UNS Tags Reference (10.1.5 draft)

New for 10.1.5 (draft preview). This is a preview of how the parent page will look in FrameworX 10.1.5 (planned April 2026). The parent page is the current 10.1.4 version. Content under review.

Define and manage tags in your UNS.

UNS Tags (Reference) represent real-time variables and their associated historical records, providing a structured way to model process information by linking to physical devices, databases, or calculated values.

Tags in the Unified Namespace provide:

- Real-time data representation
- Historical record association
- Device communication mapping
- Hierarchical organization
- Type-safe data handling
- Array and template support

Tags form the foundation of the solution's data model, connecting field devices to visualization, logic, and storage.

Tag Types
Configuration Properties
Ontology metadata columns (10.1.5+)
AI / MCP page action (10.1.5+)
Creating Tags
Tag Parameters, Type Specific Settings
Domain Configuration
Visibility Levels
Retentive Options
Format Strings
Array Configuration
DataTable Tags
Runtime Attributes
Additional Configuration
Best Practices Checklist
Troubleshooting

Tag Types

Built-in Types

Type	.NET Type	Description	Range
Digital	Int32	Binary value	0 or 1
Integer	Int32	Whole numbers	±2,147,483,647
Long	Int64	Extended integers	64-bit signed
Double	Double	Floating point	IEEE 754 double
Decimal	Decimal	High precision	28-29 significant digits
Text	String	Unicode text	2GB max
Json	String	JSON structured data	Built-in parsing
DateTime	DateTimeOffset	Date and time	Year 0001-9999
TimeSpan	TimeSpan	Duration	Days to milliseconds
Guid	Guid	Unique identifier	128-bit
DataTable	DataTable	In-memory table	Structured data
Image	Byte[]	Binary content	Any size
Timer	Int32	Time patterns	Multiple modes

Timer Types

Mode	Behavior	Use Case
SquareWave	Toggle 0/1 at interval	Blinking, heartbeat
Pulse	Momentary 010	Triggers, events
DelayOff	Hold value then reset	TOF timer
Comparer	Daily schedule	Time-based control

Reference Type

Dynamic tag addressing:

```
// Set reference target
@Tag.Reference1.Link = @Tag.TankFarm/Tank1/Level.GetName();

// Use reference
double level = @Tag.Reference1.Value;
```

Configuration Properties

Property	Description	Required
Name	Unique tag identifier	Yes
Type	Data type selection	Yes
Path	Asset tree location	No
Array	Array size (0 to N)	No
StartValue	Initial value at startup	No
Parameters	Type specific settings	No
Min/Max	Value limits	No
ScaleMin /Max	Engineering scale	No
Retentive	Value persistence	No
Domain	Server/Client scope	No
Visibility	External access level	No
Format	Display formatting	No
Units	Engineering units	No
Enumeration	Number to String (shows in Tag1.ValueAsString)	No
DefaultSymbol	Symbol used when dropping/pasting tags on display	No
Labels	Semicolon-delimited alternative names. Imports from `skos:altLabel`, searchable via `search_uns` and `runtime_search_uns`. **Runtime-mutable and retentive** — scripts, bindings, and (planned) operator UI can edit at runtime; writes persist across restarts. Unbounded text (10.1.5+).	No
SourceIri	Full source IRI for the individual. External-system join key. Populated by the ontology importer, editable at design time only. **Immutable at runtime** — the IRI is the tag's identity; if a rename is needed, re-import with `owl:sameAs`. Unbounded (10.1.5+).	No
Attributes	JSON column for ontology annotations not covered by the other columns (`dcterms:*`, alt-language labels keyed `rdfs:label@pt`, `skos:note`, `owl:versionInfo`, custom predicates). **Runtime-mutable and retentive** — edit via `@Tag.X.SetAttribute(...)` or the (planned) operator UI (10.1.5+).	No
Description	Documentation (1024 char)	No

Ontology metadata columns (10.1.5+)

Tags in 10.1.5 carry three ontology columns alongside their real-time data: `Labels`, `SourceIri`, and `Attributes`. These columns populate from RDF/OWL individuals imported by the ontology importer and round-trip back through **Export Tags RDF / JSON**.

- `Labels`. Semicolon-delimited alternative names. Imports from `skos:altLabel` and `skos:hiddenLabel`. Searchable via `search_uns` (Designer) and `runtime_search_uns` (runtime) with one canonical scoring algorithm.
- `SourceIri`. Full source IRI for the individual. External-system join key. Preserved on re-export, so round-trip is lossless.
- `Attributes`. JSON column for ontology annotations not covered by other columns (`dcterms:*`, alt-language labels keyed `rdfs:label@pt`, `skos:note`, `owl:versionInfo`, custom predicates).

Mutability & retention

Tag ontology columns follow a **class-vs-instance** split shared with UserType and member-def columns (see UNS UserTypes Reference (10.1.5 draft)):

Layer	SourceIri	Labels	Attributes
UserType	Design-time · immutable identity	Design-time	Design-time
Member-def	Design-time	Not modeled	Design-time
Tag (this page)	Design-time · immutable identity	Runtime-editable · retentive	Runtime-editable · retentive

Tag `Labels` and `Attributes` are the only ontology columns an operator / script / UI can mutate at runtime. Writes land in the retentive database and survive restarts. Use `promote_retentive_attributes` before export to roll runtime overlays back into the cold-start data so the exporter produces a deterministic RDF/OWL file. Bindings can *read* all three columns but writes go through the property setter (script) or the planned operator UI — never through binding-write directly. `SourceIri` is the tag's ontology identity and never changes at runtime: if a rename is needed, re-import with `owl:sameAs`.

See Industrial Ontology Integration How-to for the full model.

AI / MCP page action (10.1.5+)

When Designer is navigated to the Tags page, the following action is exposed via the `designer_action` MCP tool:

- `designer_action("export_graph", "rj")`. Mirrors the **Export Tags** toolbar dropdown. Accepts `format=json` (FrameworX native, round-trips with the Import Tags wizard), `format=rj` (W3C RDF/JSON, default for graph stores like GraphDB / Stardog / Fuseki), or `format=jsonld` (falls back to `rj` in 10.1.5 with a warning, native emitter in 10.1.6). The exported file lands in the solution's Exchange folder.

The action is listed in the `tabActions` array returned by `get_state()` only when the Tags tab is active. See Export your UNS to RDF/OWL /GraphDB for the broader export reference.

Creating Tags

Method 1: Asset Tree

1. Navigate to **Unified Namespace Asset Tree**
2. Right-click folder or click **New Tag** icon
3. Enter tag name and properties

Method 2: Tags Grid

1. Go to **Unified Namespace Tags**
2. Options:
    - Click **New Item** for dialog
    - Type directly in Name column
    - Paste from external source

Method 3: Import

- Copy from another solution
- Import from CSV files
- Use Solution Import Tools
- EngWrapper API

Tag Parameters, Type Specific Settings

Deadband

Limits value updates:

```
Absolute: +/- 5 units from last value
Percentage: 10% change required
```

EnumerationSet

Map values to text:

```
0 -> "Off"
1 -> "On"
2 -> "Auto"
```

Domain Configuration

Domain	Scope	Use Case
Server	Global across all clients	Device data, shared values
Client	Local to each session	User settings, local states

Visibility Levels

Level	External Access	Use Case
Private	None	Internal only
Protected	Read-only	Monitor only
Public	Read/Write	Full access

Affects: OPC UA Server, MQTT Broker, TcpDataAccess

Retentive Options

Setting	Saves	Use Case
None	Nothing	Temporary values
ValueOnly	Value only	Operator setpoints
Properties	All including value	Complete state
PropertiesOnly	All except value	Configuration

Storage: Dataset.DB.Retentive database

Format Strings

Numeric Formats

```
N0    -> 123        (no decimals)
N2    -> 123.45     (2 decimals)
C     -> $123.00    (currency)
P     -> 12.34%     (percentage)
X     -> 7B         (hexadecimal)
E     -> 1.23E+02   (scientific)
```

DateTime Formats

```
d     -> 3/15/2024  (short date)
T     -> 14:30:00   (long time)
yyyy-MM-dd HH:mm:ss -> 2024-03-15 14:30:00
```

Array Configuration

Array size N creates elements [0] to [N]:

```
Array = 5 creates:
Tag[0], Tag[1], Tag[2], Tag[3], Tag[4], Tag[5]
(6 elements total)
```

DataTable Tags

Using Queries

```
@Tag.DataTableTag = @Dataset.Query.MyQuery.SelectCommand();
```

Manual Population

```
@Tag.DataTableExample.StartBlockSet();
@Tag.DataTableExample[1].Column1 = value1;
@Tag.DataTableExample[2].Column2 = value2;
@Tag.DataTableExample.CommitBlockSet();
```

Runtime Attributes

Core Properties

Property	Type	Description
Value	Tag Type	Current value
Quality	Integer	OPC quality (0/64/192)
Timestamp	DateTimeOffset	Last change time

Quality Codes

- 0: Bad quality
- 64: Uncertain quality
- 192: Good quality

Additional Configuration

The TagNames are frequently referenced in these other configuration Tables:

DevicesPoints

- **Node**: Communication node
- **Address**: PLC/device address
- **AccessType**: Read/Write/ReadWrite

AlarmsItems

- **Condition**: Trigger logic
- **Groups**: Alarm categorization
- **Limits**: Threshold values

HistorianTags

- **Table**: Storage location

- **Deadband**: Recording threshold
- **DeadbandType**: Absolute/Percentage

Best Practices Checklist

- **Use meaningful names** - Follow naming conventions
- **Set appropriate types** - Match data characteristics
- **Configure retentive** - Preserve critical values
- **Apply deadbands** - Reduce unnecessary updates
- **Document tags** - Use Description field
- **Organize in assets** - Logical hierarchy
- **Set security** - Control access levels

Troubleshooting

Tag not updating:

- Check device communication
- Verify Quality = 192
- Review deadband settings
- Confirm not disabled

Wrong value format:

- Check Format string
- Verify Units configuration
- Review scale settings

Array issues:

- Confirm index in range
- Check array size setting
- Verify element access

Retentive not working:

- Check database connection
- Verify retentive setting
- Review database permissions