Datasets DBs Reference

Register and manage database connections.

Dataset DBs (Reference) define database connections for data storage, retrieval, and manipulation within the FrameworX platform. It provides:

- SQL database connectivity
- Pre-configured system databases
- Connection string management
- Development/production isolation
- Multi-provider support

Each DB represents a connection to a database used for tags, alarms, historian, or custom data operations.

Pre-Defined Databases
Creating Database Connections
ConnectionString Configuration
ConnectionString Macros
Supported Providers
Parameterized Queries (PARAMS prefix)
Connection Management
Time Zone Management
Development vs Production
Security Considerations
Utilities
Store & Forward Limitations
Best Practices Checklist
Troubleshooting
Related Documentation

Configuration Properties

Property	Description	Required
Name	Unique database identifier	Yes
Provider	Database technology (SQLite, SqlClient, etc.)	Yes
Database	Database type/instance	Yes
ConnectionString	Connection parameters	Yes
LogonPassword	Authentication password (admin only)	No
GatewayIP	Remote gateway for security (former ServerIP)	No
FilterColumns	Query optimization columns	No
CustomOptions	Additional configuration	No
Description	Documentation text	No

Pre-Defined Databases

Four system databases are included in every solution:

DB Name	Purpose	Default Location
Retentive	Stores retentive tag values	`<SolutionNameAndPath>.dbRetentive`
RuntimeUsers	Dynamic user management	`<SolutionNameAndPath>.dbRuntimeUsers`
AlarmHistorian	Alarm and audit trail records	`<SolutionNameAndPath>.dbAlarmHistorian`
TagHistorian	Time-series data storage	`<SolutionNameAndPath>.dbTagHistorian`

Creating Database Connections

1. Navigate to **Datasets DBs**
2. Click **Plus icon**
3. Configure:
    - **Name**: Unique identifier
    - **Provider**: Select from list
    - **Database**: Choose type
    - **Description**: Documentation

4. Click **OK**

ConnectionString Configuration

Structure

```
Provider=<provider>;Data Source=<source>;Additional Parameters
```

Examples

SQLite (Local):

```
Provider=System.Data.SQLite;
Data Source=_SolutionPathAndName_.db
```

SQL Server Express:

```
DataSource=.\SQLEXPRESS;
Initial Catalog=myDatabase;
User Id=sa;Password=pwd
```

PostgreSQL:

```
Host=localhost;Database=mydb;
Username=user;Password=pass;Port=5432
```

ConnectionString Macros

Use these macros for portable configurations:

Macro	Description	Example
`_SolutionPath_`	Solution directory	C:\Solutions\MyApp\
`_SolutionName_`	Solution name only	MyApp
`_SolutionPathAndName_`	Full path with name	C:\Solutions\MyApp\MyApp
`_ExecutionPath_`	Working directory	Current runtime folder
`_ExecutionPathAndName_`	Working path with name	Runtime location
`_ProductPath_`	Installation directory	C:\Program Files\FrameworX\
`_Exchange_`	Default transfer folder	Public\Documents\Exchange\
`_ThirdParty_`	Third-party components	MyDocuments\ProductName\ThirdParty\

Supported Providers

Provider	Supported Databases
System.Data.SQLite	SQLite
System.Data.SqlClient	SQL Server, SQL Server Express
Npgsql	PostgreSQL
MySql.Data.MySqlClient	MySQL
Oracle.DataAccess.Client	Oracle
System.Data.Odbc	Any ODBC-compatible
System.Data.OleDb	Access, Excel, CSV

MongoDB.Driver	MongoDB document database. Accepts JSON directives, aggregation pipelines, and a SQL-subset dialect. See MongoDB Database Connector (DRAFT v10.1.5).

Database-Specific SQL Syntax

Database	Syntax Example	Special Considerations
SQL Server	`SELECT TOP 10 * FROM Table`	Use TOP for limiting
SQLite	`SELECT * FROM Table LIMIT 10`	Use LIMIT clause
MySQL	`SELECT * FROM \Table` LIMIT 10`	Backticks for names
PostgreSQL	`SELECT * FROM "Table" LIMIT 10`	Case-sensitive names
Oracle	`SELECT * FROM Table WHERE ROWNUM <= 10`	ROWNUM for limiting
MongoDB	`SELECT * FROM collection LIMIT 10`	Document database. SQL subset translates to MongoDB find. Field names are case-sensitive. JSON directives and aggregation pipelines also accepted in the same Command Text field.

Parameterized Queries (PARAMS prefix)

Dataset query authors can declare bind parameters inline by prefixing the Command Text with a **`PARAMS`** block. The runtime parses the block, registers each parameter with the underlying ADO.NET command, and substitutes values at execution time — values never become part of the SQL string and are not subject to SQL-injection vectors.

Syntax

```
PARAMS @p0='value1', @p1=123, @p2=@Tag.SomeTag;
SELECT * FROM MyTable WHERE Name = @p0 AND Qty > @p1
```

Rules:

- `PARAMS` must be the first token; consecutive parameter assignments are separated by a comma (`,`), and the block is terminated by a single semicolon (`;`) that separates it from the SQL body. A newline does not end the block.
- Parameter names start with `@` and follow the host database's identifier rules (e.g. `@p0`, `@customerId`).
- Right-hand-side values may be string literals (single-quoted), numeric literals, or tag references (`@Tag.<name>`) resolved at execution time.
- The SQL body that follows references the same names; the provider performs the binding.

When to use

- **Untrusted inputs**: any value that originates from a Display field, a script argument, an external API, or a user-edited tag — use `PARAMS` to keep the value out of the SQL string.
- **Repeated execution with different values**: parameterized commands are prepared once and re-bound per execution, which is faster than re-parsing concatenated SQL.
- **Type fidelity**: the provider preserves the parameter's CLR type (string, int, DateTime, byte[]) end-to-end, avoiding culture-dependent string conversions.

PARAMS vs string concatenation

String concatenation (`"SELECT * FROM T WHERE Name = '" + userInput + "'"`) is acceptable only when every concatenated fragment is a trusted, code-controlled constant. Any user-, tag-, or network-sourced value MUST go through `PARAMS`. The PARAMS path also avoids quote-escaping bugs and DateTime locale ambiguity that concatenation invites.

Minimal C# example

```
// Preferred: feed values as tag references so PARAMS never sees the raw string.
@Tag.FilterName = filterName;       // filterName came from a Display input
@Tag.FilterMinQty = filterMinQty;
@Dataset.Query.SelectCommand.SqlStatement =
    "PARAMS @name=@Tag.FilterName, @minQty=@Tag.FilterMinQty;\n" +
    "SELECT Id, Name, Qty FROM Products WHERE Name = @name AND Qty >= @minQty";
@Dataset.Query.SelectCommand.Execute();

// The raw input never enters the SQL string; even a malformed input
// (with quotes, semicolons, or comment tokens) cannot break the parse.
```

Connection Management

Single Thread per DB

- Each DB configuration creates ONE connection
- Single execution thread per database
- Sequential command execution

Parallel Execution

For concurrent operations to same database:

```
DB1  Production Database (Thread 1)
DB2  Production Database (Thread 2)
DB3  Production Database (Thread 3)
```

Create multiple DB connections to same database for parallelism.

Connection Pooling

- Reuse existing connections
- Minimize connection overhead
- Configure pool size appropriately

Time Zone Management

Default Behavior:

- Platform stores all DateTime values as UTC
- Historian and Alarm data always in UTC
- Automatic conversion for display

Configuring External Databases:

```
DateTimeMode Settings:
- UTC: No conversion needed
- LocalTime: Platform converts automatically
- Custom: Handle in SQL statements
```

Local Time Queries:

sql

```
-- Adjust for local time zone (example: EST -5 hours)
WHERE Timestamp >= DATEADD(hour, -5, @Tag.StartTimeUTC)
```

Development vs Production

Execution Profiles

Automatic database switching for development:

Production DB	Development Override

`<name>.dbRetentive`	`<name>.dbRetentiveDev`
`<name>.dbRuntimeUsers`	`<name>.dbRuntimeUsersDev`
`<name>.dbAlarmHistorian`	`<name>.dbAlarmHistorianDev`
`<name>.dbTagHistorian`	`<name>.dbTagHistorianDev`

Configuration

1. Configure production databases normally
2. Run in Development profile
3. Data automatically redirects to Dev databases
4. No manual switching required

Security Considerations

Password Management

- Only administrators can set LogonPassword
- Passwords encrypted in configuration
- Use Windows Authentication when possible

Remote Access

Configure GatewayIP for gateway security:

- Requires TWebServices on remote computer
- Provides secure tunnel for database access
- Isolates database from direct access

Network Security Architecture

Gateway Configuration for Restricted Databases:

1. **Identify Restriction**: Database accessible only from specific servers
2. **Install Gateway**: Deploy platform with TWebServer on authorized machine
3. **Configure GatewayIP**: Point Dataset connections to gateway machine
4. **Result**: Secure database access through controlled gateway

Benefits:

- Maintains network security policies
- Centralizes database connections
- Enables audit logging
- Supports DMZ architectures

Utilities

SQLite Admin

Built-in tool for SQLite management:

- Browse tables
- Execute queries
- Import/export data

Visual Query Builder

Interactive SQL query creation:

- Drag-and-drop interface
- Join visualization
- Query testing

Store & Forward Limitations

The Store & Forward feature has specific requirements:

- **Applies Only To**: Historian and Alarm databases
- **Required Schema**: Must include control columns
- **Not Available For**: Generic application databases
- **Alternative**: Implement custom buffering for generic databases

For Store & Forward configuration, see Historian Archiving Process documentation.

Best Practices Checklist

- **Use macros** - Portable connection strings
- **Test connections** - Verify before deployment
- **Document databases** - Clear descriptions
- **Secure passwords** - Admin-only access
- **Use Dev profiles** - Protect production data
- **Regular backups** - Especially SQLite files
- **Monitor performance** - Check query execution

Troubleshooting

Connection failed:

- Verify provider installed
- Check connection string syntax
- Confirm database exists
- Test network connectivity

Access denied:

- Check credentials
- Verify permissions
- Review firewall rules

Performance issues:

- Add indexes
- Optimize queries
- Check network latency
- Review connection pooling

Related Documentation

- Datasets Query (Reference)
- Datasets Tables (Reference)
- Datasets Engine (Reference)
- MongoDB Database Connector (DRAFT v10.1.5) - document database with JSON directives, aggregation pipelines, and a SQL-subset dialect.

In this section...
