TagProvider QuestDB Example

Browse a QuestDB instance from the Designer UNS tree, register columns as type-aware UNS tags, and read live values from Scripts and Displays.

How-to **Examples** **Feature** **UNS** TagProvider QuestDB

This example shows how to expose QuestDB tables in the FrameworX UNS tree, register columns as type-aware UNS tags, and read or write them from Scripts and Displays. Companion to the Datasets QuestDB Example (which covers the Pillar 4 / SQL query path) and the QuestDB Time-Series Database Connector reference page.

Prerequisites:

- A running QuestDB 7.0+ on `localhost:8812` (PostgreSQL wire) and `localhost:9000` (ILP /HTTP and Web Console).
- A database named `qdb` with a `sensors` table - the QuestDB Time-Series Database Connector reference page covers the install walkthrough and seed schema.
- FrameworX 10.1.5 running on a Multiplatform / .NET 10 target. PostgreSQL-wire reads work on the .NET Framework 4.8 host but ILP writes require .NET 10.
- Default credentials on a fresh local install are `admin` / `quest`.

Summary

This example walks through three patterns the QuestDB UNS TagProvider supports from a single FrameworX solution: a multi-column registration that exposes several tags from one QuestDB table with the right type per column, the live read semantics that go through the PostgreSQL wire on every scan, and the append-style write semantics customers must understand before binding a UNS-mapped tag to a high-frequency writer. The page also compares the three independent FrameworX surfaces over the same QuestDB data so an integrator can pick the right one per use case.

For Pillar 4 (SQL Dataset queries against historized data, including `SAMPLE BY` server-side downsampling), see the Datasets QuestDB Example. For the per-role configuration reference (Device, UNS, Historian, Dataset), see the QuestDB Time-Series Database Connector reference page.

Technical Information

Configure one UNS TagProvider, register three columns of the `sensors` table as UNS tags with three different attribute types, and run two Script Tasks - one read-only, one that writes a value back to QuestDB and observes the ILP-append behavior.

Configure the UNS TagProvider

In **UNS / TagProviders**, create a TagProvider:

- Name: **QuestSensors**.
- Protocol: `QuestDB`.
- ServiceType: **TagDiscovery**.
- PrimaryStation:

```
localhost;8812;http://localhost:9000;admin;quest;qdb;false
```

The Designer Station editor exposes the seven fields as a structured form (Host, PgPort, IlpEndpoint, Username, Password, Database, TlsEnabled). The persisted format is the semicolon-delimited string above.

Register columns as UNS tags - column-type-aware

In Designer, expand the UNS tree under **QuestSensors**. Level 0 lists tables in the `qdb` database via `SHOW TABLES`; expand **sensors** to see each column with its native QuestDB type. The connector queries QuestDB column metadata at registration time and maps the QuestDB type to the matching FrameworX attribute type.

Right-click each column and choose **Add to UNS**:

Column registered	QuestDB type	FrameworX attribute type

`sensors.plant`	`SYMBOL`	Text
`sensors.temperature`	`DOUBLE`	Double
`sensors.quality`	`SHORT`	Integer

If the type cannot be resolved (unknown QuestDB type, table not yet created, transient connection failure), the registration falls back to **Double** with a TraceWarning logged. Re-register the tag once the table exists to pick up the actual column type. The full type mapping table is on the QuestDB Time-Series Database Connector reference page under UNS TagProvider / Register a tag.

Read semantics - latest row per scan

On each scan cycle, the connector runs one independent `SELECT <column> FROM sensors ORDER BY timestamp DESC LIMIT 1` per registered tag over the PostgreSQL wire. Quality is 192 (Good) on hit, 0 (Bad) when the table is empty or the column is absent. A tag whose underlying QuestDB column has never been written reads as Bad until the first row arrives.

Reads do not coalesce across tags that share a table prefix - three registered columns of the same table issue three independent queries per scan. For high-cardinality tag sets against a single QuestDB table, prefer a Dataset Query that reads multiple columns in one round-trip, or a Historian range query, depending on the use case.

Write semantics - ILP append, not row update

QuestDB is append-only by design. Writing a UNS-bound tag does NOT update the latest row of its source table - it appends a NEW row through ILP /HTTP with the new value, leaving older rows in place. A solution that writes the same UNS-bound QuestDB tag every scan cycle creates one new row per scan; storage grows accordingly.

This is normal for time-series workloads but surprises integrators who expect SQL-style `UPDATE`. For high-frequency historization, prefer the Historian StorageLocation surface (covered in the connector reference page) - it batches writes through the historian flush thread (one ILP `SendAll` per 500 ms tick) instead of one round-trip per tag write.

The three FrameworX surfaces over the same QuestDB data

FrameworX exposes three independent surfaces over a QuestDB table; all three reach the same rows but optimize for different read patterns. Pick by scan cadence and aggregation needs:

Surface	What it reads	Best for
UNS TagProvider (this page)	Latest single row of the table, projecting the requested column. One query per tag per scan.	Live values on Displays and in Script logic. Sub-second scan cadence on a small set of tags per table.
Historian StorageLocation	Range query over the timestamp column with ascending sort. `interval` and aggregation parameters are ignored in 10.1.5 - raw samples only.	Trend charts, time-window analysis, exporting raw history to a tag samples table.
Dataset Query (PostgreSQL provider)	Native QuestDB SQL: `SAMPLE BY`, `LATEST ON`, `ASOF JOIN`, materialized views.	Server-side downsampling, multi-table joins, analytical dashboards. See the Datasets QuestDB Example for a worked example.

Caveats

- Single-dot addresses only. `schema.table.column` is rejected at registration with the message `Multi-segment paths not supported in 10.1.5 — use 'table.column' with a single dot.`
- Writes append rather than update. A solution that writes a UNS-bound QuestDB tag every scan cycle creates one new row per scan; this can balloon storage quickly. Use the Historian StorageLocation surface (batched ILP) for high-frequency historization.
- The .NET Framework 4.8 host can read tags via the PostgreSQL wire but cannot write - the official `net-questdb-client` SDK is .NET 6+ only. Run on a Multiplatform / .NET 10 target when writes are needed; the connector fails-fast with a clear `FileNotFoundException` on net48 if a write is attempted.
- Tag type probing depends on the table existing at registration time. Tags registered against a not-yet-created table fall back to **Double**; re-register after the table exists to pick up the actual column type.
- Browse is side-effect-free. Designer tree expansions can be triggered repeatedly without changing connector state, so refreshing the tree in the Designer is safe.

In this section...
