FiberQ — Moving a design between tools

A user guide to the interchange format introduced in **FiberQ 1.5.0**, as it works from **FiberQ 1.6.0**. 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 added 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, 2 layer(s) not part of the bundle
```

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 counts what happened: how many metadata keys belonging to another tool were carried forward, and how many layers were left out. It gives the number, not the names. A vector layer is left out when FiberQ does not recognise its name as one of its own (a layer of your own, or a FiberQ layer you renamed), when QGIS cannot open its data source (a missing file, an unreachable database), or when it is empty and another layer with the same FiberQ name holds features (an empty `Poles` beside your `Stubovi`). Layers that are not vector layers, such as a raster basemap, are never part of a bundle and are not counted. Which layers were left out, and why, goes only to FiberQ's debug log, which is off by default: start QGIS with the environment variable `FIBERQ_LOG_LEVEL=DEBUG` and they are listed in **View → Panels → Log Messages**, on the FiberQ tab.

Three things are reported in a warning as well, by name:

- **A FiberQ layer whose data QGIS cannot open** is left out, and its table in an existing bundle is kept as it was. So are the relations and pass-through elements FiberQ wrote there before: with a layer unread, the project cannot say which of them it no longer has.
- **A layer whose coordinates cannot be converted to EPSG:4326** is left out: it has no CRS, or one with no transformation to EPSG:4326, and its features would otherwise land in the bundle unmoved, in the wrong place. Set the CRS its coordinates are in (**Layer Properties → Source**) and export again. Its table in an existing bundle is kept, as for a layer QGIS cannot open.
- **A column whose name FiberQ writes** is moved. A GeoPackage compares column names regardless of letter case, so a column of your own called `Name` or `name` beside FiberQ's `naziv`, say, cannot be a column of the bundle. Its values go into the feature's `fq_extra_json` under the column's own name, and the warning lists the columns. Nothing is merged or dropped. Where the feature's own `fq_extra_json` already has a key of that name, the column's value replaces it, and the export says so. The field mapping lists the names this applies to.

Two layers that are one FiberQ layer. A bundle holds one table per element type, so `Poles` and the pre-1.0 `Stubovi` in one project would both become the table `Poles`. When both hold features, FiberQ writes nothing, and says which layers they are and where each one's data lives: merge them into one layer, or remove or rename all but one, then export again. An existing bundle is left exactly as it was. The same table loaded twice is one layer, and is exported once.

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, and the members and pass-through elements of FiberQ's own relations and cables, are rewritten from this project — import the file first to keep what another tool added to them — and what FiberQ wrote there for a layer, relation or cable the project no longer has is removed. Everything else another tool stored in the file — its own metadata keys, tables and rows — is kept.

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, with one exception: what their tool added to FiberQ's own relations and cables (a member, a stop) is rewritten from your project with the rest of them. Import their file first, and it is in your project and in the next bundle. FiberQ's own half is brought up to date: the rows it wrote there last time for a relation, a relation member or a pass-through element the project no longer has are removed, and so is the table it wrote for a layer the project no longer has — one you deleted or renamed. A layer still in the project keeps its table even when it could not be exported this time, as above. A warning names each table removed. A bundle written before FiberQ 1.6.0 does not record which tables FiberQ wrote, so the first export over it removes none.

A project saved in QGIS 4 and opened in QGIS 3 cannot have its relations, pass-through elements or what an import kept read there (QGIS 3 does not understand how QGIS 4 stores them). A bundle exported from it to a new file has none of them, and the export says so. Over an existing bundle the export is refused, because it would remove them from it. Export from QGIS 4 for a complete bundle.

---

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, all by `fiberq_uuid`. `fq_path_stop` holds, for each cable whose list was saved in the pass-through dialog, the elements along it in order, each marked as passed through (`is_latent` 1) or not (0). 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.

A FiberQ layer the import has to create is a temporary layer, like every layer FiberQ creates, until it is saved. With **Auto save to GeoPackage** on, it goes into the project's GeoPackage as it is made. Otherwise use **Save all layers to GeoPackage** before you close the project: a temporary layer is empty when the project is opened again, and the features the import added with it are gone. What the import kept for other tools is in the project file itself, and survives either way.

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

Four 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), 46 already in the project, left as they are` — it does not duplicate your design, and it does not disturb the relationships the first one restored. The one exception is a feature with no `fiberq_uuid` at all: nothing says it is one already imported, so it is added again, and the import warns about it every time.

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 vector layers and your own CRS is left alone — an import adds to your work, it does not reproject it. Only ordinary vector layers count: a project holding nothing but rasters (an XYZ or WMS basemap, say), vector tiles, meshes or point clouds is treated as empty, and its CRS switches too. A basemap loaded as a vector layer counts as your work.

An import adds; it never replaces. A relation the project already has keeps its own members, and a cable the project already records pass-through elements for keeps its own; a warning counts those where the bundle's has members or elements the project's lacks. A relation or a cable the project does not have yet gets the bundle's, with the relation's own identity, category, member roles and order. A feature already in the project is not updated either — not even a slack loop's cable, which stays as you left it. A row of a side-car table FiberQ keeps for other tools, with the key of a row an earlier import kept, does not replace it; a warning counts those that differ. To see a colleague's version exactly as they sent it, import it into a new project.

A bundle it cannot read is refused before anything is imported. 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.

Inside a bundle it can read, from FiberQ 1.6.0 a feature whose coordinates cannot be reprojected into your project's CRS is skipped rather than imported with infinite coordinates, and a warning says how many, layer by layer. If a layer of the bundle cannot be read, or its features cannot be added, the import carries on with the rest of the bundle and then reports a failure; nothing it added is taken out again.

> **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.

Writing into a folder that already holds a bundle works like writing over a GeoPackage one: `_fiberq_metadata.json` is rewritten, and the keys another tool stored in it are kept. A `.geojson` file FiberQ wrote there last time for an element type this export has no layer for is removed, and the warning names it.

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 as a row of the bundle's passthrough table, `fq_extension`, rather than as a layer of its own. 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.

The colour standard a bundle was drawn to travels with it. A bundle made to a standard other than TIA-598-C, with the catalogue that defines it, comes back out of FiberQ saying so for as long as your project has no colour catalogue of its own. Once it has one, the project's own standard is written instead. Import bundles drawn to two different standards and the later one wins, and the import says so; a bundle that names only TIA-598-C does not replace one that named something else.

An attribute with no column is kept against the feature it belongs to and restored on export, an empty value included. So is a `cable_uuid` on an element FiberQ does not link to a cable, and a slack loop's or fibre break's `cable_uuid` that names a cable the project does not have. A feature with no `fiberq_uuid` has nothing to keep them against, so they are not kept for it; the import says how many.

Feature tables are read whatever their name or language. The element type comes from each feature's `fq_type`, not from the table name, so a bundle that keeps its loops in a table called "Slack loops", or its cables in "Optički kablovi", imports like any other. A name QGIS cannot open (one with `|` or a control character) is named in a warning instead.

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. That includes a table or a column a later version of the format adds, and the rows several imported bundles brought: a second import adds to what the first kept.

A pass-through element or a relation member FiberQ does not draw — a stop on a splitter it keeps only in the passthrough store, a stop on a cable that is not in the project, a relation member it cannot place — is kept and written out again, though the dialogs do not list it.

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 ten 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 — a name the project does not already use. If it does, copy the features into the existing layer instead: two layers with one FiberQ name stop the export (§1). It is manual, and it is honest: you decide what each layer is, rather than the plugin guessing. Its columns go into the bundle under their own names, except one whose name FiberQ writes on that layer, which travels in `fq_extra_json` (§1).

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.

If the project also has a layer under the modern name — an empty `Poles` beside `Stubovi` — the empty one is left out. If both hold features, the export stops and says so (§1).

---

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 passthrough row from a third tool 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.