Datasets MongoDB Example

Read, query, and write MongoDB documents from a Dataset.

How-to **Examples** **Feature** **Application** Datasets MongoDB Example

---

New in 10.1.5. This example targets FrameworX 10.1.5 and later, which introduces the MongoDB Dataset Provider.

This example shows how to read, aggregate, and write to a MongoDB collection using the Dataset Provider.

Prerequisites:

- A running mongod 6.0+ on `localhost:27017`.
- A database named `plant_data` with a `readings` collection. The MongoDB Document Database Connector reference page covers the install walkthrough and seed data.

Summary

This example walks through reading documents, running an aggregation pipeline, and inserting new records against a MongoDB collection. The work is done entirely from FrameworX, using the Dataset Provider surface of the MongoDB connector.

For other surfaces of the same connector (Device Protocol, UNS TagProvider, Historian StorageLocation), see the MongoDB Document Database Connector reference page.

Technical Information

The example creates one Dataset DB connection, three Queries, one Table, three Script Tasks, and a small set of UNS Tags. Each piece is configured in the Designer.

Configure the Dataset DB

In **Datasets / DBs**, create a connection:

- Name: `plant_data`
- Provider: `MongoDB.Driver`
- Server: `localhost`
- Port: `27017`
- Database: `plant_data`

Configure the Queries

In **Datasets / Queries**, create three queries against the `plant_data` connection.

QueryFindReadings. Returns the latest 10 readings for Plant01.

```
{
  "collection": "readings",
  "filter": { "plant": "Plant01" },
  "sort": { "ts": -1 },
  "limit": 10
}
```

QueryAggregateHourly. Hourly average of values for Plant01, expressed as an aggregation pipeline.

```
[
  { "$match": { "plant": "Plant01" } },
  { "$group": { "_id": { "$dateTrunc": { "date": "$ts", "unit": "hour" } }, "avg": { "$avg": "$value" } } },
  { "$sort": { "_id": 1 } }
]
```

QueryCount. Counts documents whose `quality` field is good.

```
{
  "count": "readings",
  "filter": { "quality": "good" }
}
```

Configure the Dataset Table

In **Datasets / Tables**, create one table named `TableReadings` bound to the `readings` collection. A Dataset Table on a MongoDB collection uses the document `_id` as the primary key. Updates apply `$set` on the tracked columns and preserve any fields outside the tracked column list, matching the behavior of SQL Dataset Tables backed by a CommandBuilder.

Then set up the column mapping. Double-click the table's **Mapping** field and click the `...` button. Here you link each column to its corresponding Tag: map `plant` to `LatestPlantCode`, `value` to `LatestValue`, and `ts` to `LatestTimestamp`. The mapping is what supplies the values on an insert, and it also retrieves the value for the selected row on a read. See the DatasetTable Mapping Example page for a full walkthrough of the mapping editor.

Configure the UNS Tags

In **Unified Namespace / Tags**, create the tags used by the tasks:

- `QueryResult`. Receives the output of the Find and Aggregate queries.
- `TriggerFind`, `TriggerAggregate`, `TriggerInsert`. Script task triggers.
- `LatestPlantCode`, `LatestValue`, `LatestTimestamp`. Values written back to MongoDB by the Insert task. These are the tags bound to the `TableReadings` mapping, so the Insert task sets them rather than writing to the table columns directly.

Configure the Script Tasks

In **Scripts / Tasks**, create three tasks, each triggered by the matching trigger tag.

Find task.

```
@Tag.QueryResult = @Dataset.Query.QueryFindReadings.SelectCommand();
@Info.Trace("Find OK: " + @Tag.QueryResult);
```

Aggregate task.

```
@Tag.QueryResult = @Dataset.Query.QueryAggregateHourly.SelectCommand();
@Info.Trace("Aggregate OK: " + @Tag.QueryResult);
```

Insert task (via the Dataset Table).

```
// Runs in a Server-domain Script Task.
// TableReadings.Mapping binds the plant, value and ts columns to these tags.
@Tag.LatestPlantCode = "Plant01";
@Tag.LatestValue     = 42.5;
@Tag.LatestTimestamp = DateTime.UtcNow;

int status = @Dataset.Table.TableReadings.InsertCommand();
@Info.Trace("Insert status: " + status);
```

`InsertCommand()` returns an `eErrorMessage` status code, not a row count. Zero means the row was inserted; any non-zero value means it was rejected, and `@Dataset.Table.TableReadings.LastStatusMessage` carries the provider's reason for the rejection.

The insert runs synchronously and blocks the calling task. `InsertCommandAsync()` is the preferred form in new code and returns the same status code.

Run the example

After you finish the configuration and create the scripts, run the solution and trigger each task. Values arrive in the MongoDB **readings** collection and the Find and Aggregate results populate the `QueryResult` tag. Trace output appears in the runtime log.

SQL subset alternative

The same queries can also be authored as SQL. The Dataset host translates SQL into Mongo Find or Aggregate operations. For the Find query above, a SQL form is:

```
SELECT * FROM readings WHERE plant = 'Plant01' ORDER BY ts DESC LIMIT 10
```

For the supported SQL subset and what falls back to the JSON form, see the SQL subset dialect section of the MongoDB Document Database Connector reference page.

In this section...
