

FiberQ 1.5.0 — Moving a design between tools

A user guide to the interchange format introduced in **FiberQ 1.5.0**. For the format itself, see the [specification](#). For every field and value, see the [field mapping](#).

A fibre design is not a pile of points and lines. A slack loop belongs to a particular cable. A cable passes through particular manholes, in order. Cables are grouped into named routes. Export a project to an ordinary GeoPackage and all of that is gone — not because GeoPackage cannot hold it, but because those relationships are recorded against identifiers that are local to one QGIS project file and mean nothing anywhere else.

FiberQ 1.5.0 adds an **interchange bundle**: one file that carries the design and its relationships, keyed by an identity that survives the trip. It is a published, openly licensed format, so the tool at the other end does not have to be FiberQ.

1. Exporting a bundle

Plugins → FiberQ → Export interchange bundle...

Choose where to save it. The file-type dropdown at the bottom of the dialog decides which of the two profiles you get:

Dropdown	You get
GeoPackage bundle (*.gpkg)	one .gpkg file — complete
GeoJSON bundle – a folder, no relations (*)	a folder of .geojson files — see §4

Your project is not modified. An export reads the layers and writes a copy; nothing is renamed, no layer is redirected at the new file, and no attribute is rewritten. You can export mid-design as often as you like.

When it finishes, the message bar says what was written:

```
FiberQ: Wrote design.gpkg – 23 layer(s), 46 feature(s), 1 foreign metadata key(s) preserved
```

Every clause is there on purpose. An export that quietly dropped a layer and an export that had nothing to drop read identically from a bare "done", so the summary names what happened — including layers left out, and how many metadata keys belonging to another tool were carried forward.

Overwriting an existing bundle

If the file already exists, FiberQ asks first, and says what replacing it does:

Replace the bundle design.gpkg? Its FiberQ layers are rewritten from this project. Anything another tool stored in the file — its own metadata keys and relational tables — is kept, not discarded.

That is the honest description of the operation, and it is worth reading once. Exporting over a file a colleague's tool also writes to does **not** wipe their half of it.

2. What is in a bundle

Open one in QGIS and it looks like an ordinary GeoPackage, because it is one. The parts that make it a bundle are:

Canonical layers and fields. Layers are named `Poles`, `Route`, `Underground cables`; fields are named `pole_type`, `length_m`, `fiber_count`. Your project keeps its own stored names — the translation happens only in the exported copy, so a receiving tool never has to learn FiberQ's internal schema. The complete table is in the [field mapping](#).

A permanent identity on every feature. `fiberq_uuid` is the join key. Not the row number — row numbers change every time anything is rewritten, and a reference built on one breaks silently.

Relationships as side-car tables. `fq_relation`, `fq_relation_member` and `fq_path_stop` record the named cable groupings and the elements a cable passes through, all by `fiberq_uuid`. A slack loop or fibre break carries `cable_uuid` instead of a project-local layer id and feature id.

A metadata table. `_fiberq_metadata` records the format and its version, the schema version, which tool wrote the file and when, the colour standard, and — importantly — `crs_epsg`, the CRS the design was authored in. Geometry itself is stored in EPSG:4326, so any tool can read it without a projection library; the authoring CRS is how the design gets back to where it was drawn.

A passthrough store. `fq_extension` holds whatever the writing tool could not model. See §5.

3. Importing a bundle

Plugins → FiberQ → Import interchange bundle...

Features land in FiberQ's own layers, with FiberQ's own field names and values. Your data model does not change to accommodate a file.

FiberQ: Imported design.gpkg – 23 layer(s), 46 feature(s)

Three behaviours are worth knowing before you rely on it.

Importing the same bundle twice changes nothing. Identity is permanent, so a feature whose `fiberq_uuid` is already in the project is the same feature, not a second one. The second import reports 0 feature(s) — it does not duplicate your design, and it does not disturb the relationships the first one restored.

Into an empty project, the authoring CRS comes back. Import into a project you have just created and it switches to the CRS the design was drawn in. Import into a project that already holds layers and your own CRS is left alone — an import adds to your work, it does not reproject it.

A bundle it cannot read fully is refused, not read partially. A bundle declaring a newer major version of the format stops with an explanation. Importing half a network and reporting success is worse than importing none of it.

What if the file is not a bundle? An ordinary GeoPackage from another GIS user has no `_fiberq_metadata` table, and the import refuses it: "This is not a FiberQ interchange bundle... Open it as ordinary layers instead." That is deliberate. Guessing which column is a cable diameter and which is a pole height produces a plausible, wrong network. See §6 for what to do with such a file.

4. The GeoJSON profile

Pick the GeoJSON entry in the file-type dropdown and type a **folder** name. You get one file per element type plus the metadata:

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

Files are named for the element type, not the layer, so a receiving tool reads `pole.geojson` without knowing FiberQ ever called that layer "Poles". Every feature still carries `fq_type` and `placement`, so nothing depends on the filename. Coordinates are WGS84 longitude/latitude, as RFC 7946 requires.

GeoJSON cannot express the relational side-car. If your project has relationships to lose, the export says so, and stamps the bundle `"profile": "geojson-lite"` so a reader can tell:

```
FiberQ: GeoJSON cannot carry the relational side-car: 12 row(s) were left out.
The bundle is marked profile=geojson-lite. Export to GeoPackage for a complete bundle.
```

A GeoJSON bundle with nothing to lose is complete and is not marked — labelling a complete bundle lossy misleads a reader as surely as not labelling a lossy one.

Use GeoJSON for web maps, a quick look, a handover where geometry and attributes are all that is wanted. **Use GeoPackage** for anything that has to come back.

5. What survives that you might expect to lose

The format's first rule is that a tool passes on what it cannot use.

An element type FiberQ has no layer for — a splitter, say — is kept whole, with its geometry and attributes, and written out again unchanged. You are told it is there:

Carried through unchanged, with no FiberQ layer to draw them in: splitter (14).
They survive the next export.

It is never reclassified to the nearest familiar type. Filing a splitter as an OTB cannot be undone, and the import would look like it worked.

An attribute with no column is kept against the feature it belongs to and restored on export.

Relational data FiberQ does not implement — fibre splices, trays, duct occupancy — is kept as the rows it arrived as, and written back into the real table on the way out, so it reaches a tool that does model it still queryable, not as an opaque blob.

Metadata keys another tool wrote are merged into the next bundle, never replaced.

The practical consequence: FiberQ can sit in the middle of a workflow between two tools that both know more than it does, and not be the place the data dies.

6. Working with other tools

If the other tool implements the format, both directions work today. The [specification](#) is published under CC-BY-4.0 with a conformance checklist, and the reader checks the file's metadata, not who sent it.

If the other tool does not, you have two options.

Ask them to implement it. The spec is short, the container is a plain GeoPackage, and the conformance list is nine items.

Or bring the data in by hand. An ordinary GeoPackage, Shapefile or DXF can be added to QGIS the usual way (**Layer → Add Layer → Add Vector Layer**), and any layer you rename to a FiberQ layer name is exported as that type from then on. It is manual, and it is honest: you decide what each layer is, rather than the plugin guessing.

A guided importer that maps a foreign file's columns onto FiberQ's fields is a sensible future feature, and is not part of this release.

7. Older projects

A project made before FiberQ's English rename exports exactly like a new one. Legacy **layer** names (`Stubovi`, `Trasa`, `Kablovi_podzemni`, `OKNA`, `Nastavci`, `Opticke_rezerve`) are recognised and exported under their canonical names, and so are legacy **field** names — including `kabl_layer_id` / `kabl_fid`, which is how a pre-1.0 project still records which cable a slack loop belongs to.

That last one matters more than it looks: unrecognised, it is not a field with an odd name, it is the cable reference going unnoticed, and the loop arrives at the next tool detached from its cable.

Your project is not renamed by any of this. Open a 2019 project, export a bundle, and the project still says `Stubovi` afterwards.

8. Quality assurance and compatibility

The round trip is asserted in CI on every commit, against a bundle deliberately richer than this plugin: it carries an element type FiberQ has no layer for, an attribute it has no column for, fibre-splice tables it does not model, a third tool's passthrough rows and a metadata key it did not write. The test imports it, exports it again, and checks that every object is still there.

FiberQ 1.5.0 was verified by hand on **QGIS 3.40 (Qt5)** and **QGIS 4.2 (Qt6)**: a modern project, a project renamed to pre-1.0 Serbian layer names, an empty project receiving an import, the GeoJSON profile, a 2019 project, and bundles written on one QGIS version and read on the other. Five defects were found that way and fixed, three of them silent data loss, each with a regression test.

Targets QGIS 3.22 LTR → QGIS 4 / Qt6.

The interchange format, its export and its import were developed with support from the [NLnet](#) NGI0 Commons Fund.