QuestDB Time-Series Database Connector

New in 10.1.5. QuestDB is integrated across three FrameworX data connection surfaces: Device Protocol, UNS TagProvider, and Historian StorageLocation. QuestDB speaks the PostgreSQL wire protocol natively, so SQL queries against historized data work through the existing PostgreSQL Dataset Provider with one connection-string flag - no separate QuestDB Dataset Provider is needed.

QuestDB time-series databases for high-throughput industrial telemetry.

- Name: **QuestDB**
- Version: 0.9.0.0
- Protocol: ILP (InfluxDB Line Protocol over HTTP) for ingestion + PostgreSQL wire protocol for queries
- Interface: TCP/IP
- Runtime: **.NET 10 only (Linux, Windows, Docker)**. The .NET Framework 4.8 host loads the connector but cannot perform ILP writes - see Runtime requirements below.
- Configuration:
  - Devices / Channels
  - UNS / TagProviders
  - Historian / Storage Locations
  - Datasets / DBs (via the existing PostgreSQL provider)
- Minimum server version: QuestDB 7.0 (recommended 8.0 or newer)

Overview

QuestDB is a high-performance open-source time-series database designed for industrial telemetry, financial market data, and IoT workloads. FrameworX integrates QuestDB across three runtime surfaces, plus a fourth path for SQL Dataset queries through the existing PostgreSQL provider.

Surface	Configure in	What it does
Device Protocol	Devices / Channels / Protocol = `QuestDB`	Map QuestDB table columns as Channel, Node, and Point tags. Reads pull the latest row of a table, writes update the latest row through ILP append.
UNS TagProvider (TagDiscovery)	UNS / TagProviders / Protocol = `QuestDB`	Browse QuestDB tables as a tag tree. Register columns as UNS tags. Updates write back via ILP.
Historian StorageLocation	Historian / Storage Locations / Protocol = `QuestDB`	Append historian tag samples to QuestDB through ILP/HTTP. High-throughput batched writes. Reads via PostgreSQL wire range queries.
Dataset Provider (existing PostgreSQL)	Datasets / DBs / Provider = `PostgreSQL` with the `NoTypeLoading` flag	Run native SQL `SELECT` queries against QuestDB tables (joins, `SAMPLE BY`, `ASOF JOIN`, materialized views). No QuestDB-branded provider entry in v1 - QuestDB is fully PostgreSQL-wire compatible.

A QuestDB UnsTagProvider row and a Historian StorageLocation row can both point at the same QuestDB instance.

Supported QuestDB versions

- Floor: **QuestDB 7.0**.
- Recommended target: **QuestDB 8.0** or newer (current stable line).
- Tested against 8.x and 9.x server lines.

Runtime requirements

.NET 10 host required for ILP writes. The official QuestDB .NET client `net-questdb-client` targets .NET 6 and newer only - it has never shipped a .NET Framework or .NET Standard 2.0 build. FrameworX hosts ILP writes from `TServer.NET10.exe` (Windows, Linux, or Docker on .NET 10).

The .NET Framework 4.8 host (`TServer.exe`) loads the QuestDB connector but cannot perform ILP writes. Connection attempts fail-fast at connect time with a clear `FileNotFoundException("net-questdb-client.dll not found")`. PostgreSQL-wire reads still work on net48 because `Npgsql` 4.0.1 supports both targets. Use a Multiplatform / .NET 10 solution target when QuestDB ILP writes are needed.

Prerequisites

QuestDB .NET clients ship unconditionally with FrameworX in `FS/ThirdPartyBin/net10.0/`: `net-questdb-client` for ILP ingestion (Apache 2.0, ~91 KB) and `Npgsql` 4.0.1 for PostgreSQL-wire queries (PostgreSQL License BSD-style, ~736 KB). No separate install and no optional pack.

Installation

Install a QuestDB server, then configure FrameworX to point at it. QuestDB is OS-native and has no FrameworX dependency on the server side.

Install the QuestDB server

Follow the official QuestDB installation guide at questdb.com/docs/get-started. Summary by target:

- **Docker (recommended for testing):** `docker run -d --name fx-questdb -p 8812:8812 -p 9000:9000 questdb/questdb:latest`.
- **Linux:** download the QuestDB no-jre tarball from questdb.com and run with Java 17 or newer, or install via the platform package manager when available.
- **Windows:** download the QuestDB `rt-windows` distribution (bundled JRE) and run the included `questdb.exe` service installer.
- **macOS:** `brew install questdb && brew services start questdb`.

Verify the server is reachable

Run the command below to confirm the server responds on the PostgreSQL wire port (8812):

```
psql "host=localhost port=8812 user=admin password=quest dbname=qdb" -c "SELECT 1 AS ping"
```

And the ILP/HTTP port (9000):

```
curl -sS "http://localhost:9000/exec?query=SELECT%201%20AS%20ping"
```

A successful ping returns a row containing `1`. Default credentials on a fresh local install are `admin` / `quest` on database `qdb`.

Create the database and starter table

QuestDB tables are created on first write through ILP. To pre-create a starter table for testing, run the following statement through the QuestDB web console (port 9000) or via psql:

```
CREATE TABLE sensors (
  timestamp TIMESTAMP,
  plant SYMBOL,
  temperature DOUBLE,
  quality SHORT
) TIMESTAMP(timestamp) PARTITION BY DAY;

INSERT INTO sensors VALUES (now(), 'Plant01', 22.3, 192);
```

Use `qdb` as the database in the FrameworX connection string in the next section, or use the database name you configured for your QuestDB install.

Connection string

All four QuestDB surfaces (Devices, UNS TagProvider, Historian, and the PostgreSQL Dataset path) share the same Station / connection string format. The Designer Station editor exposes structured fields; the persisted format is semicolon-delimited.

Station format

```
Host;PgPort;IlpEndpoint;Username;Password;Database;TlsEnabled
```

Default values:

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

Field reference

Field	Required	Description	Example
Host	Yes	Host name or IP of the QuestDB server.	localhost

PgPort	Yes	PostgreSQL wire protocol port. Default is 8812. Used for read-side queries (UNS browse, Historian range queries, Device polling).	8812
IlpEndpoint	Yes	Full URL of the ILP/HTTP ingestion endpoint. Default is `http://localhost:9000`. Used for write-side ingestion (Historian samples, Device writes, UNS tag writes).	http://localhost:9000
Username	No	Database user. Default for fresh local installs is `admin`.	admin
Password	No	User password. Stored encrypted in the solution file. Default for fresh local installs is `quest`.	********
Database	Yes	Target database name. Default is `qdb`.	qdb
TlsEnabled	No	Set to **true** for TLS/SSL connections on the PostgreSQL wire port. ILP TLS is planned for a later release.	false

Device Protocol

When to use

Use the Device Protocol surface when you want each QuestDB column addressed as an individual Tag with a Node-level connection. This is the classic SCADA Channel / Node / Point model.

Use UNS TagProvider instead (next section) when you want dynamic browse and tree-style registration.

Configure

Channel. Devices / Channels / New. Protocol = `QuestDB`. Interface is automatically **Custom** (no CommAPI).

Node. One Node per QuestDB server plus database. PrimaryStation uses the standard QuestDB Station format documented in the Connection string section above.

Example value:

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

Point. Address is a single-dot `Table.Column` string. Example: `sensors.temperature`.

Read semantics

On each scan cycle, the driver runs one query per configured address over the PostgreSQL wire protocol, ordered by the table's designated timestamp column descending and limited to 1, projecting the requested column. Quality is 192 (Good) on hit, 0 (Bad) when the table is empty or the column is absent.

Values round-trip as strings in the Device path (matching the MongoDB connector convention). Numeric or boolean tags show their value as text. Use the UNS TagProvider surface (next section) when your integration needs the value native type preserved, or the Dataset Provider section for typed query result sets.

Write semantics - APPEND through ILP

`WriteTagValue` sends an ILP/HTTP line to the IlpEndpoint, appending a row to the table with the current timestamp. Unlike the MongoDB connector, this is append-style behavior even on the Device surface - QuestDB's storage model is append-only by design. Use the Historian StorageLocation surface (later section) for batched-append behavior optimized for high-volume historian writes.

Address limits

Only single-dot addresses are accepted in 10.1.5. `Table.Column` works. `Schema.Table.Column` is rejected. Schema-level addressing is planned for a later release.

Configuration Example

This example maps a single QuestDB column to a Tag using the Channel / Node / Point model. Prerequisites: a running QuestDB server with the `sensors` table from the Installation section above, plus a UNS Tag named **SensorTemp** of type **Double**.

1. In **Devices / Channels**, create a Channel:
   - Name: **QuestDB1**.
   - Protocol: `QuestDB`. The Interface is set to **Custom** automatically.
2. In **Devices / Nodes**, create a Node under that Channel:
   - Name: **Plant_A**.
   - Channel: **QuestDB1**.
   - PrimaryStation:

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

3. In **Devices / Points**, create a Point bound to the tag **SensorTemp**:
   - TagName: **SensorTemp**.
   - Node: **Plant_A**.
   - Address: `sensors.temperature`.
   - AccessType: **ReadWrite**.
4. Run the solution on a Multiplatform / .NET 10 target. **SensorTemp** reads the latest `temperature` value from the most recent row in the `sensors` table on each scan cycle.
5. From a Script Task, write the tag back: `@Tag.SensorTemp = 75.0;`. The driver appends an ILP line to the `sensors` table with the current timestamp.

UNS TagProvider

When to use

Use the UNS TagProvider surface when you want to browse a QuestDB instance from the Designer UNS tree, discover column names dynamically, and register columns as UNS tags.

Configure

UNS / TagProviders / New. Protocol = `QuestDB`. Station uses the same delimited format as the Device Protocol and the Connection string section above.

Browse semantics

The browse view has two levels:

- **Level 0** (tree root for this TagProvider). Lists all tables in the configured database via `SHOW TABLES`.
- **Level 1** (inside a table). Queries the table schema and lists each column with its native QuestDB type. The designated timestamp column is hidden from the browse view by default.

Designer refresh does not mutate state. Browse is side-effect-free and safe to call on every tree expansion.

Register a tag

Select a column under a table and use **Add to UNS**. The registered address is `Table.Column`. The attribute type is mapped from the QuestDB column type:

QuestDB column type	FrameworX template name
`BOOLEAN`	Boolean
`BYTE`, `SHORT`, `INT`, `INTEGER`, `INT16`, `INT32`	Integer
`LONG`, `INT64`	Long
`FLOAT`, `DOUBLE`, `REAL`	Double
`CHAR`, `STRING`, `SYMBOL`, `VARCHAR`	Text
`DATE`, `TIMESTAMP`, `DATETIME`	DateTime
`BINARY`, `UUID`, `IPv4`, `GEOHASH`, `LONG256`	Text (string fallback)
Unknown / probe failure	Double (with a TraceWarning logged)

Read and write semantics

Identical to the Device Protocol section:

- Read pulls the latest row column value via PostgreSQL wire and tags quality 192 on hit, 0 on miss.
- Write appends an ILP line to the table with the current timestamp.

Address limits

Single-dot `table.column` only. The registration call rejects multi-dot addresses with the message `Multi-segment paths not supported in 10.1.5 — use 'table.column' with a single dot`.

Configuration Example

This example browses a QuestDB instance from the Designer UNS tree and registers a column as a UNS tag. Prerequisites: a running QuestDB server with the `sensors` table from the Installation section above.

1. In **UNS / TagProviders**, create a TagProvider:
   - Name: **QuestDBTags**.
   - Protocol: `QuestDB`.
   - ServiceType: **TagDiscovery**.
   - PrimaryStation:

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

2. Open the UNS tree in Designer and expand **QuestDBTags**. The tree shows the tables in `qdb` at level 0 (for example `sensors` and any other tables you have created).
3. Expand **sensors**. The browse queries the table schema and lists each column with its native QuestDB type (for example `temperature` as `DOUBLE`, `plant` as `SYMBOL`, `quality` as `SHORT`).
4. Right-click the `temperature` column and choose **Add to UNS**. A tag `QuestDBTags/sensors/temperature` appears in the UNS Tags list with attribute type `Double`.
5. Run the solution on a Multiplatform / .NET 10 target. `@Tag.QuestDBTags/sensors/temperature` reads the latest temperature value from the table on each scan cycle.
6. Writing the tag from a Script Task appends an ILP line to the `sensors` table with the current timestamp.

Historian StorageLocation

When to use

Use this surface when you want FrameworX Historian to write tag samples into QuestDB as a high-throughput append-style time series. This is the natural fit for QuestDB's append-only storage model.

Prerequisites

A QuestDB UNS TagProvider row must already exist (previous section). The Historian StorageLocation references it by name.

Configure

Historian / Storage Locations / New. The Protocol combo includes QuestDB when the TagProvider row has `IsHistorian="true"` (the default for 10.1.5). The DataRepository field selects which QuestDB UnsTagProvider to use, with the form `TagProvider.<UnsTagProviderName>`.

Storage model - QuestDB tables

The QuestDB destination is derived directly from each historized tag's **address**. An address has the single-dot form `table.column`; the connector splits it and writes ILP lines as:

```
<table> <column>=<value>,quality=<q>i  <designated-timestamp>
```

So an address of `sensors.temperature` writes to the QuestDB table `sensors`, populating the `temperature` column with the sample value. A second historized tag with address `sensors.pressure` shares the same QuestDB table - it adds its own `pressure` column. Multiple historized tags that share a left-side prefix collapse into a single QuestDB table; each contributes one column named after its right-side suffix.

QuestDB's schema-on-write creates the table on the first ILP line if it does not already exist. The resulting columns are:

- The designated timestamp column (auto-managed by QuestDB; `timestamp` by default).
- One typed column per distinct right-side suffix observed (`DOUBLE` for numeric values, the inferred type for strings or booleans).
- `quality` column of type `LONG` (the FrameworX quality code is an unsigned 16-bit value but is written through the ILP integer field path; `192` is `Good`).

Schema-on-write determines partition strategy and column types from the first row written. To choose a specific partition strategy, pre-create the table with an explicit `CREATE TABLE ... TIMESTAMP(timestamp) PARTITION BY DAY` in the QuestDB web console before the first historian write.

Write semantics - BATCHED ILP APPEND

`WriteHistorianDataEx` groups the incoming batch by table name, builds ILP lines, and sends a single `SendAll` round-trip to the IlpEndpoint per drained batch. The driver buffers writes in a background thread (drained every 500 milliseconds) so the platform's poll loop is never blocked on HTTP I/O. Partial-batch failures from QuestDB are surfaced through the per-row `bool[]` return.

Read semantics - raw samples only in 10.1.5

`GetSamples` runs a range query over the PostgreSQL wire protocol (`WHERE timestamp BETWEEN ... AND ...`) with ascending sort. The `interval` and `getSamplesMode` parameters are ignored in 10.1.5. Raw samples only.

Aggregation (`Average`, `Min`, `Max`, `Sum` with non-zero interval) is planned for a later release using QuestDB's native `SAMPLE BY` stage, which performs server-side downsampling efficiently.

Configuration Example

This example appends historian samples for a UNS tag to a QuestDB table. Prerequisites: a UNS TagProvider **QuestDBTags** exists from the previous section and points at the same QuestDB instance, plus a UNS Tag **SensorTemp** of type **Double** being updated by a Device, Script, or external source.

1. In **Historian / Storage Locations**, create a StorageLocation:
    - Name: **QuestDBStore**.
    - DataRepository: `TagProvider.QuestDBTags`.
2. In **Historian / Tags**, bind **SensorTemp** to **QuestDBStore** with a sample interval (for example one second) and a deadband appropriate for the signal.
3. Run the solution on a Multiplatform / .NET 10 target. On the first sample, the connector creates a QuestDB table in `qdb` with the canonical schema documented in the Storage model section above. Subsequent samples are appended via batched ILP writes drained every 500 milliseconds.
4. Read history with `@Tag.SensorTemp.GetSamples(...)` from a Script or Trend control. The connector runs a range query on the `timestamp` column with ascending sort and returns raw samples. Aggregation is planned for a later release.

Dataset Provider

QuestDB Dataset queries through the existing PostgreSQL provider

QuestDB speaks the PostgreSQL wire protocol natively. SQL `SELECT` queries against historized data, joins, `SAMPLE BY` downsampling, `ASOF JOIN` with tolerance, and materialized views all work through the existing **PostgreSQL Data Provider** with one connection-string flag - **no separate QuestDB Dataset Provider is required in 10.1.5**.

Configuration

1. In **Datasets / DBs**, create a Database Connection.
2. Choose **PostgreSQL Data Provider** as the **Provider**.
3. Configure the connection string targeting the QuestDB PostgreSQL wire port (default 8812). **The connection string MUST include the flag `Server Compatibility Mode=NoTypeLoading`** - without it the Npgsql client tries to load PostgreSQL system catalog tables that QuestDB does not expose, and the connection fails.
4. Click **Test** to verify the connection.
5. Create Dataset Queries with native QuestDB SQL.

Example connection string:

```
Server=localhost;Port=8812;Database=qdb;Username=admin;Password=quest;Server Compatibility
Mode=NoTypeLoading;
```

What works

- Native QuestDB SQL: `SELECT`, joins, `WHERE`, `ORDER BY`, `LIMIT`, `OFFSET`.
- QuestDB time-series extensions: `SAMPLE BY` for server-side downsampling, `LATEST ON` for latest-row-per-group, `ASOF JOIN` with tolerance for time-series alignment.
- Materialized views with calendar and timezone awareness.
- N-dimensional arrays for financial and AI workloads.
- Forward-only cursors. Pagination is supported via `LIMIT n OFFSET m`.

Caveats

- Cursor support is forward-only. Scrollable cursors (`DECLARE CURSOR ... SCROLL`) are not available.
- QuestDB's transaction semantics differ from standard PostgreSQL. Single-statement implicit transactions are the default; nested transactions are not supported.
- PostgreSQL extensions (PostGIS, pg_stat_statements, etc.) are not available - QuestDB exposes only the wire protocol, not the full PostgreSQL ecosystem.

A QuestDB-branded Dataset Provider entry may be added in a later release if customer feedback warrants a tailored experience. For 10.1.5, the PostgreSQL provider is the supported path.

Worked example

For an end-to-end Dataset configuration that reads, downsamples (using QuestDB's `SAMPLE BY` stage), counts, and inserts rows on a QuestDB table (one Dataset DB, three Queries, one Dataset Table, three Script Tasks), see the dedicated example page: Datasets QuestDB Example.

Known limitations and deferrals

The table below collects per-surface limits that apply to the 10.1.5 release.

Area	Limit	Workaround
Runtime target	ILP write path requires the .NET 10 host. The .NET Framework 4.8 host fails-fast at connect with `FileNotFoundException("net-questdb-client.dll not found")`. PostgreSQL-wire reads still work on net48 via Npgsql 4.0.1.	Run the solution on a Multiplatform / .NET 10 target. A raw HTTP ILP path that drops the SDK dependency is planned for a later release.
Device and UNS address format	Single-dot `Table.Column` only in 10.1.5.	Schema-level addressing (`Schema.Table.Column`) is planned for a later release.
Device and UNS values	Round-trip as strings on the Device path.	Use the UNS TagProvider for typed registration via the column-type mapping table, or the Dataset Provider for typed query result sets.
Historian aggregation	Raw samples only. Average, Min, Max, Sum with an interval are not supported.	Request raw samples and aggregate in FX Trend or Script. Server-side `SAMPLE BY` aggregation is planned for a later release.
Historian connection pool	One `ConnectionInfo` per call in `GetSamples` (no cache).	Scope time ranges tightly on dashboards.
UNS browse	Live `SHOW TABLES` picker is not exposed in the StationEditor in 10.1.5. Browse the tree from the UNS panel after creating the TagProvider row instead.	A live table-picker combo in the StationEditor is planned for a later release.
ILP TLS	The **TlsEnabled** field plumbs as a boolean on the PostgreSQL wire path only. ILP /HTTP TLS is not configurable in 10.1.5.	ILP TLS is planned for a later release.
Dataset Provider	No QuestDB-branded Dataset Provider entry. Use the PostgreSQL provider with the `NoTypeLoading` flag.	Configure as documented in the Dataset Provider section above.

In this section...