SKOS Simple Knowledge Organization System Reference

W3C Recommendation 18 August 2009

This version:
http://www.w3.org/TR/2009/REC-skos-reference-20090818/
Latest version:
http://www.w3.org/TR/skos-reference
Previous versions:
http://www.w3.org/TR/2009/PR-skos-reference-20090615/
Editors:
Alistair Miles, STFC Rutherford Appleton Laboratory / University of Oxford
Sean Bechhofer, University of Manchester

Please refer to the **errata** for this document, which may include some normative corrections.

See also **translations**.

Abstract

This document defines the Simple Knowledge Organization System (SKOS), a common data model for sharing and linking knowledge organization systems via the Web.

Many knowledge organization systems, such as thesauri, taxonomies, classification schemes and subject heading systems, share a similar structure, and are used in similar applications. SKOS captures much of this similarity and makes it explicit, to enable data and technology sharing across diverse applications.

The SKOS data model provides a standard, low-cost migration path for porting existing knowledge organization systems to the Semantic Web. SKOS also provides a lightweight, intuitive language for developing and sharing new knowledge organization systems. It may be used on its own, or in combination with formal knowledge representation languages such as the Web Ontology language (OWL).

This document is the normative specification of the Simple Knowledge Organization System. It is intended for readers who are involved in the design and implementation of information systems, and who already have a good understanding of Semantic Web technology, especially RDF and OWL.

For an informative guide to using SKOS, see the [SKOS-PRIMER].

Synopsis

Using SKOS, **concepts** can be identified using URIs, **labeled** with lexical strings in one or more natural languages, assigned **notations** (lexical codes), **documented** with various types of note, **linked to other concepts** and organized into informal hierarchies and association networks, aggregated into **concept schemes**, grouped into labeled and/or ordered **collections**, and **mapped** to concepts in other schemes.

[show quick access panel]

Status of This Document

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/.

This document is a W3C Recommendation developed by the Semantic Web Deployment Working Group, part of the W3C Semantic Web Activity. This document reflects an editorial change arising during the Proposed Recommendation review: a non-normative example and preceding text was removed that suggested one means to reference a system of notation (e.g. a symbolic notation) in a label where the system of notation does not correspond to a natural language. This suggestion was deemed inconsistent with IETF Best Current Practice 47 on the use of tags for identifying languages. Users should consider the SKOS Extension vocabulary for support of alternate systems of notation. An implementation report documents known uses of SKOS during the Candidate Recommendation review period. An updated SKOS Primer is being published concurrently with this Recommendation.

Changes Since 15 June 2009 Proposed Recommendation:

- Removed final paragraph and example from section 6.5.4 on use of private use language sub-tags.
- Editorial changes to references section and citation of SKOS Primer.

Comments on this document may be sent to public-swd-wg@w3.org with public archive.

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who

Table of Contents

[show quick access panel]

1. Introduction

1.1. Background and Motivation

The Simple Knowledge Organization System is a data-sharing standard, bridging several different fields of knowledge, technology and practice.

In the library and information sciences, a long and distinguished heritage is devoted to developing tools for organizing large collections of objects such as books or museum artifacts. These tools are known generally as "knowledge organization systems" (KOS) or sometimes as "controlled structured vocabularies". Several similar yet distinct traditions have emerged over time, each supported by a community of practice and set of agreed standards. Different families of knowledge organization systems, including thesauri, classification schemes, subject heading systems, and taxonomies are widely recognized and applied in both modern and traditional information systems. In practice it can be hard to draw an absolute distinction between thesauri and classification schemes or taxonomies, although some properties can be used to broadly characterize these different families (see e.g., [BS8723-3]). The important point for SKOS is that, in addition to their unique features, each of these families shares much in common, and can often be used in similar ways [SKOS-UCR]. However, there is currently no widely deployed standard for representing these knowledge organization systems as data and exchanging them between computer systems.

The W3C's Semantic Web Activity [SW] has stimulated a new field of integrative research and technology development, at the boundaries between database systems, formal logic and the World Wide Web. This work has led to the development of foundational standards for the Semantic Web. The Resource Description Framework (RDF) provides a common data abstraction and syntax for the Web [RDF-PRIMER]. The RDF Vocabulary Description language (RDFS) and the Web Ontology language (OWL) together provide a common data modeling (schema) language for data in the Web [RDFS] [OWL-GUIDE]. The SPARQL Query Language and Protocol provide a standard means for interacting with data in the Web [SPARQL].

These technologies are being applied across diverse applications, because many applications require a common framework for publishing, sharing, exchanging and integrating ("joining up") data from different sources. The ability to link data from different sources is motivating many projects, as different communities seek to exploit the hidden value in data previously spread across isolated sources.

One facet of the Semantic Web vision is the hope of better organizing the vast amounts of unstructured (i.e., human-readable) information in the Web, providing new routes to discovering and sharing that information. RDFS and OWL are formally defined knowledge representation languages, providing ways of expressing meaning that are amenable to computation, meaning that complements and gives structure to information already present in the Web [RDF-PRIMER] [OWL-GUIDE]. However, to actually apply these technologies over large bodies of information requires the construction of detailed maps of particular domains of knowledge, in addition to the accurate description (i.e., annotation or cataloging) of information resources on a large scale, much of which cannot be done automatically. The accumulated experience and best practices in the library and information sciences regarding the organization of information and knowledge are obviously complementary and applicable to this vision, as are the many existing knowledge organization systems already developed and in use, such as the Library of Congress Subject Headings [LCSH] or the United Nations Food and Agriculture Organization's AGROVOC Thesaurus [AGROVOC].

The Simple Knowledge Organization System therefore aims to provide a bridge between different communities of practice within the library and information sciences involved in the design and application of knowledge organization systems. In addition, SKOS aims to provide a bridge between these communities and the Semantic Web, by transferring existing models of knowledge organization to the Semantic Web technology context, and by providing a low-cost migration path for porting existing knowledge organization systems to RDF.

Looking to the future, SKOS occupies a position between the exploitation and analysis of unstructured information, the informal and socially-mediated organization of information on a large scale, and the formal representation of knowledge. By making the accumulated experience and wisdom of knowledge organization in the library and information sciences accessible, applicable and transferable to the technological context of the Semantic Web, in a way that is complementary to existing Semantic Web technology (and in particular formal systems of knowledge representation such as OWL), it is hoped that SKOS will enable many new and valuable applications, and will also lead to new integrative lines of research and development in both technology and practice.

1.2. SKOS Overview

The Simple Knowledge Organization System is a common data model for knowledge organization systems such as thesauri, classification schemes, subject heading systems and taxonomies. Using SKOS, a knowledge organization system can be expressed **as machine-readable data**. It can then be exchanged between computer applications and published in a machine-readable format in the Web.

The SKOS data model is formally defined in this specification as an OWL Full ontology [OWL-SEMANTICS]. SKOS data are expressed as RDF triples [RDF-CONCEPTS], and may be encoded using any concrete RDF syntax (such as RDF/XML [RDF-XML] or Turtle [TURTLE]). For more on the relationships between SKOS, RDF and OWL, see the next sub-section below.

The SKOS data model views a knowledge organization system as a **concept scheme** comprising a set of **concepts**. These SKOS concept schemes and SKOS concepts are identified by URIs, enabling anyone to refer to them unambiguously from any context, and making them a part of the World Wide Web. See Section 3. The skos:Concept Class for more on identifying and describing SKOS concepts, and Section 4. Concept Schemes for more on concept schemes.

SKOS concepts can be **labeled** with any number of lexical (UNICODE) strings, such as "romantic love" or "れんあい", in any given natural language, such as English or Japanese (written here in hiragana). One of these labels in any given language can be indicated as the preferred label for that language, and the others as alternative labels. Labels may also be "hidden", which is useful where a knowledge organization system is being queried via a text index. See Section 5. Lexical Labels for more on the SKOS lexical labeling properties.

SKOS concepts can be assigned one or more **notations**, which are lexical codes used to uniquely identify the concept within the scope of a given concept scheme. While URIs are the preferred means of identifying SKOS concepts within computer systems, notations provide a bridge to other systems of identification already in use such as classification codes used in library catalogs. See Section 6. Notations for more on notations.

SKOS concepts can be **documented** with notes of various types. The SKOS data model provides a basic set of documentation properties, supporting scope notes, definitions and editorial notes, among others. This set is not meant to be exhaustive, but rather to provide a framework that can be extended by third parties to provide support for more specific types of note. See Section 7. Documentation Properties for more on notes.

SKOS concepts can be **linked** to other SKOS concepts via semantic relation properties. The SKOS data model provides support for hierarchical and associative links between SKOS concepts. Again, as with any part of the SKOS data model, these can be extended by third parties to provide support for more specific needs. See Section 8. Semantic Relations for more on linking SKOS concepts.

SKOS concepts can be grouped into **collections**, which can be labeled and/or ordered. This feature of the SKOS data model is intended to provide support for node labels within thesauri, and for situations where the ordering of a set of concepts is meaningful or provides some useful information. See Section 9. Concept Collections for more on collections.

SKOS concepts can be **mapped** to other SKOS concepts in different concept schemes. The SKOS data model provides support for four basic types of mapping link: hierarchical, associative, close equivalent and exact equivalent. See Section 10. Mapping Properties for more on mapping.

Finally, an optional extension to SKOS is defined in Appendix B. SKOS eXtension for Labels (SKOS-XL). SKOS-XL provides more support for identifying, describing and linking lexical entities.

1.3. SKOS, RDF and OWL

The elements of the SKOS data model are classes and properties, and the structure and integrity of the data model is defined by the logical characteristics of, and interdependencies between, those classes and properties. This is perhaps one of the most powerful and yet potentially confusing aspects of SKOS, because SKOS can, in more advanced applications, also be used side-by-side with OWL to express and exchange knowledge about a domain. However, SKOS is **not** a formal knowledge representation language.

To understand this distinction, consider that the "knowledge" made explicit in a formal ontology is expressed as sets of axioms and facts. A thesaurus or classification scheme is of a completely different nature, and does not assert any axioms or facts. Rather, a thesaurus or classification scheme identifies and describes, through natural language and other informal means, a set of distinct ideas or meanings, which are sometimes conveniently referred to as "concepts". These "concepts" may also be arranged and organized into various structures, most commonly hierarchies and association networks. These structures, however, do not have any formal semantics, and cannot be reliably interpreted as either formal axioms or facts about the world. Indeed they were never intended to be so, for they serve only to provide a convenient and intuitive map of some subject domain, which can then be used as an aid to organizing and finding objects, such as documents, which are relevant to that domain.

To make the "knowledge" embedded in a thesaurus or classification scheme explicit in any formal sense requires that the thesaurus or classification scheme be *re-engineered* as a formal ontology. In other words, some person has to do the work of transforming the structure and intellectual content of a thesaurus or classification scheme into a set of formal axioms and facts. This work of transformation is both intellectually demanding and time consuming, and therefore costly. Much can be gained from using thesauri, etc., as-is, as informal, convenient structures for navigation within a subject domain. Using them as-is does not require any re-engineering and is therefore much less costly. In addition, some KOS are, by design, not intended to represent a logical view of their domain. Converting such KOS to a formal logic-based representation may, in practice, involve changes which result in a representation that no longer meets the originally intended purpose.

OWL does, however, provide a powerful data modeling language. We can, therefore, use OWL to construct a data model for representing thesauri or classification schemes as-is. This is exactly what SKOS does. Taking this approach, the "concepts" of a thesaurus or classification scheme are modeled as individuals in the SKOS data model, and the informal descriptions about and links between those "concepts" as given by the thesaurus or classification scheme are modeled as facts about those individuals, never as class or property axioms. Note that these are facts *about* the thesaurus or classification scheme *itself*, such as "concept X has preferred label 'Y' and is part of thesaurus Z"; these are **not** facts about the way the world is arranged within a particular subject domain, as might be expressed in a formal ontology.

SKOS data are then expressed as RDF triples. The RDF graph below (in [TURTLE] as discussed in Section 1.7.3) expresses some facts about a thesaurus.

```
<A> rdf:type skos:Concept ;
  skos:prefLabel "love"@en ;
  skos:altLabel "adoration"@en ;
  skos:broader <B> ;
  skos:inScheme <S> .

<B> rdf:type skos:Concept ;
  skos:prefLabel "emotion"@en ;
  skos:altLabel "feeling"@en ;
  skos:topConceptOf <S> .

<S> rdf:type skos:ConceptScheme ;
  dct:title "My First Thesaurus" ;
  skos:hasTopConcept <B> .
```

This point is vital to understanding the formal definition of the SKOS data model and how it may be implemented in software systems. This point is also vital to more advanced applications of SKOS, especially where SKOS and OWL are used in combination as part of a hybrid formal/semi-formal design.

From a user's point of view, however, the distinction between a formal knowledge representation system and an informal or semi-formal knowledge organization system may naturally become blurred. In other words, it may not be relevant to a user that `<A>` and `<B>` in the graph below are individuals (instances of `skos:Concept`), and `<C>` and `<D>` are classes (instances of `owl:Class`) .

```
<A> rdf:type skos:Concept ;
  skos:prefLabel "love"@en ;
  skos:broader <B> .

<B> rdf:type skos:Concept ;
  skos:prefLabel "emotion"@en .

<C> rdf:type owl:Class ;
  rdfs:label "mammals"@en ;
  rdfs:subClassOf <D> .

<D> rdf:type owl:Class ;
  rdfs:label "animals"@en .
```

An information system that has any awareness of the SKOS data model will, however, need to appreciate the distinction.

RDF schemas for SKOS and the SKOS eXtension for Labels (SKOS-XL) are described in Appendix C. SKOS and SKOS-XL Namespace Documents. Note that, as there are constraints that cannot be completely captured in the schema, the RDF/XML document provides a normative subset of this specification.

1.4. Consistency and Integrity

Under the RDF and OWL Full semantics, the formal meaning (*interpretation*) of an RDF graph is a truth value [RDF-SEMANTICS] [OWL-SEMANTICS], i.e., an RDF graph is interpreted as either true or false.

In general, an RDF graph is said to be *inconsistent* if it cannot possibly be true. In other words, an RDF graph is inconsistent if it contains a contradiction.

Using the RDF and RDFS vocabularies alone, it is virtually impossible to make a contradictory statement. When the OWL vocabulary is used as well, there are many ways to state a contradiction, e.g., consider the RDF graph below.

```
<Dog> rdf:type owl:Class .
<Cat> rdf:type owl:Class .
<Dog> owl:disjointWith <Cat> .
<dogcat> rdf:type <Dog> , <Cat> .
```

The graph states that `<Dog>` and `<Cat>` are both classes, and that they are disjoint, i.e., that they do not have any members in common. This is contradicted by the statement that `<dogcat>` has type both `<Dog>` and `<Cat>`. There is no OWL Full interpretation which can satisfy this graph, and therefore this graph is **not** OWL Full consistent.

When OWL Full is used as a knowledge representation language, the notion of inconsistency is useful because it reveals contradictions within the axioms and facts that are asserted in an ontology. By resolving these inconsistencies we learn more about a domain of knowledge, and come to a better model of that domain from which interesting and valid inferences can be drawn.

When OWL Full is used as a data modeling (i.e., schema) language, the notion of inconsistency is again useful, but in a different way. Here we are not concerned with the logical consistency of human knowledge itself. We are simply interested in formally defining a data model, so that we can establish with certainty whether or not some given data fit with (i.e., conform to) the given data model. If the data are inconsistent with respect to the data model, then the data does not fit.

Here, we are not concerned with whether or not some given data have any correspondence with the real world, i.e., whether they are true or false in any absolute sense. We are simply interested in whether or not the data fit the data model, because interoperability within a given class of applications depends on data conforming to a common data model.

Another way to express this view is via the notion of *integrity*. Integrity conditions are statements within the formal definition of a data model, which are used to establish whether or not given data are consistent with respect to the data model, e.g., the statement that `<Dog>` and `<Cat>` are disjoint classes can be viewed as an integrity condition on a data model. Given this condition, the data below are then not consistent.

```
<dogcat> rdf:type <Dog> , <Cat> .
```

The definition of the SKOS data model given in this document contains a limited number of statements that are intended as integrity conditions. These integrity conditions are included to promote interoperability, by defining the circumstances under which data are **not consistent** with respect to the SKOS data model. Tools can then be implemented which check whether some or all of these integrity conditions are met for given data, and therefore whether the data are consistent with the SKOS data model.

These integrity conditions are part of the formal definition of the classes and properties of the SKOS data model. However, they are presented separately from other parts of the formal definition because they serve a different purpose. Integrity conditions serve primarily to establish whether given data are consistent with the SKOS data model. All other statements within the definition of the SKOS data model serve **only** to support logical inferences. (See also the next sub-section.)

Integrity conditions are defined for the SKOS data model in a way that is independent of strategies for their implementation, in so far as that is possible. This is because there are several different ways in which a procedure to find inconsistencies with the SKOS data model could be implemented. Inconsistencies could be found using an OWL reasoner. Alternatively, some inconsistencies could be found by searching for specific patterns within the data, or by a hybrid strategy (e.g., drawing inferences using an RDFS or OWL reasoner, then searching for patterns in the inferred graph).

The integrity conditions on the SKOS data model are fewer than might be expected, especially for those used to working within the closed world of database systems. See also the next sub-section, and the notes in sections 3-10 below.

1.5. Inference, Dependency and the Open World Assumption

This document defines the SKOS data model as an OWL Full ontology. There are other ways in which the SKOS data model could have been defined, for example as an entity-relationship model, or a UML class model. Although OWL Full as a data modeling language appears intuitively similar in many ways to these other modeling approaches, there is an important fundamental distinction.

RDF and OWL Full are designed for systems in which data may be widely distributed (e.g., the Web). As such a system becomes larger, it becomes both impractical and virtually impossible to know where all of the data in the system are located. Therefore, one cannot generally assume that data obtained from such a system are complete. If some data appear to be missing, one has to assume, in general, that the data *might* exist somewhere else in the system. This assumption, roughly speaking, is known as the **open world assumption** [OWL-GUIDE].

This means in practice that, for a data model defined as an OWL Full ontology, some definitions can have a counter-intuitive meaning. No conclusions can be drawn from missing data, and removing something will never make the remaining data inconsistent. This is illustrated, for example, by the definition of `skos:semanticRelation` in Section 8 below. The property `skos:semanticRelation` is defined to have domain and range `skos:Concept`. These domain and range definitions give license to **inferences**. Consider the graph below.

```
<A> skos:semanticRelation <B>.
```

In this case, the graph above entails the following graph.

```
<A> rdf:type skos:Concept .
<B> rdf:type skos:Concept .
```

Thus, we do not need to *explicitly* state here that `<A>` and `<B>` are instances of `skos:Concept`, because such statements are entailed by the definition of `skos:semanticRelation`.

In the SKOS data model, most statements of definition are **not** integrity conditions, but are statements of logical dependency between different elements of the data model, which (under the open world assumption) give license to a number of simple inferences. This is illustrated, for example, by the statement in Section 7 below that `skos:broader` and `skos:narrower` are inverse properties. This statement means that

```
<A> skos:narrower <B> .
```

entails

```
<B> skos:broader <A> .
```

Both of these two graphs are, by themselves, consistent with the SKOS data model.

Knowledge organization systems such as thesauri and classification schemes are applied in a wide range of situations, and an individual knowledge organization system can be used in many different information systems. By defining the SKOS data model as an OWL Full ontology, the Semantic Web can then be used as a medium for publishing, exchanging, sharing and linking data involving these knowledge organization systems. For this reason, for the expressiveness of OWL Full as a data modeling language, and for the possibility of using thesauri, classification schemes, etc., side-by-side with formal ontologies, OWL Full has been used to define the SKOS data model. The open world assumption is therefore a fundamental premise of the SKOS data model, and should be borne in mind when reading this document.

See also [RDF-PRIMER] and [OWL-GUIDE].

1.6. Design Rationale

As discussed above, the notion of a Knowledge Organization System encompasses a wide range of artifacts. There is thus a danger of overcommitment in the SKOS schema, which could preclude the use of SKOS in some applications. In order to alleviate this, in situations where there is doubt about the inclusion of a formal constraint (for example, see the discussion about `skos:hasTopConcept`), the constraint has not been stated formally. In some cases, although no formal constraint is stated, usage conventions are recommended. Applications that require more constrained behaviour may define extensions to the SKOS vocabulary. See also the [SKOS-PRIMER].

1.7. How to Read this Document

This document formally defines the Simple Knowledge Organization System data model as an OWL Full ontology. The elements of the SKOS data model are OWL classes and properties, and a Uniform Resource Identifier (URI) is provided for each of these classes and properties so that they may be used unambiguously in the Web. This set of URIs is the SKOS vocabulary.

The complete SKOS vocabulary is given in section 2 below. Sections 3 to 10 then formally define the SKOS data model. The definition of the data model is broken down into a number of sections purely for convenience. Each of these sections 3 to 10 follows a common layout:

- **Preamble** — the main ideas covered in the section are introduced informally.
- **Vocabulary** — URIs from the SKOS vocabulary which are defined in the section are given.
- **Class & Property Definitions** — the logical characteristics and interdependencies between the classes and properties denoted by those URIs are formally defined.
- **Integrity Conditions** — if there are any integrity conditions, those are given.
- **Examples** — some canonical examples are given, both of data which **are** consistent with the SKOS data model, and (where appropriate) of data which are **not** consistent with the SKOS data model.
- **Notes** — any further notes and discussion are presented.

1.7.1. Formal Definitions

Most of the class and property definitions and integrity conditions stated in this document could be stated as RDF triples, using the RDF, RDFS and OWL vocabularies. However, a small number cannot, either because of limitations in the expressiveness of OWL Full or lack of a standard URI for some class. To improve the overall readability of this document, rather than mix RDF triples and other notations, the formal definitions and integrity conditions are stated throughout using prose.

The style of this prose generally follows the style used in [RDFS], and should be clear to a reader with a working knowledge of RDF and OWL.

So, for example, "`ex:Person` is an instance of `owl:Class`" means

```
ex:Person rdf:type owl:Class .
```

"`ex:hasParent` and `ex:hasMother` are each instances of `owl:ObjectProperty`" means:

```
ex:hasParent rdf:type owl:ObjectProperty .
ex:hasMother rdf:type owl:ObjectProperty .
```

"`ex:hasMother` is a sub-property of `ex:hasParent`" means

```
ex:hasMother rdfs:subPropertyOf ex:hasParent .
```

"the `rdfs:range` of `ex:hasParent` is the class `ex:Person`" means:

```
ex:hasParent rdfs:range ex:Person .
```

Where some formal aspects of the SKOS data model cannot be stated as RDF triples using either RDF, RDFS or OWL vocabularies, it should be clear to a reader with a basic understanding of the RDF and OWL semantics how these statements might be translated into formal conditions on the interpretation of an RDF vocabulary (e.g., from Section 5, "A resource has no more than one value of `skos:prefLabel` per language tag" means for any resource x, no two members of the set { y | <x,y> is in IEXT(I(`skos:prefLabel`)) } share the same language tag, where I and IEXT are functions as defined in [RDF-SEMANTICS]).

1.7.2. URI Abbreviations

Full URIs are cited in the text of this document in monospace font, enclosed by angle brackets, e.g., `<http://example.org/ns/example>`. Relative URIs are cited in the same way, and are relative to the base URI `<http://example.org/ns/>`, e.g., `<example>` and `<http://example.org/ns/example>` are the same URI.

URIs are also cited in the text of this document in an abbreviated form. Abbreviated URIs are cited in monospace font without angle brackets, and should be expanded using the table of abbreviations below.

Table 1. URI Abbreviations

URI	Abbreviation
http://www.w3.org/2004/02/skos/core#	**skos:**

http://www.w3.org/1999/02/22-rdf-syntax-ns#	rdf:
http://www.w3.org/2000/01/rdf-schema#	rdfs:
http://www.w3.org/2002/07/owl#	owl:

So, for example, `skos:Concept` is an abbreviation of `<http://www.w3.org/2004/02/skos/core#Concept>`.

1.7.3. Examples

Examples of RDF graphs are given using the Terse RDF Triple language (Turtle) [TURTLE]. All examples assume that they are preceded by the following prefix and URI base directives:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@base <http://example.org/ns/> .
```

Therefore, the example given below

Example 1

```
<MyConcept> rdf:type skos:Concept .
```

is equivalent to the following Turtle document

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@base <http://example.org/ns/> .

<MyConcept> rdf:type skos:Concept .
```

which is equivalent to the following RDF/XML document [RDF-XML]

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:skos="http://www.w3.org/2004/02/skos/core#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xml:base="http://example.org/ns/">

    <skos:Concept rdf:about="MyConcept"/>

</rdf:RDF>
```

which is equivalent to the following N-TRIPLES document [NTRIPLES]

```
<http://example.org/ns/MyConcept> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2004/02/skos/core#Concept> .
```

Note the use in Turtle of the ";" and "," characters to abbreviate multiple triples with the same subject or predicate. Some examples also make use of the Turtle syntax "(...)", representing an RDF Collection.

1.8. Conformance

This specification does not define a formal notion of conformance.

However, an RDF graph will be **inconsistent** with the SKOS data model if that graph and the SKOS data model (as defined formally below) taken together lead to a logical contradiction.

Where URIs are used to identify resources of type `skos:Concept`, `skos:ConceptScheme`, `skos:Collection` or `skosxl:Label`, this specification **does not** require specific behavior when dereferencing those URIs via the Web [WEBARCH]. It is, however, strongly recommended that publishers of SKOS data follow the guidelines given in [COOLURIS] and [RECIPES].

2. SKOS Namespace and Vocabulary

The SKOS namespace URI is:

- **http://www.w3.org/2004/02/skos/core#**

The SKOS vocabulary is a set of URIs, given in the left-hand column in the table below.

Table 1. SKOS Vocabulary

URI	Definition
skos:Concept	Section 3. The skos:Concept Class
skos:ConceptScheme	Section 4. Concept Schemes
skos:inScheme	Section 4. Concept Schemes

skos:hasTopConcept	Section 4. Concept Schemes
skos:topConceptOf	Section 4. Concept Schemes
skos:altLabel	Section 5. Lexical Labels
skos:hiddenLabel	Section 5. Lexical Labels
skos:prefLabel	Section 5. Lexical Labels
skos:notation	Section 6. Notations
skos:changeNote	Section 7. Documentation Properties
skos:definition	Section 7. Documentation Properties
skos:editorialNote	Section 7. Documentation Properties
skos:example	Section 7. Documentation Properties
skos:historyNote	Section 7. Documentation Properties
skos:note	Section 7. Documentation Properties
skos:scopeNote	Section 7. Documentation Properties
skos:broader	Section 8. Semantic Relations
skos:broaderTransitive	Section 8. Semantic Relations
skos:narrower	Section 8. Semantic Relations
skos:narrowerTransitive	Section 8. Semantic Relations
skos:related	Section 8. Semantic Relations
skos:semanticRelation	Section 8. Semantic Relations
skos:Collection	Section 9. Concept Collections
skos:OrderedCollection	Section 9. Concept Collections
skos:member	Section 9. Concept Collections
skos:memberList	Section 9. Concept Collections
skos:broadMatch	Section 10. Mapping Properties
skos:closeMatch	Section 10. Mapping Properties
skos:exactMatch	Section 10. Mapping Properties
skos:mappingRelation	Section 10. Mapping Properties
skos:narrowMatch	Section 10. Mapping Properties
skos:relatedMatch	Section 10. Mapping Properties

All URIs in the SKOS vocabulary are constructed by appending a local name (e.g., "prefLabel") to the SKOS namespace URI.

See also the SKOS overview in Appendix B and the quick access panel.

3. The skos:Concept Class

3.1. Preamble

The class `skos:Concept` is the class of SKOS concepts.

A SKOS concept can be viewed as an idea or notion; a unit of thought. However, what constitutes a unit of thought is subjective, and this definition is meant to be suggestive, rather than restrictive.

The notion of a SKOS concept is useful when describing the conceptual or intellectual structure of a knowledge organization system, and when referring to specific ideas or meanings established within a KOS.

Note that, because SKOS is designed to be a vehicle for representing semi-formal KOS, such as thesauri and classification schemes, a certain amount of flexibility has been built in to the formal definition of this class.

See the [SKOS-PRIMER] for more examples of identifying and describing SKOS concepts.

3.2. Vocabulary

`skos:Concept`

3.3. Class & Property Definitions

S1	`skos:Concept` is an instance of `owl:Class`.

3.4. Examples

The graph below states that `<MyConcept>` is a SKOS concept (i.e., an instance of `skos:Concept`).

Example 2 (consistent)

```
<MyConcept> rdf:type skos:Concept .
```

3.5. Notes

3.5.1. SKOS Concepts, OWL Classes and OWL Properties

Other than the assertion that `skos:Concept` is an instance of `owl:Class`, this specification does **not** make any additional statement about the formal relationship between the class of SKOS concepts and the class of OWL classes. The decision **not** to make any such statement has been made to allow applications the freedom to explore different design patterns for working with SKOS in combination with OWL.

In the example graph below, `<MyConcept>` is an instance of `skos:Concept` **and** an instance of `owl:Class`.

Example 3 (consistent)

```
<MyConcept> rdf:type skos:Concept , owl:Class .
```

This example is **consistent** with the SKOS data model.

Similarly, this specification does **not** make any statement about the formal relationship between the class of SKOS concepts and the class of OWL properties.

In the example graph below, `<MyConcept>` is an instance of `skos:Concept` **and** an instance of `owl:ObjectProperty`.

Example 4 (consistent)

```
<MyConcept> rdf:type skos:Concept , owl:ObjectProperty .
```

This example is **consistent** with the SKOS data model.

4. Concept Schemes

4.1. Preamble

A SKOS concept scheme can be viewed as an aggregation of one or more SKOS concepts. Semantic relationships (links) between those concepts may also be viewed as part of a concept scheme. This definition is, however, meant to be suggestive rather than restrictive, and there is some flexibility in the formal data model stated below.

The notion of a concept scheme is useful when dealing with data from an unknown source, and when dealing with data that describes two or more different knowledge organization systems.

See the [SKOS-PRIMER] for more examples of identifying and describing concept schemes.

4.2. Vocabulary

`skos:ConceptScheme`
`skos:inScheme`
`skos:hasTopConcept`
`skos:topConceptOf`

4.3. Class & Property Definitions

S2	`skos:ConceptScheme` is an instance of `owl:Class`.
S3	`skos:inScheme`, `skos:hasTopConcept` and `skos:topConceptOf` are each instances of `owl:ObjectProperty`.

S4	The `rdfs:range` of `skos:inScheme` is the class `skos:ConceptScheme`.
S5	The `rdfs:domain` of `skos:hasTopConcept` is the class `skos:ConceptScheme`.
S6	The `rdfs:range` of `skos:hasTopConcept` is the class `skos:Concept`.
S7	`skos:topConceptOf` is a sub-property of `skos:inScheme`.
S8	`skos:topConceptOf` is `owl:inverseOf` the property `skos:hasTopConcept`.

4.4. Integrity Conditions

S9	`skos:ConceptScheme` is disjoint with `skos:Concept`.

4.5. Examples

The graph below describes a concept scheme with two SKOS concepts, one of which is a top-level concept in that scheme.

Example 5 (consistent)

```
<MyScheme> rdf:type skos:ConceptScheme ;
  skos:hasTopConcept <MyConcept> .

<MyConcept> skos:topConceptOf <MyScheme> .

<AnotherConcept> skos:inScheme <MyScheme> .
```

4.6. Notes

4.6.1. Closed vs. Open Systems

The notion of an individual SKOS concept scheme corresponds **roughly** to the notion of an individual thesaurus, classification scheme, subject heading system or other knowledge organization system.

However, in most current information systems, a thesaurus or classification scheme is treated as a **closed system** — conceptual units defined within that system cannot take part in other systems (although they can be *mapped* to units in other systems).

Although SKOS does take a similar approach, there are **no** conditions preventing a SKOS concept from taking part in zero, one, or more than one concept scheme.

So, for example, in the graph below the SKOS concept `<MyConcept>` takes part in two different concept schemes — this is **consistent** with the SKOS data model.

Example 6 (consistent)

```
<MyScheme> rdf:type skos:ConceptScheme .

<AnotherScheme> rdf:type skos:ConceptScheme ;
  owl:differentFrom <MyScheme> .

<MyConcept> skos:inScheme <MyScheme> , <AnotherScheme> .
```

This flexibility is desirable because it allows, for example, new concept schemes to be described by linking two or more existing concept schemes together.

Also, note that there is no way to close the boundary of a concept scheme. So, while it is possible using `skos:inScheme` to say that SKOS concepts X, Y and Z take part in concept scheme A, there is no way to say that **only** X, Y and Z take part in A.

Therefore, while SKOS can be used to **describe** a concept scheme, SKOS does not provide any mechanism to completely **define** a concept scheme.

4.6.2. SKOS Concept Schemes and OWL Ontologies

This specification does **not** make any statement about the formal relationship between the class of SKOS concept schemes and the class of OWL ontologies. The decision **not** to make any such statement has been made to allow different design patterns to be explored for using SKOS in combination with OWL [OWL-GUIDE].

In the example graph below, `<MyScheme>` is both a SKOS concept scheme and an OWL ontology. This is **consistent** with the SKOS data model.

Example 7 (consistent)

```
<MyScheme> rdf:type skos:ConceptScheme , owl:Ontology .

<MyConcept> skos:inScheme <MyScheme> .
```

4.6.3. Top Concepts and Semantic Relations

The property `skos:hasTopConcept` is, by convention, used to link a concept scheme to the SKOS concept(s) which are topmost in the hierarchical relations for that scheme. However, there are no integrity conditions enforcing this convention. Therefore, the graph below, whilst not strictly adhering to the usage convention for `skos:hasTopConcept`, is nevertheless **consistent** with the SKOS data model.

Example 8 (consistent)

```
<MyScheme> skos:hasTopConcept <MyConcept> .
<MyConcept> skos:broader <AnotherConcept> .
<AnotherConcept> skos:inScheme <MyScheme> .
```

An application may reject such data but is not required to.

4.6.4. Scheme Containment and Semantic Relations

A link between two SKOS concepts **does not** entail containment within the same concept scheme. This is illustrated in the example below.

Example 9 (non-entailment)

```
<A> skos:narrower <B> .
<A> skos:inScheme <MyScheme> .
```

does not entail

```
<B> skos:inScheme <MyScheme> .
```

See also Section 8 below.

4.6.5. Domain of skos:inScheme

Note that **no domain is stated** for the property `skos:inScheme`, i.e., the domain is effectively the class of all resources (`rdfs:Resource`). The decision not to state any domain has been made to provide some flexibility, enabling extensions to SKOS to define new classes of resource but still use `skos:inScheme` to link them to a `skos:ConceptScheme`. See also example 82 below.

5. Lexical Labels

5.1. Preamble

A lexical label is a string of UNICODE characters, such as "romantic love" or "れんあい", in a given natural language, such as English or Japanese (written here in hiragana).

The Simple Knowledge Organization System provides some basic vocabulary for associating lexical labels with resources of any type. In particular, SKOS enables a distinction to be made between the preferred, alternative and "hidden" lexical labels for any given resource.

The preferred and alternative labels are useful when generating or creating human-readable representations of a knowledge organization system. These labels provide the strongest clues as to the meaning of a SKOS concept.

The hidden labels are useful when a user is interacting with a knowledge organization system via a text-based search function. The user may, for example, enter mis-spelled words when trying to find a relevant concept. If the mis-spelled query can be matched against a hidden label, the user will be able to find the relevant concept, but the hidden label won't otherwise be visible to the user (so further mistakes aren't encouraged).

Formally, a lexical label is an RDF plain literal [RDF-CONCEPTS]. An RDF plain literal is composed of a lexical form, which is a string of UNICODE characters, and an optional language tag, which is a string of characters conforming to the syntax defined by [BCP47].

See the [SKOS-PRIMER] for more examples of labeling SKOS concepts. Note especially that the examples below serve only to illustrate general features of the SKOS data model, and **do not** necessarily indicate best practice for the provision of labels with different language tags. The SKOS Reference aims to establish a data model that is applicable across a range of situations, which may then be refined and/or constrained by usage conventions for more specific situations. Application- and language-specific usage conventions with respect to labels and language tags are out of scope for the SKOS Reference.

5.2. Vocabulary

`skos:prefLabel`
`skos:altLabel`
`skos:hiddenLabel`

5.3. Class & Property Definitions

S10	`skos:prefLabel`, `skos:altLabel` and `skos:hiddenLabel` are each instances of `owl:AnnotationProperty`.
S11	`skos:prefLabel`, `skos:altLabel` and `skos:hiddenLabel` are each sub-properties of `rdfs:label`.
S12	The `rdfs:range` of each of `skos:prefLabel`, `skos:altLabel` and `skos:hiddenLabel` is the class of RDF plain literals.

5.4. Integrity Conditions

S13	`skos:prefLabel`, `skos:altLabel` and `skos:hiddenLabel` are pairwise disjoint properties.
S14	A resource has no more than one value of `skos:prefLabel` per language tag.

5.5. Examples

The following graph is **consistent**, and illustrates the provision of lexical labels in two different languages (French and English).

Example 10 (consistent)

```
<MyResource>
  skos:prefLabel "animals"@en ;
  skos:altLabel "fauna"@en ;
  skos:hiddenLabel "aminals"@en ;
  skos:prefLabel "animaux"@fr ;
  skos:altLabel "faune"@fr .
```

The following graph is **consistent** and illustrates the provision of lexical labels in four different variations (Japanese written with kanji, the hiragana script, the katakana script or with latin characters (rōmaji)).

Example 11 (consistent)

```
<AnotherResource>
  skos:prefLabel "東"@ja-Hani ;
  skos:prefLabel "ひがし"@ja-Hira ;
  skos:altLabel "あずま"@ja-Hira ;
  skos:prefLabel "ヒガシ"@ja-Kana ;
  skos:altLabel "アズマ"@ja-Kana ;
  skos:prefLabel "higashi"@ja-Latn ;
  skos:altLabel "azuma"@ja-Latn .
```

The following graph is **not consistent** with the SKOS data model, because two different preferred lexical labels have been given with the same language tag.

Example 12 (not consistent)

```
<Love> skos:prefLabel "love"@en ; skos:prefLabel "adoration"@en .
```

The following graph is **not consistent** with the SKOS data model, because there is a clash between the preferred and alternative lexical labels.

Example 13 (not consistent)

```
<Love> skos:prefLabel "love"@en ; skos:altLabel "love"@en .
```

The following graph is **not consistent** with the SKOS data model, because there is a clash between alternative and hidden lexical labels.

Example 14 (not consistent)

```
<Love> skos:altLabel "love"@en ; skos:hiddenLabel "love"@en .
```

The following graph is **not consistent** with the SKOS data model, because there is a clash between preferred and hidden lexical labels.

Example 15 (not consistent)

```
<Love> skos:prefLabel "love"@en ; skos:hiddenLabel "love"@en .
```

5.6. Notes

5.6.1. Domain of SKOS Lexical Labeling Properties

Note that **no domain is stated** for `skos:prefLabel`, `skos:altLabel` and `skos:hiddenLabel`. Thus, the effective domain of these properties is the class of all resources (`rdfs:Resource`).

Therefore, using the properties `skos:prefLabel`, `skos:altLabel` and `skos:hiddenLabel` to **label any type of resource** is **consistent** with the SKOS data model.

In the example graph below, `skos:prefLabel`, `skos:altLabel` and `skos:hiddenLabel` have been used to label a resource of type `owl:Class` — this is **consistent** with the SKOS data model.

Example 16 (consistent)

```
<MyClass> rdf:type owl:Class ;
  skos:prefLabel "animals"@en ;
  skos:altLabel "fauna"@en ;
  skos:hiddenLabel "aminals"@en ;
  skos:prefLabel "animaux"@fr ;
  skos:altLabel "faune"@fr .
```

5.6.2. Range of SKOS Lexical Labeling Properties

Note that the range of `skos:prefLabel`, `skos:altLabel` and `skos:hiddenLabel` is the class of RDF plain literals [RDF-CONCEPTS].

By convention, RDF plain literals are always used in the object position of a triple, where the predicate is one of `skos:prefLabel`, `skos:altLabel` or `skos:hiddenLabel`. If a graph **does not** follow this usage convention an application may reject such data but is not required to. See also the note below.

5.6.3. Defining Label Relations

Some applications require additional functionality relating to labels, for example allowing the description of those labels or the definition of additional relations between the labels (such as acronyms). This can be achieved through the identification of labels using URIs. The SKOS eXtension for Labels defined in Appendix A provides support for this.

5.6.4. Alternates Without Preferred

In the graph below, a resource has two alternative lexical labels, but no preferred lexical label. This is **consistent** with the SKOS data model, and there are no additional entailments which follow from the data model. However, note that many applications will require a preferred lexical label in order to generate an optimum human-readable display.

Example 17 (consistent)

```
<Love> skos:altLabel "adoration"@en , "desire"@en .
```

5.6.5. Labeling and Language Tags

[BCP47] defines tags for identifying languages. Note that "en", "en-GB", "en-US" are three different language tags, used with English, British English and US English respectively. Similarly "ja", "ja-Hani", "ja-Hira", "ja-Kana" and "ja-Latn" are five different language tags used with Japanese, Japanese written with kanji, the hiragana script, the katakana script or with latin characters (rōmaji) respectively.

The graph below is **consistent** with the SKOS data model, because "en", "en-US" and "en-GB" are different language tags.

Example 18 (consistent)

```
<Colour> skos:prefLabel "color"@en , "color"@en-US , "colour"@en-GB .
```

In the graph below, there is no clash between the lexical labeling properties, again because "en" and "en-GB" are different language tags, and therefore the graph is **consistent** with the SKOS data model.

Example 19 (consistent)

```
<Love> skos:prefLabel "love"@en ; skos:altLabel "love"@en-GB .
```

Note however that, as stated above in section 5.1, these examples serve only to illustrate general features of the SKOS data model, and **do not** necessarily indicate best practice for the provision of labels with different language tags. Application- and language-specific usage conventions with respect to labels and language tags are out of scope for the SKOS Reference.

It is suggested that applications match requests for labels in a given language to labels with related language tags that are provided by a SKOS concept scheme, e.g., by implementing the "lookup" algorithm defined by [BCP 47]. Applications that perform matching in this way do not require labels to be provided in all possible language variations (of which there could be many), and are compatible with SKOS concept schemes that provide only those labels whose lexical forms are distinct for a given language or collection of languages.

6. Notations

6.1. Preamble

A notation is a string of characters such as "T58.5" or "303.4833" used to uniquely identify a concept within the scope of a given concept scheme.

A notation is different from a lexical label in that a notation is not normally recognizable as a word or sequence of words in any natural language.

This section defines the `skos:notation` property. This property is used to assign a notation as a typed literal [RDF-CONCEPTS].

6.2. Vocabulary

```
skos:notation
```

6.3. Class & Property Definitions

S15	`skos:notation` is an instance of `owl:DatatypeProperty`.

6.4. Examples

The example below illustrates a resource `<http://example.com/ns/MyConcept>` with a notation whose lexical form is the UNICODE string "303.4833" and whose datatype is denoted by the URI `<http://example.com/ns/MyNotationDatatype>`.

Example 20 (consistent)

```
<MyConcept> skos:notation "303.4833"^^<MyNotationDatatype> .
```

6.5. Notes

6.5.1. Notations, Typed Literals and Datatypes

A typed literal is a UNICODE string combined with a datatype URI [RDF-CONCEPTS].

Typed literals are commonly used to denote values such as integers, floating point numbers and dates, and there are a number of datatypes pre-defined by the XML Schema specification [XML-SCHEMA] such as `xs:integer`, `xs:float` and `xs:date`.

For other situations, new datatypes can be defined, and these are commonly called "user-defined datatypes" [SWBP-DATATYPES].

By convention, the property `skos:notation` is only used with a typed literal in the object position of the triple, where the datatype URI denotes a user-defined datatype corresponding to a particular system of notations or classification codes.

For many situations it may be sufficient to simply coin a datatype URI for a particular notation system, and define the datatype informally via a document that describes how the notations are constructed and/or which lexical forms are allowed. Note, however, that it is also possible to define at least the lexical space of a datatype more formally via the XML Schema language, see [SWBP-DATATYPES] section 2. Users should be aware that tools may vary in their support of datatypes. However, as discussed in [OWL-REFERENCE] section 6.3, tools should at least treat lexically identical literals as equal.

6.5.2. Multiple Notations

There are no constraints on the cardinality of the `skos:notation` property. A concept may have zero, 1 or more notations.

Where a concept has more than 1 notation, these may be from the same or different notation systems. In the case where notations are from different systems, different datatypes may be used to indicate this. It is not common practice to assign more than one notation from the same notation system (i.e., with the same datatype URI).

6.5.3. Unique Notations in Concept Schemes

By convention, no two concepts in the same concept scheme are given the same notation. If they were, it would not be possible to use the notation to uniquely refer to a concept (i.e., the notation would become ambiguous).

6.5.4. Notations and Preferred Labels

There are no constraints on the combined use of `skos:notation` and `skos:prefLabel`. In the example below, the same string is given both as the lexical form of a notation and as a the lexical form of a preferred label.

Example 21 (consistent)

```
<Potassium>
  skos:prefLabel "K"@en ;
  skos:notation "K"^^<ChemicalSymbolNotation> .
```

Typed literals consist of a string of characters and a datatype URI. By convention, `skos:notation` is only used with **typed literals** in the object position of the triple.

Plain literals consist of a string of characters and a language tag. By convention, `skos:prefLabel` (and `skos:altLabel` and `skos:hiddenLabel`) are only used with **plain literals** in the object position of the triple.

There is no such thing as an RDF literal with both a language tag and a datatype URI, i.e., a typed literal does not have a language tag, and a plain literal does not have a datatype URI.

6.5.5. Domain of skos:notation

Note that **no domain is stated** for `skos:notation`. Thus, the effective domain is the class of all resources (`rdfs:Resource`). Therefore, using `skos:notation` with any type of resource is consistent with the SKOS data model.

7. Documentation Properties (Note Properties)

7.1. Preamble

Notes are used to provide information relating to SKOS concepts. There is no restriction on the nature of this information, e.g., it could be plain text, hypertext, or an image; it could be a definition, information about the scope of a concept, editorial information, or any other type of information.

There are seven properties in SKOS for associating notes with concepts, defined formally in this section. For more information on the recommended usage of each of the SKOS documentation properties, see the [SKOS-PRIMER].

These seven properties are not intended to cover every situation, but rather to be useful in some of the most common situations, and to provide a set of extension points for defining more specific types of note. For more information on recommended best practice for extending SKOS, see the [SKOS-PRIMER].

Three different usage patterns are recommended in the [SKOS-PRIMER] for the SKOS documentation properties — "documentation as an RDF literal", "documentation as a related resource description" and "documentation as a document reference". The data model defined in this section is intended to accommodate all three design patterns.

7.2. Vocabulary

`skos:note`
`skos:changeNote`
`skos:definition`
`skos:editorialNote`
`skos:example`
`skos:historyNote`
`skos:scopeNote`

7.3. Class & Property Definitions

S16	`skos:note`, `skos:changeNote`, `skos:definition`, `skos:editorialNote`, `skos:example`, `skos:historyNote` and `skos:scopeNote` are each instances of `owl:AnnotationProperty`.
S17	`skos:changeNote`, `skos:definition`, `skos:editorialNote`, `skos:example`, `skos:historyNote` and `skos:scopeNote` are each sub-properties of `skos:note`.

7.4. Examples

The graph below gives an example of the "documentation as an RDF literal" pattern.

Example 22 (consistent)

```
<MyResource> skos:note "this is a note"@en .
```

The graph below gives an example of the "documentation as a document reference" pattern.

Example 23 (consistent)

```
<MyResource> skos:note <MyNote> .
```

7.5. Notes

7.5.1. Domain of the SKOS Documentation Properties

Note that **no domain is stated** for the SKOS documentation properties. Thus, the effective domain for these properties is the class of all resources (`rdfs:Resource`). Therefore, using the SKOS documentation properties to provide information on **any type of resource** is consistent with the SKOS data model.

In the example graph below, `skos:definition` has been used to provide a plain text definition for a resource of type `owl:Class` — this is consistent with the SKOS data model.

Example 24 (consistent)

```
<Protein> rdf:type owl:Class ;
  skos:definition """A physical entity consisting of a sequence of amino-acids; a protein monomer; a single polypeptide chain. An
example is the EGFR protein."""@en .
```

7.5.2. Range of the SKOS Documentation Properties

Note that no range is stated for the SKOS documentation properties, and thus the range of these properties is effectively the class of all resources (`rdfs:Resource`). Under the RDF and OWL Full semantics, everything is a resource, including RDF plain literals.

---

8. Semantic Relations

8.1. Preamble

SKOS semantic relations are links between SKOS concepts, where the link is inherent in the meaning of the linked concepts.

The Simple Knowledge Organization System distinguishes between two basic categories of semantic relation: **hierarchical** and **associative**. A hierarchical link between two concepts indicates that one is in some way more general ("broader") than the other ("narrower"). An associative link between two concepts indicates that the two are inherently "related", but that one is **not** in any way more general than the other.

The properties `skos:broader` and `skos:narrower` are used to assert a direct hierarchical link between two SKOS concepts. A triple `<A> skos:broader <B>` asserts that `<B>`, the object of the triple, is a broader concept than `<A>`, the subject of the triple. Similarly, a triple `<C> skos:narrower <D>` asserts that `<D>`, the object of the triple, is a narrower concept than `<C>`, the subject of the triple.

By convention, `skos:broader` and `skos:narrower` are **only** used to assert a **direct** (i.e., immediate) hierarchical link between two SKOS concepts. This provides applications with a convenient and reliable way to access the direct broader and narrower links for any given concept. Note that, to support this usage convention, the properties `skos:broader` and `skos:narrower` are **not** declared as transitive properties.

Some applications need to make use of **both direct and indirect** hierarchical links between concepts, for instance to improve search recall through query expansion. For this purpose, the properties `skos:broaderTransitive` and `skos:narrowerTransitive` are provided. A triple `<A> skos:broaderTransitive <B>` represents a direct or indirect hierarchical link, where `<B>` is a broader "ancestor" of `<A>`. Similarly a triple `<C> skos:narrowerTransitive <D>` represents a direct or indirect hierarchical link, where `<D>` is a narrower "descendant" of `<C>`.

By convention, the properties `skos:broaderTransitive` and `skos:narrowerTransitive` are **not** used to make assertions. Rather, these properties are used to infer the transitive closure of the hierarchical links, which can then be used to access direct or indirect hierarchical links between concepts.

The property `skos:related` is used to assert an associative link between two SKOS concepts.

For more examples of stating hierarchical and associative links, see the [SKOS-PRIMER].

8.2. Vocabulary

skos:semanticRelation
skos:broader
skos:narrower
skos:related
skos:broaderTransitive
skos:narrowerTransitive

8.3. Class & Property Definitions

S18	`skos:semanticRelation`, `skos:broader`, `skos:narrower`, `skos:related`, `skos:broaderTransitive` and `skos:narrowerTransitive` are each instances of `owl:ObjectProperty`.
S19	The `rdfs:domain` of `skos:semanticRelation` is the class `skos:Concept`.
S20	The `rdfs:range` of `skos:semanticRelation` is the class `skos:Concept`.
S21	`skos:broaderTransitive`, `skos:narrowerTransitive` and `skos:related` are each sub-properties of `skos:semanticRelation`.
S22	`skos:broader` is a sub-property of `skos:broaderTransitive`, and `skos:narrower` is a sub-property of `skos:narrowerTransitive`.
S23	`skos:related` is an instance of `owl:SymmetricProperty`.
S24	`skos:broaderTransitive` and `skos:narrowerTransitive` are each instances of `owl:TransitiveProperty`.
S25	`skos:narrower` is `owl:inverseOf` the property `skos:broader`.
S26	`skos:narrowerTransitive` is `owl:inverseOf` the property `skos:broaderTransitive`.

8.4. Integrity Conditions

S27	`skos:related` is disjoint with the property `skos:broaderTransitive`.

Note that because `skos:related` is a symmetric property, and `skos:broaderTransitive` and `skos:narrowerTransitive` are inverses, `skos:related` is therefore also disjoint with `skos:narrowerTransitive`.

8.5. Examples

The graph below asserts a direct hierarchical link between `<A>` and `<B>` (where `<B>` is broader than `<A>`), and an associative link between `<A>` and `<C>`, and is **consistent** with the SKOS data model.

Example 25 (consistent)

```
<A> skos:broader <B> ; skos:related <C> .
```

The graph below is **not consistent** with the SKOS data model, because there is a clash between associative links and hierarchical links.

Example 26 (not consistent)

```
<A> skos:broader <B> ; skos:related <B> .
```

The graph below is **not consistent** with the SKOS data model, again because there is a clash between associative links and hierarchical links.

Example 27 (not consistent)

```
<A> skos:broader <B> ; skos:related <C> .
<B> skos:broader <C> .
```

In the example above, the clash is not immediately obvious. The clash becomes apparent when inferences are drawn, based on the class and property definitions above, giving the following graph.

Example 28 (not consistent)

```
<A> skos:broaderTransitive <C> ; skos:related <C> .
```

The graph below is **not consistent** with the SKOS data model, again because there is a clash between associative links and hierarchical links, which can be inferred from the class and property definitions given above.

Example 29 (not consistent)

```
<A> skos:narrower <B> ; skos:related <C> .
<B> skos:narrower <C> .
```

8.6. Notes

8.6.1. Sub-Property Relationships

The diagram below illustrates informally the sub-property relationships between the SKOS semantic relation properties.

```
skos:semanticRelation
 |
 +— skos:related
 |
 +— skos:broaderTransitive
 |    |
 |    +— skos:broader
 |
 +— skos:narrowerTransitive
      |
      +— skos:narrower
```

8.6.2. Domain and Range of SKOS Semantic Relation Properties

Note that the domain and range of `skos:semanticRelation` is the class `skos:Concept`. Because `skos:broader`, `skos:narrower` and `skos:related` are each sub-properties of `skos:semanticRelation`, the graph in example 26 above entails that `<A>`, `<B>` and `<C>` are each instances of `skos:Concept`.

8.6.3. Symmetry of skos:related

`skos:related` is a symmetric property. The example below illustrates an entailment which follows from this condition.

Example 30 (entailment)

```
<A> skos:related <B> .
```

entails

```
<B> skos:related <A> .
```

Note that, although `skos:related` is a symmetric property, this condition does **not** place any restrictions on sub-properties of `skos:related` (i.e., sub-properties of `skos:related` could be symmetric, not symmetric or antisymmetric, and still be consistent with the SKOS data model).

To illustrate this point, in the example below, two new properties which are **not** symmetric are declared as sub-properties of `skos:related`. The example, which is **consistent** with the SKOS data model, also shows some of the entailments which follow.

Example 31 (entailment)

```
<cause> rdf:type owl:ObjectProperty ;
  rdfs:subPropertyOf skos:related .

<effect> rdf:type owl:ObjectProperty ;
 rdfs:subPropertyOf skos:related ;
  owl:inverseOf <cause> .

<A> <cause> <B> .
```

entails

```
<A> skos:related <B> .

<B> <effect> <A> ; skos:related <A> .
```

See also the [SKOS-PRIMER] for best practice recommendations on extending SKOS.

8.6.4. skos:related and Transitivity

Note that `skos:related` is **not** a transitive property. Therefore, the SKOS data model does **not** support an entailment as illustrated in the example below.

Example 32 (non-entailment)

```
<A> skos:related <B> .
<B> skos:related <C> .
```

does not entail

```
<A> skos:related <C> .
```

8.6.5. skos:related and Reflexivity

Note that this specification does not state that `skos:related` is a reflexive property, **neither** does it state that `skos:related` is an irreflexive property.

Because `skos:related` is **not** defined as an irreflexive property, the graph below is **consistent** with the SKOS data model.

Example 33 (consistent)

```
<A> skos:related <A> .
```

However, for many applications that use knowledge organization systems, statements of the form X `skos:related` X are a potential problem. Where this is the case, an application may wish to search for such statements prior to processing SKOS data, although how an application should handle such statements is not defined in this specification and may vary between applications.

8.6.6. skos:broader and Transitivity

Note that `skos:broader` is **not** a transitive property. Similarly, `skos:narrower` is **not** a transitive property. Therefore, the SKOS data model does **not** support an entailment as illustrated in the example below.

Example 34 (non-entailment)

```
<A> skos:broader <B> .
<B> skos:broader <C> .
```

does not entail

```
<A> skos:broader <C> .
```

However, `skos:broader` is a sub-property of `skos:broaderTransitive`, which **is** a transitive property. Similarly, `skos:narrower` is a sub-property of `skos:narrowerTransitive`, which **is** a transitive property. Therefore the SKOS data model **does** support the entailments illustrated below.

Example 35 (entailment)

```
<A> skos:broader <B> .
<B> skos:broader <C> .
```

entails

```
<A> skos:broaderTransitive <B> .
<B> skos:broaderTransitive <C> .
<A> skos:broaderTransitive <C> .
```

Note especially that, by convention, `skos:broader` and `skos:narrower` are **only** used to assert immediate (i.e., direct) hierarchical links between two SKOS concepts. By convention, `skos:broaderTransitive` and `skos:narrowerTransitive` are **not** used to make assertions, but are instead used only to draw inferences.

This pattern allows the information about direct (i.e., immediate) hierarchical links to be preserved, which is necessary for many tasks (e.g., building various types of visual representation of a knowledge organization system), whilst also providing a mechanism for conveniently querying the transitive closure of those hierarchical links (which will include both direct and indirect links), which is useful in other situations (e.g., query expansion algorithms).

Note also that a sub-property of a transitive property is **not** necessarily transitive.

See also the note on alternative paths below.

8.6.7. skos:broader and Reflexivity

Note that this specification makes no statements regarding the reflexive characteristics of the `skos:broader` relationship. It does not state that `skos:broader` is a reflexive property, **neither** does it state that `skos:broader` is an irreflexive property. Thus for any graph and resource `<A>`, the triple:

Example 36 (consistent)

```
<A> skos:broader <A> .
```

may or may not be present. This conservative position allows SKOS to be used to model both KOS where the interpretation of `skos:broader` is reflexive (e.g., a direct translation of an inferred OWL sub-class hierarchy), or KOS where `skos:broader` could be considered irreflexive (as would be appropriate for most thesauri or classification schemes).

Similarly, there are no assertions made as to the reflexivity or irreflexivity of `skos:narrower`.

However, for many applications that use knowledge organization systems, statements of the form X `skos:broader` X or Y `skos:narrower` Y represent a potential problem. Where this is the case, an application may wish to search for such statements prior to processing SKOS data, although how an application should handle such statements is not defined in this specification and may vary between applications.

8.6.8. Cycles in the Hierarchical Relation (skos:broaderTransitive and Reflexivity)

In the graph below, a cycle has been stated in the hierarchical relation. Note that this graph is **consistent** with the SKOS data model, i.e., there is **no** condition requiring that `skos:broaderTransitive` be irreflexive.

Example 37 (consistent)

```
<A> skos:broader <B> .
<B> skos:broader <A> .
```

However, for many applications where knowledge organization systems are used, a cycle in the hierarchical relation represents a potential problem. For these applications, computing the transitive closure of `skos:broaderTransitive` then looking for statements of the form X `skos:broaderTransitive` X is a convenient strategy for finding cycles in the hierarchical relation. How an application should handle such statements is not defined in this specification and may vary between applications.

8.6.9. Alternate Paths in the Hierarchical Relation

In the graph below, there are two alternative paths from A to C in the hierarchical relation.

Example 38 (consistent)

```
<A> skos:broader <B> , <C> .
<B> skos:broader <C> .
```

In the graph below, there are two alternative paths from A to D in the hierarchical relation.

Example 39 (consistent)

```
<A> skos:broader <B> , <C> .
<B> skos:broader <D> .
<C> skos:broader <D> .
```

This is a pattern which arises naturally in poly-hierarchical knowledge organization systems.

Both of these graphs are **consistent** with the SKOS data model, i.e., there is **no** condition requiring that there be only one path between any two nodes in the hierarchical relation.

8.6.10. Disjointness of skos:related and skos:broaderTransitive

This specification treats the hierarchical and associative relations as fundamentally distinct in nature. Therefore a clash between hierarchical and associative links is **not** consistent with the SKOS data model. The examples above illustrate some situations in which a clash is seen to arise.

This position follows the usual definitions given to hierarchical and associative relations in thesaurus standards [ISO2788] [BS8723-2], and supports common practice in many existing knowledge organization systems.

Note that this specification takes the stronger position that, not only are the immediate (i.e., direct) hierarchical and associative links disjoint, but associative links are also disjoint with *indirect* hierarchical links. This is captured formally in the integrity condition asserting that `skos:related` and `skos:broaderTransitive` are disjoint properties.

---

9. Concept Collections

9.1. Preamble

SKOS concept collections are labeled and/or ordered groups of SKOS concepts.

Collections are useful where a group of concepts shares something in common, and it is convenient to group them under a common label, or where some concepts can be placed in a meaningful order.

9.2. Vocabulary

`skos:Collection`
`skos:OrderedCollection`
`skos:member`
`skos:memberList`

9.3. Class & Property Definitions

S28	`skos:Collection` and `skos:OrderedCollection` are each instances of `owl:Class`.
S29	`skos:OrderedCollection` is a sub-class of `skos:Collection`.
S30	`skos:member` and `skos:memberList` are each instances of `owl:ObjectProperty`.
S31	The `rdfs:domain` of `skos:member` is the class `skos:Collection`.

S32	The `rdfs:range` of `skos:member` is the union of classes `skos:Concept` and `skos:Collection`.
S33	The `rdfs:domain` of `skos:memberList` is the class `skos:OrderedCollection`.
S34	The `rdfs:range` of `skos:memberList` is the class `rdf:List`.
S35	`skos:memberList` is an instance of `owl:FunctionalProperty`.
S36	For any resource, every item in the list given as the value of the `skos:memberList` property is also a value of the `skos:member` property.

9.4. Integrity Conditions

S37	`skos:Collection` is disjoint with each of `skos:Concept` and `skos:ConceptScheme`.

9.5. Examples

The graph below illustrates a SKOS collection with 3 members.

Example 40 (consistent)

```
<MyCollection> rdf:type skos:Collection ;
  skos:member <X> , <Y> , <Z> .
```

The graph below illustrates an ordered SKOS collection with 3 members. Note the use of the Turtle syntax `(...)`, representing an RDF Collection (list).

Example 41 (consistent)

```
<MyOrderedCollection> rdf:type skos:OrderedCollection ;
  skos:memberList ( <X> <Y> <Z> ) .
```

9.6. Notes

9.6.1. Inferring Collections from Ordered Collections

Statement S36 states the logical relationship between the `skos:memberList` and `skos:member` properties. This relationship means that a collection can be inferred from an ordered collection. This is illustrated in the example below.

Example 42 (entailment)

```
<MyOrderedCollection> rdf:type skos:OrderedCollection ;
  skos:memberList ( <X> <Y> <Z> ) .
```

entails

```
<MyOrderedCollection> rdf:type skos:Collection ;
  skos:member <X> , <Y> , <Z> .
```

Note that SKOS does not provide any way to explicitly state that a collection is **not** ordered.

9.6.2. skos:memberList Integrity

Note that `skos:memberList` is a functional property, i.e., it does not have more than one value. This is intended to capture within the SKOS data model that it doesn't make sense for an ordered collection to have more than one member list. Unfortunately, there is no way to use this condition as an integrity condition without explicitly stating that two lists are different objects. In other words, although the graph below is **consistent** with the SKOS data model, it entails nonsense (a list with two first elements and a forked tail).

Example 43 (entailment)

```
<OrderedCollectionResource>
  skos:memberList ( <A> <B> ) , ( <X> <Y> ) .
```

entails

```
<OrderedCollectionResource>
  skos:memberList [ rdf:first <A> , <X> ; rdf:rest [ rdf:first <B> ; rdf:rest rdf:nil ] , [ rdf:first <Y> ; rdf:rest rdf:nil ] ] .
```

However, as stated in [RDF-SEMANTICS] section 3.3.3, semantic extensions to RDF may place extra syntactic well-formedness restrictions on the use of the RDF collection vocabulary (`rdf:first`, `rdf:rest`, `rdf:nil`) in order to rule out such graphs.

9.6.3. Nested Collections

In the example below, a collection is nested within another collection.

Example 44 (consistent)

```
<MyCollection> rdf:type skos:Collection ;
  skos:member <A> , <B> , <MyNestedCollection> .
```

```
<MyNestedCollection> rdf:type skos:Collection ;
  skos:member <X> , <Y> , <Z> .
```

This example is **consistent** with the SKOS data model, because the range of `skos:member` is the union of `skos:Concept` and `skos:Collection`.

9.6.4. SKOS Concepts, Concept Collections and Semantic Relations

In the SKOS data model, `skos:Concept` and `skos:Collection` are disjoint classes. The domain and range of the SKOS semantic relation properties is `skos:Concept`. Therefore, if any of the SKOS semantic relation properties (e.g., `skos:narrower`) are used to link to or from a collection, the graph will **not** be consistent with the SKOS data model.

This is illustrated in the example below, which is **not** consistent with the SKOS data model.

Example 45 (not consistent)

```
<A> skos:narrower <B> .
<B> rdf:type skos:Collection .
```

Similarly, the graph below is **not** consistent with the SKOS data model.

Example 46 (not consistent)

```
<A> skos:broader <B> .
<B> rdf:type skos:Collection .
```

Similarly, the graph below is **not** consistent with the SKOS data model.

Example 47 (not consistent)

```
<A> skos:related <B> .
<B> rdf:type skos:Collection .
```

However, the graph below is consistent with the SKOS data model.

Example 48 (consistent)

```
<A> skos:narrower <B> , <C> , <D> .

<ResourceCollection> rdfs:label "Resource Collection"@en ;
  skos:member <B> , <C> , <D> .
```

This means that, for thesauri and other knowledge organization systems where node labels are used within the systematic display for that thesaurus, the appropriate SKOS representation requires careful consideration. Furthermore, where node labels are used in the systematic display, it may not always be possible to fully reconstruct the systematic display from a SKOS representation alone. Fully representing all of the information represented in a systematic display of a thesaurus or other knowledge organization system, including details of layout and presentation, is beyond the scope of SKOS.

10. Mapping Properties

10.1. Preamble

The SKOS mapping properties are `skos:closeMatch`, `skos:exactMatch`, `skos:broadMatch`, `skos:narrowMatch` and `skos:relatedMatch`. These properties are used to state mapping (alignment) links between SKOS concepts in different concept schemes, where the links are inherent in the meaning of the linked concepts.

The properties `skos:broadMatch` and `skos:narrowMatch` are used to state a hierarchical mapping link between two concepts.

The property `skos:relatedMatch` is used to state an associative mapping link between two concepts.

The property `skos:closeMatch` is used to link two concepts that are sufficiently similar that they can be used interchangeably in **some** information retrieval applications. In order to avoid the possibility of "compound errors" when combining mappings across more than two concept schemes, `skos:closeMatch` is **not** declared to be a transitive property.

The property `skos:exactMatch` is used to link two concepts, indicating a high degree of confidence that the concepts can be used interchangeably across a wide range of information retrieval applications. `skos:exactMatch` is a transitive property, and is a sub-property of `skos:closeMatch`.

10.2. Vocabulary

skos:mappingRelation
skos:closeMatch
skos:exactMatch
skos:broadMatch
skos:narrowMatch
skos:relatedMatch

10.3. Class & Property Definitions

S38	`skos:mappingRelation`, `skos:closeMatch`, `skos:exactMatch`, `skos:broadMatch`, `skos:narrowMatch` and `skos:relatedMatch` are each instances of `owl:ObjectProperty`.
S39	`skos:mappingRelation` is a sub-property of `skos:semanticRelation`.
S40	`skos:closeMatch`, `skos:broadMatch`, `skos:narrowMatch` and `skos:relatedMatch` are each sub-properties of `skos:mappingRelation`.
S41	`skos:broadMatch` is a sub-property of `skos:broader`, `skos:narrowMatch` is a sub-property of `skos:narrower`, and `skos:relatedMatch` is a sub-property of `skos:related`.
S42	`skos:exactMatch` is a sub-property of `skos:closeMatch`.
S43	`skos:narrowMatch` is `owl:inverseOf` the property `skos:broadMatch`.
S44	`skos:relatedMatch`, `skos:closeMatch` and `skos:exactMatch` are each instances of `owl:SymmetricProperty`.
S45	`skos:exactMatch` is an instance of `owl:TransitiveProperty`.

10.4. Integrity Conditions

S46	`skos:exactMatch` is disjoint with each of the properties `skos:broadMatch` and `skos:relatedMatch`.

Note that because `skos:exactMatch` is a symmetric property, and `skos:broadMatch` and `skos:narrowMatch` are inverses, `skos:exactMatch` is therefore also disjoint with `skos:narrowMatch`.

10.5. Examples

The graph below asserts an exact equivalence mapping link between `<A>` and `<B>`.

Example 49 (consistent)

```
<A> skos:exactMatch <B> .
```

The graph below asserts a close equivalence mapping link between `<A>` and `<B>`.

Example 50 (consistent)

```
<A> skos:closeMatch <B> .
```

The graph below asserts a hierarchical mapping link between `<A>` and `<B>` (where `<B>` is broader than `<A>`), and an associative mapping link between `<A>` and `<C>`.

Example 51 (consistent)

```
<A> skos:broadMatch <B> ; skos:relatedMatch <C> .
```

The graph below is **not consistent** with the SKOS data model, because there is a clash between exact and hierarchical mapping links.

Example 52 (not consistent)

```
<A> skos:exactMatch <B> ; skos:broadMatch <B> .
```

The graph below is **not consistent** with the SKOS data model, because there is a clash between exact and associative mapping links.

Example 53 (not consistent)

```
<A> skos:exactMatch <B> ; skos:relatedMatch <B> .
```

10.6. Notes

10.6.1. Mapping Properties, Semantic Relation Properties and Concept Schemes

By convention, the SKOS mapping properties are only used to link concepts in **different** concept schemes. However, note that using the SKOS semantic relation properties (`skos:broader`, `skos:narrower`, `skos:related`) to link concepts in **different** concept schemes is also **consistent** with the SKOS data model (see Section 8).

The mapping properties `skos:broadMatch`, `skos:narrowMatch` and `skos:relatedMatch` are provided as a convenience, for situations where the provenance of data is known, and it is useful to be able to tell at a glance the difference between internal links within a concept scheme and mapping links between concept schemes.

However, using the SKOS mapping properties is **no substitute** for the careful management of RDF graphs or the use of provenance mechanisms.

The rationale behind this design is that it is hard to draw an absolute distinction between internal links within a concept scheme and mapping links between concept schemes. This is especially true in an open environment where different people might re-organize concepts into concept schemes in different ways. What one person views as two concept schemes with mapping links between, another might view as one single concept scheme with internal links only. This specification allows both points of view to co-exist, which (it is hoped) will promote flexibility and innovation in the re-use of SKOS data in the Web.

There is therefore an intimate connection between the SKOS semantic relation properties and the SKOS mapping properties. The property `skos:broadMatch` is a sub-property of `skos:broader`, `skos:narrowMatch` is a sub-property of `skos:narrower`, and `skos:relatedMatch` is a sub-property of `skos:related`. The full set of sub-property relationships is illustrated below.

```
skos:semanticRelation
 |
 +- skos:related
 |   |
 |   +- skos:relatedMatch
 |
 +- skos:broaderTransitive
 |   |
 |   +- skos:broader
 |       |
 |       +- skos:broadMatch
 |
 +- skos:narrowerTransitive
 |   |
 |   +- skos:narrower
 |       |
 |       +- skos:narrowMatch
 |
 +- skos:mappingRelation
     |
     +- skos:closeMatch
     |   |
     |   +- skos:exactMatch
     |
     +- skos:relatedMatch
     |
     +- skos:broadMatch
     |
     +- skos:narrowMatch
```

Examples below illustrate some entailments which follow from this sub-property tree, and from the domain and range of `skos:semanticRelation`.

Example 54 (entailment)

```
<A> skos:broadMatch <B> .
```

entails

```
<A> skos:mappingRelation <B> .
<A> skos:broader <B> .
<A> skos:broaderTransitive <B> .
<A> skos:semanticRelation <B> .
<A> rdf:type skos:Concept .
<B> rdf:type skos:Concept .
```

Example 55 (entailment)

```
<A> skos:narrowMatch <B> .
```

entails

```
<A> skos:mappingRelation <B> .
<A> skos:narrower <B> .
<A> skos:narrowerTransitive <B> .
<A> skos:semanticRelation <B> .
<A> rdf:type skos:Concept .
<B> rdf:type skos:Concept .
```

Example 56 (entailment)

```
<A> skos:relatedMatch <B> .
```

entails

```
<A> skos:mappingRelation <B> .
<A> skos:related <B> .
<A> skos:semanticRelation <B> .
<A> rdf:type skos:Concept .
<B> rdf:type skos:Concept .
```

Example 57 (entailment)

```
<A> skos:exactMatch <B> .
```

entails

```
<A> skos:closeMatch <B> .
<A> skos:mappingRelation <B> .
<A> skos:semanticRelation <B> .
<A> rdf:type skos:Concept .
<B> rdf:type skos:Concept .
```

Note also that, because different people might re-organize concepts into concept schemes in different ways, a graph might assert **mapping** links between concepts in the **same** concept scheme, and there are **no** formal integrity conditions in the SKOS data model that would make such a graph inconsistent, e.g., the graph below is **consistent** with the SKOS data model. However, in practice it is expected that such a graph would only ever arise from the merge of two or more graphs from different sources.

Example 58 (consistent)

```
<A> skos:broadMatch <B> ; skos:relatedMatch <C> .

<A> skos:inScheme <MyScheme> .
<B> skos:inScheme <MyScheme> .
<C> skos:inScheme <MyScheme> .
```

10.6.2. Clashes Between Hierarchical and Associative Links

Examples below illustrate "clashes" between hierarchical and associative mapping links, which are **not consistent** with the SKOS data model (because of the sub-property relationships illustrated above, and because of the data model for SKOS semantic relation properties defined in Section 8).

Example 59 (not consistent)

```
<A> skos:broadMatch <B> ; skos:relatedMatch <B> .
```

Example 60 (not consistent)

```
<A> skos:narrowMatch <B> ; skos:relatedMatch <B> .
```

Example 61 (not consistent)

```
<A> skos:broadMatch <B> .
<B> skos:broadMatch <C> .
<A> skos:relatedMatch <C> .
```

10.6.3. Mapping Properties and Transitivity

The only SKOS mapping property which is declared as transitive is `skos:exactMatch`. An example entailment is illustrated below:

Example 62 (entailment)

```
<A> skos:exactMatch <B> .
<B> skos:exactMatch <C> .
```

entails

```
<A> skos:exactMatch <C> .
```

All other SKOS mapping properties are not transitive. Therefore, entailments as illustrated in examples below are **not** supported by the SKOS data model.

Example 63 (non-entailment)

```
<A> skos:broadMatch <B> .
<B> skos:broadMatch <C> .
```

does not entail

```
<A> skos:broadMatch <C> .
```

Example 64 (non-entailment)

```
<A> skos:relatedMatch <B> .
<B> skos:relatedMatch <C> .
```

does not entail

```
<A> skos:relatedMatch <C> .
```

Example 65 (non-entailment)

```
<A> skos:closeMatch <B> .
<B> skos:closeMatch <C> .
```

does not entail

```
<A> skos:closeMatch <C> .
```

10.6.4. Mapping Properties and Reflexivity

None of the SKOS mapping properties are reflexive, **neither** are they irreflexive.

Because `skos:exactMatch`, `skos:broadMatch` and `skos:relatedMatch` are **not irreflexive**, the graph below is **consistent** with the SKOS data model.

Example 66 (consistent)

```
<A> skos:exactMatch <A> .
<B> skos:broadMatch <B> .
<C> skos:relatedMatch <C> .
```

However, see also Section 8 on the reflexivity of SKOS semantic relation properties.

10.6.5. Cycles and Alternate Paths Involving skos:broadMatch

There are no formal integrity conditions preventing either cycles or alternative paths in a graph of hierarchical mapping links.

In the graph below there are two cycles involving `skos:broadMatch`. This graph is **consistent** with the SKOS data model.

Example 67 (consistent)

```
<A> skos:broadMatch <B> .
<B> skos:broadMatch <A> .

<X> skos:broadMatch <Y> .
<Y> skos:broadMatch <Z> .
<Z> skos:broadMatch <X> .
```

In the graph below there are two alternative paths involving `skos:broadMatch`. This graph is **consistent** with the SKOS data model.

Example 68 (consistent)

```
<A> skos:broadMatch <B> .
<B> skos:broadMatch <C> .
<A> skos:broadMatch <C> .
```

See however Section 8 on cycles and alternative paths involving `skos:broader`.

10.6.6. Cycles Involving skos:exactMatch and skos:closeMatch

Example 69 (entailment)

```
<A> skos:exactMatch <B>
```

entails

```
<A> skos:exactMatch <A> .
<A> skos:closeMatch <A> .
```

Due to the entailment above (which arises through a combination of S42, S44 and S45), applications must be able to cope with cycles in `skos:exactMatch` and `skos:closeMatch`.

10.6.7. Sub-Property Chains Involving skos:exactMatch

There are no sub-property chain axioms in the SKOS data model involving the `skos:exactMatch` or `skos:closeMatch` properties. Hence the entailments illustrated below are **not** supported.

Example 70 (non-entailment)

```
<A> skos:exactMatch <B> .
<B> skos:broadMatch <C> .
```

does not entail

```
<A> skos:broadMatch <C> .
```

Example 71 (non-entailment)

```
<A> skos:exactMatch <B> .
<B> skos:relatedMatch <C> .
```

does not entail

```
<A> skos:relatedMatch <C> .
```

Example 72 (non-entailment)

```
<A> skos:closeMatch <B> .
<B> skos:broadMatch <C> .
```

does not entail

```
<A> skos:broadMatch <C> .
```

Example 73 (non-entailment)

```
<A> skos:closeMatch <B> .
<B> skos:relatedMatch <C> .
```

does not entail

```
<A> skos:relatedMatch <C> .
```

10.6.8. skos:closeMatch, skos:exactMatch, owl:sameAs, owl:equivalentClass, owl:equivalentProperty

OWL provides three properties which might, at first glance, appear similar to `skos:closeMatch` or `skos:exactMatch`. `owl:sameAs` is used to link two individuals in an ontology, and indicates that they are the same individual (i.e., the same resource). `owl:equivalentClass` is used to link two classes in an ontology, and indicates that those classes have the same class extension. `owl:equivalentProperty` is used to link two properties in an ontology and indicates that both properties have the same property extension.

`skos:closeMatch` and `skos:exactMatch` are used to link SKOS concepts in different schemes. A `skos:closeMatch` link indicates that two concepts are sufficiently similar that they can be used interchangeably in **some** information retrieval applications. A `skos:exactMatch` link indicates a high degree of confidence that two concepts can be used interchangeably across a wide range of information retrieval applications.

`owl:sameAs`, `owl:equivalentClass` or `owl:equivalentProperty` would typically be inappropriate for linking SKOS concepts in different concept schemes, because the formal consequences that follow could be undesirable.

The example below illustrates some undesirable entailments that would follow from using `owl:sameAs` in this way.

Example 74 (entailment)

```
<A> rdf:type skos:Concept ;
  skos:prefLabel "love"@en ;
  skos:inScheme <MyScheme> .

<B> rdf:type skos:Concept ;
  skos:prefLabel "adoration"@en ;
  skos:inScheme <AnotherScheme> .

<A> owl:sameAs <B> .
```

entails

```
<A>
  skos:prefLabel "love"@en ;
  skos:prefLabel "adoration"@en ;
  skos:inScheme <MyScheme> ;
  skos:inScheme <AnotherScheme> .

<B>
  skos:prefLabel "love"@en ;
  skos:prefLabel "adoration"@en ;
  skos:inScheme <MyScheme> ;
  skos:inScheme <AnotherScheme> .
```

In this example, using `owl:sameAs` to link two SKOS concepts in different concept schemes does actually lead to an inconsistency with the SKOS data model, because both `<A>` and `<B>` now have two preferred lexical labels in the same language. This will not always be the case, however.

11. References

[AGROVOC]
AGROVOC Thesaurus, Food and Agriculture Organization of the United Nations (FAO). Available at http://www.fao.org/agrovoc

[BCP47]
Tags for Identifying Languages, A. Phillips and M. Davis, Editors, September 2006. Available at http://www.rfc-editor.org/rfc/bcp/bcp47.txt

[BS8723-2]
BS8723 Structured Vocabularies for Information Retrieval Part 2: Thesauri, British Standards Institution (BSI), 2005.

[BS8723-3]
BS8723 Structured Vocabularies for Information Retrieval Part 3: Vocabularies Other Than Thesauri, British Standards Institution (BSI), 2005.

[COOLURIS]
Cool URIs for the Semantic Web, Leo Sauermann and Richard Cyganiak, Editors, W3C Interest Group Note, 31 March 2008, http://www.w3.org/TR/2008/NOTE-cooluris-20080331/. Latest version available at http://www.w3.org/TR/cooluris/

[ISO2788]
ISO 2788:1986 Documentation -- Guidelines for the establishment and development of monolingual thesauri, International Organization for Standardization (ISO), 1986.

[LCSH]
Library of Congress Subject Headings, The Library of Congress Cataloging Distribution Service. Available at http://www.loc.gov/cds/lcsh.html and at http://id.loc.gov/

[NTRIPLES]
RDF Test Cases, Jan Grant and Dave Beckett, Editors, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-testcases-20040210/. Latest version available at http://www.w3.org/TR/rdf-testcases/

[OWL-GUIDE]
OWL Web Ontology Language Guide, Michael K. Smith, Chris Welty and Deborah L. McGuinness, Editors, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-owl-guide-20040210/. Latest version available at http://www.w3.org/TR/owl-guide/

[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-SEMANTICS]
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/

[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-PRIMER]
RDF Primer, Frank Manola and Eric Miller, Editors, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-primer-20040210/. Latest version available at http://www.w3.org/TR/rdf-primer/

[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-XML]
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/

[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/

[RECIPES]
: *Best Practice Recipes for Publishing RDF Vocabularies*, Diego Berrueta and Jon Phipps, Editors, W3C Working Draft, 23 January 2008, http://www.w3.org/TR/2008/WD-swbp-vocab-pub-20080123/. Latest version available at http://www.w3.org/TR/swbp-vocab-pub/

[SKOS-HTML]
: *SKOS Namespace Document - HTML Variant*. Latest version available at http://www.w3.org/TR/skos-reference/skos.html

[SKOS-PRIMER]
: *SKOS Simple Knowledge Organization System Primer*, Antoine Isaac and Ed Summers, Editors, W3C Working Group Note, 18 August 2009, http://www.w3.org/TR/2009/NOTE-skos-primer-20090818. Latest version available at http://www.w3.org/TR/skos-primer

[SKOS-RDF]
: *SKOS Namespace Document - RDF/XML Variant*. Latest version available at http://www.w3.org/TR/skos-reference/skos.rdf

[SKOS-RDF-OWL1-DL]
: *SKOS RDF Schema - OWL 1 DL Sub-set*. Latest version available at http://www.w3.org/TR/skos-reference/skos-owl1-dl.rdf

[SKOS-UCR]
: *SKOS Use Cases and Requirements*, Antoine Isaac, Jon Phipps and Daniel Rubin, Editors, W3C Working Group Note, 18 August 2009, http://www.w3.org/TR/2009/NOTE-skos-ucr-20090818. Latest version available at http://www.w3.org/TR/skos-ucr

[SKOS-XL-HTML]
: *SKOS-XL Namespace Document - HTML Variant*. Latest version available at http://www.w3.org/TR/skos-reference/skos-xl.html

[SKOS-XL-RDF]
: *SKOS-XL Namespace Document - RDF/XML Variant*. Latest version available at http://www.w3.org/TR/skos-reference/skos-xl.rdf

[SPARQL]
: *SPARQL Query Language for RDF*, Eric Prud'hommeaux and Andy Seaborne, Editors, W3C Recommendation, 15 January 2008, http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/. Latest version available at http://www.w3.org/TR/rdf-sparql-query/

[SW]
: *W3C Semantic Web Activity*. Available at http://www.w3.org/2001/sw/

[SWBP-DATATYPES]
: *XML Schema Datatypes in RDF and OWL*, Jeremy J. Carroll and Jeff Z. Pan, Editors, W3C Working Group Note, 14 March 2006, http://www.w3.org/TR/2006/NOTE-swbp-xsch-datatypes-20060314/. Latest version available at http://www.w3.org/TR/swbp-xsch-datatypes/

[TURTLE]
: *Turtle - Terse RDF Triple Language*, David Beckett and Tim Berners-Lee, W3C Team Submission, 14 January 2008, http://www.w3.org/TeamSubmission/2008/SUBM-turtle-20080114/. Latest version available at http://www.w3.org/TeamSubmission/turtle/

[WEBARCH]
: *Architecture of the World Wide Web, Volume One*, Ian Jacobs and Norman Walsh, Editors, W3C Recommendation, 15 December 2004, http://www.w3.org/TR/2004/REC-webarch-20041215/. Latest version available at http://www.w3.org/TR/webarch/

[XML-SCHEMA]
: *XML Schema Part 2: Datatypes Second Edition*, Paul V. Biron and Ashok Malhotra, Editors, W3C Recommendation, 28 October 2004, http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/. Latest version available at http://www.w3.org/TR/xmlschema-2/

12. Acknowledgments

This document is the result of extensive discussions within the W3C Semantic Web Deployment Working Group. The document drew on the experiences of earlier groups and projects, including SWAD-Europe and the W3C Semantic Web Best Practices and Deployment Working Group. Members of the W3C's public-esw-thes mailing list also made valuable contributions.

Appendix A. SKOS Properties and Classes

A.1. Classes in the SKOS Data Model

	skos:Collection
URI:	`http://www.w3.org/2004/02/skos/core#Collection`
Definition:	Section 9. Concept Collections
Label:	*Collection*
Disjoint classes:	`skos:Concept` `skos:ConceptScheme`
	skos:Concept
URI:	`http://www.w3.org/2004/02/skos/core#Concept`
Definition:	Section 3. The skos:Concept Class
Label:	*Concept*
Disjoint classes:	`skos:Collection` `skos:ConceptScheme`
	skos:ConceptScheme
URI:	`http://www.w3.org/2004/02/skos/core#ConceptScheme`
Definition:	Section 4. Concept Schemes
Label:	*Concept Scheme*

Disjoint classes:	`skos:Collection` `skos:Concept`
	skos:OrderedCollection
URI:	`http://www.w3.org/2004/02/skos/core#OrderedCollection`
Definition:	Section 9. Concept Collections
Label:	*Ordered Collection*
Super-classes:	`skos:Collection`

A.2. Properties in the SKOS Data Model

	skos:altLabel
URI:	`http://www.w3.org/2004/02/skos/core#altLabel`
Definition:	Section 5. Lexical Labels
Label:	*alternative label*
Super-properties:	`http://www.w3.org/2000/01/rdf-schema#label`
	skos:broadMatch
URI:	`http://www.w3.org/2004/02/skos/core#broadMatch`
Definition:	Section 10. Mapping Properties
Label:	*has broader match*
Super-properties:	`skos:broader` `skos:mappingRelation`
Inverse of:	`skos:narrowMatch`
	skos:broader
URI:	`http://www.w3.org/2004/02/skos/core#broader`
Definition:	Section 8. Semantic Relations
Label:	*has broader*
Super-properties:	`skos:broaderTransitive`
Inverse of:	`skos:narrower`
	skos:broaderTransitive
URI:	`http://www.w3.org/2004/02/skos/core#broaderTransitive`
Definition:	Section 8. Semantic Relations
Label:	*has broader transitive*
Super-properties:	`skos:semanticRelation`
Inverse of:	`skos:narrowerTransitive`
Other characteristics:	Transitive
	skos:changeNote
URI:	`http://www.w3.org/2004/02/skos/core#changeNote`
Definition:	Section 7. Documentation Properties
Label:	*change note*
Super-properties:	`skos:note`
	skos:closeMatch
URI:	`http://www.w3.org/2004/02/skos/core#closeMatch`

Definition:	Section 10. Mapping Properties
Label:	*has close match*
Super-properties:	`skos:mappingRelation`
Other characteristics:	Symmetric
skos:definition	
URI:	`http://www.w3.org/2004/02/skos/core#definition`
Definition:	Section 7. Documentation Properties
Label:	*definition*
Super-properties:	`skos:note`
skos:editorialNote	
URI:	`http://www.w3.org/2004/02/skos/core#editorialNote`
Definition:	Section 7. Documentation Properties
Label:	*editorial note*
Super-properties:	`skos:note`
skos:exactMatch	
URI:	`http://www.w3.org/2004/02/skos/core#exactMatch`
Definition:	Section 10. Mapping Properties
Label:	*has exact match*
Super-properties:	`skos:closeMatch`
Other characteristics:	Transitive Symmetric
skos:example	
URI:	`http://www.w3.org/2004/02/skos/core#example`
Definition:	Section 7. Documentation Properties
Label:	*example*
Super-properties:	`skos:note`
skos:hasTopConcept	
URI:	`http://www.w3.org/2004/02/skos/core#hasTopConcept`
Definition:	Section 4. Concept Schemes
Label:	*label*
Domain:	`skos:ConceptScheme`
Range:	`skos:Concept`
Inverse of:	`skos:topConceptOf`
skos:hiddenLabel	
URI:	`http://www.w3.org/2004/02/skos/core#hiddenLabel`
Definition:	Section 5. Lexical Labels
Label:	*hidden label*
Super-properties:	`http://www.w3.org/2000/01/rdf-schema#label`
skos:historyNote	
URI:	`http://www.w3.org/2004/02/skos/core#historyNote`

Definition:	Section 7. Documentation Properties
Label:	*history note*
Super-properties:	`skos:note`
skos:inScheme	
URI:	`http://www.w3.org/2004/02/skos/core#inScheme`
Definition:	Section 4. Concept Schemes
Label:	*is in scheme*
Range:	`skos:ConceptScheme`
skos:mappingRelation	
URI:	`http://www.w3.org/2004/02/skos/core#mappingRelation`
Definition:	Section 10. Mapping Properties
Label:	*is in mapping relation with*
Super-properties:	`skos:semanticRelation`
skos:member	
URI:	`http://www.w3.org/2004/02/skos/core#member`
Definition:	Section 9. Concept Collections
Label:	*has member*
Domain:	`skos:Collection`
Range:	union of `skos:Concept` and `skos:Collection`
skos:memberList	
URI:	`http://www.w3.org/2004/02/skos/core#memberList`
Definition:	Section 9. Concept Collections
Label:	*has member list*
Domain:	`skos:OrderedCollection`
Range:	`http://www.w3.org/1999/02/22-rdf-syntax-ns#List`
Other characteristics:	Functional
skos:narrowMatch	
URI:	`http://www.w3.org/2004/02/skos/core#narrowMatch`
Definition:	Section 10. Mapping Properties
Label:	*has narrower match*
Super-properties:	`skos:mappingRelation` `skos:narrower`
Inverse of:	`skos:broadMatch`
skos:narrower	
URI:	`http://www.w3.org/2004/02/skos/core#narrower`
Definition:	Section 8. Semantic Relations
Label:	*has narrower*
Super-properties:	`skos:narrowerTransitive`
Inverse of:	`skos:broader`
skos:narrowerTransitive	

URI:	`http://www.w3.org/2004/02/skos/core#narrowerTransitive`
Definition:	Section 8. Semantic Relations
Label:	*has narrower transitive*
Super-properties:	`skos:semanticRelation`
Inverse of:	`skos:broaderTransitive`
Other characteristics:	Transitive
skos:notation	
URI:	`http://www.w3.org/2004/02/skos/core#notation`
Definition:	Section 6. Notations
Label:	*notation*
skos:note	
URI:	`http://www.w3.org/2004/02/skos/core#note`
Definition:	Section 7. Documentation Properties
Label:	*note*
skos:prefLabel	
URI:	`http://www.w3.org/2004/02/skos/core#prefLabel`
Definition:	Section 5. Lexical Labels
Label:	*preferred label*
Super-properties:	`http://www.w3.org/2000/01/rdf-schema#label`
skos:related	
URI:	`http://www.w3.org/2004/02/skos/core#related`
Definition:	Section 8. Semantic Relations
Label:	*has related*
Super-properties:	`skos:semanticRelation`
Other characteristics:	Symmetric
skos:relatedMatch	
URI:	`http://www.w3.org/2004/02/skos/core#relatedMatch`
Definition:	Section 10. Mapping Properties
Label:	*has related match*
Super-properties:	`skos:mappingRelation` `skos:related`
Other characteristics:	Symmetric
skos:scopeNote	
URI:	`http://www.w3.org/2004/02/skos/core#scopeNote`
Definition:	Section 7. Documentation Properties
Label:	*scope note*
Super-properties:	`skos:note`
skos:semanticRelation	
URI:	`http://www.w3.org/2004/02/skos/core#semanticRelation`
Definition:	Section 8. Semantic Relations

Label:	*is in semantic relation with*
Domain:	`skos:Concept`
Range:	`skos:Concept`
skos:topConceptOf	
URI:	`http://www.w3.org/2004/02/skos/core#topConceptOf`
Definition:	Section 4. Concept Schemes
Label:	*is top concept in scheme*
Super-properties:	`skos:inScheme`
Inverse of:	`skos:hasTopConcept`

Appendix B. SKOS eXtension for Labels (SKOS-XL)

This appendix defines an **optional** extension to the Simple Knowledge Organization System, called the SKOS eXtension for Labels (SKOS-XL). This extension provides additional support for identifying, describing and linking lexical entities.

A special class of lexical entities, called `skosxl:Label`, is defined. Each instance of this class has a single RDF plain literal form, but two instances of this class are not necessarily the same individual if they share the same literal form.

Three labeling properties, `skosxl:prefLabel`, `skosxl:altLabel` and `skosxl:hiddenLabel`, are defined. These properties are used to label SKOS concepts with instances of `skosxl:Label`, and are otherwise analogous to the properties of the same local name defined in SKOS (`skos:prefLabel`, `skos:altLabel` and `skos:hiddenLabel` respectively).

The SKOS data model also defines the property `skosxl:labelRelation`. This property can be used to assert a direct (binary) link between instances of `skosxl:Label`. It is primarily intended as an extension point, to be refined for more specific types of link. No built-in refinements of `skosxl:labelRelation` are provided, although some examples of how this could be done are given.

B.1. SKOS-XL Namespace and Vocabulary

The SKOS-XL namespace URI is:

- **http://www.w3.org/2008/05/skos-xl#**

Here the prefix `skosxl:` is used as an abbreviation for the SKOS-XL namespace URI.

The SKOS-XL vocabulary is the set of URIs given in the left-hand column of the table below.

Table 2. The SKOS-XL Vocabulary

URI	Defined by (section of this appendix)
`skosxl:Label`	The skosxl:Label Class
`skosxl:literalForm`	The skosxl:Label Class
`skosxl:prefLabel`	Preferred, Alternate and Hidden skosxl:Labels
`skosxl:altLabel`	Preferred, Alternate and Hidden skosxl:Labels
`skosxl:hiddenLabel`	Preferred, Alternate and Hidden skosxl:Labels
`skosxl:labelRelation`	Links Between skosxl:Labels

Here "the SKOS-XL vocabulary" refers to the union of the SKOS vocabulary and the SKOS-XL vocabulary.

Here "the XL data model" refers to the class and property definitions stated in this appendix only. "The SKOS+XL data model" refers to the union of the data model defined in sections 3-10 above and the XL data model.

B.2. The skosxl:Label Class

B.2.1. Preamble

The class `skosxl:Label` is a special class of lexical entities.

An instance of the class `skosxl:Label` is a resource and may be named with a URI.

An instance of the class `skosxl:Label` has a single literal form. This literal form is an RDF plain literal (which is a string of UNICODE characters and an optional language tag [RDF-CONCEPTS]). The property `skosxl:literalForm` is used to give the literal form of an `skosxl:Label`.

If two instances of the class `skosxl:Label` have the same literal form, they are **not** necessarily the same resource.

B.2.2. Class and Property Definitions

S47	`skosxl:Label` is an instance of `owl:Class`.
S48	`skosxl:Label` is disjoint with each of `skos:Concept`, `skos:ConceptScheme` and `skos:Collection`.
S49	`skosxl:literalForm` is an instance of `owl:DatatypeProperty`.
S50	The `rdfs:domain` of `skosxl:literalForm` is the class `skosxl:Label`.
S51	The `rdfs:range` of `skosxl:literalForm` is the class of RDF plain literals.
S52	`skosxl:Label` is a sub-class of a restriction on `skosxl:literalForm` cardinality exactly 1.

B.2.3. Examples

The example below describes a `skosxl:Label` named with the URI `<http://example.com/ns/A>`, with the literal form "love" in English.

Example 75 (consistent)

```
<A> rdf:type skosxl:Label ; skosxl:literalForm "love"@en .
```

The four examples below are each **not consistent** with the XL data model, because an `skosxl:Label` is described with two different literal forms.

Example 76 (not consistent)

```
<B> rdf:type skosxl:Label ; skosxl:literalForm "love" ; skosxl:literalForm "adoration" .
```

Example 77 (not consistent)

```
<B> rdf:type skosxl:Label ; skosxl:literalForm "love"@en ; skosxl:literalForm "love"@fr .
```

Example 78 (not consistent)

```
<B> rdf:type skosxl:Label ; skosxl:literalForm "love"@en-GB ; skosxl:literalForm "love"@en-US .
```

Example 79 (not consistent)

```
<B> rdf:type skosxl:Label ; skosxl:literalForm "東"@ja-Hani ; skosxl:literalForm "ひがし"@ja-Hira .
```

B.2.4. Notes

B.2.4.1. Identity and Entailment

As stated above, each instance of the class `skosxl:Label` has **one and only one literal form**. In other words, there is a function mapping the class extension of `skosxl:Label` to the set of RDF plain literals. This function is defined by the property extension of `skosxl:literalForm`. Note especially two facts about this function.

First, the function is **not** injective. In other words, there is **not** a one-to-one mapping from instances of `skosxl:Label` to the set of RDF plain literals (in fact it is many-to-one). This means that two instances of `skosxl:Label` which have the same literal form are **not necessarily** the same individual. So, for example, the entailment illustrated below is **not** supported by the XL data model.

Example 80 (non-entailment)

```
<A> skosxl:literalForm "love"@en .
<B> skosxl:literalForm "love"@en .
```

does not entail

```
<A> owl:sameAs <B> .
```

Second, the function is **not** surjective. In other words, for a given plain literal `l`, there might not be any instances of `skosxl:Label` with literal form `l`.

B.2.4.2. Membership of Concept Schemes

The membership of an instance of `skosxl:Label` within a SKOS concept scheme can be asserted using the `skos:inScheme` property.

Example 81 (consistent)

```
<A> rdf:type skosxl:Label ; skosxl:literalForm "love"@en ; skos:inScheme <MyScheme> .
```

B.3. Preferred, Alternate and Hidden skosxl:Labels

B.3.1. Preamble

The three properties `skosxl:prefLabel`, `skosxl:altLabel` and `skosxl:hiddenLabel` are used to give the preferred, alternative and hidden labels of a resource respectively, where those labels are instances of the class `skosxl:Label`. These properties are analogous to the properties of the same local name defined in the SKOS vocabulary, and there are logical dependencies between these two sets of properties defined below.

B.3.2. Class and Property Definitions

S53	`skosxl:prefLabel`, `skosxl:altLabel` and `skosxl:hiddenLabel` are each instances of `owl:ObjectProperty`.
S54	The `rdfs:range` of each of `skosxl:prefLabel`, `skosxl:altLabel` and `skosxl:hiddenLabel` is the class `skosxl:Label`.
S55	The property chain (`skosxl:prefLabel`, `skosxl:literalForm`) is a sub-property of `skos:prefLabel`.
S56	The property chain (`skosxl:altLabel`, `skosxl:literalForm`) is a sub-property of `skos:altLabel`.
S57	The property chain (`skosxl:hiddenLabel`, `skosxl:literalForm`) is a sub-property of `skos:hiddenLabel`.
S58	`skosxl:prefLabel`, `skosxl:altLabel` and `skosxl:hiddenLabel` are pairwise disjoint properties.

B.3.3. Examples

The example below illustrates the use of all three XL labeling properties, and is consistent with the SKOS+XL data model.

Example 82 (consistent)

```
<Love>
  skosxl:prefLabel <A> ;
  skosxl:altLabel <B> ;
  skosxl:hiddenLabel <C> .

<A> rdf:type skosxl:Label ;
  skosxl:literalForm "love"@en .

<B> rdf:type skosxl:Label ;
  skosxl:literalForm "adoration"@en .

<C> rdf:type skosxl:Label ;
  skosxl:literalForm "luv"@en .
```

B.3.4. Notes

B.3.4.1. Dumbing-Down to SKOS Lexical Labels

The sub-property chain axioms S55, S56 and S57 support the dumbing-down of XL labels to vanilla SKOS lexical labels via inference. This is illustrated in the example below.

Example 83 (entailment)

```
<Love>
  skosxl:prefLabel <A> ;
  skosxl:altLabel <B> ;
  skosxl:hiddenLabel <C> .

<A> rdf:type skosxl:Label ;
  skosxl:literalForm "love"@en .

<B> rdf:type skosxl:Label ;
  skosxl:literalForm "adoration"@en .

<C> rdf:type skosxl:Label ;
  skosxl:literalForm "luv"@en .
```

entails

```
<Love>
  skos:prefLabel "love"@en ;
  skos:altLabel "adoration"@en ;
  skos:hiddenLabel "luv"@en .
```

B.3.4.2. SKOS+XL Labeling Integrity

In Section 5, two integrity conditions were defined on the basic SKOS labeling properties. First, the properties `skos:prefLabel`, `skos:altLabel` and `skos:hiddenLabel` are pairwise disjoint. Second, a resource has no more than one value of `skos:prefLabel` per language. Because of the sub-property chain axioms defined above, the following four examples, whilst consistent w.r.t. the XL data model alone, are **not** consistent with the SKOS+XL data model.

Example 84 (not consistent)

```
# Two different preferred labels in the same language

<Love> skosxl:prefLabel <A> ; skosxl:prefLabel <B> .
<A> skosxl:literalForm "love"@en .
<B> skosxl:literalForm "adoration"@en .
```

Example 85 (not consistent)

```
# Clash between preferred and alternative labels

<Love> skosxl:prefLabel <A> ; skosxl:altLabel <B> .
<A> skosxl:literalForm "love"@en .
```

```
<B> skosxl:literalForm "love"@en .
```

Example 86 (not consistent)

```
# Clash between alternative and hidden labels

<Love> skosxl:altLabel <A> ; skosxl:hiddenLabel <B> .
<A> skosxl:literalForm "love"@en .
<B> skosxl:literalForm "love"@en .
```

Example 87 (not consistent)

```
# Clash between preferred and hidden labels

<Love> skosxl:prefLabel <A> ; skosxl:hiddenLabel <B> .
<A> skosxl:literalForm "love"@en .
<B> skosxl:literalForm "love"@en .
```

B.4. Links Between skosxl:Labels

B.4.1. Preamble

This section defines a pattern for representing binary links between instances of the class `skosxl:Label`.

Note that the vocabulary defined in this section is not intended to be used directly, but rather as an extension point which can be refined for more specific labeling scenarios.

B.4.2. Class and Property Definitions

S59	`skosxl:labelRelation` is an instance of `owl:ObjectProperty`.
S60	The `rdfs:domain` of `skosxl:labelRelation` is the class `skosxl:Label`.
S61	The `rdfs:range` of `skosxl:labelRelation` is the class `skosxl:Label`.
S62	`skosxl:labelRelation` is an instance of `owl:SymmetricProperty`.

B.4.3. Examples

The example below illustrates a link between two instances of the class `skosxl:Label`.

Example 88 (consistent)

```
<A> rdf:type skosxl:Label ; skosxl:literalForm "love" .
<B> rdf:type skosxl:Label ; skosxl:literalForm "adoration" .
<A> skosxl:labelRelation <B> .
```

B.4.4. Notes

B.4.4.1. Refinements of this Pattern

As mentioned above, the `skosxl:labelRelation` property serves as an extension point, which can be refined for more specific labeling scenarios.

In the example below, a third party has refined the property `skos:labelRelation` to express acronym relationships, and used it to express the fact that "FAO" is an acronym for "Food and Agriculture Organization".

Example 89 (consistent)

```
# First define an extension to skosxl:labelRelation
ex:acronym rdfs:subPropertyOf skosxl:labelRelation .

# Now use it
<A> rdf:type skosxl:Label ; skosxl:literalForm "FAO"@en .
<B> rdf:type skosxl:Label ; skosxl:literalForm "Food and Agriculture Organization"@en .
<B> ex:acronym <A> .
```

Note that a sub-property of a symmetric property is not necessarily symmetric.

B.5. SKOS-XL Schema Overview

The following table gives an overview of the SKOS-XL vocabulary.

B.5.1 Classes in the SKOS-XL Data Model

skosxl:Label	
URI:	`http://www.w3.org/2008/05/skos-xl#Label`
Definition:	Section B.2. The skosxl:Label Class

Label:	*Label*
Super-classes:	restriction on `skosxl:literalForm` cardinality exactly 1
Disjoint classes:	`skos:Collection` `skos:Concept` `skos:ConceptScheme`

B.5.2.Properties in the SKOS-XL Data Model

skosxl:altLabel	
URI:	`http://www.w3.org/2008/05/skos-xl#altLabel`
Definition:	Section B.3. Preferred, Alternate and Hidden skosxl:Labels
Label:	*alternative label*
Range:	`skosxl:Label`
skosxl:hiddenLabel	
URI:	`http://www.w3.org/2008/05/skos-xl#hiddenLabel`
Definition:	Section B.3. Preferred, Alternate and Hidden skosxl:Labels
Label:	*hidden label*
Range:	`skosxl:Label`
skosxl:labelRelation	
URI:	`http://www.w3.org/2008/05/skos-xl#labelRelation`
Definition:	Section B.4. Links Between skosxl:Labels
Label:	*label relation*
Domain:	`skosxl:Label`
Range:	`skosxl:Label`
Other characteristics:	Symmetric
skosxl:literalForm	
URI:	`http://www.w3.org/2008/05/skos-xl#literalForm`
Definition:	Section B.2. The skosxl:Label Class
Label:	*literal form*
Domain:	`skosxl:Label`
skosxl:prefLabel	
URI:	`http://www.w3.org/2008/05/skos-xl#prefLabel`
Definition:	Section B.3. Preferred, Alternate and Hidden skosxl:Labels
Label:	*preferred label*
Range:	`skosxl:Label`

Appendix C. SKOS and SKOS-XL Namespace Documents

Following Architecture of the World Wide Web, Volume One [WEBARCH], a "namespace document" is an "information resource that contains useful information, machine-usable and/or human-usable, about terms in the namespace".

The SKOS vocabulary is a conceptual resource identified by the namespace URI `http://www.w3.org/2004/02/skos/core#`. The normative definition of the SKOS vocabulary is found in SKOS Reference (this document).

The following namespace documents provide alternative representations of the SKOS vocabulary:

C.1. SKOS Namespace Document - HTML Variant (normative)

The SKOS vocabulary is summarized in SKOS Namespace Document - HTML Variant [SKOS-HTML], which is served from the namespace URI `http://www.w3.org/2004/02/skos/core#` via content negotiation using Recipe 3 of "Best Practice Recipes for Publishing Vocabularies" [RECIPES]. Clients requiring HTML or XHTML should include an appropriate "Accept" header in the HTTP request. Alternatively, the SKOS Namespace Document - HTML Variant can be referenced directly by citing its URI: `http://www.w3.org/TR/skos-reference/skos.html`.

The SKOS Namespace Document - HTML Variant replicates Appendix A. SKOS Properties and Classes of this document.

C.2. SKOS Namespace Document - RDF/XML Variant (normative)

The SKOS Namespace Document - RDF/XML Variant [SKOS-RDF] expresses the SKOS vocabulary and data model (in so far as possible) as RDF triples. It is served via content negotiation using Recipe 3 of "Best Practice Recipes for Publishing Vocabularies" [RECIPES]. Clients requiring RDF/XML should include an appropriate "Accept" header in the HTTP request. Alternatively, the RDF schema can be referenced directly (and downloaded) by citing its URI: `http://www.w3.org/TR/skos-reference/skos.rdf`.

It is not possible to express all of the statements of the SKOS data model as RDF triples, so this schema forms a normative subset of SKOS Reference. The RDF schema defines an OWL Full ontology [OWL-SEMANTICS] [OWL-REFERENCE]. SKOS vocabularies can be defined as instances of this ontology.

C.3. SKOS RDF Schema - OWL 1 DL Sub-set (informative)

For the convenience of tools and applications that wish to work within the constraints of OWL DL, the SKOS RDF Schema - OWL 1 DL Sub-set [SKOS-RDF-OWL1-DL] provides a modified, informative, schema which conforms to those constraints. Note that this schema is obtained through the deletion of triples representing axioms that violate OWL DL constraints. Alternative prunings could be performed.

The SKOS OWL 1 DL Sub-set is available by citing its URI: `http://www.w3.org/TR/skos-reference/skos-owl1-dl.rdf`

C.4. SKOS-XL Namespace Document - HTML Variant (normative)

The SKOS-XL vocabulary is summarized in SKOS-XL Namespace Document - HTML Variant [SKOS-XL-HTML], which is served from the namespace URI `http://www.w3.org/2008/05/skos-xl#` via content negotiation using Recipe 3 of "Best Practice Recipes for Publishing Vocabularies" [RECIPES]. Clients requiring HTML or XHTML should include an appropriate "Accept" header in the HTTP request. Alternatively, the SKOS-XL Namespace Document - HTML Variant can be referenced directly by citing its URI: `http://www.w3.org/TR/skos-reference/skos-xl.html`.

The SKOS-XL HTML Variant Namespace Document replicates Appendix B.5 SKOS-XL Schema Overview of this document.

C.5. SKOS-XL Namespace Document - RDF/XML Variant (normative)

This RDF schema document expresses the SKOS vocabulary and data model (in so far as possible) as RDF triples. It is served from the namespace URI `http://www.w3.org/2008/05/skos-xl#` via content negotiation using Recipe 3 of "Best Practice Recipes for Publishing Vocabularies" [RECIPES]. Clients requiring RDF/XML should include an appropriate "Accept" header in the HTTP request. Alternatively, the RDF schema can be referenced directly (and downloaded) by citing its URI: `http://www.w3.org/TR/skos-reference/skos-xl.rdf`.

Appendix D. SKOS Namespace: a historical note

The SKOS schema defines vocabulary using the namespace http://www.w3.org/2004/02/skos/core#. This namespace was used to define the original SKOS schema which served as a starting point for this Recommendation. As a result of this, elements present in previous versions of the machine-readable schema have been removed from the current version. In a number of cases, the definition or semantics of elements in the schema has changed.

Retaining the existing SKOS namespace avoids some issues with existing KOS that are already using the SKOS schema. Users should, however, be aware of the change in the semantics of `skos:broader` (and `skos:narrower`) which *may* impact on SKOS applications.

Where elements have been removed, no explicit deprecation axioms have been expressed in the schema. Historical versions of the SKOS schema, are, however, available from the SKOS Web site "version history" page, and those elements which have been removed from the recent version of the vocabulary are listed below.

- `skos:symbol`
- `skos:prefSymbol`
- `skos:altSymbol`
- `skos:CollectableProperty`
- `skos:subject`
- `skos:isSubjectOf`
- `skos:primarySubject`
- `skos:isPrimarySubjectOf`
- `skos:subjectIndicator`

In the case of `skos:broader` and `skos:narrower`, the semantics of the vocabulary elements have been changed — these properties are no longer declared to be transitive. Thus the follow entailment does not hold.

Example 90 (non-entailment)

```
<A> skos:broader <B> .
<B> skos:broader <C> .
```

does not entail

```
<A> skos:broader <C> .
```

A transitive super property of `skos:broader` — `skos:broaderTransitive` — is provided which allows for query across the transitive closure of `skos:broader` relations. A similar property — `skos:narrowerTransitive` — is provided for query across the transitive closure of `skos:narrower`.