Industrial Ontology Integration How-to

Bring RDF/OWL ontologies into FrameworX as UserTypes, Tags, and AssetTree folders; round-trip the full Unified Namespace back to a graph file for diff, archive, or upload to a triple store.

How-to Guides **Compliance** Industrial Ontology

Version 10.1.5+

Industrial ontology integration in FrameworX is a layered stack of W3C Semantic Web standards, consumed and emitted by the built-in graph import/export engine. The engine reads an RDF graph, classifies each triple against a small set of supported predicates, and materializes the result into native Unified Namespace rows (UserTypes, Tags, AssetTree folders, UDT members). Round-trip is symmetric at the data-model level: the exporter walks the same UNS and emits an equivalent graph.

This section covers the standards actually implemented in 10.1.5, the policy templates that ship with the product, and the pages that show you how to use them.

Standards Supported

The following W3C standards have first-class implementations in 10.1.5. Every row is backed by real parsing or emitting code in the graph import /export engine, not by marketing claim. Predicates listed here are consumed by the importer and restored by the exporter on round-trip.

Standard	Role in FrameworX	Status
RDF 1.1 (W3C)	Core triple model. Every ontology artifact in the UNS traces back to an RDF triple.	Full
RDF Schema (RDFS) (W3C)	`rdfs:subClassOf` drives the `BaseUserType` chain; `rdfs:label` populates `DisplayText`; `rdfs:comment` populates `Description`; `rdfs:domain` and `rdfs:range` are preserved on member definitions.	Full
OWL 2 (W3C)	`owl:Class` UserType, `owl:NamedIndividual` Tag, `owl:ObjectProperty` and `owl:DatatypeProperty` UDT members. Blank-node class expressions walked for `owl:Restriction`, `owl:intersectionOf`, and `owl:unionOf`. TBox fundamentals only — OWL reasoning is not performed.	Full (TBox subset)
SKOS (W3C)	`skos:prefLabel` `DisplayText`; `skos:altLabel` and `skos:hiddenLabel` `Labels` (semicolon-delimited); `skos:definition` `Description`; `skos:note`, `skos:scopeNote`, and `skos:example` preserved in `Attributes`.	Full (label and definition subset)
XSD datatypes 1.1 (W3C)	Literal datatype mapping for member definitions. `xsd:string`, `xsd:int`, `xsd:long`, `xsd:double`, `xsd:decimal`, `xsd:boolean`, `xsd:dateTime`, `xsd:anyURI`, and the full common-types set map to FrameworX native types.	Full (common types)
RDF/JSON (`.rj`, W3C WG Note)	W3C RDF/JSON. Default round-trip format — the only format both the importer and exporter speak in 10.1.5. Deterministic ordering, blank-node support, language tags, and typed literals round-trip cleanly.	Full (import + export)
JSON-LD 1.1 (`.jsonld`, W3C Recommendation)	Flattened form with top-level `@context` + `@graph`. Predicate IRIs compact to `prefix:local` against the standard prefixes (rdf, rdfs, owl, xsd, skos, dcterms, dc) plus the solution's `@vocab`. `rdf:type` folds to `@type`. Datatype-tagged and language-tagged literals serialize via `@value` + `@type` / `@language`.	Full (export only in 10.1.5; import via dotNetRDF in a follow-up release)
Turtle (`.ttl`, W3C Recommendation)	Prefix-abbreviated subject-grouped serialization with `;` and `,` shorthand and a for `rdf:type`. The most-used RDF format in industrial-ontology workflows (IOF, ISA-88).	Full (export only in 10.1.5; import via dotNetRDF in a follow-up release)
N-Triples (`.nt`, W3C Recommendation)	One fully-expanded triple per line, no prefix abbreviation. The textual form that mirrors the UNS triple-store mental model. Most universally accepted; best for diff and audit.	Full (export only in 10.1.5; import via dotNetRDF in a follow-up release)

Importer / exporter format asymmetry (10.1.5)

The exporter ships all four W3C-standard RDF formats in 10.1.5 GA: RDF/JSON, JSON-LD 1.1, Turtle, and N-Triples. All four are pure-serialization writers sharing the same triple stream — choosing a format is purely a serialization decision; the exported RDF semantics are identical.

The importer is **format-asymmetric** in 10.1.5: RDF/JSON is the only parser. JSON-LD / Turtle / N-Triples readers are queued for a follow-up release backed by dotNetRDF. Practical implications:

- **Round-trip checks** (export then re-import to verify) must use RDF/JSON end-to-end.
- **Publishing to a triple store** works with any of the four formats — pick whichever your destination accepts.
- **Sharing with an ontology team** that wants Turtle or JSON-LD: export those directly. The receiving team can convert back to RDF/JSON via dotNetRDF, Apache Jena, or rdflib if they need to import into FrameworX before the importer parsers ship.

Related and planned standards

Other standards that the ontology feature interacts with, or that are on the roadmap. These are documented here for context.

- **Dublin Core Terms** (`dcterms:*`, `dc:*`) — annotation predicates are round-tripped through the `Attributes` JSON column keyed as `prefix:localName`. No term-specific handling; every `dcterms:*` predicate is preserved uniformly as an opaque annotation.
- **RDF/XML** — planned for a future release on both import and export sides via dotNetRDF.
- **SPARQL** — the exporter can wrap its output in a SPARQL `INSERT DATA {…}` envelope for upload to a triple store. `SELECT` queries are the caller's job; the engine does not parse or evaluate SPARQL queries. Fetch query results through WebData and pass the response to the importer as inline content.
- **SHACL**, **PROV-O**, **QUDT** — not implemented. Shape validation, provenance tracking, and unit quantification are not performed by the import engine. If your source graph includes these vocabularies, their triples are either skipped or preserved as opaque annotations depending on predicate shape.

Policy templates shipped with the product

Two starter policy templates ship in 10.1.5 to simplify onboarding from well-known industrial vocabularies. Both are JSON configuration files consumed by the generic import engine — they set prefix mappings, module routing, typed-container predicates, and containment predicates, but do not introduce standard-specific code paths. Pick one as a starting point and edit the JSON to fit your ontology.

- **IOF** (Industrial Ontologies Foundry) — `policy-iof.json`. Real IOF IRIs: BFO_0000051 for containment, module routing for IOF_Core, IOF_Biopharma, IOF_SupplyChain, IOF_Construct, and the `IOF_` prefix. Suitable as a starter for any IOF-derived ontology.
- **ISA-88** (IEC 61512, batch control) — `policy-isa88.json`. `s88:` prefix, containment mappings (`hasSite`, `hasArea`, `hasProcessCell`, `hasUnit`, `hasEquipmentModule`), and a `typeToUserType` map covering the ISA-88 physical and procedural models. The Local AI Ontology Demo uses this template end-to-end.

FrameworX architecture and ontology alignment

The UNS is a triple store by construction. Every row in the Unified Namespace — every UserType, Tag, AssetTree folder, UDT member, and metadata value — corresponds to one or more RDF triples. N-Triples, the W3C line-based textual form of a triple (`<subject> <predicate> <object> .`), is the canonical way to describe and reason about that structure.

The Unified Namespace is the landing surface for imported ontology content. The mapping is intentionally small, so that graph files stay readable and round-trips stay symmetric:

- **UserTypes** hold class-level metadata. Every `owl:Class` becomes one row; `rdfs:subClassOf` sets `BaseUserType`; cycle detection prevents infinite loops; multi-parent OWL graphs pick the first named parent and duck-type the rest.
- **Tags** hold instance-level metadata. Every `owl:NamedIndividual` becomes one Tag at path `<containmentPath>/<entityName>/Attr`. The `/Attr` convention is enforced on import and stripped on export so that OWL entity IRIs reflect identity, not storage layout.
- **AssetTree folders** are materialized automatically from the slashed paths. The importer never writes folder rows directly.
- **UDT members** hold property-level metadata. Each `owl:ObjectProperty` or `owl:DatatypeProperty` on a class becomes one member; on export, the original predicate IRI is restored from the `SourceIri` column.
- **Metadata columns** preserve annotations. `DisplayText`, `Labels`, `SourceIri`, `Description`, and the `Attributes` JSON blob together hold everything a round-trip needs.

Related pages

- Import an OWL/RDF ontology into your UNS. Designer wizard and MCP reference for the import step, including the full OWL-to-FrameworX predicate mapping table.
- Export your UNS to RDF/OWL/GraphDB. Format selector, round-trip guarantees, and SPARQL `INSERT DATA` envelope details.
- Generate a visual report of your UNS. Relationship Graph button and Mermaid output reference.
- Local AI Ontology Demo. Reference solution shipping with 10.1.5 — a hand-authored ISA-88 mini-ontology that exercises every column in the 10.1.5 schema additions.

In this section...
