sidecar table a reader does not implement returns as rows of that table;
- §6.1 called the type codes dot-free, but five of them (`cable.aerial`, `cable.underground`, `duct.pe`, `duct.transition`, `closure.joint`) contain a dot;
- §6.2 gave `fq_container_slot` a column called `index`, but the column is `slot_index`;
- §6.2's column list for `fq_fiber_connection` left out `source_color_hex` and `dest_color_hex`, the advisory columns the paragraph below the list describes.

In each case the reference implementation has always done what the text now says for the tables and columns this version defines, and a reader or writer built from the earlier text should follow the text above. From FiberQ 1.6.0 a column called `index` passes through FiberQ like any column a table did not define, but a reader built from the corrected text looks for `slot_index`: rename the column.

The FiberQ plugin now checks in CI (`tests/test_interchange_format_doc.py`) that the `fq_*` tables in the §3 tree, the §6.1 type codes, the column lists in §6.2, §6.4 and §8 and the required §7 keys match its implementation, that every other §7 key is one it owns, and that every back-quoted `fq_*` name in this text is a table or a column the format has. Nothing else on the page is checked.

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.