Datasets Tables Reference

Perform direct table-level operations with automatic CRUD, tag mapping, and access control.

Datasets Tables (Reference) provide direct table-level database operations with automatic CRUD functionality and tag mapping. DatasetTable enables:

- Direct table access without SQL
- Automatic INSERT, UPDATE, DELETE operations
- Row-to-tag mapping
- WhereCondition filtering
- Access control management
- Batch operations

Tables simplify database interactions by eliminating manual SQL for common operations.

Configuration Properties
Creating Tables
Access Levels
WhereCondition Configuration
Tag Mapping
Execution Methods
Advanced Operations
DataTable Usage Patterns
DateTime Handling
Display Integration
Performance Optimization
Best Practices Checklist
Troubleshooting
Related Documentation

Configuration Properties

Property	Description	Required
Name	Unique table identifier (no spaces)	Yes
DB	Database connection	Yes
TableName	Actual database table name	Yes
WhereCondition	Filter clause for operations	No
Access	Permission level (Read/Insert/ReadWrite/Unrestricted)	Yes
Mapping	Tag-to-column assignments	No
MappingDateTime	Time interpretation (UTC/Local)	No
Description	Documentation text	No

Creating Tables

1. Navigate to **Datasets Tables**
2. Click **Plus icon**
3. Configure:
    - **Name**: Unique identifier (use underscores)
    - **Description**: Documentation
4. Click **OK**
5. Set properties:
    - **DB**: Select database
    - **TableName**: Target table
    - **Access**: Permission level

Access Levels

Level	Permissions	Use Case
Read	SELECT only	Viewing data
Insert	INSERT only	Adding records
ReadWrite	SELECT, INSERT, UPDATE, DELETE	Full CRUD
Unrestricted	All operations + ALTER	Development only

WhereCondition Configuration

Filter data using SQL WHERE syntax:

Static Conditions

sql

```sql
Status = 1
Department = 'Production'
Timestamp > '2024-01-01'
```

Dynamic with Tags

sql

```sql
BatchID = {Tag.Production/Batch_ID}
Temperature > {Tag.TankFarm/Tank1/Temp}
Operator = '{Tag.Current_Operator}'
```

Complex Conditions

sql

```sql
Status IN (1, 2, 3)
  AND Department = '{Tag.Department}'
  AND Timestamp > {Tag.StartDate}
```

Tag Mapping

Configuration

1. Click **Mapping** ellipsis (...)
2. Assign columns to tags:

Column	Tag	Usage
ID	Tag.RecordID	Primary key
Name	Tag.ProductName	Display value
Count	Tag.Quantity	Numeric data
Status	Tag.Status	State tracking

Mapping Behavior

- **First row** mapped on SELECT
- **All rows** available via DataTable
- **Bidirectional** for updates
- **Auto-refresh** on changes

Tag Domain Mapping

Tag mapping occurs in the **original call domain**:

Client-Initiated Calls:

- From displays or client scripts
- Mapping uses client domain
- Results visible to specific client
- Isolated from other clients

Server-Initiated Calls:

- From server scripts or tasks
- Mapping uses server domain
- Results visible project-wide
- Shared across all clients

Domain Selection Strategy

Requirement	Domain	Example
User-specific data	Client	Personal preferences
Shared data	Server	Production values
Session data	Client	Login information
Global state	Server	System status

Execution Methods

Select Operations

csharp

```csharp
// Trigger select
@Dataset.Table.ProductionTable.Select = true;

// Get all results
DataTable data = @Dataset.Table.ProductionTable.SelectCommand();

// Check row count
int rows = @Dataset.Table.ProductionTable.RowCount;
```

Insert Operations

csharp

```csharp
// Set values in mapped tags
@Tag.ProductName = "Widget-A";
@Tag.Quantity = 100;
@Tag.Status = 1;

// Execute insert
@Dataset.Table.ProductionTable.Insert = true;
```

Update Operations

csharp

```csharp
// Modify mapped tags
@Tag.Quantity = 150;
@Tag.Status = 2;

// Execute update (uses WhereCondition)
@Dataset.Table.ProductionTable.Update = true;
```

Delete Operations

csharp

```csharp
// Set identifying values
@Tag.RecordID = 12345;

// Execute delete (uses WhereCondition)
@Dataset.Table.ProductionTable.Delete = true;
```

Advanced Operations

Batch Processing

csharp

```
public void BatchUpdate()
{
    DataTable table = @Dataset.Table.MyTable.SelectCommand();

    foreach(DataRow row in table.Rows)
    {
        row["Status"] = "Processed";
        row["ProcessDate"] = DateTime.Now;
    }

    // Update all modified rows
    @Dataset.Table.MyTable.UpdateCommand(table);
}
```

Dynamic WhereCondition

csharp

```
// Change filter at runtime
@Dataset.Table.MyTable.WhereCondition =
    string.Format("Date = '{0}'", DateTime.Today);

// Execute with new filter
@Dataset.Table.MyTable.Select = true;
```

Transaction Support

Dataset Tables do not expose transaction control to scripts. There is no Begin / Commit / Rollback API on `@Dataset` or on `@Dataset.Table.<Name>`. Each `InsertCommand()`, `UpdateCommand()` and `DeleteCommand()` is applied on its own, so if one command in a sequence fails, the commands that already succeeded are not undone.

DataTable Usage Patterns

Pattern 1: Direct Script Processing

csharp

```
DataTable result = @Dataset.Query.Query1.SelectCommand();
// Process data locally without server domain impact
foreach(DataRow row in result.Rows) {
    // Local processing
}
```

Pattern 2: Tag Distribution

csharp

```
// Assign to DataTable tag for module sharing
@Tag.MyDataTable = @Dataset.Query.Query1.SelectCommand();
// Now available to displays, reports, etc.
```

Pattern 3: Mapped Navigation

csharp

```
// Configure mapping, then navigate rows
@Dataset.Query.Query1.Select();
@Dataset.Query.Query1.Next(); // Moves to next row
```

DateTime Handling

MappingDateTime Options

Setting	Behavior	Use Case
Local	Convert to local time	User displays
UTC	Keep as UTC	Server operations

Example Configuration

csharp

```
// Store as UTC, display as local
@Dataset.Table.EventLog.MappingDateTime = "UTC";

// Tag receives UTC, display converts
@Tag.EventTime = // Automatically converted
```

Display Integration

DataGrid Binding

xml

```
<DataGrid DataSource="{Dataset.Table.ProductionTable}" />
```

ComboBox Population

csharp

```
// Fill combo with table data
@Display.ComboBox.DataSource =
    @Dataset.Table.LookupTable.SelectCommand();
```

Performance Optimization

Memory & Traffic Optimization

Control Data Volume:

- Always use WHERE clauses to filter results
- Implement pagination for large datasets
- Use SELECT specific columns instead of SELECT *
- Consider indexed views for complex queries

Resource Planning:

```
Small Query: < 1,000 rows = Minimal impact
Medium Query: 1,000-10,000 rows = Monitor memory usage
Large Query: > 10,000 rows = Implement pagination
```

Optimize WhereCondition

sql

```
-- Use indexed columns
ID = {Tag.ID}  -- Good if ID is indexed

-- Avoid functions on columns
YEAR(Date) = 2024  -- Bad
Date >= '2024-01-01' AND Date < '2025-01-01'  -- Good
```

Batch Operations

csharp

```
// Instead of multiple single updates
for(int i = 0; i < 100; i++)
{
    @Tag.ID = i;
    @Dataset.Table.MyTable.Update = true;  // Slow
}

// Use batch update
DataTable batch = PrepareData();
@Dataset.Table.MyTable.UpdateCommand(batch);  // Fast
```

Best Practices Checklist

- [ ] **Use WhereCondition** - Filter at database level
- [ ] **Set appropriate Access** - Minimum required permissions

- [ ] **Map key columns** - For updates and deletes
- [ ] **Handle nulls** - Check before operations ?
- [ ] **Sequence related writes carefully** - Each command is applied on its own; there is no rollback
- [ ] **Limit result sets** - Add TOP/LIMIT in WhereCondition
- [ ] **Index columns** - Used in WhereCondition
- [ ] **Use appropriate domains** - Client for user, Server for shared
- [ ] **Pool connections** - Reuse database connections

Troubleshooting

No data returned:

- Check WhereCondition syntax
- Verify table exists
- Confirm permissions
- Test in SQL tool

Insert/Update fails:

- Check Access level
- Verify required columns
- Review data types
- Check constraints

Mapping not working:

- Confirm column names
- Check tag types match
- Verify tag exists
- Review case sensitivity
- Verify domain selection
- Check tag existence
- Review mapping configuration
- Confirm execution context

Performance issues:

- Add database indexes
- Limit WhereCondition scope
- Use pagination
- Check network latency
- Use asynchronous execution
- Check execution plan

Related Documentation

- Datasets DBs (Reference)
- Datasets Queries (Reference)
- Datasets Engine (Reference)

In this section...
