UML to Equivalent OWL in OWL Functional Syntax

There are various syntaxes available for encoding OWL ontologies. The Concept Modeler can export UML to an OWL ontology using the following syntaxes:

- **RDF/XML**. It is the original standard syntax for writing RDF (Resource Description Framework), which is a general-purpose language for representing information on the Web. Although it is wordy and difficult to read, it is the only syntax written in various concrete formats called RDF serialization formats, and it is mandatory to be supported by OWL 2 tools. It provides an XML representation of an RDF graph. This syntax is the default syntax used in the Concept Modeler.
- **JSON-LD** or JavaScript Object Notation for Linked Data is a method of encoding linked data using JSON, which is a concrete RDF syntax. JSON-LD is used to map JSON terms (keys and values) to IRIs, giving them a global context. A JSON-LD document is both an RDF document and a JSON document and correspondingly represents an instance of an RDF data model.
- **OWL Functional**. It is a simple text-based syntax designed to be easier for specification purposes, and to provide a foundation for the implementation of OWL 2 tools such as APIs and reasoners. It is used in most of the OWL 2 specification documents as the primary presentation syntax that translates the structural specification into other concrete syntaxes. A functional-style syntax ontology document consists of sequences of Unicode characters and is encoded in UTF-8.
- **Turtle**. A concrete syntax for RDF, Turtle (Terse RDF Triple Language) is a plain-text RDF representation. It is more concise, and easier to read and edit manually than RDF/XML. A Turtle document is a collection of RDF-triples, which groups three URIs to make a triple and provides ways to abbreviate information, e.g., by factoring out common portions of URIs. Each triple has the format: <subject> <predicate> <object>. Each statement ends with a period, and each element in the triple is an URI, except the <object>, which can be a bit of text or a number.
- **Manchester**. It provides a compact textual-based representation of OWL ontologies that is easy to read and write. It uses IRIs as term identifiers. The syntax for entering and displaying annotations and descriptions in the Manchester OWL syntax closely corresponds to the syntax in the OWL Functional syntax. A Manchester OWL document consists of sequences of Unicode characters, and is encoded in UTF-8.

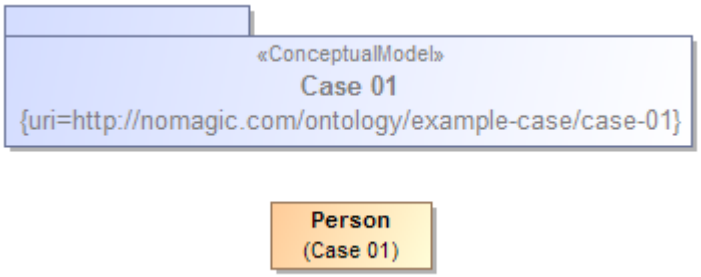
This section gives examples that show the transformation of UML modeled in the Concept Modeler to an exported OWL ontology. The OWL ontologies are presented in OWL Functional Syntax.

Note

- A model may contain elements, (e.g. classes, properties, or datatypes) that belong to other models. When exporting the model, the Concept Modeler will include the OWL declaration of the elements that exist in the current model only, not those of the other models.
- However, if the entity that belongs to another model is an object property with an inverse property defined, you will see the OWL declaration of the inverse property in the current OWL ontology upon export.

Class

For a simple UML class, the diagram below shows that the ontology is transformed as the package containing the UML class. For the sake of brevity, subsequent diagrams do not show the package in the diagram.



A class diagram in Concept Modeler.

```
Ontology(<http://nomagic.com/ontology/example-case/case-01>
                Declaration(
                                Class(:Person)
                )
                AnnotationAssertion(rdfs:label :Person "Person"@en)
)
```

Related pages

- Setting the concept model export syntax
- Exporting your concept model to an OWL ontology
- Usage

- Class
- Class generalization
- Class with Datatype property
- Class with object property
- Class with Self-Referential Object Property
- Class with object property without range
- Class with subproperty
- Class with universal quantification constraint on property I

- Class with universal quantification constraint on property II
- Class with existential quantification constraint on property
- Class with subproperty without a range
- Class with necessary and sufficient property
- Class with property having unspecified multiplicity
- Class with inverse property
- Class with Asymmetric Object Property
- Class with Functional Object Property
- Class with Inverse Functional Object Property
- Class with Irreflexive Object Property
- Class with Reflexive Object Property
- Class with symmetric object property
- Class with Transitive Object Property
- Generalization with disjoint subclasses
- Generalization with subclass completeness
- Anonymous union class
- Association classes
- Property holder with datatype property
- Property holder with self-referential object property
- Property holder with object property
- Property holder with self-referential subproperty
- Property holder with subproperty
- Property with a maximum but no minimum cardinality
- Property with multiple domains and ranges
- Annotation and annotation property
- Asymmetrical inverse property
- Disjoint classes
- Property chains
- Equivalent property
- Equivalent class
- Property restriction from a different namespace
- Necessary and sufficient conditions of anonymous subclasses