UNS Tag Discovery Reference

Automatically discover new UNS tags.

Reference **Modules** **UNS** UI | Engine | Tag Discovery

UNS Tag Discovery provides three distinct patterns for connecting external data to the Unified Namespace. Each pattern is optimized for different data characteristics and operational requirements, and production systems typically use multiple patterns simultaneously based on the specific needs of each data set.

Key Understanding: These are not developmental stages but three equally valid architectural patterns. Choose based on your data's governance needs, stability, and control requirements.

Three Connection Patterns
Pattern Selection Guide
Pattern 1: Local Tags & Device Points
Pattern 2: Linked Tags
Pattern 3: Dynamic Tags
Mixed-Mode Production Systems
Technical Implementation
Best Practices
Device Module and TagProviders — Comparison
See Also

Advanced Topic

This content covers advanced functionality not required for most projects. Start with Local Tags mapped to Devices first. Use this section when dynamic discovery or auto-managed communications is required.

Three Connection Patterns

Each pattern serves distinct operational needs:

Pattern	Control Model	Best For	Access Syntax
Local Tags & Device Points	Explicit control	Critical control, legacy protocols, regulated systems	`Tag.TagName`
Linked Tags	Governed flexibility	Stable models, changing sources, enterprise data	`Tag.TagName`
Dynamic Tags	Dynamic discovery	IIoT monitoring, ephemeral assets, unknown structures	`Asset("path")`

Pattern Selection Guide

Choose based on data characteristics, not development phase:

If Your Data...	Use This Pattern	Why
Requires deterministic polling	Local Tags & Device Points	Explicit control over timing
Uses legacy/proprietary protocols	Local Tags & Device Points	Only option for non-discovery protocols
Needs local governance with flexible sources	Linked Tags	Stable names, changing connections
Has stable structure but multiple sites	Linked Tags	Template once, link many
Is temporary or diagnostic	Dynamic Tags	No configuration overhead
Has unknown/changing structure	Dynamic Tags	Dynamic discovery at runtime
Comes from an IIoT broker or enterprise UNS	Dynamic Tags	External system governs the namespace

Pattern 1: Local Tags & Device Points

The Explicit Control Pattern

Traditional SCADA approach where you define tags locally and explicitly map them to device addresses through the Devices Module.

Characteristics:

- Full local governance of names and types

- Explicit communication configuration via Devices module (Channel Node Points)
- Deterministic polling and scan rates
- Works with any protocol (~100 available in the Devices module)
- Complete offline configuration capability

When This Pattern is Optimal:

- Regulatory compliance (FDA, EPA, etc.)
- Safety-critical control loops
- Legacy protocol integration (Modbus, DNP3, BACnet, etc.)
- Explicit timing requirements
- Full audit trail requirements

Local Tags & Device Points — Configuration Approach

1. Create Tags in UNS
2. Configure Device Channel (select protocol, set ProtocolOptions)
3. Create Device Node (set PrimaryStation — the connection endpoint)
4. Create Device Points mapped to Tags (set Address for each point)
5. Define AccessType (read, write, or read/write) and customize polling rate as needed

---

Pattern 2: Linked Tags

The Governed Flexibility Pattern

Maintain local tag definitions while linking them to external sources via the **DataLink** property. The runtime automatically manages communications when tags are accessed — on-demand, no Device Points needed.

Characteristics:

- Local governance with flexible sourcing
- Auto-managed communications via the TagProvider runtime
- On-demand activation (only communicates when tag is consumed)
- Same tag names across changing sources
- Full UNS features (alarms, historian, scaling, units) because the tag exists locally

When This Pattern is Optimal:

- Multi-site deployments with standard models
- Cloud/edge hybrid architectures
- Vendor equipment that may change
- Enterprise data integration
- Systems with many data points but only a subset is consumed at any time

Linked Tags — Configuration Approach

1. Create Tags in UNS (define Name, Type, and any metadata like alarms, historian, units)
2. Create a TagProvider connection (select protocol, set PrimaryStation)
3. Set the **DataLink** property on each tag to the external path
4. Runtime auto-manages communication — no Device Points needed

DataLink Example:

```
Tag Name:    Plant/Tank1/Level
Type:        Double
DataLink:    MyMQTT.("sensors/tank1/level")
```

When any module (display, alarm, historian) reads `Tag.Plant/Tank1/Level`, the runtime automatically activates communication to the MQTT broker path specified in DataLink. When no module is consuming the tag, communication is released.

Pattern 3: Dynamic Tags

The Dynamic Discovery Pattern

Direct binding to external paths without creating local tags. The system dynamically creates tags in the real-time database (RTDB) as paths are accessed via `Asset()` syntax. Tags are created on demand and cleaned up automatically when no longer needed.

Characteristics:

- No upfront configuration — zero local tags needed
- Dynamic discovery at runtime
- Lightweight memory footprint
- Direct path access to any node in the TagProvider's namespace
- Ideal for unknown or changing structures

When This Pattern is Optimal:

- IIoT monitoring where the broker defines the data model
- System diagnostics and troubleshooting
- Temporary test connections
- Fleet/rental equipment with changing IDs
- Exploring new data sources before committing to local tag definitions
- Enterprise UNS integration with dynamic external governance

Access Method:

Asset path access

```
// Direct access in C# code (mqtt: a folder under Plant linked to the MQTT TagProvider's root)
object value = TK.Data.AssetValue("/Plant/mqtt/line1/tank01/level");
```

Display binding

```
// Direct binding in displays (opcua: a folder under Plant linked to the OPC UA TagProvider's root)
Asset("/Plant/opcua/machine\status")
```

Mixed-Mode Production Systems

Most production systems use multiple patterns based on specific data requirements:

Typical Architecture

```
Production System
??? Local Tags
?   ??? Safety interlocks
?   ??? Control loops
?   ??? Legacy PLCs and protocols
??? Linked Tags
?   ??? Enterprise metrics
?   ??? Standard equipment (OPC UA, MQTT)
?   ??? Multi-site template data
??? Dynamic Tags
    ??? Enterprise UNS / external governance
    ??? Temporary sensors or diagnostic overlays
    ??? Maintenance tools
```



The "Typical" Architecture

Rarely does a "typical" profile match exactly the field requirements — this is why all three patterns are available. The distribution varies widely by industry and application. Reach out to the community and support as needed when applying these patterns to a specific use case.

Example Scenarios

Manufacturing Line:

- **Local Tags:** PLC control for conveyor, safety stops
- **Linked Tags:** MES data, quality metrics, OEE calculations
- **Dynamic Tags:** Portable test equipment, temporary sensors

Water Treatment Plant:

- **Local Tags:** Pump control, valve positions, flow control
- **Linked Tags:** Laboratory data, weather services, reporting
- **Dynamic Tags:** Portable analyzers, maintenance diagnostics

---

Technical Implementation

Unified Communication Stack

Linked Tags and Dynamic Tags share the same runtime infrastructure:

```
User Configuration
    Linked Tag  ? DataLink field  ? Communication Table
    Asset("path") ? Auto-created entry ? Communication Table
                            ?
                  Same Runtime Engine
                  (Polling, Subscription, Caching)
```

Communication Behavior

- **Activation** — First use triggers communication
- **Caching** — Values cached for optimal performance
- **Deactivation** — Unused paths release resources after timeout
- **Resource Management** — Automatic optimization

Error Handling

All patterns handle errors consistently:

- Failed reads return null values with Bad quality
- UI controls handle nulls gracefully
- Scripts should validate returned values
- Quality indicators available for diagnostics

Best Practices

Pattern Selection

- Analyze each data set independently
- Don't force one pattern for everything
- Consider future maintenance requirements
- Evaluate governance needs per data type

Common Pitfalls to Avoid

❌ Using Dynamic Tags for safety-critical control
❌ Forcing Local Tags when discovery would simplify
❌ Setting both DevicePoint and DataLink on the same tag
❌ Ignoring protocol capabilities (not all protocols support discovery)

Performance Considerations

- All patterns support thousands of tags
- Linked Tags and Dynamic Tags activate on-demand — only consume resources when read
- Local Tags with Device Points poll regardless of whether any module is consuming them
- Consider startup time vs runtime efficiency

Device Module and TagProviders — Comparison

While both the Devices module and TagProvider connections enable data exchange with remote systems, they are completely independent and serve different workflows. A solution can use both simultaneously.

Configuration Similarities

In both cases, you establish a connection with a remote system, and the core connection settings are essentially the same:

- **Devices:** Create a Channel (select **Protocol**, configure **ProtocolOptions**), then create a Node (configure **PrimaryStation**)
- **TagProviders:** Select a **Protocol**, and configure **ProtocolOptions** and **PrimaryStation** on the same dialog

The TagProvider combines what is a Channel and Node in Devices into a single configuration. The fields for ProtocolOptions and PrimaryStation use the same format in both cases — which is why the protocol documentation applies to both.

Where the Configuration Differs

Devices Module:

- AccessType can customize read and write cycles and events
- Multiple AccessTypes can be used on the same Node
- You create Points and map them to Tags — every address is explicitly configured

TagProvider Services:

- ReadCycle and WriteCycle apply to all communication on that connection
- No Points needed — you access addresses directly via DataLink or Asset() syntax
- Discovery-capable protocols only (~16 connectors)

Which One to Use Depends on Requirements

Best for Devices Module	Best for TagProvider Services
Local tag definition with full metadata (Min, Max, Retentive, Units, etc.)	Access data already defined in other systems
Validation of address availability and diagnostics	Dynamic data definition that adapts without manual changes
Control over communication performance, cycle times, read/write events	No detailed control required over communication messages
The solution is the Publisher that creates topics in the remote system	The solution is a subscriber — reading from an existing namespace
Regulatory compliance requiring explicit address mapping	Engineering speed — minimize configuration for monitoring use cases

In many solutions, some data is better suited for the Devices module while other data is better acquired using TagProvider Services. Both approaches coexist naturally in the same solution.

See Also

- TagProvider Services — Concept overview
- TagProvider Services Technology — Architecture deep dive, service types comparison
- UNS TagProvider Services Reference — All properties and configuration
- TagProvider Services How-to — Step-by-step procedures
- UNS Tags Reference — Tag properties including DataLink
- Devices Module — Device Points configuration

In this section...