Binding to Tag Configuration Columns

Read UnsTags row configuration columns (Description, Units, Min, Max, Labels, SourceIri, Attributes, and more) directly from display bindings and script expressions using `{Tag.<path>.<ColumnName>}` syntax.

Version 10.1.5+.

Overview

Tag configuration columns - fields stored on the `UnsTags` row such as `Description`, `Units`, `Min`, `Max`, `Labels`, and `SourceIri` - are readable from any context where the FrameworX expression compiler runs: display element bindings, expression bindings, and Script Expressions. The same syntax used to read the tag's runtime value extends to its configuration metadata.

This collapses a class of patterns where operators previously had to mirror configuration into runtime tags or re-type configured values into the display. The configuration row is the single source of truth and the binding pulls it directly.

Syntax

Display and expression bindings

Use the standard tag-binding syntax with the column name as the terminal identifier:

```
{Tag.<path>.<ColumnName>}
```

For example, to display the engineering units configured on a tag:

```
{Tag.Tank1/Level.Units}
```

The same dotted form works inside Script Expressions (one-line server-side or client-side expressions, one row per binding) without the curly braces:

```
Tag.Tank1/Level.Units
```

Script Tasks and Classes (C# / VB)

Script Tasks and Script Classes compile through the C# / VB compilers against the runtime `TagObj` handle, which does not expose configuration columns as managed members. From these contexts, call the toolkit helper directly:

```
object units = TK.GetTagRowColumn("Tank1/Level", "Units");
object description = TK.GetTagRowColumn("Tank1/Level", "Description");
```

The helper returns `object`; cast or convert to the expected type at the call site.

Supported columns

Eighteen `UnsTags` row columns are reachable from bindings. The set is fixed - columns outside this list are not exposed even if they exist on the row.

Column	Purpose
SourceIri	RDF / OWL source identity URI for the tag (immutable design-time value, populated by GraphIO imports).
Labels	Semicolon-separated alternate names used for search and discovery.
Attributes	JSON metadata payload attached to the tag (annotations, provenance, free-form fields).
DisplayText	Human-readable label intended for display elements.
Description	Free-form description of the tag's purpose.
StartValue	Initial value loaded into the tag at runtime startup.

Min	Minimum allowed value (engineering range lower bound).
Max	Maximum allowed value (engineering range upper bound).
ScaleMin	Scaling lower bound applied between raw and engineering units.
ScaleMax	Scaling upper bound applied between raw and engineering units.
Units	Engineering units string (for example `°C`, `kPa`, `kWh`).
Format	Numeric or date format string used when rendering the value.
Visibility	Domain-scope token controlling where the tag is visible.
ActiveColor	Color rendered when a digital tag evaluates true.
InactiveColor	Color rendered when a digital tag evaluates false.
EditSecurity	Security policy gating Designer-time edit of the tag.
ReadSecurity	Security policy gating runtime read of the tag value.
WriteSecurity	Security policy gating runtime write of the tag value.

Identifier matching is case-insensitive: `units`, `Units`, and `UNITS` all resolve to the same column. The canonical casing shown above is what gets emitted into the compiled binding.

Behavior

Read-only

Row columns are readable but never writable from bindings. Assigning to one - for example `Tag.Tank1/Level.Units = "kPa"` - fails at compile time. To change a configuration column, use the Designer Tags grid, `write_objects` through DesignerMCP or ConsoleMCP, or any other authoring path that goes through the standard configuration write surface.

UDT member precedence

When the tag is typed by a UserType (UDT) and a UDT member has the same name as a row column, the UDT member wins. `Tag.MyTag.Description` on a UDT-typed tag whose UDT defines a `Description` member resolves to that member's value, not to the configuration row's `Description` column.

This makes UDT design the override surface: define a member to take over a name; leave the member off to fall through to the row column.

Missing tag, missing column, empty cell

If the tag path does not exist, the column is not present in the schema (older schema version), or the cell is empty, the binding silently renders empty. No exception is raised at runtime. Treat empty results as a probable typo or schema-version mismatch and verify the configured value through the Designer Tags grid.

Examples

Show the engineering units next to a value

A TextBox text bound to `{Tag.Tank1/Level.Value} {Tag.Tank1/Level.Units}` shows `72.4 kPa` when the tag's value is 72.4 and its `Units` column is `kPa`.

Drive a color from the tag's configured ActiveColor

A Rectangle FillColor dynamic bound to `Tag.Pump1/Status.ActiveColor` uses the color stored on the tag's configuration row, so palette changes propagate through configuration without re-touching every display.

Use the configured description as a tooltip

A control's ToolTip bound to `{Tag.Tank1/Level.Description}` surfaces the tag's documented purpose without duplicating the text into the display element.

Read a configuration column from a Script Task

```
string units = TK.GetTagRowColumn("Tank1/Level", "Units") as string;
string description = TK.GetTagRowColumn("Tank1/Level", "Description") as string;
```

Known limitation - Script Tasks and Classes

The `Tag.<path>.<ColumnName>` dotted-suffix syntax works only where the FrameworX expression compiler runs: display bindings, expression bindings, and Script Expressions. C# and VB Script Tasks and Classes compile through the standard managed compilers (Roslyn for C#, the VB compiler for VB) against the live `TagObj` runtime handle, which does not expose row columns as managed members.

Writing `@Tag.Tank1/Level.SourceIri` in a C# Script Task fails with:

```
CS1061: 'TagObj' does not contain a definition for 'SourceIri'
```

The VB equivalent fails with `BC30456`. Use the toolkit helper `TK.GetTagRowColumn(tagPath, columnName)` from Script Tasks and Classes - it reads the same configuration row through the same lookup path.
