Import an OWL/RDF ontology into your UNS

Bring an RDF/OWL ontology into FrameworX as UserTypes, Tags, and AssetTree folders.

Technical Reference **Solution** **Designer** **Settings and Tools** **Import Tags** OWL/RDF Ontology

Version 10.1.5+

N-Triples as the mental model. The Unified Namespace is a triple store by construction. Every UserType, Tag, AssetTree folder, and member that this wizard writes corresponds to one or more RDF triples of the form `<subject> <predicate> <object> .` — the W3C N-Triples textual form. The import here is a serialization step onto that underlying triple model. See Industrial Ontology Integration How-to for the full standards map.

Prerequisites

- A solution open in Designer (new or existing).
- A source file: RDF/JSON (`.rj`). JSON-LD and Turtle support arrives in a future release.
- (Optional) a policy template. Pick `iof`, `isa88`, or write your own JSON. Defaults work for generic OWL files.

Primary path: the Designer wizard

1. In Designer, go to **Solution Import Tags Graph / RDF file**.
2. Pick the source file. The wizard previews class count, individual count, and any warnings.
3. Pick a policy (`iof` / `isa88` / `custom`) or leave blank for default.
4. Choose the mode. **Auto** is the default and inspects the file to decide schema-only, instances-only, or both.
5. Click **Import**. UserTypes and Tags appear in the UNS with full ontology metadata populated.

Alternative path: AI-driven with the MCP tool

If the AI assistant is connected to Designer, use the skill Skill Import Industrial Ontology. The assistant runs the same engine via the `import_graph_mode` MCP tool.

```
import_graph_model(source="file", file_path="C:/.../biotech.rj", policy="iof", dry_run=true)
# review the summary, then re-run with dry_run=false
```

Always run `dry_run=true` once before committing. The preview reports class count, instance count, `BaseUserType` cycle warnings, and label collisions.

What the import writes

OWL construct	FrameworX column
`owl:Class`	New row in **UnsUserTypes**
`owl:NamedIndividual`	New row in **UnsTags** at path `<containment>/<entity>/Attr`
`owl:ObjectProperty` / `owl:DatatypeProperty`	Member row inside the UDT's member-def table
Primary `rdfs:label` / `skos:prefLabel`	`DisplayText`
`skos:altLabel` / `skos:hiddenLabel` / custom alt-names	`Labels` (semicolon-delimited)
`rdfs:comment` / `skos:definition`	`Description`
Class / individual / property IRI	`SourceIri` (full IRI, never truncated)
`rdfs:subClassOf <named class>`	`BaseUserType` (UDT only)
Everything else: `dcterms:*`, alt-language `rdfs:label`, `skos:note`, `owl:versionInfo`, custom predicates	`Attributes` JSON column, keyed as `prefix:localName` (with `@lang` suffix for language tags)

The `/Attr` convention

Every typed OWL individual imports as a Tag named `<containmentPath>/<entityName>/Attr`. AssetTree folders auto-create from the slashes. The importer never writes folder rows directly. The `/Attr` leaf collects the entity's typed members so one click reveals everything about the entity in one place. On re-export, the `/Attr` suffix is stripped so OWL entity IRIs reflect identity, not storage layout.

Common gotchas

- **Base-class cycles** (`A subClassOf B`, `B subClassOf A`). FX single-parent `BaseUserType` does not model cycles. Neither class gets `BaseUserType` set. Duck-typed member copy-down still flattens members from both ancestors. Look for `"cycle detected"` in the warnings.
- **Multi-parent inheritance**. OWL allows multiple parents. FX picks the first named parent for `BaseUserType`, the rest get warning-logged. All ancestors' members are still copied down.
- **Re-import collision**. By default, user-authored `Description` text is preserved (`descriptionCollisionPolicy: "skip"`). Change to `"overwrite"` if the ontology is the source of truth.

Related: Industrial Ontology Integration How-to · Export your UNS to RDF/OWL/GraphDB · Generate a visual report of your UNS · Local AI Ontology Demo

---