Unified Namespace Reference

Model and organize your plant data in a Unified Namespace.

Reference Unified Namespace | Tutorial | How-to Guide | Reference

---

Unified Namespace (Reference) provides a centralized real-time data infrastructure where all solution variables, assets, and external data sources converge into a single hierarchical structure.

The Unified Namespace enables:

- Centralized data organization
- Hierarchical asset modeling
- Real-time data exchange
- External system integration
- Reusable data templates
- Event-driven architecture

The UNS serves as the solution's data backbone, organizing tags, assets, and connections in a unified structure accessible to all modules.

Configuration Workflow
Core Components
Runtime Architecture
Asset Tree Structure
Integration Points
Common Issues and Solutions
Namespace Properties
Security Considerations
Performance Optimization
Best Practices Checklist
Troubleshooting
Unified Namespace Links

Configuration Workflow

Typical Process

1. **Create UserTypes** - Define reusable structures
2. **Create Tags and Folders** - Organize data hierarchy
3. **Configure TagProviders** - Connect external sources
4. **Build Asset Tree** - Model physical/logical structure

Quick Actions

Action	Method
New Folder	Right-click or toolbar
New Tag	Select folder, then create
New TagProvider	Toolbar or File menu
Link External Data	Right-click folder
Move Tags	Drag and drop

Core Components

UserTypes

Create custom data types:

- Reusable structures
- Hierarchical organization
- Property inheritance
- Consistent modeling

Tags

Real-time variables with:

- Multiple data types
- Arrays support
- Retentive options
- Security settings

Enumerations

Value-to-text mappings:

```
0  "Off"
1  "On"
2  "Fault"
```

TagProviders

External connections:

- OPC UA, MQTT
- Historian systems
- Other solutions
- Dynamic discovery

Runtime Architecture

Execution Model

At startup, the UNS loads into an in-memory real-time database (TServer.exe):

- Event-driven updates
- Module data exchange
- Real-time synchronization
- Optimized performance

Data Access Methods

Direct Tag Access:

csharp

```
@Tag.Machine1.Temperature
@Tag.Line1/Pump1/Status
```

Asset() Function:

`Asset()` is the form for displays and expressions. C# and VB.NET code reads the same path with `TK.Data.AssetValue`.

```
Asset("/Line1/Machine1.Temperature")
Asset(@Client.Context.AssetPath + "/Status")
```

Asset Tree Structure

Organization Examples

By Location:

```
/Plant
  /Area1
    /Line1
      /Machine1
        Temperature
        Pressure
        Status
```

By Function:

```
/Production
  /Mixing
  /Packaging
/Utilities
  /Power
  /Water
```

Integration Points

Module Access

All modules can access UNS data:

- **Displays** - Visualization
- **Alarms** - Event monitoring
- **Historian** - Data logging
- **Scripts** - Logic execution
- **Reports** - Data presentation

External Systems

TagProviders enable:

- Protocol integration
- Historian connections
- Database links
- API interfaces

Common Issues and Solutions

Recursive Data Model Crash

Issue: Templates containing circular references **Solution:** Avoid recursive structures; validate template hierarchy

Missing External Data

Issue: TagProvider data not updating **Solution:**

- Verify connection status
- Check external system
- Review security settings
- Monitor network connectivity

Import Inconsistencies

Issue: Data missing after import **Solution:**

- Validate file format
- Check column mapping
- Verify data types
- Review import logs

Performance Issues

Issue: Slow data updates **Solution:**

- Optimize tag count
- Review update rates
- Check network latency
- Monitor resource usage

Namespace Properties

Access runtime properties:

csharp

```
// Tag count
int tagCount = @Info.Model.TagCount;

// Check if tag exists
bool exists = @Info.Model.HasTag("Machine1.Temperature");

// Get tag reference
var tag = @Info.Model.GetTag("Machine1.Temperature");
```

Security Considerations

Tag Security

- Read/Write permissions per tag

- User group restrictions
- Visibility settings (Private/Protected/Public)

Connection Security

- TagProvider authentication
- Encrypted communications
- Certificate validation

Performance Optimization

Design Guidelines

- Limit hierarchy depth (3-5 levels)
- Group related tags
- Use templates efficiently
- Optimize polling rates

Runtime Tips

- Monitor memory usage
- Check update frequencies
- Review connection counts
- Optimize queries

Best Practices Checklist

Data Structure Organization

- **Plan hierarchy** - Consider relationships carefully
- **Use templates** - Reduce redundancy
- **Clear naming** - Consistent conventions
- **Logical folders** - Intuitive navigation
- **Document structure** - Maintain documentation

System Monitoring

- **Add displays** - Monitor performance
- **Implement alarms** - Critical connections
- **Track status** - Connection health
- **Log events** - Audit trail

Troubleshooting

Connection Issues

Check network connectivity

Verify credentials

Review firewall settings

Test with simple query

Data Quality

Check Quality property (192 = good)

Verify timestamps

Review error messages

Monitor communication logs

Performance

Check CPU/memory usage

Review tag count

Optimize update rates

Monitor network traffic

Unified Namespace Links

Explanation - about foundational technologies

Platform Overview / Technology / Real-Time Tag Database

Platform Overview / Technology / Native Unified Namespace

Platform Overview / Technology / TagProvider Services

Explanation - to understand concepts

Modules / Data Foundation / Unified Namespace

Tutorials - to learn by doing

Tutorials / Data Foundation / Unified Namespace Tutorial

How-to Guides - to accomplish specific tasks

How-to Guides / Data Foundation / Unified Namespace How-to Guide

Reference - technical details

Technical Reference / Platform Modules / Unified Namespace

In this section...
