Datasets Engine Reference

Understand the datasets engine architecture.

Reference **Modules** **Datasets** UI | Engine

Datasets Engine (Reference) manages database interactions, query execution, and data synchronization within the FrameworX runtime environment.

Advanced Topic: This document provides deep technical insight into the Dataset module execution engine. Most solutions don't require this level of understanding - the default engine behavior handles typical database operations automatically.

Overview
Concurrency & Server Domain Management
Advanced Data Management Strategies
Backup & Recovery
Production Deployment Considerations

Overview

The Dataset Engine orchestrates the internal mechanics of database operations through a multi-layered architecture. This reference covers the deep technical aspects of how the engine processes requests internally.

The engine orchestrates:

- Synchronous and asynchronous operation coordination
- Thread management and request queuing
- Client/server domain resolution
- Query execution pipeline
- Result set propagation mechanisms
- Multi-database coordination

Understanding the engine internals helps when:

- Debugging complex concurrency issues
- Optimizing for extreme performance scenarios
- Building custom extensions
- Troubleshooting edge cases

Concurrency & Server Domain Management

Understanding Server Domain Attributes

All Dataset Module properties exist in the server domain, creating a shared resource environment:

```
Example Risk Scenario:
1. Client A sets: SQLStatement = "SELECT * FROM Orders WHERE Status='Open'"
2. Client B sets: SQLStatement = "SELECT * FROM Orders WHERE Status='Closed'"
3. Execute command runs with Client B's statement (last write wins)
```

Preventing Concurrency Conflicts

Strategy 1: Dedicated Query Objects

- Create separate query objects for different operations
- Assign unique queries to specific tasks or displays
- Avoid sharing query objects between concurrent processes

Strategy 2: Synchronization Patterns

- Use semaphores or locks in scripts
- Implement request queuing for shared resources
- Design state machines for complex operations

Strategy 3: Client-Side Processing

- Execute queries in scripts with local variables
- Process DataTables before assignment to tags
- Minimize server domain property modifications

Advanced Data Management Strategies

Concurrency Patterns

The Dataset Module provides three primary patterns for handling concurrent access:

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
```

Backup & Recovery

SQLite Backup Strategies

Option 1: Command Line Backup

bash

```
sqlite3 source.db ".backup backup.db"
```

- Simple and reliable
- Requires database file access
- Best for scheduled maintenance

Option 2: Online Backup API

- Backup while database is active
- Support for incremental backups
- Progress monitoring capability

Option 3: File System Copy

- Only when database is offline
- Fastest for large databases
- Requires downtime

Backup Best Practices

1. **Schedule**: Automate backups during low-activity periods
2. **Verify**: Test restore procedures monthly
3. **Rotate**: Maintain multiple backup generations
4. **Secure**: Store backups in separate physical location
5. **Document**: Maintain restore procedure documentation

Production Deployment Considerations

Design Principles

1. **Isolation**: Use dedicated query objects for different operations
2. **Filtering**: Always limit result sets with WHERE clauses
3. **Security**: Use parameterized queries exclusively
4. **Monitoring**: Track performance metrics and errors
5. **Planning**: Design for concurrent access from the start

Production Checklist

- [ ] Before deploying to production:
- [ ] Parameterized all dynamic queries
- [ ] Implemented error handling for all operations
- [ ] Tested concurrent access scenarios
- [ ] Configured appropriate timeouts
- [ ] Established backup procedures
- [ ] Documented recovery processes
- [ ] Verified timezone handling
- [ ] Optimized query performance
- [ ] Planned for data growth
- [ ] Secured network access

In this section...
