Runtime Client-Server Domains Reference

Configure distributed runtime domains.

Reference **Solution** **Runtime** UI | Engine | Troubleshooting | Diagnostics | Client-Server

---

Runtime Client-Server Domains (Reference): Understanding the client-server architecture and domain separation in FrameworX runtime execution.

FrameworX uses a client-server architecture that clearly separates server-side processes (data acquisition, alarms, historian) from client-side processes (displays, local scripts). This separation enables:

- Distributed application deployment
- Optimized performance through appropriate process allocation
- Secure data access with proper isolation
- Flexible scaling of client connections

Domain Concepts
Namespace Architecture
Tag Domain Configuration
Module Domain Behavior
Dataset Operations
Report Operations
Script Execution
Built-in Protections
Best Practices

Domain Concepts

Server Domain

- **Scope**: Global across all modules and clients
- **Process**: TServer.exe
- **Contains**: Device communications, alarms, historian, server scripts
- **Tag Access**: Server tags only
- **Use For**: PLC data, shared values, system-wide calculations

Client Domain

- **Scope**: Local to each client computer
- **Process**: Client.exe (Rich/Smart/Web)
- **Contains**: Displays, client scripts, local variables
- **Tag Access**: Both server and client tags
- **Use For**: Operator input, display parameters, local calculations

---

Namespace Architecture

Namespace	Description	Access	Example
@Server	Server computer properties	Global to all clients	`@Server.Date`
@Client	Client computer properties	Local to each client	`@Client.Username`
Tag	All project tags	Based on tag domain	`Tag.Temperature.Value`
Device	Communication points	Server only	`Device.PLC1.Status`
Alarm	Alarm management	Server only	`Alarm.Group.Critical.TotalCount`
Dataset	Database access	Both domains	`Dataset.Query1.Execute()`

Tag Domain Configuration

Server Tags (Default)

Characteristics:

- Same value across all clients
- Synchronized automatically
- Stored in server memory

Use Cases:

- PLC/device data
- Process variables
- Alarm triggers
- Historian points

- Shared calculations

Client Tags

Characteristics:

- Local to each client
- Not synchronized
- Client memory only

Use Cases:

- Operator input fields
- Display navigation
- Report parameters
- Local calculations
- UI state variables

Module Domain Behavior

Module	Domain	Tag Access	Execution
Devices	Server only	Server tags	TServer.exe
Alarms	Server only	Server tags	TServer.exe
Historian	Server only	Server tags	TServer.exe
Displays	Client only	Server + Client	Client.exe
Script Tasks	Configurable	Based on domain	Server or Client
Script Classes	Configurable	Based on domain	Server or Client
Script Expressions	Automatic	Based on tags used	Auto-allocated

---

Dataset Operations

Synchronous Execution

csharp

```
// Executes where called (client or server)
@Dataset.Table.Table1.SelectCommand()
```

- Runs in calling domain
- Immediate execution
- Blocks until complete

Asynchronous Execution

csharp

```
// Always executes on server
@Dataset.Table.Table1.Select += 1
```

- Server-side only
- Non-blocking
- Fire-and-forget

Important: When using client tags in SQL queries or mappings, use synchronous methods from client domain.

Report Operations

Similar to datasets, reports support both synchronous and asynchronous execution:

Method	Domain	File Location	Usage
`SaveCommand()`	Calling domain	Server	Direct method call
`Save += 1`	Server only	Server	Property toggle

Files always save on the server, regardless of where the command originated.

Script Execution

Server Scripts

- Run in TServer.exe process or separate AppDomain
- Access server tags only
- Can be distributed to remote computers
- Protected execution environment

Client Scripts

- Run in client process threads
- Access both server and client tags
- Local to each client
- UI interaction capable

Calling Server Methods from Client

When calling server methods from client scripts, use async/await:

C#:

csharp

```csharp
public async Task DisplayOpening()
{
    await @Script.Class.ServerMain.Method1();
}
```

VB.NET:

vbnet

```vbnet
Public Async Function DisplayOpening() As Task
    Await @Script.Class.ServerMain.Method1()
End Function
```

Built-in Protections

Script Safety

- Automatic try-catch wrapping
- Reentrancy prevention
- Task queue management
- Thread pool optimization
- Real-time data synchronization

Performance

- Separate execution threads per task
- CPU core-based thread pooling
- Non-blocking architecture
- Automatic load balancing

Best Practices

Domain Selection

Use Server Domain for:

- Shared data across clients
- Device communications
- System-wide calculations
- Historical data

Use Client Domain for:

- User interface state
- Local preferences
- Temporary calculations
- Display parameters

Performance Optimization

- Minimize client-server tag synchronization
- Use appropriate domain for calculations
- Batch database operations
- Cache frequently accessed server data locally

Security Considerations

- Server process has database credentials
- Clients access data through server
- No direct database access from clients
- Domain separation ensures data isolation

In this section...
