



OWL Web Ontology Language Guide

W3C Recommendation 10 February 2004

This version:

<http://www.w3.org/TR/2004/REC-owl-guide-20040210/>

Latest version:

<http://www.w3.org/TR/owl-guide/>

Previous version:

<http://www.w3.org/TR/2003/PR-owl-guide-20031215/>

Editors:

Michael K. Smith, Electronic Data Systems, michael.smith@eds.com

Chris Welty, IBM Research, chris.welty@us.ibm.com

Deborah L. McGuinness, Stanford University, dln@ksl.stanford.edu

Please refer to the [errata](#) for this document, which may include some normative corrections.

See also [translations](#).

Copyright © 2004 W3C[®] (MIT, ERCIM, Keio), All Rights Reserved. W3C [liability](#), [trademark](#), [document use](#) and [software licensing](#) rules apply.

Abstract

The World Wide Web as it is currently constituted resembles a poorly mapped geography. Our insight into the documents and capabilities available are based on keyword searches, abetted by clever use of document connectivity and usage patterns. The sheer mass of this data is unmanageable without powerful tool support. In order to map this terrain more precisely, computational agents require machine-readable descriptions of the content and capabilities of Web accessible resources. These descriptions must be in addition to the human-readable versions of that information.

The OWL Web Ontology Language is intended to provide a language that can be used to describe the classes and relations between them that are inherent in Web documents and applications.

This document demonstrates the use of the OWL language to

1. formalize a domain by defining classes and properties of those classes,
2. define individuals and assert properties about them, and

3. reason about these classes and individuals to the degree permitted by the formal semantics of the OWL language.

The sections are organized to present an incremental definition of a set of classes, properties and individuals, beginning with the fundamentals and proceeding to more complex language components.

Status of This Document

This document has been reviewed by W3C Members and other interested parties, and it has been endorsed by the Director as a [W3C Recommendation](#). W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

This is one of [six parts](#) of the W3C Recommendation for OWL, the Web Ontology Language. It has been developed by the [Web Ontology Working Group](#) as part of the [W3C Semantic Web Activity](#) ([Activity Statement](#), [Group Charter](#)) for publication on 10 February 2004.

The design of OWL expressed in earlier versions of these documents has been widely reviewed and satisfies the Working Group's [technical requirements](#). The Working Group has addressed [all comments received](#), making changes as necessary. Changes to this document since [the Proposed Recommendation version](#) are detailed in the [change log](#).

Comments are welcome at public-webont-comments@w3.org ([archive](#)) and general discussion of related technology is welcome at www-rdf-logic@w3.org ([archive](#)).

A list of [implementations](#) is available.

The W3C maintains a list of [any patent disclosures related to this work](#).

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the [W3C technical reports index](#) at <http://www.w3.org/TR/>.

Contents

- [Abstract](#)
- [Status of This Document](#)
- [Contents](#)
- [1. Introduction](#)
 - [1.1. The Species of OWL](#)
 - [1.2. Structure of the Document](#)
- [2. The Structure of Ontologies](#)
 - [2.1. Namespaces](#)
 - [2.2. Ontology Headers](#)
 - [2.3. Data Aggregation and Privacy](#)
- [3. Basic Elements](#)
 - [3.1. Simple Classes and Individuals](#)
 - [3.1.1. Simple Named Classes](#)
 - [3.1.2. Individuals](#)
 - [3.1.3. Design for Use](#)

- [3.2. Simple Properties](#)
 - [3.2.1. Defining Properties](#)
 - [3.2.2. Properties and Datatypes](#)
 - [3.2.3. Properties of Individuals](#)
 - [3.3. Property Characteristics](#)
 - [3.3.1. TransitiveProperty](#)
 - [3.3.2. SymmetricProperty](#)
 - [3.3.3. FunctionalProperty](#)
 - [3.3.4. inverseOf](#)
 - [3.3.5. InverseFunctionalProperty](#)
 - [3.4. Property Restrictions](#)
 - [3.4.1. allValuesFrom, someValuesFrom](#)
 - [3.4.2. Cardinality](#)
 - [3.4.3. hasValue](#)
 - [4. Ontology Mapping](#)
 - [4.1. Equivalence between Classes and Properties](#)
 - [4.2. Identity between Individuals](#)
 - [4.3. Different Individuals](#)
 - [5. Complex Classes](#)
 - [5.1. Set Operators](#)
 - [5.2. Enumerated Classes](#)
 - [5.3. Disjoint Classes](#)
 - [6. Ontology Versioning](#)
 - [7. Usage Examples](#)
 - [7.1. Wine Portal](#)
 - [7.2. Wine Agent](#)
 - [Acknowledgements](#)
 - [OWL Glossary](#)
 - [Term Index and Cross Reference](#)
 - [References](#)
 - [Appendix A: XML + RDF Basics](#)
 - [Appendix B: History](#)
 - [Appendix C: Change Log Since Last Call Release](#)
-

1. Introduction

"Tell me what wines I should buy to serve with each course of the following menu. And, by the way, I don't like Sauternes."

It would be difficult today to construct a Web agent that would be capable of performing a search for wines on the Web that satisfied this query. Similarly, consider actually assigning a software agent the task of making a coherent set of travel arrangements. (For more use cases see the [OWL requirements document](#).)

To support this sort of computation, it is necessary to go beyond keywords and specify the meaning of the resources described on the Web. This additional layer of interpretation captures the *semantics* of the data.

The OWL Web Ontology Language is a language for defining and instantiating *Web ontologies*. *Ontology* is a term borrowed from philosophy that refers to the science of describing the kinds of entities in the world and how they are related. An *OWL ontology* may include descriptions of *classes*, *properties* and their instances. Given such an ontology, the [OWL formal semantics](#) specifies how to derive its logical consequences, i.e. facts not literally present in the ontology, but *entailed* by the semantics. These entailments may be based on a single document or multiple distributed documents that have been combined

using defined [OWL mechanisms](#).

This document is one component of the description of OWL, the Web Ontology Language, being produced by the W3C Web Ontology Working Group (WebOnt). The [Document Roadmap](#) section of the Overview ([\[Overview\]](#), 1.1) describes each of the different parts and how they fit together.

One question that comes up when describing yet another XML/Web standard is "What does this buy me that XML and XML Schema don't?" There are two answers to this question.

- An ontology differs from an XML schema in that it is a knowledge representation, not a message format. Most industry based Web standards consist of a combination of message formats and protocol specifications. These formats have been given an operational semantics, such as, "Upon receipt of this `PurchaseOrder` message, transfer `Amount` dollars from `AccountFrom` to `AccountTo` and ship `Product`." But the specification is not designed to support reasoning outside the transaction context. For example, we won't in general have a mechanism to conclude that because the `Product` is a type of Chardonnay it must also be a white wine.
- One advantage of OWL ontologies will be the availability of tools that can reason about them. Tools will provide *generic* support that is not specific to the particular subject domain, which would be the case if one were to build a system to reason about a specific industry-standard XML schema. Building a sound and useful reasoning system is not a simple effort. Constructing an ontology is much more tractable. It is our expectation that many groups will embark on ontology construction. They will benefit from third party tools based on the formal properties of the OWL language, tools that will deliver an assortment of capabilities that most organizations would be hard pressed to duplicate.

1.1. The Species of OWL

The OWL language provides three increasingly expressive sublanguages designed for use by specific communities of implementers and users.

- *OWL Lite* supports those users primarily needing a classification hierarchy and simple constraint features. For example, while OWL Lite supports cardinality constraints, it only permits cardinality values of 0 or 1. It should be simpler to provide tool support for OWL Lite than its more expressive relatives, and provide a quick migration path for thesauri and other taxonomies.
- *OWL DL* supports those users who want the maximum expressiveness without losing computational completeness (all entailments are guaranteed to be computed) and decidability (all computations will finish in finite time) of reasoning systems. OWL DL includes all OWL language constructs with restrictions such as type separation (a class can not also be an individual or property, a property can not also be an individual or class). OWL DL is so named due to its correspondence with *description logics* [\[Description Logics\]](#), a field of research that has studied a particular decidable fragment of first order logic. OWL DL was designed to support the existing Description Logic business segment and has desirable computational properties for reasoning systems.
- *OWL Full* is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For example, in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right. Another significant difference from OWL DL is that a `owl:DatatypeProperty`

can be marked as an `owl:InverseFunctionalProperty`. OWL Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary. It is unlikely that any reasoning software will be able to support every feature of OWL Full.

Each of these sublanguages is an extension of its simpler predecessor, both in what can be legally expressed and in what can be validly concluded. The following set of relations hold. Their inverses do not.

- Every legal OWL Lite ontology is a legal OWL DL ontology.
- Every legal OWL DL ontology is a legal OWL Full ontology.
- Every valid OWL Lite conclusion is a valid OWL DL conclusion.
- Every valid OWL DL conclusion is a valid OWL Full conclusion.

Ontology developers adopting OWL should consider which species best suits their needs. The choice between OWL Lite and OWL DL depends on the extent to which users require the more expressive restriction constructs provided by OWL DL. Reasoners for OWL Lite will have desirable computational properties. Reasoners for OWL DL, while dealing with a decidable sublanguage, will be subject to higher worst-case complexity. The choice between OWL DL and OWL Full mainly depends on the extent to which users require the meta-modelling facilities of RDF Schema (i.e. defining classes of classes). When using OWL Full as compared to OWL DL, reasoning support is less predictable. For more information about this issue see the [OWL semantics document](#).

Users migrating from RDF to OWL DL or OWL Lite need to take care to ensure that the original RDF document complies with the constraints imposed by OWL DL and OWL Lite. The details of these constraints are explained in [Appendix E](#) of the OWL Reference.

When we introduce constructs that are only permitted in OWL DL or OWL Full, they are marked by "[OWL DL]".

1.2. Structure of the Document

In order to provide a consistent set of examples throughout the guide, we have created a [wine](#) and [food](#) ontology. This is an OWL DL ontology. Some of our discussion will focus on OWL Full capabilities and is so marked. The wine and food ontology is a significant modification of an [element](#) of the DAML ontology library with a long history. It was originally developed by McGuinness as a CLASSIC description logic [example](#), expanded to a description logic [tutorial](#), and expanded to an ontology [tutorial](#).

In this document we present examples using the [RDF/XML syntax](#) ([RDF], 5), assuming XML will be familiar to the largest audience. The normative OWL exchange syntax is RDF/XML. Note that OWL has been designed for maximal compatibility with RDF and RDF Schema. These XML and RDF formats are part of the OWL standard.

All of the examples presented in this document are taken from the ontologies contained in [wine.rdf](#) and [food.rdf](#), except those marked with \rightarrow in the bottom right corner.

2. The Structure of Ontologies

OWL is a component of the [Semantic Web](#) activity. This effort aims to make Web resources more readily accessible to automated processes by adding information about the resources that describe or provide Web content. As the Semantic Web is inherently distributed, OWL must allow for information to be gathered from distributed sources. This is partly done by allowing ontologies to be related, including explicitly importing information

from other ontologies.

In addition, OWL makes an *open world* assumption. That is, descriptions of resources are not confined to a single file or scope. While class `c1` may be defined originally in ontology `O1`, it can be extended in other ontologies. The consequences of these additional propositions about `c1` are *monotonic*. New information cannot retract previous information. New information can be contradictory, but facts and entailments can only be *added*, never *deleted*.

The possibility of such contradictions is something the designer of an ontology needs to take into consideration. It is expected that tool support will help detect such cases.

In order to write an ontology that can be interpreted unambiguously and used by software agents we require a syntax and formal semantics for OWL. OWL is a vocabulary extension [[RDF Semantics](#)] of RDF. The OWL semantics are defined in [OWL Web Ontology Language Semantics and Abstract Syntax](#).

2.1. Namespaces

Before we can use a set of terms, we need a precise indication of what specific vocabularies are being used. A standard initial component of an ontology includes a set of [XML namespace](#) declarations enclosed in an opening `<rdf:RDF` tag. These provide a means to unambiguously interpret identifiers and make the rest of the ontology presentation much more readable. A typical OWL ontology begins with a [namespace declaration](#) similar to the following. Of course, the URIs of the defined ontologies will not usually be `w3.org` references.

```
<rdf:RDF
  xmlns      = "http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#"
  xmlns:vin  = "http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#"
  xml:base   = "http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#"
  xmlns:food = "http://www.w3.org/TR/2004/REC-owl-guide-20040210/food#"
  xmlns:owl  = "http://www.w3.org/2002/07/owl#"
  xmlns:rdf  = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#"
  xmlns:xsd  = "http://www.w3.org/2001/XMLSchema#">
```

The first two declarations identify the namespace associated with this ontology. The first makes it the *default* namespace, stating that unprefixed qualified names refer to the current ontology. The second identifies the namespace of the current ontology with the prefix `vin:`. The third identifies the base URI for this document (see [below](#)). The fourth identifies the namespace of the supporting food ontology with the prefix `food:`.

The fifth namespace declaration says that in this document, elements prefixed with `owl:` should be understood as referring to things drawn from the namespace called `http://www.w3.org/2002/07/owl#`. This is a conventional OWL declaration, used to introduce the OWL vocabulary.

OWL depends on constructs defined by RDF, RDFS, and XML Schema datatypes. In this document, the `rdf:` prefix refers to things drawn from the namespace called `http://www.w3.org/1999/02/22-rdf-syntax-ns#`. The next two namespace declarations make similar statements about the RDF Schema (`rdfs:`) and XML Schema datatype (`xsd:`) namespaces.

As an aid to writing lengthy URLs it can often be useful to provide a set of entity definitions in a document type declaration (DOCTYPE) that precedes the ontology definitions. The

names defined by the namespace declarations only have significance as parts of XML tags. Attribute values are *not* namespace sensitive. But in OWL we frequently reference ontology identifiers using attribute values. They can be written down in their fully expanded form, for example "http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#merlot". Alternatively, abbreviations can be defined using an ENTITY definition, for example:

```
<!DOCTYPE rdf:RDF [
  <!ENTITY vin "http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#" >
  <!ENTITY food "http://www.w3.org/TR/2004/REC-owl-guide-20040210/food#" > ]>
```

After this pair of ENTITY declarations, we could write the value "&vin;merlot" and it would expand to "http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#merlot".

Perhaps more importantly, the `rdf:RDF` namespace declarations can then be simplified so that changes made to the entity declarations will propagate through the ontology consistently.

```
<rdf:RDF
  xmlns      ="&vin;"
  xmlns:vin ="&vin;"
  xml:base   ="&vin;"
  xmlns:food ="&food;"
  xmlns:owl ="http://www.w3.org/2002/07/owl#"
  xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:xsd ="http://www.w3.org/2001/XMLSchema#">
```

2.2. Ontology Headers

Once namespaces are established we normally include a collection of assertions about the ontology grouped under an `owl:Ontology` tag. These tags support such critical housekeeping tasks as comments, version control and inclusion of other ontologies.

```
<owl:Ontology rdf:about="">
  <rdfs:comment>An example OWL ontology</rdfs:comment>
  <owl:priorVersion rdf:resource="http://www.w3.org/TR/2003/PR-owl-guide-20031215/wine"
  <owl:imports rdf:resource="http://www.w3.org/TR/2004/REC-owl-guide-20040210/food"/>
  <rdfs:label>Wine Ontology</rdfs:label>
  ...
```

Note that we use '...' to indicate that there is additional text that has been elided for purposes of the example.

The `owl:Ontology` element is a place to collect much of the OWL meta-data for the document. It does not guarantee that the document describes an ontology in the traditional sense. In some communities, ontologies are not about individuals but only the classes and properties that define a domain. When using OWL to describe a collection of instance data the `owl:Ontology` tag may be needed in order to record version information and to import the definitions that the document depends on. Thus, in OWL the term *ontology* has been broadened to include instance data ([see above](#)).

The `rdf:about` attribute provides a name or reference for the ontology. Where the value of the attribute is "", the standard case, the name of the ontology is the base URI of the `owl:Ontology` element. Typically, this is the URI of the document containing the ontology. An exception to this is a context that makes use of `xml:base` which may set the base URI for an element to something other than the URI of the current document.

`rdfs:comment` provides the obvious needed capability to annotate an ontology.

`owl:priorVersion` is a standard tag intended to provide hooks for version control systems working with ontologies. Ontology versioning is discussed further [below](#).

`owl:imports` provides an include-style mechanism. `owl:imports` takes a single argument, identified by the `rdf:resource` attribute.

Importing another ontology brings the entire set of assertions provided by that ontology into the current ontology. In order to make best use of this imported ontology it would normally be coordinated with a namespace declaration. Notice the distinction between these two mechanisms. The namespace declarations provide a convenient means to *reference* names defined in other OWL ontologies. Conceptually, `owl:imports` is provided to indicate your intention to *include* the assertions of the target ontology. Importing another ontology, *O2*, will also import all of the ontologies that *O2* imports.

Note that `owl:imports` may not always succeed. As you would expect when dealing with the Semantic Web, access to resources distributed across the Web may not always be possible. Tools will respond to this situation in an implementation defined manner.

Note that in order to use the OWL vocabulary you do not need to import the [owl.rdf](#) ontology. In fact, such an import is not recommended.

One common set of additional tags that could reasonably be included here are some of the standard [Dublin Core](#) metadata tags. The subset includes those that take simple types or strings as values. Examples include Title, Creator, Description, Publisher, and Date (see [RDF declarations](#)).

Properties that are used as annotations should be declared using `owl:AnnotationProperty`. E.g.

```
<owl:AnnotationProperty rdf:about="&dc;creator" />
```

OWL provides several other mechanisms to tie the current ontology and imported ontologies together (see [ontology mapping](#)).

We also include an `rdfs:label` to support a natural language label for our ontology.

The ontology header definition is closed with the following tag.

```
</owl:Ontology>
```

This prelude is followed by the actual definitions that make up the ontology and is ultimately closed by

```
</rdf:RDF>
```

2.3. Data Aggregation and Privacy

OWL's ability to express ontological information about instances appearing in multiple documents supports linking of data from diverse sources in a principled way. The underlying semantics provides support for inferences over this data that may yield unexpected results. In particular, the ability to express equivalences using `owl:sameAs` can be used to state that seemingly different individuals are actually the same.

`owl:InverseFunctionalProperty` can also be used to link individuals together. For example, if a property such as "SocialSecurityNumber" is an `owl:InverseFunctionalProperty`, then two separate individuals could be inferred to be identical based on having the same value of that property. When individuals are determined to be the same by such means, information about them from different sources can be merged. This *aggregation* can be used to determine facts that are not *directly* represented in any one source.

The ability of the Semantic Web to link information from multiple sources is a desirable and powerful feature that can be used in many [applications](#). However, the capability to merge data from multiple sources, combined with the inferential power of OWL, does have potential for abuse. Users of OWL should be alert to the potential privacy implications. Detailed security solutions were considered out of scope for the Working Group. A number of organizations are addressing these issues with a variety of security and preference solutions. See for example [SAML](#) and [P3P](#).

3. Basic Elements

Most of the elements of an OWL ontology concern classes, properties, instances of classes, and relationships between these instances. This section presents the language components essential to introducing these elements.

3.1. Simple Classes and Individuals

Many uses of an ontology will depend on the ability to reason about individuals. In order to do this in a useful fashion we need to have a mechanism to describe the classes that individuals belong to and the properties that they inherit by virtue of class membership. We can always assert specific properties about individuals, but much of the power of ontologies comes from class-based reasoning.

Sometimes we want to emphasize the distinction between a class as an object and a class as a set containing elements. We call the set of individuals that are members of a class the *extension* of the class.

3.1.1. Simple Named Classes

`Class`, `rdfs:subClassOf`

The most basic concepts in a domain should correspond to classes that are the roots of various taxonomic trees. Every individual in the OWL world is a member of the class `owl:Thing`. Thus each user-defined class is implicitly a subclass of `owl:Thing`. Domain specific root classes are defined by simply declaring a named class. OWL also defines the empty class, `owl:Nothing`.

For our sample wines domain, we create three root classes: `Winery`, `Region`, and `ConsumableThing`.

```
<owl:Class rdf:ID="Winery"/>
<owl:Class rdf:ID="Region"/>
<owl:Class rdf:ID="ConsumableThing"/>
```

Note that we have only said that there exist classes that have been given these names, indicated by the `'rdf:ID='` syntax. Formally, we know almost nothing about these classes other than their existence, despite the use of familiar English terms as labels. And while the classes exist, they may have no members. For all we know at the moment, these classes

might as well have been called `Thing1`, `Thing2`, and `Thing3`.

It is important to remember that definitions may be incremental and distributed. In particular, we will have more to say about `Winery` later.

The syntax `rdf:ID="Region"` is used to introduce a name, as part of its definition. This is the [rdf:ID attribute](#) ([RDF], 7.2.22) that is like the familiar ID attribute defined by XML. Within this document, the `Region` class can now be referred to using `#Region`, e.g.

`rdf:resource="#Region"`. Other ontologies may reference this name using its complete form, `"http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#Region"`.

Another form of reference uses the syntax `rdf:about="#Region"` to [extend](#) the definition of a resource. This use of the `rdf:about="&ont;#x"` syntax is a critical element in the creation of a distributed ontology. It permits the extension of the imported definition of `x` without modifying the original document and supports the incremental construction of a larger ontology.

It is now possible to refer to the classes we defined in other OWL constructs using their given identifier. For the first class, within this document, we can use the relative identifier, `#Winery`. Other documents may need to reference this class as well. The most reasonable way to do so is to provide namespace and entity definitions that include the defining document as a source:

```
...
<!ENTITY vin "http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#" >
<!ENTITY food "http://www.w3.org/TR/2004/REC-owl-guide-20040210/food#" >
...
<rdf:RDF xmlns:vin ="http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#"
        xmlns:food="http://www.w3.org/TR/2004/REC-owl-guide-20040210/food#" ... >
...

```

Given these definitions we can refer to the winery class either using the XML tag `vin:Winery` or the attribute value `&vin;Winery`. More literally, it is always possible to reference a resource using its full URI, here

`http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#Winery`.

The fundamental taxonomic constructor for classes is `rdfs:subClassOf`. It relates a more specific class to a more general class. If `X` is a subclass of `Y`, then every instance of `X` is also an instance of `Y`. The `rdfs:subClassOf` relation is transitive. If `X` is a subclass of `Y` and `Y` a subclass of `Z` then `X` is a subclass of `Z`.

```
<owl:Class rdf:ID="PotableLiquid">
  <rdfs:subClassOf rdf:resource="#ConsumableThing" />
  ...
</owl:Class>

```

We define `PotableLiquid` (liquids suitable for drinking) to be a subclass of `ConsumableThing`.

In the world of Web-based ontologies, both of these classes can be defined in a separate ontology that would provide the basic building blocks for a wide variety of food and drink ontologies, which is what we have done - they are defined in the [food](#) ontology, which is [imported](#) into the wine ontology. The food ontology includes a number of classes, for example `Food`, `EdibleThing`, `MealCourse`, and `Shellfish`, that do not belong in a collection of wine facts, but must be connected to the wine vocabulary if we are going to perform useful reasoning. Food and wine are mutually dependent, in order to satisfy our need to identify wine/food matches.

A class definition has two parts: a name introduction or reference and a list of restrictions. Each of the immediate contained expressions in the class definition further restricts the instances of the defined class. Instances of the class belong to the intersection of the restrictions. (Though see the details of [owl:equivalentClass](#).) So far we have only seen examples that include a single restriction, forcing the new class to be a subclass of some other named class.

At this point it is possible to create a simple (and incomplete) definition for the class `Wine`. `Wine` is a `PotableLiquid`. We also define `Pasta` as an `EdibleThing`.

```
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource="#food;PotableLiquid" />
  <rdfs:label xml:lang="en">wine</rdfs:label>
  <rdfs:label xml:lang="fr">vin</rdfs:label>
  ...
</owl:Class>

<owl:Class rdf:ID="Pasta">
  <rdfs:subClassOf rdf:resource="#EdibleThing" />
  ...
</owl:Class>
```

The `rdfs:label` entry provides an optional human readable name for this class. Presentation tools can make use of it. The "lang" attribute provides support for multiple languages. A label is like a comment and contributes nothing to the logical interpretation of an ontology.

Our wine definition is still very incomplete. We know nothing about wines except that they are things and potable liquids, but we have sufficient information to create and reason about individuals.

3.1.2. Individuals

In addition to classes, we want to be able to describe their members. We normally think of these as individuals in our universe of things. An individual is minimally introduced by declaring it to be a member of a class.

```
<Region rdf:ID="CentralCoastRegion" />
```

Note that the following is identical in meaning to the example above.

```
<owl:Thing rdf:ID="CentralCoastRegion" />
<owl:Thing rdf:about="#CentralCoastRegion">
  <rdf:type rdf:resource="#Region"/>
</owl:Thing>
```

`rdf:type` is an RDF property that ties an individual to a class of which it is a member.

There are a couple of points to be made here. First, we have decided that `CentralCoastRegion` (a specific area) is member of `Region`, the class containing all geographical regions. Second, there is no requirement in the two-part example that the two elements need to be adjacent to one another, or even in the same file (though the names would need to be extended with a URI in such a case). We design Web ontologies to be distributed. They can be imported and augmented, creating derived ontologies.

In order to have available a few more classes for the properties introduced in the next sections, we define a branch of the `Grape` taxonomy, with an individual denoting the Cabernet Sauvignon grape varietal. Grapes are defined in the food ontology:

```
<owl:Class rdf:ID="Grape">
  ...
</owl:Class>
```

And then in the wine ontology we have:

```
<owl:Class rdf:ID="WineGrape">
  <rdfs:subClassOf rdf:resource="&food;Grape" />
</owl:Class>

<WineGrape rdf:ID="CabernetSauvignonGrape" />
```

As discussed in the next section, `CabernetSauvignonGrape` is an individual because it denotes a single grape varietal.

3.1.3. Design for Use

There are important issues regarding the distinction between a *class* and an *individual* in OWL. A class is simply a name and collection of properties that describe a set of individuals. Individuals are the members of those sets. Thus classes should correspond to naturally occurring sets of things in a domain of discourse, and individuals should correspond to actual entities that can be grouped into these classes.

In building ontologies, this distinction is frequently blurred in two ways:

- *Levels of representation:* In certain contexts something that is obviously a class can itself be considered an instance of something else. For example, in the wine ontology we have the notion of a `Grape`, which is intended to denote the set of all *grape varieties*. `CabernetSauvignonGrape` is an example instance of this class, as it denotes the actual grape varietal called Cabernet Sauvignon. However, `CabernetSauvignonGrape` could itself be considered a class, the set of all actual Cabernet Sauvignon grapes.
- *Subclass vs. instance:* It is very easy to confuse the instance-of relationship with the subclass relationship. For example, it may seem arbitrary to choose to make `CabernetSauvignonGrape` an individual that is an instance of `Grape`, as opposed to a subclass of `Grape`. This is not an arbitrary decision. The `Grape` class denotes the set of all *grape varieties*, and therefore any subclass of `Grape` should denote a subset of these varieties. Thus, `CabernetSauvignonGrape` should be considered an *instance of* `Grape`, and not a subclass. It does not describe a subset of `Grape` varieties, it *is* a grape varietal.

Note that the same distinction arises with the treatment of the `Wine` class. The `Wine` class actually denotes the set of all *varieties* of wine, not the set of actual bottles that someone may purchase. In an alternate ontology, each instance of `Wine` in the current ontology could instead designate a class consisting of all the bottles of wine of that type. It is easy to imagine an information system, such as an inventory system for a wine merchant, that needs to consider individual bottles of wine. The wine ontology as it currently exists would require the ability to treat classes as instances in order to support such an interpretation. Note that OWL Full permits such expressivity, allowing us to treat an instance of a wine variety simultaneously as a class whose instances are bottles of wine.

In a similar vein, the wines produced by wineries in specific years are considered vintages. In order to represent the notion of a vintage, we must determine where it fits in the current ontology. An instance of the `Wine` class, as discussed above, represents a single variety of wine produced by a single winery, for example `FormanChardonnay`.

Adding that the wine produced in the year 2000 is considered a vintage poses a challenge, because we don't have the ability to represent a subset of a given wine individual. This vintage is not a new variety of wine, it is a special subset of the wine - that produced in the year 2000. An option would be to use OWL Full and treat the wine instances as classes with subclasses (subsets) denoting vintages. Another option is to use a workaround and to consider `Vintage` as a separate class whose instances have a relationship to the `Wine` they are a vintage of. For example, `FormanChardonnay2000` is an individual `Vintage` with a `vintageOf` property whose value is the `Wine, FormanChardonnay`. We define the [vintage](#) class below.

The point of this discussion is to note that the development of an ontology should be firmly driven by the intended usage. These issues also underlie one major difference between OWL Full and OWL DL. OWL Full allows the use of classes as instances and OWL DL does not. The wine ontology is designed to work in OWL DL, and as a result individuals like `FormanChardonnay` are not simultaneously treated as classes.

3.2. Simple Properties

This world of classes and individuals would be pretty uninteresting if we could only define taxonomies. *Properties* let us assert general facts about the members of classes and specific facts about individuals.

3.2.1. Defining Properties

`ObjectProperty`, `DatatypeProperty`, `rdfs:subPropertyOf`, `rdfs:domain`, `rdfs:range`

A property is a binary relation. Two types of properties are distinguished:

- *datatype properties*, relations between instances of classes and RDF literals and XML Schema datatypes
- *object properties*, relations between instances of two classes. Note that the name *object property* is not intended to reflect a connection with the [RDF](#) term [rdf:object](#) ([\[RDF\]](#), 5.3.4).

When we define a property there are a number of ways to restrict the relation. The domain and range can be specified. The property can be defined to be a specialization (subproperty) of an existing property. More elaborate restrictions are possible and are described [later](#).

```
<owl:ObjectProperty rdf:ID="madeFromGrape">
  <rdfs:domain rdf:resource="#Wine"/>
  <rdfs:range rdf:resource="#WineGrape"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="course">
  <rdfs:domain rdf:resource="#Meal" />
  <rdfs:range rdf:resource="#MealCourse" />
</owl:ObjectProperty>
```

In OWL, a sequence of elements without an explicit operator represents an implicit

conjunction. The property `madeFromGrape` has a domain of `Wine` *and* a range of `WineGrape`. That is, it relates instances of the class `Wine` to instances of the class `WineGrape`. Multiple domains mean that the domain of the property is the intersection of the identified classes (and similarly for range).

Similarly, the property `course` ties a `Meal` to a `MealCourse`.

Note that the use of range and domain information in OWL is different from type information in a programming language. Among other things, types are used to check consistency in a programming language. In OWL, a range may be used to infer a type. For example, given:

```
<owl:Thing rdf:ID="LindemansBin65Chardonnay">
  <madeFromGrape rdf:resource="#ChardonnayGrape" />
</owl:Thing>
```

we can infer that `LindemansBin65Chardonnay` is a wine because the domain of `madeFromGrape` is `Wine`.

Properties, like classes, can be arranged in a hierarchy.

```
<owl:Class rdf:ID="WineDescriptor" />

<owl:Class rdf:ID="WineColor">
  <rdfs:subClassOf rdf:resource="#WineDescriptor" />
  ...
</owl:Class>

<owl:ObjectProperty rdf:ID="hasWineDescriptor">
  <rdfs:domain rdf:resource="#Wine" />
  <rdfs:range rdf:resource="#WineDescriptor" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasColor">
  <rdfs:subPropertyOf rdf:resource="#hasWineDescriptor" />
  <rdfs:range rdf:resource="#WineColor" />
  ...
</owl:ObjectProperty>
```

`WineDescriptor` properties relate wines to their color and components of their taste, including sweetness, body, and flavor. `hasColor` is a subproperty of the `hasWineDescriptor` property, with its range further restricted to `WineColor`. The `rdfs:subPropertyOf` relation in this case means that anything with a `hasColor` property with value X also has a `hasWineDescriptor` property with value X.

Next we introduce the `locatedIn` property, which relates things to the regions they are located in.

```
<owl:ObjectProperty rdf:ID="locatedIn">
  ...
  <rdfs:domain rdf:resource="http://www.w3.org/2002/07/owl#Thing" />
  <rdfs:range rdf:resource="#Region" />
</owl:ObjectProperty>
```

Notice how the domain and range of `locatedIn` are defined. The domain permits anything to be located in a region, including regions themselves. And the [transitive](#) composition of this relation essentially creates a network of geographically included subregions and things. Those things that have nothing located in them can be of any class, while those that contain

others must be regions.

It is now possible to expand the definition of `Wine` to include the notion that a wine is made from at least one `WineGrape`. As with property definitions, class definitions have multiple subparts that are implicitly conjoined.

```
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource="&food;PotableLiquid"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#madeFromGrape"/>
      <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  ...
</owl:Class>
```

The highlighted subclass restriction above

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#madeFromGrape"/>
  <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
</owl:Restriction>
```

defines an unnamed class that represents the set of things with at least one `madeFromGrape` property. We call these *anonymous* classes. Including this restriction in the `Wine` class definition body states that things that are wines are also members of this anonymous class. That is, every individual wine must participate in at least one `madeFromGrape` relation.

We can now describe the class of `Vintages`, discussed [previously](#).

```
<owl:Class rdf:ID="Vintage">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#vintageOf"/>
      <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

The property `vintageOf` ties a `Vintage` to a `Wine`.

```
<owl:ObjectProperty rdf:ID="vintageOf">
  <rdfs:domain rdf:resource="#Vintage" />
  <rdfs:range rdf:resource="#Wine" />
</owl:ObjectProperty>
```

We relate `Vintages` to their years in the next section.

3.2.2. Properties and Datatypes

We distinguish properties according to whether they relate individuals to individuals (object properties) or individuals to datatypes (datatype properties). Datatype properties may range over RDF literals or simple types defined in accordance with [XML Schema datatypes](#).

OWL uses most of the built-in XML Schema datatypes. References to these datatypes are by means of the URI reference for the datatype, <http://www.w3.org/2001/XMLSchema>. The following datatypes are *recommended* for use with OWL:

xsd:string	xsd:normalizedString	xsd:boolean	
xsd:decimal	xsd:float	xsd:double	
xsd:integer	xsd:nonNegativeInteger	xsd:positiveInteger	
xsd:nonPositiveInteger	xsd:negativeInteger		
xsd:long	xsd:int	xsd:short	xsd:byte
xsd:unsignedLong	xsd:unsignedInt	xsd:unsignedShort	xsd:unsignedByte
xsd:hexBinary	xsd:base64Binary		
xsd:dateTime	xsd:time	xsd:date	xsd:gYearMonth
xsd:gYear	xsd:gMonthDay	xsd:gDay	xsd:gMonth
xsd:anyURI	xsd:token	xsd:language	
xsd:NMTOKEN	xsd:Name	xsd:NCName	

The above datatypes, plus `rdfs:Literal`, form the built-in OWL datatypes. All OWL reasoners are required to support the `xsd:integer` and `xsd:string` datatypes.

Other built-in XML Schema datatypes may be used in OWL Full, but with caveats described in the [OWL Semantics and Abstract Syntax](#) documentation.

```
<owl:Class rdf:ID="VintageYear" />

<owl:DatatypeProperty rdf:ID="yearValue">
  <rdfs:domain rdf:resource="#VintageYear" />
  <rdfs:range rdf:resource="&xsd;positiveInteger"/>
</owl:DatatypeProperty>
```

The `yearValue` property relates `VintageYears` to positive integer values. We introduce the `hasVintageYear` property, which relates a `Vintage` to a `VintageYear` [below](#).

The [OWL Reference](#) ([Reference], 6.2) describes the use of `owl:oneOf` and `rdf:List` and `rdf:rest` to define an enumerated datatype. The example shows how to construct the `owl:DatatypeProperty`, `tennisGameScore`, with a range equal to the elements of the list of integer values {0, 15, 30, 40}.

3.2.3. Properties of Individuals

First we describe `Region` and `Winery` individuals, and then we define our first wine, a `Cabernet Sauvignon`.

```
<Region rdf:ID="SantaCruzMountainsRegion">
  <locatedIn rdf:resource="#CaliforniaRegion" />
</Region>

<Winery rdf:ID="SantaCruzMountainVineyard" />

<CabernetSauvignon
  rdf:ID="SantaCruzMountainVineyardCabernetSauvignon" >
  <locatedIn rdf:resource="#SantaCruzMountainsRegion"/>
```

```
<hasMaker    rdf:resource="#SantaCruzMountainVineyard" />
</CabernetSauvignon>
```

This is still incomplete. There are other aspects of the wine flavor that are defined in the full ontology. But the pieces are falling together. We could begin reasoning about what menu items in our food ontology this wine might accompany. We know from the definition above that the Santa Cruz Mountain Vineyard makes it. Because it is a Cabernet Sauvignon (see [wine.rdf](#)), we know it is a dry, red wine.

Datatype properties can be added to individuals in a similar fashion. Below we describe an instance of `VintageYear` and tie it to a specific value of type `&xsd:positiveInteger`.

```
<VintageYear rdf:ID="Year1998">
  <yearValue rdf:datatype="&xsd:positiveInteger">1998</yearValue>
</VintageYear>
```

3.3. Property Characteristics

The next few sections describe the mechanisms used to further specify properties. It is possible to specify property *characteristics*, which provides a powerful mechanism for enhanced reasoning about a property.

3.3.1. TransitiveProperty

If a property, `P`, is specified as transitive then for any `x`, `y`, and `z`:

$$P(x,y) \text{ and } P(y,z) \text{ implies } P(x,z)$$

The property `locatedIn` is transitive.

```
<owl:ObjectProperty rdf:ID="locatedIn">
  <rdf:type rdf:resource="&owl;TransitiveProperty" />
  <rdfs:domain rdf:resource="&owl;Thing" />
  <rdfs:range rdf:resource="#Region" />
</owl:ObjectProperty>

<Region rdf:ID="SantaCruzMountainsRegion">
  <locatedIn rdf:resource="#CaliforniaRegion" />
</Region>

<Region rdf:ID="CaliforniaRegion">
  <locatedIn rdf:resource="#USRegion" />
</Region>
```

Because the `SantaCruzMountainsRegion` is `locatedIn` the `CaliforniaRegion`, then it must also be `locatedIn` the `USRegion`, since `locatedIn` is transitive.

3.3.2. SymmetricProperty

If a property, `P`, is tagged as symmetric then for any `x` and `y`:

$$P(x,y) \text{ iff } P(y,x)$$

The property `adjacentRegion` is symmetric, while `locatedIn` is not. To be more precise, `locatedIn` is not intended to be symmetric. Nothing in the wine ontology at present prevents

it from being symmetric.

```
<owl:ObjectProperty rdf:ID="adjacentRegion">
  <rdf:type rdf:resource="&owl:SymmetricProperty" />
  <rdfs:domain rdf:resource="#Region" />
  <rdfs:range rdf:resource="#Region" />
</owl:ObjectProperty>

<Region rdf:ID="MendocinoRegion">
  <locatedIn rdf:resource="#CaliforniaRegion" />
  <adjacentRegion rdf:resource="#SonomaRegion" />
</Region>
```

The `MendocinoRegion` is adjacent to the `SonomaRegion` and vice-versa. The `MendocinoRegion` is located in the `CaliforniaRegion` but not vice versa.

3.3.3. FunctionalProperty

If a property, P , is tagged as functional then for all x , y , and z :

$$P(x,y) \text{ and } P(x,z) \text{ implies } y = z$$

In our wine ontology, `hasVintageYear` is functional. A wine has a unique vintage year. That is, a given individual `Vintage` can only be associated with a single year using the `hasVintageYear` property. It is not a requirement of a `owl:FunctionalProperty` that all elements of the domain have values. See the discussion of [Vintage cardinality](#).

```
<owl:Class rdf:ID="VintageYear" />

<owl:ObjectProperty rdf:ID="hasVintageYear">
  <rdf:type rdf:resource="&owl:FunctionalProperty" />
  <rdfs:domain rdf:resource="#Vintage" />
  <rdfs:range rdf:resource="#VintageYear" />
</owl:ObjectProperty>
```

3.3.4. inverseOf

If a property, P_1 , is tagged as the `owl:inverseOf` P_2 , then for all x and y :

$$P_1(x,y) \text{ iff } P_2(y,x)$$

Note that the syntax for `owl:inverseOf` takes a property name as an argument. $A \text{ iff } B$ means $(A \text{ implies } B)$ and $(B \text{ implies } A)$.

```
<owl:ObjectProperty rdf:ID="hasMaker">
  <rdf:type rdf:resource="&owl:FunctionalProperty" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="producesWine">
  <owl:inverseOf rdf:resource="#hasMaker" />
</owl:ObjectProperty>
```

Wines have makers, which in the definition of `Wine` are restricted to `Winerys`. Then each `Winery` produces the set of wines that identify it as maker.

3.3.5. InverseFunctionalProperty

If a property, P , is tagged as `InverseFunctional` then for all x , y and z :

$$P(y,x) \text{ and } P(z,x) \text{ implies } y = z$$

Notice that `producesWine` in the preceding section is inverse functional. The reason is that the inverse of a functional property must be inverse functional. We could have defined `hasMaker` and `producesWine` as follows and achieved the identical effect as the preceding example.

```
<owl:ObjectProperty rdf:ID="hasMaker" />

<owl:ObjectProperty rdf:ID="producesWine">
  <rdf:type rdf:resource="&owl;InverseFunctionalProperty" />
  <owl:inverseOf rdf:resource="#hasMaker" />
</owl:ObjectProperty>
```

Think of the elements of the range in an inverse functional property as defining a unique key in the database sense. `owl:InverseFunctional` implies that the elements of the range provide a unique identifier for each element of the domain.

In OWL Full, we can tag a `DatatypeProperty` as `inverseFunctional`. This permits us to identify a string as a unique key. In OWL DL literals are disjoint from `owl:Thing`, which is why OWL DL does not permit `InverseFunctional` to be applied to `DatatypeProperty`.

3.4. Property Restrictions

In addition to designating property characteristics, it is possible to further constrain the range of a property in specific contexts in a variety of ways. We do this with *property restrictions*. The various forms described below can only be used within the context of an `owl:Restriction`. The `owl:onProperty` element indicates the restricted property.

3.4.1. allValuesFrom, someValuesFrom

We have already seen one way to restrict the types of the elements that make up a property. The mechanisms to date have been *global* in that they apply to all instances of the property. These next two, `allValuesFrom` and `someValuesFrom`, are *local* to their containing class definition.

The `owl:allValuesFrom` restriction requires that for every instance of the class that has instances of the specified property, the values of the property are all members of the class indicated by the `owl:allValuesFrom` clause.

```
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource="&food;PotableLiquid" />
  ...
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasMaker" />
      <owl:allValuesFrom rdf:resource="#Winery" />
    </owl:Restriction>
  </rdfs:subClassOf>
  ...
```

```
</owl:Class>
```

The maker of a `Wine` must be a `Winery`. The `allValuesFrom` restriction is on the `hasMaker` property of this `Wine` class *only*. Makers of `Cheese` are not constrained by this local restriction.

`owl:someValuesFrom` is similar. If we replaced `owl:allValuesFrom` with `owl:someValuesFrom` in the example above, it would mean that at least *one* of the `hasMaker` properties of a `Wine` must point to an individual that is a `Winery`.

```
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource="&food;PotableLiquid" />
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasMaker" />
      <owl:someValuesFrom rdf:resource="#Winery" />
    </owl:Restriction>
  </rdfs:subClassOf>
  ...
</owl:Class>
```

The difference between the two formulations is the difference between a universal and existential quantification.

Relation	Implications
<code>allValuesFrom</code>	For all wines, if they have makers, all the makers are wineries.
<code>someValuesFrom</code>	For all wines, they have at least one maker that is a winery.

The first does not require a wine to have a maker. If it does have one or more, they must all be wineries. The second requires that there be at least one maker that is a winery, but there may be makers that are not wineries.

3.4.2. Cardinality

We have already seen examples of cardinality constraints. To date, they have been assertions about minimum cardinality. Even more straight-forward is `owl:cardinality`, which permits the specification of *exactly* the number of elements in a relation. For example, we specify `Vintage` to be a class with exactly one `VintageYear`.

```
<owl:Class rdf:ID="Vintage">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasVintageYear" />
      <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

We specified `hasVintageYear` to be a functional property, which is the same as saying that every `Vintage` has at most one `VintageYear`. This application of that property to `Vintage` using the cardinality restriction asserts something stronger, that *every* `Vintage` has *exactly one* `VintageYear`.

Cardinality expressions with values limited to 0 or 1 are part of OWL Lite. This permits the user to indicate 'at least one', 'no more than one', and 'exactly one'. Positive integer values

other than 0 and 1 are permitted in OWL DL. `owl:maxCardinality` can be used to specify an *upper* bound. `owl:minCardinality` can be used to specify a *lower* bound. In combination, the two can be used to limit the property's cardinality to a numeric interval.

3.4.3. `hasValue` [\[OWL DL\]](#)

`hasValue` allows us to specify classes based on the existence of *particular* property values. Hence, an individual will be a member of such a class whenever at least *one* of its property values is equal to the `hasValue` resource.

```
<owl:Class rdf:ID="Burgundy">
  ...
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasSugar" />
      <owl:hasValue rdf:resource="#Dry" />
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Here we declare that all `Burgundy` wines are dry. That is, their `hasSugar` property must have at least one value that is equal to `Dry`.

As for `allValuesFrom` and `someValuesFrom`, this is a local restriction. It holds for `hasSugar` as applied to `Burgundy`.

4. Ontology Mapping

In order for ontologies to have the maximum impact, they need to be widely shared. In order to minimize the intellectual effort involved in developing an ontology they need to be re-used. In the best of all possible worlds they need to be composed. For example, you might adopt a date ontology from one source and a physical location ontology from another and then extend the notion of location to include the time period during which it holds.

It is important to realize that much of the effort of developing an ontology is devoted to hooking together classes and properties in ways that maximize implications. We want simple assertions about class membership to have broad and useful implications. This is the hardest part of ontology development. If you can find an existing ontology that has already undergone extensive use and refinement, it makes sense to adopt it.

It will be challenging to merge a collection of ontologies. Tool support will almost certainly be required to maintain consistency.

4.1. Equivalence between Classes and Properties

`equivalentClass`, `equivalentProperty`

To tie together a set of component ontologies as part of a third it is frequently useful to be able to indicate that a particular class or property in one ontology is equivalent to a class or property in a second ontology. This capability must be used with care. If the combined ontologies are contradictory (all A's are B's vs. all A's are not B's) there will be no extension (no individuals and relations) that satisfies the resulting combination.

In the food ontology we want to link wine features in the descriptions of dining courses back to the wine ontology. One way to do this is by defining a class in the food ontology

(&food;Wine) and then declaring it equivalent to an existing wine class in the wine ontology.

```
<owl:Class rdf:ID="Wine">
  <owl:equivalentClass rdf:resource="#vin;Wine" />
</owl:Class>
```

The property `owl:equivalentClass` is used to indicate that two classes have precisely the same instances. Note that in OWL DL, classes simply denote sets of individuals, and are not individuals themselves. In OWL Full, however, we can use [owl:sameAs](#) between two classes to indicate that they are identical in every way.

Of course the example above is somewhat contrived, since we can always use `&vin;Wine` anywhere we would use `#Wine` and get the same effect without redefinition. A more likely use would be in a case where we depend on two independently developed ontologies, and note that they use the URI's `O1:foo` and `O2:bar` to reference the same class.

`owl:equivalentClass` could be used to collapse these together so that the entailments from the two ontologies are combined.

We have already seen that class expressions can be the targets of `rdfs:subClassOf` constructors. They can also be the target of `owl:equivalentClass`. Again, this avoids the need to contrive names for every class expression and provides a powerful definitional capability based on satisfaction of a property.

```
<owl:Class rdf:ID="TexasThings">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#locatedIn" />
      <owl:someValuesFrom rdf:resource="#TexasRegion" />
    </owl:Restriction>
  </owl:equivalentClass>
</owl:Class>
```

`TexasThings` are *exactly* those things located in the Texas region. The difference between using `owl:equivalentClass` here and using `rdfs:subClassOf` is the difference between a necessary condition and a necessary and sufficient condition. With `subClassOf`, things that are located in Texas are not necessarily `TexasThings`. But, using `owl:equivalentClass`, if something is located in Texas, then it must be in the class of `TexasThings`.

Relation	Implications
<code>subClassOf</code>	<code>TexasThings(x)</code> implies <code>locatedIn(x,y)</code> and <code>TexasRegion(y)</code>
<code>equivalentClass</code>	<code>TexasThings(x)</code> implies <code>locatedIn(x,y)</code> and <code>TexasRegion(y)</code> <code>locatedIn(x,y)</code> and <code>TexasRegion(y)</code> implies <code>TexasThings(x)</code>

To tie together properties in a similar fashion, we use `owl:equivalentProperty`.

4.2. Identity between Individuals

[sameAs](#)

This mechanism is similar to that for classes, but declares two individuals to be identical. An example would be:

```
<Wine rdf:ID="MikesFavoriteWine">
  <owl:sameAs rdf:resource="#StGenevieveTexasWhite" />
</Wine>
```

This example does not have great utility. About all we learn from this is that Mike likes an inexpensive local wine. A more typical use of `sameAs` would be to equate individuals defined in different documents to one another, as part of unifying two ontologies.

This brings up an important point. OWL does not have a *unique name* assumption. Just because two names are different does not mean they refer to different individuals.

In the example above, we *asserted* identity between two distinct names. But it is just as possible for this sort of identity to be inferred. Remember the implications that can be derived from a functional property. Given that `hasMaker` is functional, the following is not necessarily a conflict.

```
<owl:Thing rdf:about="#BancroftChardonnay">
  <hasMaker rdf:resource="#Bancroft" />
  <hasMaker rdf:resource="#Beringer" />
</owl:Thing>
```

Unless this conflicts with other information in our ontology, it simply means that `Bancroft = Beringer`.

Note that using `sameAs` to equate two classes is **not** the same as equating them with `equivalentClass`; instead, it causes the the classes to be interpreted as individuals, and is therefore sufficient to categorize an ontology as OWL Full. In OWL Full `sameAs` may be used to equate anything: a class and an individual, a property and a class, etc., and causes both arguments to be interpreted as individuals.

4.3. Different Individuals

`differentFrom`, `AllDifferent`

This mechanism provides the opposite effect from `sameAs`.

```
<WineSugar rdf:ID="Dry" />

<WineSugar rdf:ID="Sweet">
  <owl:differentFrom rdf:resource="#Dry"/>
</WineSugar>

<WineSugar rdf:ID="OffDry">
  <owl:differentFrom rdf:resource="#Dry"/>
  <owl:differentFrom rdf:resource="#Sweet"/>
</WineSugar>
```

This is one way to assert that these three values are mutually distinct. There will be cases where it is important to ensure such distinct identities. Without these assertions we could describe a wine that was both `Dry` and `Sweet`. We have stated that the `hasSugar` property applied to a wine has no more than one value. If we erred, and asserted that a wine was both `Dry` and `Sweet`, without the `differentFrom` elements above, this would imply that `Dry` and `Sweet` are identical. With the elements above, we would instead get a contradiction.

A more convenient mechanism exists to define a set of mutually distinct individuals. The following asserts that `Red`, `White`, and `Rose` are pairwise distinct.

```
<owl>AllDifferent>
  <owl:distinctMembers rdf:parseType="Collection">
```

```

    <vin:WineColor rdf:about="#Red" />
    <vin:WineColor rdf:about="#White" />
    <vin:WineColor rdf:about="#Rose" />
  </owl:distinctMembers>
</owl:AllDifferent>

```

Note that `owl:distinctMembers` can only be used in combination with `owl:AllDifferent`

In the wine ontology we provide an `owl:AllDifferent` assertion for all of the `WineDescriptorS`. We also state that the `WineryS` are all different. If we wanted to add a new winery in some other ontology and assert that it was disjoint from all of those that have already been defined, we would need to cut and paste the original `owl:AllDifferent` assertion and add the new maker to the list. There is not a simpler way to extend an `owl:AllDifferent` collection in OWL DL. In OWL Full, using RDF triples and the `rdf:List` constructs, other approaches are possible.

5. Complex Classes [\[OWL DL\]](#)

OWL provides additional constructors with which to form classes. These constructors can be used to create so-called *class expressions*. OWL supports the basic set operations, namely union, intersection and complement. These are named `owl:unionOf`, `owl:intersectionOf`, and `owl:complementOf`, respectively. Additionally, classes can be *enumerated*. Class extensions can be stated explicitly by means of the `oneOf` constructor. And it is possible to assert that class extensions must be disjoint.

Note that Class expressions can be nested without requiring the creation of names for every intermediate class. This allows the use of set operations to build up complex classes from anonymous classes or classes with value restrictions.

5.1. Set Operators

`intersectionOf`, `unionOf`, `complementOf`

Remember that OWL class extensions are sets consisting of the individuals that are members of the class. OWL provides the means to manipulate class extensions using basic set operators.

5.1.1. Intersection [\[some uses of OWL DL\]](#)

The following examples demonstrate the use of the *intersectionOf* construct.

```

<owl:Class rdf:ID="WhiteWine">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Wine" />
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasColor" />
      <owl:hasValue rdf:resource="#White" />
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>

```

Classes constructed using the set operations are more like definitions than anything we have seen to date. The members of the class are completely specified by the set operation. The construction above states that `WhiteWine` is *exactly* the intersection of the class `Wine`

and the set of things that are white in color. This means that if something is white and a wine, then it is an instance of `WhiteWine`. Without such a definition we can know that white wines are wines and white, but not vice-versa. This is an important tool for categorizing individuals. (Note that `'rdf:parseType="Collection"'` is a required syntactic element.)

```
<owl:Class rdf:about="#Burgundy">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Wine" />
    <owl:Restriction>
      <owl:onProperty rdf:resource="#locatedIn" />
      <owl:hasValue rdf:resource="#BourgogneRegion" />
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

Here we define `Burgundy` to include exactly those wines that have at least one `locatedIn` relation to the `BourgogneRegion`. We could have declared a new class `ThingsFromBourgogneRegion` and used it as a class in the `owl:intersectionOf` construct. Since we do not have any other use for `ThingsFromBourgogneRegion`, the declaration above is shorter, clearer and doesn't require the creation of a contrived name.

```
<owl:Class rdf:ID="WhiteBurgundy">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Burgundy" />
    <owl:Class rdf:about="#WhiteWine" />
  </owl:intersectionOf>
</owl:Class>
```

Finally, the class `WhiteBurgundy` is exactly the intersection of white wines and Burgundies. Burgundies in turn are grown in the French region of `Bourgogne` and are dry wines. Accordingly all individual wines that meet these criteria are part of the class extension of `WhiteBurgundy`.

5.1.2. Union [\[OWL DL\]](#)

The following example demonstrates the use of the *unionOf* construct. It is used exactly like the *intersectionOf* construct:

```
<owl:Class rdf:ID="Fruit">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#SweetFruit" />
    <owl:Class rdf:about="#NonSweetFruit" />
  </owl:unionOf>
</owl:Class>
```

The class `Fruit` includes *both* the extension of `SweetFruit` and the extension of `NonSweetFruit`.

Note how completely different this union type construct is from the following.

```
<owl:Class rdf:ID="Fruit">
  <rdfs:subClassOf rdf:resource="#SweetFruit" />
  <rdfs:subClassOf rdf:resource="#NonSweetFruit" />
</owl:Class>
```

This says that the instances of `Fruit` are a subset of the *intersection* of sweet and

non-sweet fruit, which we would expect to be the empty set.

5.1.3. Complement [\[OWL DL\]](#)

The *complementOf* construct selects all individuals from the domain of discourse that do not belong to a certain class. Usually this refers to a very large set of individuals:

```
<owl:Class rdf:ID="ConsumableThing" />

<owl:Class rdf:ID="NonConsumableThing">
  <owl:complementOf rdf:resource="#ConsumableThing" />
</owl:Class>
```

The class of `NonConsumableThing` includes as its members all individuals that do not belong to the extension of `ConsumableThing`. This set includes all `Wines`, `Regions`, etc. It is literally the set difference between `owl:Thing` and `ConsumableThing`. Therefore, a typical usage pattern for *complementOf* is in combination with other set operators:

```
<owl:Class rdf:ID="NonFrenchWine">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Wine"/>
    <owl:Class>
      <owl:complementOf>
        <owl:Restriction>
          <owl:onProperty rdf:resource="#locatedIn" />
          <owl:hasValue rdf:resource="#FrenchRegion" />
        </owl:Restriction>
      </owl:complementOf>
    </owl:Class>
  </owl:intersectionOf>
</owl:Class>
```

This defines the class `NonFrenchWine` to be the intersection of `Wine` with the set of all things *not* located in France.

5.2. Enumerated Classes

oneOf [\[OWL DL\]](#)

OWL provides the means to specify a class via a direct enumeration of its members. This is done using the *oneOf* construct. Notably, this definition completely specifies the class extension, so that no other individuals can be declared to belong to the class.

The following defines a class `WineColor` whose members are the individuals `White`, `Rose`, and `Red`.

```
<owl:Class rdf:ID="WineColor">
  <rdfs:subClassOf rdf:resource="#WineDescriptor"/>
  <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:about="#White"/>
    <owl:Thing rdf:about="#Rose"/>
    <owl:Thing rdf:about="#Red"/>
  </owl:oneOf>
</owl:Class>
```

The first thing to understand here is that no other individuals can be a valid `WineColor` since

the class has been defined by enumeration.

Each element of the `oneOf` construct must be a validly declared individual. An individual has to belong to some class. In the above example, each individual was referenced by name. We used `owl:Thing` as a simple cliché to introduce the reference. Alternatively, we could have referenced the elements of the set according to their specific type, `WineColor`, by:

```
<owl:Class rdf:ID="WineColor">
  <rdfs:subClassOf rdf:resource="#WineDescriptor" />
  <owl:oneOf rdf:parseType="Collection">
    <WineColor rdf:about="#White" />
    <WineColor rdf:about="#Rose" />
    <WineColor rdf:about="#Red" />
  </owl:oneOf>
</owl:Class>
```

Other, more complex descriptions of individuals are also valid elements of the `oneOf` construct, for example:

```
<WineColor rdf:about="#White">
  <rdfs:label>White</rdfs:label>
</WineColor>
```

For additional examples of the use of `oneOf`, see the [Reference](#).

5.3. Disjoint Classes

`disjointWith` [\[OWL DL\]](#)

The disjointness of a set of classes can be expressed using the `owl:disjointWith` constructor. It guarantees that an individual that is a member of one class cannot simultaneously be an instance of a specified other class.

```
<owl:Class rdf:ID="Pasta">
  <rdfs:subClassOf rdf:resource="#EdibleThing" />
  <owl:disjointWith rdf:resource="#Meat" />
  <owl:disjointWith rdf:resource="#Fowl" />
  <owl:disjointWith rdf:resource="#Seafood" />
  <owl:disjointWith rdf:resource="#Dessert" />
  <owl:disjointWith rdf:resource="#Fruit" />
</owl:Class>
```

The `Pasta` example demonstrates multiple disjoint classes. Note that this only asserts that `Pasta` is disjoint from all of these other classes. It does not assert, for example, that `Meat` and `Fruit` are disjoint. In order to assert that a set of classes is mutually disjoint, there must be an `owl:disjointWith` assertion for every pair.

A common requirement is to define a class as the union of a set of mutually disjoint subclasses.

```
<owl:Class rdf:ID="SweetFruit">
  <rdfs:subClassOf rdf:resource="#EdibleThing" />
</owl:Class>

<owl:Class rdf:ID="NonSweetFruit">
  <rdfs:subClassOf rdf:resource="#EdibleThing" />
```

```

    <owl:disjointWith rdf:resource="#SweetFruit" />
  </owl:Class>

  <owl:Class rdf:ID="Fruit">
    <owl:unionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#SweetFruit" />
      <owl:Class rdf:about="#NonSweetFruit" />
    </owl:unionOf>
  </owl:Class>

```

Here we define `Fruit` to be exactly the union of `SweetFruit` and `NonSweetFruit`. And we know that these subclasses exactly partition `Fruit` into two distinct subclasses because they are disjoint. As the number of mutually disjoint classes grows, the number of disjointness assertions grows proportionally to n^2 . However, in the use cases we have seen, n is typically small.

When n is large, alternate approaches can be used to avoid quadratic growth in the number of assertions. One such method is illustrated in the [OWL test suite](#)

The illustrated method works as follows. We describe a parent class whose elements have a property with cardinality equal to one. That is, each instance must have one and only one value for this property. Then, for every subclass of the parent we require that its instances must have a particular unique value for the property. In which case none of the distinct subclasses can have members in common.

6. Ontology Versioning

Ontologies are like software, they will be maintained and thus will change over time. Within an `owl:Ontology` element (discussed [above](#)), it is possible to link to a previous version of the ontology being defined. The `owl:priorVersion` property is intended to provide this link, and can be used to track the version history of an ontology.

```

<owl:Ontology rdf:about="">
  ...
  <owl:priorVersion rdf:resource="http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine"
  ...
</owl:Ontology>

```

The indicated ontology is a previous version of the one being defined.

Ontology versions may not be compatible with each other. For example, a prior version of an ontology may contain statements that contradict the current version. Within an `owl:Ontology` element, we use the tags `owl:backwardCompatibleWith` and `owl:incompatibleWith` to indicate compatibility or the lack thereof with previous ontology versions. If `owl:backwardCompatibleWith` is not declared, then compatibility should not be assumed. In addition, `owl:versionInfo` provides a hook suitable for use by versioning systems. As opposed to the previous three tags, the object of `owl:versionInfo` is a literal and the tag can be used to annotate classes and properties in addition to ontologies.

For many purposes, doing version tracking at the granularity of an entire ontology is not enough. Maintainers may wish to keep version information for classes, properties, and individuals - and even that may not be sufficient. The incremental nature of class expressions in OWL implies that one ontology may add restrictions to a (named) class defined in another ontology, and these additional restrictions themselves may require

version information.

OWL Full provides the expressive power to make any sort of assertion about a class, i.e. that it is an instance of another class, or that it (and not its instances) has a property and a value for that property. This framework can be used to build an ontology of classes and properties for tracking version information. The OWL namespace includes two pre-defined classes that can be used for this purpose: `owl:DeprecatedClass` and `owl:DeprecatedProperty`. They are intended to indicate that the class or property will likely be changing in an incompatible manner in a forthcoming release:

```
...
  <owl:DeprecatedClass rdf:ID="&vin;JugWine" />
  <owl:DeprecatedProperty rdf:ID="&vin;hasSeeds" />
...
```

It is important to note that `owl:DeprecatedClass` and `owl:DeprecatedProperty` have no additional semantics and it is up to tool developers and OWL users to ensure they are used as intended.

7. Usage Examples

Once an initial domain ontology is available, a large number of applications can be developed that exploit the ontology. In this section, we describe some sample uses in the domain of wines.

7.1. Wine Portal

A number of sites exist today that call themselves wine portals. Google for example, provides 152,000 matches for the query "wine portal". One of the top matches, a site called "[Wine-Portal.com](#)", provides access to a number of sites. Many sites claiming to be wine portals are mostly informational sites. For example, wine-portal.com's first featured site, called 'cork cuisine' (www.corkcuisine.com/), provides information about matching wines and foods, wines as gifts, etc.

Perusing any of the topic areas, one finds a collection of pages containing information and sometimes services related to the topic. For example, 'accessories and gifts' contains information about what to look for when buying particular wine items and also contains a significant number of online retailers. Another top level area called 'shopping' has a subarea called 'wine shopping' from which a user can find online (or 'street shopping') stores (categorized by country). These two sites are just two of the many examples today and are representative of the general notion of a wine portal providing a collection of information and services connected to a particular topic area.

When looking at these sites in some detail, it is not clear how much they depend on ontologies today. For example, viewing the source for the html does not reveal evidence of ontological usage. However, it is clear that the sites could exploit ontologies had some wine ontologies been available.

One simple use of ontologies in portal sites is for organization and browsing. The listing of categories above could be generated from the top few levels of wine related classes. Queries could exploit wine ontologies to *retrieve* wine relevant information. If one did a search for a term contained in the ontology, the query could be expanded with subclass information in order to find more relevant answers. Portals could be made to automatically update themselves with (candidate) information in topic areas. With very powerful

reasoning capabilities they could even identify likely wine sales sites and negotiate to include them as part of the portal.

7.2. Wine Agent

We have started a [wine agent](#) for expository purposes. In our initial design, the wine agent's goal is to recommend wines to accompany meal courses. This application exploits the ontology used as the basis of this guide. This wine ontology is available in the DAML ontology library and is entitled [wines](#).

A personalized wine agent can provide a number of services for a human. The agent may be used to recommend wines given a set of constraints (such as a meal being served), the agent may find information about a particular wine or a particular class of wines, it may look for appropriate accessories for a wine (such as a particular kind of glass suited for that wine varietal, etc.).

Below, we describe an example in a simple prototype system that is being written as a student project.

Consider the following scenario:

Someone is planning a dinner party and at least one of the guests is wine knowledgeable. The host would like to serve wine that is well matched to the course(s) on the menu. The host would also like to appear knowledgeable about the wines served at the event. The host would also like to have appropriate accessories at the dinner. The host may have decided to serve a special tomato based pasta sauce with fresh pasta as the main course.

In order to serve wines appropriate to the meal, the host needs information concerning wine and food pairings. In order to appear knowledgeable about wines, the host would benefit from having access to wine information relevant to the event. In order to have appropriate wine accessories, the host would need to have information about what accessories are relevant to the situation (and are within the host's price range).

With a background wine ontology, given a description of a meal, a wine agent can suggest the type of wine to serve with the meal. The wine agent may suggest a zinfandel as the varietal of choice for the meal. Additionally, given a background ontology, the wine agent may suggest a particular zinfandel, possibly Marietta Zinfandel. Given the information that the wine should be a zinfandel, a wine agent may look for a place to acquire either a selection of zinfandels or it may look for a particular zinfandel wine, such as Marietta. Given a background ontology containing appropriate sources for wine purchases (possibly filtered by the location of the host and the location of the wine seller), the wine agent could go to a site such as [wine.com](#) and do a search for 'zinfandels' returning a listing of zinfandels for sale on that site. The wine agent could attempt to find [Marietta Zinfandel](#) either from the winery itself or from other resellers. It could, for example, find (by a search on Google or a structured search of selected Web sites) that winelibrary.com has a sale on Marietta Zinfandel 1999 vintage for a discounted price of \$13.99. The wine agent could use additional filtering information such as price ranges provided either by the consumer or as suggestions based on varietal.

The wine agent may now attempt to provide information concerning zinfandel in general or Marietta Zinfandel in particular. It could use a background ontology of wine sites to find information about particular wines. For example, the winery [description](#) of their most recent Zinfandel may be of use. Additionally reviews from respected sources such as the [Wine Spectator](#) may be of use. If no review of Marietta Zinfandel is available on a favorite wine review site, it may be useful to look for related information such as reviews on zinfandels from the same region, in this case zinfandels from Sonoma County, California.

General background information may also be of use. The host may also want to do some reading and may be interested in books on wine in general or zinfandels in particular. For example, the host may be interested in the books that Amazon.com has for [sale](#) on zinfandel. The host may also be interested in information concerning wines from the same region, and thus may be interested in Sonoma zinfandels. A wine agent may have typical background information available that is related to its main knowledge areas. For example, this wine agent is concerned with matching foods and wines, so it may have both free and purchasable information on this topic such as the Wine Spectator's article on [matching food and wine](#).

The dinner host may also want to acquire appropriate wine accessories prior to the event. Wine is served in wine glasses and different wine varietals are best served in different kinds of glasses. For example, if the host has chosen a meal course for which a zinfandel is appropriate, the host may want to know that [Riedel](#) is a well-known manufacturer of wine glassware. The host may also want to be linked to the Wine Enthusiast (a well respected supplier of wine merchandise) and be told that the Wine Enthusiast has [Riedel's Vinum Zinfandel glass](#) for sale as a set of 4 for \$63.95 (with a discount to \$59.95 if you buy two sets of 4 glasses). The host may also be interested to know that Amazon.com has [Reidel's Sommelier Zinfandel single stem glass](#) available for \$49.99 (and claims a list price of \$65.00). Amazon also has the same Vinum glass for sale in sets of 6 (instead of 4 on the wine enthusiast) for \$79.99 (and claims a list price of \$119.40). A wine agent could provide a comparison listing of glassware that is matched to the meal (i.e., is appropriate to be used to serve zinfandel) and then is compared by price or other criteria chosen from a list of properties in the ontology.

The dinner host may want to consider other wine accessories. From the ontology, we know that corkscrews are wine accessories. The background ontology may encode subclasses of corkscrews or such information could be found from relevant wine sites as well. The Wine Enthusiast has a set of [corkscrews](#) they [recommend](#) (with descriptions of the types and price ranges). They also [distinguish](#) corkscrews by type (level, waiter, stationary, twist, and pump) and the dinner host may want to get information about those styles.

The wine agent may be taken to many levels of sophistication depending upon background ontology knowledge of the domain and information and services sites. In this example, we only exploited information concerning wines, varietal type, food and wine combinations, some wine accessories and their related properties. We could of course expand this to include more information and more constraints by the customer.

An evolving example of this wine agent is [available](#).

Acknowledgements

This document is the result of extensive discussions within the [Web Ontology Working Group](#) as a whole. The participants in this Working Group included: Yasser alSafadi, Jean-François Baget, James Barnette, Sean Bechhofer, Jonathan Borden, Frederik Brysse, Stephen Buswell, Jeremy Carroll, Dan Connolly, Peter Crowther, Jonathan Dale, Jos De Roo, David De Roure, Mike Dean, Larry Eshelman, Jérôme Euzenat, Tim Finin, Nicholas Gibbins, Sandro Hawke, Patrick Hayes, Jeff Heflin, Ziv Hellman, James Hendler, Bernard Horan, Masahiro Hori, Ian Horrocks, Jane Hunter, Francesco Iannuzzelli, Rüdiger Klein, Natasha Kravtsova, Ora Lassila, Massimo Marchiori, Deborah McGuinness, Enrico Motta, Leo Obrst, Mehrdad Omidvari, Martin Pike, Marwan Sabbouh, Guus Schreiber, Noboru Shimizu, Michael Sintek, Michael K. Smith, John Stanton, Lynn Andrea Stein, Herman ter Horst, David Trastour, Frank van Harmelen, Bernard Vatant, Raphael Volz, Evan Wallace, Christopher Welty, Charles White, and John Yanosy.

Some critical early text on complex restrictions was written by [Raphael Volz](#), Forschungszentrum Informatik (FZI). Substantial insight was provided by the [DAML+OIL Walkthru](#). Jeremy Carroll, Jerome Euzenat, Jeff Heflin, Kevin Page and Peter F. Patel-Schneider provided extensive reviews. At the WG Face to Face, 8 October 2002, Stephen Buswell, Ruediger Klein, Enrico Motta, and Evan Wallace provided a detailed review of the ontology resulting in substantial changes. At the WG Face to Face, 10 January 2003, Jonathan Dale, Bernard Horan, Guus Schreiber, and Jeff Heflin provided detailed reviews of the Guide resulting in changes. The [public reviews](#) provided numerous helpful suggestions and corrections.

OWL Glossary

Attribute

as in XML

Class Definition

informal term for an owl:Class element

Class Description

describes an OWL class, either by a class name or by specifying a class extension of an unnamed anonymous class

Class name

informal term for an owl:Class rdf:ID attribute value.

Class

as in RDF

Component

for parts of a definition e.g. the arguments to intersection-of in a class definition

Concept

informal term for the abstractions "in the world" that ontologies describe

Constraint

informal term for discussing the effect of a restriction

Data-valued Property

alternative term for DataType Property

Datatype Property

an OWL property that relates individuals to data values

Datatype

an RDFS datatype, almost always one of the built-in non-list XML Schema datatypes

Element

- (1) as in XML
- (2) an element of a set

Entity

as in XML

Imports Closure

the information in an ontology document, plus the information in the imports closure of ontology documents that are imported by the document

Individual-valued Property

alternative term for Object Property

Individual

an instance of an OWL class, i.e., a resource that belongs to the class extension of an OWL class

Instance Of

the relation between an individual and a class

Instance

a member of the class extension of an OWL class

Name

as in XML Namespaces

Named Class

an OWL class with an associated identifier

Node

as in RDF Graphs

OWL Class

an RDFS class that belongs to the class extension of owl:Class

Object Property

an OWL property that relates individuals to other individuals

Object

(1) the object of an RDF triple

(2) an alternative term for individual (used for historical reasons)

Ontology Document

a Web document that contains an ontology, generally indicated by the presence of an owl:Ontology element in the document

Ontology

(1) collection of information, generally including information about classes and properties

(2) the information contained in an ontology document

Property Definition

informal term for an owl:ObjectProperty element and or owl:DatatypeProperty element

Resource

an element of the RDF domain of discourse

Restriction, global

reserved for discussions of the `domain` and `range` of properties

Restriction, local

[see above]

Restriction

usually a piece of a class expression, a statement that expresses a constraint, local by default

Set

a mathematical set

Statement

as in RDF Graphs

Type

as in RDF (`rdf:type`)

URI reference

as in RDF

Unnamed Class

an OWL class without an associated identifier, normally components of restrictions.

Vocabulary

a set of URI references

Term Index and Cross Reference

Term Index

Term	Section
anonymous class	3.2.1.
class	3.1.3.
cardinality	3.4.2.

complement	5.1.3.
datatype	3.2.1.
datatype property	3.2.1.
domain	3.2.1.
entailed	1.
enumerated	5.
extension	3.1.
instance of	3.1.3.
intersectionOf	5.1.1.
imports	2.2.
individual	3.1.3.
instance	3.1.3.
monotonic	2.
object properties	3.2.1.
ontology	1.
open world	2.
OWL DL	1.1.
OWL Full	1.1.
OWL Lite	1.1.
property	3.2.1.
range	3.2.1.
restriction class	3.4.1.
union	5.1.2.
unique names	4.2.

Guide, Reference and Semantics Cross Reference

OWL Guide	OWL Reference	OWL Semantics
owl:AllDifferent / 4.3.	owl:AllDifferent	owl:AllDifferent
owl:allValuesFrom / 3.4.1.	owl:allValuesFrom	owl:allValuesFrom
owl:AnnotationProperty / 2.2.	owl:AnnotationProperty	owl:AnnotationProperty
owl:backwardCompatibleWith / 6.	owl:backwardCompatibleWith	owl:backwardCompatibleWith
owl:cardinality / 3.4.2.	owl:cardinality	owl:cardinality
owl:Class / 3.1.1.	owl:Class	owl:Class
owl:complementOf / 5.1.3.	owl:complementOf	owl:complementOf
	owl:DataRange	owl:DataRange
owl:DatatypeProperty / 3.2.2.	owl:DatatypeProperty	owl:DatatypeProperty
owl:DeprecatedClass / 6.	owl:DeprecatedClass	
owl:DeprecatedProperty / 6.	owl:DeprecatedProperty	
owl:differentFrom / 4.3.	owl:differentFrom	owl:differentFrom
owl:disjointWith / 5.3.	owl:disjointWith	owl:disjointWith
owl:distinctMembers / 4.3.	owl:distinctMembers	owl:distinctMembers

owl:equivalentClass / 4.1.	owl:equivalentClass	owl:equivalentClass
owl:equivalentProperty / 4.1.	owl:equivalentProperty	owl:equivalentProperty
owl:FunctionalProperty / 3.3.	owl:FunctionalProperty	owl:FunctionalProperty
owl:hasValue / 3.4.3.	owl:hasValue	owl:hasValue
owl:imports / 2.2.	owl:imports	owl:imports
owl:incompatibleWith / 6.	owl:incompatibleWith	owl:incompatibleWith
owl:intersectionOf / 5.1.1.	owl:intersectionOf	owl:intersectionOf
owl:InverseFunctionalProperty / 3.3.	owl:InverseFunctionalProperty	owl:InverseFunctionalProperty
owl:inverseOf / 3.3.	owl:inverseOf	owl:inverseOf
owl:maxCardinality / 3.4.2.	owl:maxCardinality	owl:maxCardinality
owl:minCardinality / 3.4.2.	owl:minCardinality	owl:minCardinality
owl:Nothing / 3.1.1.	owl:Nothing	owl:Nothing
owl:ObjectProperty / 3.2.1.	owl:ObjectProperty	owl:ObjectProperty
owl:oneOf / 5.2.	owl:oneOf	owl:oneOf
owl:onProperty / 3.4.	owl:onProperty	owl:onProperty
owl:Ontology / 2.2.	owl:Ontology	owl:Ontology
	owl:OntologyProperty	owl:OntologyProperty
owl:priorVersion / 6.	owl:priorVersion	owl:priorVersion
owl:Restriction / 3.4.	owl:Restriction	owl:Restriction
owl:sameAs / 4.2.	owl:sameAs	owl:sameAs
owl:someValuesFrom / 3.4.1.	owl:someValuesFrom	owl:someValuesFrom
owl:SymmetricProperty / 3.3.	owl:SymmetricProperty	owl:SymmetricProperty
owl:Thing / 3.1.1.	owl:Thing	owl:Thing
owl:TransitiveProperty / 3.3.	owl:TransitiveProperty	owl:TransitiveProperty
owl:unionOf / 5.1.2.	owl:unionOf	owl:unionOf
owl:versionInfo / 6.	owl:versionInfo	owl:versionInfo
		rdf:List
		rdf:nil
rdf:type		rdf:type
rdfs:comment / 2.2.		rdfs:comment
rdfs:Datatype / 3.2.2.		rdfs:Datatype
rdfs:domain / 3.2.1.	rdfs:domain	rdfs:domain
rdfs:label / 3.1.1.		rdfs:label
rdfs:Literal / 3.3.		rdfs:Literal
rdfs:range / 3.2.1.	rdfs:range	rdfs:range
rdfs:subClassOf / 3.1.1.	rdfs:subClassOf	rdfs:subClassOf
rdfs:subPropertyOf / 3.2.1.	rdfs:subPropertyOf	rdfs:subPropertyOf

References

OWL

[OWL Semantics and Abstract Syntax]

[OWL Web Ontology Language Semantics and Abstract Syntax](#), Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks, Editors. W3C Recommendation, 10 February 2004,
<http://www.w3.org/TR/2004/REC-owl-semantics-20040210/>.
[Latest version](#) available at <http://www.w3.org/TR/owl-semantics/>.

[OWL Overview]

[OWL Web Ontology Language Overview](#), Deborah L. McGuinness and Frank van Harmelen, Editors. W3C Recommendation, 10 February 2004,
<http://www.w3.org/TR/2004/REC-owl-features-20040210/>.
[Latest version](#) available at <http://www.w3.org/TR/owl-features/>.

[OWL Reference]

[OWL Web Ontology Language Reference](#), Mike Dean and Guus Schreiber, Editors. W3C Recommendation, 10 February 2004,
<http://www.w3.org/TR/2004/REC-owl-ref-20040210/>.
[Latest version](#) available at <http://www.w3.org/TR/owl-ref/>.

[OWL Requirements]

[OWL Web Ontology Language Use Cases and Requirements](#), Jeff Heflin, Editor. W3C Recommendation, 10 February 2004,
<http://www.w3.org/TR/2004/REC-webont-req-20040210/>.
[Latest version](#) available at <http://www.w3.org/TR/webont-req/>.

[OWL Test Cases]

[OWL Web Ontology Language Test Cases](#), Jeremy J. Carroll and Jos De Roo, Editors. W3C Recommendation, 10 February 2004,
<http://www.w3.org/TR/2004/REC-owl-test-20040210/>.
[Latest version](#) available at <http://www.w3.org/TR/owl-test/>.

Related W3C Standards

[RDF]

[Resource Description Framework \(RDF\) Model and Syntax Specification](#), Ora Lassila, Ralph R. Swick, Editors. World Wide Web Consortium Recommendation, 1999,
<http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/>.
[Latest version](#) available at <http://www.w3.org/TR/REC-rdf-syntax/>.

[RDFS]

[RDF Vocabulary Description Language 1.0: RDF Schema](#), Dan Brickley and R.V. Guha, Editors. W3C Recommendation, 10 February 2004,
<http://www.w3.org/TR/2004/REC-rdf-schema-20040210/>.
[Latest version](#) available at <http://www.w3.org/TR/rdf-schema/>.

[RDF Concepts]

[Resource Description Framework \(RDF\): Concepts and Abstract Syntax](#), Graham Klyne and Jeremy J. Carroll, Editors. W3C Recommendation, 10 February 2004,
<http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/>.
[Latest version](#) available at <http://www.w3.org/TR/rdf-concepts/>.

[RDF Semantics]

[RDF Semantics](#), Patrick Hayes, Editor. W3C Recommendation, 10 February 2004,
<http://www.w3.org/TR/2004/REC-rdf-mt-20040210/>.
[Latest version](#) available at <http://www.w3.org/TR/rdf-mt/>.

[RDF Syntax]

[RDF/XML Syntax Specification \(Revised\)](#), Dave Beckett, Editor. W3C Recommendation, 10 February 2004,
<http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/>.
[Latest version](#) available at <http://www.w3.org/TR/rdf-syntax-grammar/>.

[URI]

[Uniform Resource Identifiers \(URI\): Generic Syntax](#), T. Berners-Lee, R. Fielding, and L. Masinter, IETF Draft Standard August, 1998 (RFC 2396).

[XML Base]

[XML Base](#), Jonathan Marsh, Editor. W3C Recommendation, 27 June 2001, <http://www.w3.org/TR/2001/REC-xmlbase-20010627/>.

[Latest version](#) available at <http://www.w3.org/TR/xmlbase/>.

[XML Namespaces]

[Namespaces in XML](#), Tim Bray, Dave Hollander, Andrew Layman, Editors. W3C Recommendation, Jan 1999, <http://www.w3.org/TR/1999/REC-xml-names-19990114/>.

[Latest version](#) available at <http://www.w3.org/TR/REC-xml-names>.

[XML]

[Extensible Markup Language \(XML\) 1.0](#), Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Editors. W3C Recommendation, 10 February 1998, <http://www.w3.org/TR/1998/REC-xml-19980210>.

[Latest version](#) available at <http://www.w3.org/TR/REC-xml>.

[XML Schema 1]

[XML Schema Part 1: Structures](#), Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn, Editors. W3C Recommendation, 2 May 2001, <http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/>.

[Latest version](#) available at <http://www.w3.org/TR/xmlschema-1/>.

[XML Schema 2]

[XML Schema Part 2: Datatypes](#), Paul V. Biron, Ashok Malhotra, Editors. W3C Recommendation, 2 May 2001, <http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/>.

[Latest version](#) available at <http://www.w3.org/TR/xmlschema-2/>.

Sample Ontologies and Applications

[Integrating Applications]

[Integrating Applications on the Semantic Web](#), James Hendler, Tim Berners-Lee, and Eric Miller. *Journal of the Institute of Electrical Engineers of Japan*, Vol 122(10), October, 2002, p. 676-680.

[VerticalNet]

[Industrial Strength Ontology Management](#), Aseem Das, Wei Wu, and Deborah L. McGuinness. Stanford Knowledge Systems Laboratory Technical Report KSL-01-09 2001. In the *Proceedings of the International Semantic Web Working Symposium*, Stanford, CA, July 2001.

[Wine Ontology From Daml.org]

<http://www.daml.org/ontologies/76>

[Wine Ontology / CLASSIC Tutorial]

[Classic Knowledge Representation System Tutorial](#), Deborah L. McGuinness, Peter F. Patel-Schneider, Richmond H. Thomason, Merryll K. Abrahams, Lori Alperin Resnick, Violetta Cavalli-Sforza, and Cristina Conati. AT&T Bell Laboratories and University of Pittsburgh, 1994.

[Wine Ontology Tutorial]

[Ontology Development 101: A Guide to Creating Your First Ontology](#), Natalya Fridman Noy and Deborah L. McGuinness. Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880, March 2001.

[Wine Ontology in CLASSIC]

[Living with CLASSIC: When and How to Use a KL-ONE-Like Language](#), Ronald J. Brachman, Deborah L. McGuinness, Peter F. Patel-Schneider, Lori Alperin Resnick, and Alex Borgida. In *Principles of Semantic Networks: Explorations in the representation of knowledge*, John Sowa, Editor. Morgan-Kaufmann, San Mateo, California, 1991, pages 401--456.

Related KR Language Research

[DAML+OIL]

[DAML+OIL W3C Submission](#), Includes reference description, both model theoretic and axiomatic semantics, annotated walkthrough and examples.

[Annotated DAML+OIL Ontology Markup](#), Dan Connolly, Frank van Harmelen, Ian Horrocks, Deborah McGuinness, Peter F. Patel-Schneider, Lynn Andrea Stein. December 2001.

[DAML-ONT]

[DAML-ONT initial release](#) at <http://www.daml.org/2000/10/daml-ont.html>.

[DAML-ONT KIF]

[Partial DAML-ONT axiomatization](#) at <http://www.daml.org/2000/10/DAML-Ont-kif-axioms-001107.html>.
Defined in [KIF](#) (<http://logic.stanford.edu/kif/kif.html>).

[Description Logics]

[The Description Logic Handbook: Theory, Implementation and Application](#), Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, Editors. Cambridge University Press, 2002.

[MCF]

[Meta Content Framework Using XML](#), R.V. Guha and Tim Bray. Netscape Communications, 6 June 1997.

[Part Whole]

A Taxonomy of Part-Whole Relations. M. Winston, R. Chaffin & D. Herrmann. *Cognitive Science*, 11:417-444, 1987.

[XOL]

[XOL: An XML-Based Ontology Exchange Language](#), Peter D. Karp, Vinay K. Chaudhri, and Jerome F. Thomere. Technical Report 559. AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, Jul 1999.

Background Information and Home Pages

[Dublin Core]

[Dublin Core Metadata](#) at home page, <http://dublincore.org/>.

[Dublin Core XML]

[Expressing Dublin Core in RDF/XML](#), Dave Beckett, Eric Brickley, and Dan Brickley. Dublin Core Metadata Initiative. July 31, 2002, <http://dublincore.org/documents/2001/11/28/dcmes-xml/>.

[KAON]

[The Karlsruhe Ontology and Semantic Web Tool Suite](#) at <http://kaon.semanticweb.org>.

[OIL]

[OIL Home Page](#) at <http://oil.semanticweb.org/>.

[Ontobroker]

[The Ontobroker home page](#) at http://ontobroker.aifb.uni-karlsruhe.de/index_ob.html.
Institute AIFB, University of Karlsruhe.

[Ontoknowledge]

[Ontoknowledge Home Page](#) at <http://www.ontoknowledge.org/>.

[RDF Home]

[RDF: Resource Description Framework](#). Background information at <http://www.w3.org/RDF/>.

[SHOE]

[Simple HTML Ontology Extensions \(SHOE\) home page](#) at <http://www.cs.umd.edu/projects/plus/SHOE/>. University of Maryland.

[KR]

[KR Home Page](#) at <http://kr.org/>.

Appendix A: XML + RDF Basics

This appendix provides links to introductions to the standards that OWL depends on.

To fully understand the OWL syntax and semantics you should be familiar with the basics of the related W3C and IETF standards listed below. A minimal guide to XML and RDF is provided by the first two links below.

- [Appendix](#) to the [DAML+OIL Walkthru](#)
- [URI - Uniform Resource Identifier](#)
- [XML - eXtensible Markup Language](#)
- [XML Namespaces](#)
- [XML Schema](#)
- [RDF - Resource Description Framework](#)
- [RDF Schema](#)

Appendix B: History

[The Resource Description Framework \(RDF\)](#) was the first language specified by the W3C for representing semantic information about arbitrary resources. [RDF Schema \(RDFS\)](#) is a W3C candidate recommendation for an extension to RDF to describe RDF vocabularies. RDFS can be used to create ontologies, but it is purposefully lightweight, with less expressive power than OWL.

Like OWL, RDFS includes classes and properties, as well as range and domain constraints on properties. It provides inheritance hierarchies for both classes and properties. Upon its release users began requesting additional features, including data types, enumerations and the ability to define properties more rigorously.

Other efforts in the research community were already examining exactly these sorts of features. For those who wish to delve more deeply into this background, a partial list of projects and languages includes:

- [DAML - DARPA Agent Markup Language](#)
- [DAML-ONT](#)
- [MCF - Meta Content Framework](#).
- [Ontobroker](#)
- [On-To-Knowledge](#)
- [OIL - Ontology Inference Layer](#)
- [SHOE - Simple HTML Ontology Extensions](#)
- [XOL](#)

Instead of continuing with separate ontology languages for the Semantic Web, a group of researchers, including many of the main participants in both the OIL and DAML-ONT efforts, got together in the [Joint US/EU ad hoc Agent Markup Language Committee](#) to create a new Web ontology language. This language [DAML+OIL](#) built on both OIL and DAML-ONT, was [submitted](#) to the W3C as a proposed basis for OWL, and was subsequently selected as the starting point for OWL.

In addition to ontology languages, various taxonomies and existing ontologies are already in

use commercially. In e-Commerce sites they facilitate machine-based communication between buyer and seller, enable vertical integration of markets and allow descriptions to be reused in different marketplaces. Examples of sites that are actually making commercial use ontologies include:

- [VerticalNet](#) Vertical Net currently hosts 59 industry-specific e-marketplaces that span diverse industries such as manufacturing, communications, energy, and healthcare.

Various medical or drug-related ontologies have been developed to help manage the overwhelming mass of current medical and biochemical research data that can be difficult to tie together into a cohesive whole. One major resource is the [Gene Ontology Consortium](#) which is defining ontologies for

- Molecular Function,
- Biological Process, and
- Cellular Components.

That site also has pointers to ontologies for

- sequence attributes,
- gene product attributes,
- chemical substances,
- pathways,
- anatomies,
- pathology,
- physical characteristics,
- experiment attributes,
- classification, and
- pathology.

There exist large taxonomies in use today that would be ripe for extension into the OWL space. For example, the North American Industry Classification System (NAICS) defines a hierarchy of over 1900 items that identify industry types. NAICS is also tied to the International Standard Industrial Classification System (ISIC, Revision 3), developed and maintained by the United Nations.

Appendix C: Change Log Since Proposed Recommendation, 15 December 2003

- Corrected a typo (allValuesFrom to someValuesFrom) in the TexasThings example in 4.1.
- Added anchor for owl:AnnotationProperty.
- In 3.1.1. added explicit ontology prefix for 'about' syntax example.
- Added a cross reference to the description of the use of owl:oneOf in the *Reference* at the end of 5.2.
- Made a number of minor corrections suggested by a [message](#) to public-webont-comments.
- Modified citations for consistency with other OWL documents.
- Fixed typos related to numbering in namespace references.