



SKOS Simple Knowledge Organization System Reference

W3C Working Draft 29 August 2008

This version:

<http://www.w3.org/TR/2008/WD-skos-reference-20080829/>

Latest version:

<http://www.w3.org/TR/skos-reference>

Previous version:

<http://www.w3.org/TR/2008/WD-skos-reference-20080609/>

Editors:

[Alistair Miles](#), STFC Rutherford Appleton Laboratory / University of Oxford

[Sean Bechhofer](#), University of Manchester

Copyright © 2008 W3C® ([MIT](#), [ERCIM](#), [Keio](#)), All Rights Reserved. W3C [liability](#), [trademark](#) and [document use](#) rules apply.

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 light weight, 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 Last Call Working Draft published by the [Semantic Web Deployment Working Group](#), part of the [W3C Semantic Web Activity](#). This Working Draft updates the the [9 Jun 2008 Working Draft](#) with the Working Group decisions since that time. The changes are summarized below.

The Working Group solicits review and feedback on this draft specification. Comments are requested by 3 October 2008, at which time the Working Group intends to close Last Call.

All comments are welcome and may be sent to public-swd-wg@w3.org; please include the text "SKOS comment" in the subject line. All messages received at this address are viewable in a [public archive](#).

The Working Group intends to advance the SKOS Reference to W3C Recommendation after further review and comment. This Last Call Working Draft signals the Working Group's belief that it has met its design objectives for SKOS and has resolved all open issues.

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 has actual knowledge of a patent which the individual believes contains [Essential Claim\(s\)](#) must disclose the information in accordance with [section 6 of the W3C Patent Policy](#).

Publication as a Working Draft does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

Features at Risk of Change

The Working Group have identified a number of features which it considers to be "at risk of change". These features include:

- The URI of the SKOS namespace. The current draft introduces a new namespace for the SKOS vocabulary. Comments have been made that this may cause difficulty for existing SKOS implementations or vocabularies. As a result, the choice of SKOS namespace should be considered "at risk of change".

- Relationship between `skos:exactMatch`, `skos:broadMatch` and `skos:narrowMatch`. The Working group consider that there is insufficient implementation experience or evidence to be able to make a firm decision (see resolution of ISSUE 75). The current situation is that there are no axioms stating relationships between `skos:exactMatch`, `skos:broadMatch` and `skos:narrowMatch`. Axioms could be stated, for example, asserting that the composition of `skos:broadMatch` and `skos:exactMatch` is a subproperty of `skos:broadMatch`. Note that this would, however, require OWL 2 features which are not present in OWL.
- The naming of the property `skos:topConceptOf` may change.

Any feature at risk of change is marked in the body of this document with a comment in a red box, such as:

At Risk of Change: This feature may change in subsequent versions of this document...

Changes

Changes Since [9 June 2008 Working Draft](#):

- [Section 1.7 \(Conformance\)](#): Wording on requirements for URI dereference behaviour have been added.
- [Section 4 \(Concept Schemes\)](#): A new property `skos:topConceptOf` is added, which is the inverse of `skos:hasTopConcept`, and which is a sub-property of `skos:inScheme`.
- [Section 5 \(Lexical Labels\)](#): Assertions that `rdfs:label` is a super-property of the SKOS lexical labelling properties `skos:altLabel`, `skos:hiddenLabel` and `skos:prefLabel` added.
- [Section 8 \(Semantic Relations\)](#): The preamble has been revised for clarity.
- [Section 10 \(Mapping Properties\)](#): A new property `skos:closeMatch` is added, used to link two concepts that are sufficiently similar that they can be used interchangeably in some information retrieval applications. The informal definition of `skos:exactMatch` is modified, such that the property now indicates a high degree of confidence that two linked concepts can be used interchangeably across a wide range of information retrieval applications. The formal definition of `skos:exactMatch` is modified, such that it is now transitive, a sub-property of `skos:closeMatch`, and disjoint with each of `skos:broadMatch` and `skos:relatedMatch`.
- Editorial Changes
 - Meaningful examples
 - Minor rewording and addressing grammatical issues throughout
 - Changes to xl prefix used in prose and examples

Changes since [25 January 2008 Working Draft](#):

- The namespace used for the SKOS Vocabulary has been changed to `<http://www.w3.org/2008/05/skos#>`.
- A new section on notations has been included.
- A new appendix "SKOS eXtension for Labels" has been included.
- The section on label relations has been dropped. This is replaced by the `skosxl:labelRelation` property in the eXtension for Labels appendix.
- The section on mapping properties has been rewritten to reflect the decision to make `skos:broadMatch`, `skos:narrowMatch` and `skos:relatedMatch` sub-properties of `skos:broader`, `skos:narrower` and `skos:related` respectively.
- Appendices on SKOS Rules of Thumb, SKOS/OWL Patterns, SKOS/SPARQL Patterns and Extending SKOS have been dropped as out of scope for this document, to be covered in the SKOS Primer and/or other specifications.

Table of Contents

[\[show quick access panel\]](#)

- [1. Introduction](#)**
 - [1.1. Background and Motivation](#)
 - [1.2. SKOS Overview](#)
 - [1.3. SKOS, RDF and OWL](#)
 - [1.4. Consistency and Integrity](#)
 - [1.5. Inference, Dependency and the Open-World Assumption](#)
 - [1.6. Design Rationale](#)
 - [1.7. How to Read this Document](#)
 - [1.7.1. Formal Definitions](#)
 - [1.7.2. URI Abbreviations](#)
 - [1.7.3. Examples](#)
 - [1.8. Conformance](#)
- [2. SKOS Namespace and Vocabulary](#)**
- [3. The `skos:Concept` Class](#)**
 - [3.1. Preamble](#)
 - [3.2. Vocabulary](#)
 - [3.3. Class & Property Definitions](#)
 - [3.4. Examples](#)
 - [3.5. Notes](#)
 - [3.5.1. SKOS Concepts, OWL Classes and OWL Properties](#)
- [4. Concept Schemes](#)**
 - [4.1. Preamble](#)
 - [4.2. Vocabulary](#)
 - [4.3. Class & Property Definitions](#)
 - [4.4. Integrity Conditions](#)
 - [4.5. Examples](#)
 - [4.6. Notes](#)
 - [4.6.1. Closed vs. Open Systems](#)
 - [4.6.2. SKOS Concept Schemes and OWL Ontologies](#)
 - [4.6.3. Top Concepts and Semantic Relations](#)
 - [4.6.4. Scheme Containment and Semantic Relations](#)
- [5. Lexical Labels](#)**
 - [5.1. Preamble](#)
 - [5.2. Vocabulary](#)
 - [5.3. Class & Property Definitions](#)
 - [5.4. Integrity Conditions](#)
 - [5.5. Examples](#)

- [5.6. Notes](#)
 - [5.6.1. Domain of SKOS Lexical Labeling Properties](#)
 - [5.6.2. Range of SKOS Lexical Labeling Properties](#)
 - [5.6.3. Defining Label Relations](#)
 - [5.6.4. Alternates Without Preferred](#)
 - [5.6.5. Definition of a "Language"](#)
- **[6. Notations](#)**
 - [6.1. Preamble](#)
 - [6.2. Vocabulary](#)
 - [6.3. Class & Property Definitions](#)
 - [6.4. Examples](#)
 - [6.5. Notes](#)
 - [6.5.1. Notations, Typed Literals and Datatypes](#)
 - [6.5.2. Multiple Notations](#)
 - [6.5.3. Unique Notations in Concept Schemes](#)
 - [6.5.4. Notations and Preferred Labels](#)
- **[7. Documentation Properties \(Note Properties\)](#)**
 - [7.1. Preamble](#)
 - [7.2. Vocabulary](#)
 - [7.3. Class & Property Definitions](#)
 - [7.4. Examples](#)
 - [7.5. Notes](#)
 - [7.5.1. Domain of the SKOS Documentation Properties](#)
 - [7.5.2. Type & Range of the SKOS Documentation Properties](#)
- **[8. Semantic Relations](#)**
 - [8.1. Preamble](#)
 - [8.2. Vocabulary](#)
 - [8.3. Class & Property Definitions](#)
 - [8.4. Integrity Conditions](#)
 - [8.5. Examples](#)
 - [8.6. Notes](#)
 - [8.6.1. Sub-Property Relationships](#)
 - [8.6.2. Domain and Range of SKOS Semantic Relation Properties](#)
 - [8.6.3. Symmetry of skos:related](#)
 - [8.6.4. Transitivity of skos:related](#)
 - [8.6.5. Reflexivity of skos:related](#)
 - [8.6.6. Transitivity of skos:broader](#)
 - [8.6.7. Reflexivity of skos:broader](#)
 - [8.6.8. Cycles in the Hierarchical Relation \(Reflexivity of skos:broaderTransitive\)](#)
 - [8.6.9. Alternate Paths in the Hierarchical Relation](#)
 - [8.6.10. Disjointness of skos:related and skos:broaderTransitive](#)
- **[9. Concept Collections](#)**
 - [9.1. Preamble](#)
 - [9.2. Vocabulary](#)
 - [9.3. Class & Property Definitions](#)
 - [9.4. Integrity Conditions](#)
 - [9.5. Examples](#)
 - [9.6. Notes](#)
 - [9.6.1. Inferring Collections from Ordered Collections](#)
 - [9.6.2. skos:memberList Integrity](#)
 - [9.6.3. Nested Collections](#)
 - [9.6.4. SKOS concepts, Concept Collections and Semantic Relations](#)
- **[10. Mapping Properties](#)**
 - [10.1. Preamble](#)
 - [10.2. Vocabulary](#)
 - [10.3. Class & Property Definitions](#)
 - [10.4. Integrity Conditions](#)
 - [10.5. Examples](#)
 - [10.6. Notes](#)
 - [10.6.1. Mapping Properties, Semantic Relation Properties and Concept Schemes](#)
 - [10.6.2. "Clashes" Between Hierarchical and Associative Links](#)
 - [10.6.3. Transitivity of Mapping Properties](#)
 - [10.6.4. Reflexivity of Mapping Properties](#)
 - [10.6.5. Cycles and Alternate Paths Involving skos:broadMatch](#)
 - [10.6.6. Property Chains Involving skos:exactMatch](#)
 - [10.6.7. skos:closeMatch, skos:exactMatch, owl:sameAs, owl:equivalentClass, owl:equivalentProperty](#)
- **[11. References](#)**
- **[Appendix A. SKOS eXtension for Labels \(XL\)](#)**
 - [A.1. XL Namespace and Vocabulary](#)
 - [A.2. The skosxl:Label Class](#)
 - [A.2.1. Preamble](#)
 - [A.2.2. Class and Property Definitions](#)
 - [A.2.3. Examples](#)
 - [A.2.4. Notes](#)
 - [A.2.4.1. Identity and Entailment](#)
 - [A.2.4.2. Membership of Concept Schemes](#)
 - [A.3. Preferred, Alternate and Hidden skosxl:Labels](#)
 - [A.3.1. Preamble](#)
 - [A.3.2. Class and Property Definitions](#)
 - [A.3.3. Examples](#)
 - [A.3.4. Notes](#)
 - [A.3.4.1. "Dumbing-Down" to SKOS Lexical Labels](#)
 - [A.3.4.2. SKOS+XL Labeling Integrity](#)
 - [A.4. Links Between skosxl:Labels](#)
 - [A.4.1. Preamble](#)
 - [A.4.2. Class and Property Definitions](#)
 - [A.4.3. Examples](#)

- [A.4.4. Notes](#)
 - [A.4.4.1. Refinements of this Pattern](#)
- [Appendix B. SKOS Overview](#)
 - [B.1. Classes in the SKOS Data Model](#)
 - [B.2. Properties in the SKOS Data Model](#)
- [Appendix C. SKOS Data Model as RDF Triples](#)

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 "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. 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. This ability to "join up" data from different sources is motivating many projects, as different communities seek to exploit the hidden value in linking 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 [\[RDF-PRIMER\]](#) [\[OWL-GUIDE\]](#). They go a long way towards supporting that vision, but the story doesn't end there. 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 is 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. It is hoped that, by making the accumulated experience and wisdom of knowledge organization in the library and information sciences accessible, applicable within 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), 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, 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 Hiragana. One of these labels in any given language can be indicated as the "preferred" label for that language, and the others as "alternate" labels. Labels may also be "hidden", which is useful e.g. 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 catalogues. 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 A. SKOS eXtension for Labels \(XL\)](#). 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.

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 "facts" 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. For example, the RDF graph below 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 ;
      dc: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 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.

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. For example, 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 conform to or "fit" with 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 has any correspondence with the "real world", i.e. whether it is 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 is consistent with respect to the data model.

For example, 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 (OWL Full) 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 "fit" 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. For example, 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 (draw inferences using an RDFS or OWL reasoner, then search 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 3-11 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, alternate 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 is located. Therefore, one cannot generally assume that data obtained from such a system is "complete". I.e. if some data appears 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.

For example, 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 (RDFS and OWL Full) 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 `` are instances of `skos:Concept`.

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. For example, in [section 7](#) below, `skos:broader` and `skos:narrower` are defined as 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 "joining up" 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 Organisation System encompasses a wide range of artefacts. There is thus a danger of overcommitment in the SKOS schema, which could preclude the use of SKOS for a particular application. In order to alleviate this, in situations where there is doubt about the inclusion of a formal constraint (for example, see discussion about `skos:hasTopConcept`), the constraint has not been stated formally. In such cases, usage conventions may be suggested, or specialisations of the SKOS vocabulary may be used in order to enforce constraints (see 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 comprises 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 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 .
```

etc.

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.

For example, from [Section 5](#), "A resource has no more than one value of `skos:prefLabel` per language" means for any resource *x*, no two members of the set { *y* | <*x*,*y*> is in IEXT(*l*(`skos:prefLabel`)) } share the same language tag (where *l* 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. For example, `<http://example.org/ns/example>`. Relative URIs are cited in the same way, and are relative to the base URI `<http://example.org/ns/>`. For example, `<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
<code>http://www.w3.org/2008/05/skos#</code>	<code>skos:</code>
<code>http://www.w3.org/1999/02/22-rdf-syntax-ns#</code>	<code>rdf:</code>
<code>http://www.w3.org/2000/01/rdf-schema#</code>	<code>rdfs:</code>

http://www.w3.org/2002/07/owl#	owl:
--------------------------------	------

So, for example, `skos:Concept` is an abbreviation of `<http://www.w3.org/2008/05/skos#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/2008/05/skos#> .
@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/2008/05/skos#> .
@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/2008/05/skos#"
  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/2008/05/skos#Concept> .
```

Note the use in Turtle of the ";" and "," characters to abbreviate multiple triples with the same subject or predicate.

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 behaviour 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/2008/05/skos#>

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

Table 1. SKOS Vocabulary

URI	Definition
<code>skos:Concept</code>	Section 3. The <code>skos:Concept</code> Class
<code>skos:ConceptScheme</code>	Section 4. Concept Schemes
<code>skos:inScheme</code>	Section 4. Concept Schemes
<code>skos:hasTopConcept</code>	Section 4. Concept Schemes
<code>skos:topConceptOf</code>	Section 4. Concept Schemes
<code>skos:altLabel</code>	Section 5. Lexical Labels
<code>skos:hiddenLabel</code>	Section 5. Lexical Labels
<code>skos:prefLabel</code>	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](#).

At Risk of Change: The SKOS namespace URI may be changed in future versions of this specification. In particular, the namespace URI may be changed back to `<http://www.w3.org/2004/02/skos/core#>`. Defining a new namespace URI for SKOS potentially introduces difficulties in handling legacy data. Retaining the existing namespace may also be problematic given the changes in the semantics of the properties.

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	<code>skos:Concept</code> is an instance of <code>owl:Class</code> .
----	--

3.4. Examples

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

Example 2 (consistent)
<code><MyConcept> rdf:type skos:Concept .</code>

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.

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

Example 3 (consistent)
<code><MyConcept> rdf:type skos:Concept , owl:Class .</code>

This example is **consistent** with the SKOS data model. However, note that it is **not** compatible with OWL DL, because the URI `<MyConcept>` has been used to denote both a class and an individual [\[OWL-SEMANTICS\]](#). To work within the OWL DL language, take care **not** to use the same URI to denote both an OWL class and a SKOS concept.

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

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

Example 4 (consistent)
<code><MyConcept> rdf:type skos:Concept , owl:ObjectProperty .</code>

This example is **consistent** with the SKOS data model. However, it is **not** compatible with OWL DL, because the URI `<MyConcept>` has been used to denote both an object property and an individual [\[OWL-SEMANTICS\]](#).

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

<code>skos:ConceptScheme</code>
<code>skos:inScheme</code>
<code>skos:hasTopConcept</code>
<code>skos:topConceptOf</code>

4.3. Class & Property Definitions

S2	<code>skos:ConceptScheme</code> is an instance of <code>owl:Class</code> .
S3	<code>skos:inScheme</code> , <code>skos:hasTopConcept</code> and <code>skos:topConceptOf</code> are each instances of <code>owl:ObjectProperty</code> .
S4	The <code>rdfs:range</code> of <code>skos:inScheme</code> is the class <code>skos:ConceptScheme</code> . 
S5	The <code>rdfs:domain</code> of <code>skos:hasTopConcept</code> is the class <code>skos:ConceptScheme</code> .
S6	The <code>rdfs:range</code> of <code>skos:hasTopConcept</code> is the class <code>skos:Concept</code> .
S7	<code>skos:topConceptOf</code> is a sub-property of <code>skos:inScheme</code> .

S8  skos:topConceptOf is the owl:inverseOf the property skos:hasTopConcept.

At Risk of Change: The property `skos:topConceptOf` is at risk of change or removal in future versions of this specification.

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 [\[OWL-GUIDE\]](#).

For example, in the graph below, `<MyScheme>` is both a 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> .
```

How an application should handle such data is not defined in this specification.

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)
<pre><A> skos:narrower . <A> skos:inScheme <MyScheme> . does not entail skos:inScheme <MyScheme> .</pre>

See also [Section 8](#) 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 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", "alternate" and "hidden" lexical labels for any given resource.

The "preferred" and "alternate" 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.



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:DatatypeProperty.
S11	skos:prefLabel, skos:altLabel and skos:hiddenLabel are each subproperties 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. 

N.B. the conditions above assume that each distinct language tag allowed by [\[BCP47\]](#) denotes a different "language". E.g. "en", "en-GB" and "en-US" denote three different languages (English, British English and US English respectively). E.g. "ja", "ja-Hani", "ja-Hira", "ja-Kana" and "ja-Latn" denote five different languages (Japanese, Japanese Kanji, Japanese Hiragana, Japanese Katakana and Japanese Rōmaji respectively).

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

The following graph is **consistent**, and illustrates the provision of lexical labels in four different languages (Japanese Kanji, Japanese Hiragana, Japanese Katakana and Japanese 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 for the same language.

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 alternate 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 alternate 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.

For example, in the 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 .
```

This example is, however, incompatible with OWL DL, because a datatype property has been used to make assertions about a class [OWL-SEMANTICS].



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`. However, there is nothing in the RDF or OWL Full semantics which prevents the use of a URI or a blank node to denote an RDF plain literal. Therefore, although the graph below does not adhere to the usage convention stated above, it is nevertheless **consistent** with the SKOS data model.

Example 17 (consistent)

```
<Love> skos:prefLabel <love> .
```

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 alternate 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 semantics. However, note that many applications will require a preferred lexical label in order to generate an optimum human-readable display.

Example 18 (consistent)

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

5.6.5. Definition of a "Language"

Note how "language" has been defined for the conditions above. Each different language tag permissible by [\[BCP47\]](#) is taken to denote a different "language". Therefore the graph below is **consistent** with the SKOS data model, because "en", "en-US" and "en-GB" are taken to denote different languages.

Example 19 (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 taken to denote different languages, and therefore the graph is **consistent**.

Example 20 (consistent)

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

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 to a concept 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 21 (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 X³ schema language, see [\[SWBP-DATATYPES\]](#) section 2.

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 (i.e. datatypes). However, 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).

However, this usage convention is not part of the formal SKOS data model. I.e. there are no integrity conditions on the use of `skos:notation`.

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 the lexical form of a preferred label.

Example 22 (consistent)
<pre><Potassium> skos:prefLabel "K"@en ; skos:notation "K"^^<ChemicalSymbolNotation> .</pre>

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.

Usually a notation system does not have any correspondence with a natural language — it is language-independent (and hence typed literals are an appropriate representation). However, in some cases a system of notation might be associated with a particular language and/or script. One possible strategy in this situation is to use the private use language sub-tags [BCP47] with `skos:prefLabel`, as illustrated in the example below — this is consistent with the SKOS data model. An alternative strategy is to define one or more sub-properties of `skos:prefLabel` and/or `skos:altLabel`.

Example 23 (consistent)
<pre><Potassium> skos:prefLabel "Potassium"@en ; skos:prefLabel "K"@en-x-chemicalsymbol ; skos:notation "K"^^<ChemicalSymbolNotation> .</pre>

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. For example, 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 formal semantics stated in this section are intended to accommodate all three design patterns.

7.2. Vocabulary

<code>skos:note</code>
<code>skos:changeNote</code>
<code>skos:definition</code>
<code>skos:editorialNote</code>
<code>skos:example</code>
<code>skos:historyNote</code>
<code>skos:scopeNote</code>

7.3. Class & Property Definitions

S16	<code>skos:note</code> , <code>skos:changeNote</code> , <code>skos:definition</code> , <code>skos:editorialNote</code> , <code>skos:example</code> , <code>skos:historyNote</code> and <code>skos:scopeNote</code> are each instances of <code>owl:ObjectProperty</code> .
S17	<code>skos:changeNote</code> , <code>skos:definition</code> , <code>skos:editorialNote</code> , <code>skos:example</code> , <code>skos:historyNote</code> and <code>skos:scopeNote</code> are each sub-properties of <code>skos:note</code> .

7.4. Examples

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

Example 24 (consistent)

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

The graph below gives an example of the "documentation as a related resource description" pattern.

Example 25 (consistent)

```
<MyResource> skos:note [ rdf:value "this is a note"@en ] .
```

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

Example 26 (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.

For example, in the 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 27 (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 .
```

This example is, however, incompatible with OWL DL, because an object property has been used to make an assertion about a class [[OWL-SEMANTICS](#)].

7.5.2. Type & Range of the SKOS Documentation Properties

Note that the basis for the SKOS data model is the RDF-compatible model-theoretic semantics for OWL Full [[OWL-SEMANTICS](#)]. Under the OWL Full semantics, the class of OWL datatype properties is a sub-class of the class of OWL object properties [[OWL-REFERENCE](#)]. Therefore, stating that the SKOS documentation properties are all instances of `owl:ObjectProperty` is the most general statement that can be made, and is consistent with all three usage patterns described in the [SKOS Primer](#).

Note also 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 ` asserts that ``, 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 ` represents a direct or indirect hierarchical link, where `` 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 relation, 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 the owl:inverseOf the property skos:broader.
S26	skos:narrowerTransitive is the owl:inverseOf the property skos:broaderTransitive.

8.4. Integrity Conditions

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

Note that as skos:related is a symmetric relation, and skos:broaderTransitive and skos:narrowerTransitive are inverses, skos:related can be inferred to be disjoint with skos:narrowerTransitive.

8.5. Examples

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

Example 28 (consistent)	
<A> skos:broader ; 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 29 (not consistent)	
<A> skos:broader ; skos:related .	

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

Example 30 (not consistent)	
<A> skos:broader ; skos:related <C> . 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 31 (not consistent)	
<A> skos:broaderTransitive <C> ; skos:related <C> .	

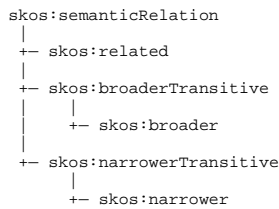
The graph below is **not consistent** with the SKOS data model, again because there is a "clash" between associative links and hierarchical links, which can be inferred from the class and property definitions given above.

Example 32 (not consistent)	
<A> skos:narrower ; skos:related <C> . 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.



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 28 above entails that `<A>`, `` 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 33 (entailment)
<pre><A> skos:related . entails skos:related <A> .</pre>

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 34 (entailment)
<pre><cause> rdf:type owl:ObjectProperty ; rdfs:subPropertyOf skos:related . <effect> rdf:type owl:ObjectProperty ; rdfs:subPropertyOf skos:related ; owl:inverseOf <cause> . <A> <cause> . entails <A> skos:related . <effect> <A> ; skos:related <A> .</pre>

See also the [SKOS Primer](#) for best practice recommendations on extending SKOS.

8.6.4. Transitivity of `skos:related`



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 35 (non-entailment)
<pre><A> skos:related . skos:related <C> . does not entail <A> skos:related <C> .</pre>

8.6.5. Reflexivity of `skos:related`

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 36 (consistent)
<pre><A> skos:related <A> .</pre>

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. Transitivity of `skos:broader`

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 37 (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 38 (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 alternate paths below.

8.6.7. Reflexivity of `skos:broader`

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 39 (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 (Reflexivity of `skos:broaderTransitive`)

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 40 (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 alternate paths from A to C in the hierarchical relation.

Example 41 (consistent)

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

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

Example 42 (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

<code>skos:Collection</code>
<code>skos:OrderedCollection</code>
<code>skos:member</code>
<code>skos:memberList</code>

9.3. Class & Property Definitions

S28	<code>skos:Collection</code> and <code>skos:OrderedCollection</code> are each instances of <code>owl:Class</code> .
S29	<code>skos:OrderedCollection</code> is a sub-class of <code>skos:Collection</code> .
S30	<code>skos:member</code> and <code>skos:memberList</code> are each instances of <code>owl:ObjectProperty</code> .
S31	The <code>rdfs:domain</code> of <code>skos:member</code> is the class <code>skos:Collection</code> .
S32	The <code>rdfs:domain</code> of <code>skos:memberList</code> is the class <code>skos:OrderedCollection</code> .
S33	The <code>rdfs:range</code> of <code>skos:memberList</code> is the class <code>rdf:List</code> .
S34	<code>skos:memberList</code> is an instance of <code>owl:FunctionalProperty</code> .
S35	For any resource, every item in the list given as the value of the <code>skos:memberList</code> property is also a value of the <code>skos:member</code> property.

9.4. Integrity Conditions

S36	<code>skos:Collection</code> is disjoint with each of <code>skos:Concept</code> and <code>skos:ConceptScheme</code> .
-----	---

9.5. Examples

The graph below illustrates a SKOS collection with 3 members.

Example 43 (consistent)
<pre><MyCollection> rdf:type skos:Collection ; skos:member <X> , <Y> , <Z> .</pre>

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

Example 44 (consistent)
<pre><MyOrderedCollection> rdf:type skos:OrderedCollection ; skos:memberList (<X> <Y> <Z>) .</pre>

9.6. Notes

9.6.1. Inferring Collections from Ordered Collections

Statement S34 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 45 (entailment)
<pre><MyOrderedCollection> rdf:type skos:OrderedCollection ; skos:memberList (<X> <Y> <Z>) . <i>entails</i> <MyOrderedCollection> rdf:type skos:Collection ; skos:member <X> , <Y> , <Z> .</pre>

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 46 (entailment)
<pre><OrderedCollectionResource> skos:memberList (<A>) , (<X> <Y>) . <i>entails</i> <OrderedCollectionResource> skos:memberList [rdf:first <A> , <X> ; rdf:rest [rdf:first ; rdf:rest rdf:nil] , [rdf:first <Y> ; rdf:rest rdf:nil]] .</pre>

9.6.3. Nested Collections

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

Example 47 (consistent)
<pre><MyCollection> rdf:type skos:Collection ; skos:member <A> , , <MyNestedCollection> . <MyNestedCollection> rdf:type skos:Collection ; skos:member <X> , <Y> , <Z> .</pre>

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

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 48 (not consistent)
<pre><A> skos:narrower . rdf:type skos:Collection .</pre>

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

Example 49 (not consistent)
<pre><A> skos:broader . rdf:type skos:Collection .</pre>

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

Example 50 (not consistent)
<pre><A> skos:related . rdf:type skos:Collection .</pre>

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

Example 51 (consistent)
<pre><A> skos:narrower , <C> , <D> . <ResourceCollection> rdfs:label "Resource Collection"@en ; skos:member , <C> , <D> .</pre>

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

<code>skos:mappingRelation</code>
<code>skos:closeMatch</code>
<code>skos:exactMatch</code>
<code>skos:broadMatch</code>
<code>skos:narrowMatch</code>
<code>skos:relatedMatch</code>

10.3. Class & Property Definitions

S37	<code>skos:mappingRelation</code> , <code>skos:closeMatch</code> , <code>skos:exactMatch</code> , <code>skos:broadMatch</code> , <code>skos:narrowMatch</code> and <code>skos:relatedMatch</code> are each instances of <code>owl:ObjectProperty</code> .
S38	The <code>rdfs:domain</code> of <code>skos:mappingRelation</code> is the class <code>skos:Concept</code> .
S39	The <code>rdfs:range</code> of <code>skos:mappingRelation</code> is the class <code>skos:Concept</code>
S40	<code>skos:closeMatch</code> , <code>skos:broadMatch</code> , <code>skos:narrowMatch</code> and <code>skos:relatedMatch</code> are each sub-properties of <code>skos:mappingRelation</code> .
S41	<code>skos:broadMatch</code> is a sub-property of <code>skos:broader</code> , <code>skos:narrowMatch</code> is a sub-property of <code>skos:narrower</code> , and <code>skos:relatedMatch</code> is a sub-property of <code>skos:related</code> .
S42	<code>skos:exactMatch</code> is a sub-property of <code>skos:closeMatch</code> .
S43	<code>skos:narrowMatch</code> is the <code>owl:inverseOf</code> the property <code>skos:broadMatch</code> .
S44	<code>skos:relatedMatch</code> is an instance of <code>owl:SymmetricProperty</code> .
S45	<code>skos:closeMatch</code> and <code>skos:exactMatch</code> are each instances of <code>owl:SymmetricProperty</code> . 
S46	<code>skos:exactMatch</code> is an instance of <code>owl:TransitiveProperty</code> .

10.4. Integrity Conditions

S47	<code>skos:exactMatch</code> is disjoint with each of the properties <code>skos:broadMatch</code> and <code>skos:relatedMatch</code> .
-----	--

Note that as `skos:exactMatch` is a symmetric relation, and `skos:broadMatch` and `skos:narrowMatch` are inverses, `skos:exactMatch` can be inferred to be disjoint with `skos:narrowMatch`.

10.5. Examples

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

Example 52 (consistent)	
<code><A></code>	<code>skos:exactMatch</code> <code></code> .

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

Example 53 (consistent)

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

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

Example 54 (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 55 (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 56 (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-organise 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.

Example 57 (entailment)

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

entails

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


Example 58 (entailment)

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

entails

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

Example 59 (entailment)

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

entails

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

Example 60 (entailment)

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

entails

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

Note also that, because different people might re-organise 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 61 (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 62 (not consistent)

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

Example 63 (not consistent)

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

Example 64 (not consistent)

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

10.6.3. Transitivity of Mapping Properties

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

Example 65 (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 66 (non-entailment)

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

does not entail

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

Example 67 (non-entailment)

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

does not entail

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

Example 68 (non-entailment)

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

does not entail

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

10.6.4. Reflexivity of Mapping Properties

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 69 (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 alternate 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 70 (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 alternate paths involving `skos:broadMatch`. This graph is **consistent** with the SKOS data model.

Example 71 (consistent)

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

See however [Section 8](#) on cycles and alternate paths involving `skos:broader`.

10.6.6. Property Chains Involving `skos:exactMatch`

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

Example 72 (non-entailment)

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

does not entail

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

Example 73 (non-entailment)

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

does not entail

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

Example 74 (non-entailment)

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

does not entail

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

Example 75 (non-entailment)

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

does not entail

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

At Risk of Change: The relationships between the properties `skos:exactMatch`, `skos:broadMatch` and `skos:narrowMatch` may be changed in future versions of this specification. In particular, property chain axioms may be introduced.

10.6.7. `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 76 (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 `` now have two preferred lexical labels in the same language. This will not always be the case, however.

11. References

[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.

[SW]

W3C Semantic Web Activity. Available at <http://www.w3.org/2001/sw/>

[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/>

[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/>

[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/>

[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/>

[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://lcsh.info/>

[AGROVOC]

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

[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-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/>

[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/>

[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/>

[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/>

[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/>

[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/>

[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/>

[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/>

[BCP47]

[Tags for Identifying Languages](#), P. V. Biron, A. Malhotra, Editors, W3C Recommendation, 28 October 2004, <http://www.rfc-editor.org/rfc/bcp/bcp47.txt>.

[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/>

[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/>

[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/>

[ISO2788]

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

Appendix A. SKOS eXtension for Labels (XL)

This appendix defines an **optional** extension to the Simple Knowledge Organization System, called the SKOS eXtension for Labels (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.

A.1. XL Namespace and Vocabulary

The XL namespace URI is:

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

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

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

Table 2. The XL Vocabulary

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

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

A.2. The skosxl:Label Class

A.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.

A.2.2. Class and Property Definitions

S48	<code>skosxl:Label</code> is an instance of <code>owl:Class</code> .
S49	<code>skosxl:Label</code> is disjoint with each of <code>skos:Concept</code> , <code>skos:ConceptScheme</code> and <code>skos:Collection</code> .
S50	<code>skosxl:literalForm</code> is an instance of <code>owl:DatatypeProperty</code> .
S51	The <code>rdfs:domain</code> of <code>skosxl:literalForm</code> is the class <code>skosxl:Label</code> .
S52	The <code>rdfs:range</code> of <code>skosxl:literalForm</code> is the class of RDF plain literals.
S53	<code>skosxl:Label</code> is a sub-class of a restriction on <code>skosxl:literalForm</code> cardinality exactly 1.

A.2.3. Examples

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

Example 77 (consistent)
<pre><A> rdf:type skosxl:Label ; skosxl:literalForm "love"@en .</pre>

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 78 (not consistent)
<pre> rdf:type skosxl:Label ; skosxl:literalForm "love" ; skosxl:literalForm "adoration" .</pre>

Example 79 (not consistent)
<pre> rdf:type skosxl:Label ; skosxl:literalForm "love"@en ; skosxl:literalForm "love"@fr .</pre>

Example 80 (not consistent)
<pre> rdf:type skosxl:Label ; skosxl:literalForm "love"@en-GB ; skosxl:literalForm "love"@en-US .</pre>

Example 81 (not consistent)
<pre> rdf:type skosxl:Label ; skosxl:literalForm "東"@ja-Hani ; skosxl:literalForm "ひがし"@ja-Hira .</pre>

A.2.4. Notes

A.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 82 (non-entailment)
<pre><A> skosxl:literalForm "love"@en . skosxl:literalForm "love"@en . does not entail <A> owl:sameAs .</pre>

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`.

A.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 83 (consistent)

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

A.3. Preferred, Alternate and Hidden `skosxl:Labels`

A.3.1. Preamble

The three properties `skosxl:prefLabel`, `skosxl:altLabel` and `skosxl:hiddenLabel` are used to give the preferred, alternate 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.

A.3.2. Class and Property Definitions

S54	<code>skosxl:prefLabel</code> , <code>skosxl:altLabel</code> and <code>skosxl:hiddenLabel</code> are each instances of <code>owl:ObjectProperty</code> .
S55	The <code>rdfs:range</code> of each of <code>skosxl:prefLabel</code> , <code>skosxl:altLabel</code> and <code>skosxl:hiddenLabel</code> is the class <code>skosxl:Label</code> .
S56	The property chain (<code>skosxl:prefLabel</code> , <code>skosxl:literalForm</code>) is a sub-property of <code>skos:prefLabel</code> .
S57	The property chain (<code>skosxl:altLabel</code> , <code>skosxl:literalForm</code>) is a sub-property of <code>skos:altLabel</code> .
S58	The property chain (<code>skosxl:hiddenLabel</code> , <code>skosxl:literalForm</code>) is a sub-property of <code>skos:hiddenLabel</code> .

A.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 84 (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 .
```

A.3.4. Notes

A.3.4.1. "Dumbing-Down" to SKOS Lexical Labels

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

Example 85 (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 .
```

A.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 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 86 (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 87 (not consistent)

```
# "Clash" between preferred and alternate labels

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

Example 88 (not consistent)

```
# "Clash" between alternate and hidden labels

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

Example 89 (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 .
```

A.4. Links Between skosxl:Labels

A.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.

A.4.2. Class and Property Definitions

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

A.4.3. Examples

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

Example 90 (consistent)

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

A.4.4. Notes

A.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.

For 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 91 (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.

Appendix B. SKOS Overview

B.1. Classes in the SKOS Data Model

<u>skos:Collection</u>	
URI:	http://www.w3.org/2008/05/skos#Collection
Definition:	Section 9. Concept Collections
Disjoint classes:	skos:Concept skos:ConceptScheme
<u>skos:Concept</u>	
URI:	http://www.w3.org/2008/05/skos#Concept
Definition:	Section 3. The skos:Concept Class
Disjoint classes:	skos:Collection skos:ConceptScheme
<u>skos:ConceptScheme</u>	
URI:	http://www.w3.org/2008/05/skos#ConceptScheme
Definition:	Section 4. Concept Schemes
Disjoint classes:	skos:Collection skos:Concept
<u>skos:OrderedCollection</u>	
URI:	http://www.w3.org/2008/05/skos#OrderedCollection
Definition:	Section 9. Concept Collections
Super-classes:	skos:Collection

B.2. Properties in the SKOS Data Model

<u>skos:altLabel</u>	
URI:	http://www.w3.org/2008/05/skos#altLabel
Definition:	Section 5. Lexical Labels
Super-properties:	http://www.w3.org/2000/01/rdf-schema#label
<u>skos:broadMatch</u>	
URI:	http://www.w3.org/2008/05/skos#broadMatch
Definition:	Section 10. Mapping Properties
Super-properties:	skos:broader skos:mappingRelation
Inverse of:	skos:narrowMatch
<u>skos:broader</u>	
URI:	http://www.w3.org/2008/05/skos#broader
Definition:	Section 8. Semantic Relations
Super-properties:	skos:broaderTransitive
Inverse of:	skos:narrower
<u>skos:broaderTransitive</u>	
URI:	http://www.w3.org/2008/05/skos#broaderTransitive
Definition:	Section 8. Semantic Relations
Super-properties:	skos:semanticRelation
Inverse of:	skos:narrowerTransitive
Other characteristics:	Transitive
<u>skos:changeNote</u>	
URI:	http://www.w3.org/2008/05/skos#changeNote
Definition:	Section 7. Documentation Properties
Super-properties:	skos:note

<u>skos:closeMatch</u>	
URI:	http://www.w3.org/2008/05/skos#closeMatch
Definition:	Section 10. Mapping Properties
Super-properties:	skos:mappingRelation
Other characteristics:	Symmetric
<u>skos:definition</u>	
URI:	http://www.w3.org/2008/05/skos#definition
Definition:	Section 7. Documentation Properties
Super-properties:	skos:note
<u>skos:editorialNote</u>	
URI:	http://www.w3.org/2008/05/skos#editorialNote
Definition:	Section 7. Documentation Properties
Super-properties:	skos:note
<u>skos:exactMatch</u>	
URI:	http://www.w3.org/2008/05/skos#exactMatch
Definition:	Section 10. Mapping Properties
Super-properties:	skos:closeMatch
Other characteristics:	Transitive Symmetric
<u>skos:example</u>	
URI:	http://www.w3.org/2008/05/skos#example
Definition:	Section 7. Documentation Properties
Super-properties:	skos:note
<u>skos:hasTopConcept</u>	
URI:	http://www.w3.org/2008/05/skos#hasTopConcept
Definition:	Section 4. Concept Schemes
Domain:	skos:ConceptScheme
Range:	skos:Concept
Inverse of:	skos:topConceptOf
<u>skos:hiddenLabel</u>	
URI:	http://www.w3.org/2008/05/skos#hiddenLabel
Definition:	Section 5. Lexical Labels
Super-properties:	http://www.w3.org/2000/01/rdf-schema#label
<u>skos:historyNote</u>	
URI:	http://www.w3.org/2008/05/skos#historyNote
Definition:	Section 7. Documentation Properties
Super-properties:	skos:note
<u>skos:inScheme</u>	
URI:	http://www.w3.org/2008/05/skos#inScheme
Definition:	Section 4. Concept Schemes
Range:	skos:ConceptScheme
<u>skos:mappingRelation</u>	
URI:	http://www.w3.org/2008/05/skos#mappingRelation

Definition:	Section 10. Mapping Properties
Domain:	skos:Concept
Range:	skos:Concept
skos:member	
URI:	http://www.w3.org/2008/05/skos#member
Definition:	Section 9. Concept Collections
Domain:	skos:Collection
skos:memberList	
URI:	http://www.w3.org/2008/05/skos#memberList
Definition:	Section 9. Concept Collections
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/2008/05/skos#narrowMatch
Definition:	Section 10. Mapping Properties
Super-properties:	skos:mappingRelation skos:narrower
Inverse of:	skos:broadMatch
skos:narrower	
URI:	http://www.w3.org/2008/05/skos#narrower
Definition:	Section 8. Semantic Relations
Super-properties:	skos:narrowerTransitive
Inverse of:	skos:broader
skos:narrowerTransitive	
URI:	http://www.w3.org/2008/05/skos#narrowerTransitive
Definition:	Section 8. Semantic Relations
Super-properties:	skos:semanticRelation
Inverse of:	skos:broaderTransitive
Other characteristics:	Transitive
skos:notation	
URI:	http://www.w3.org/2008/05/skos#notation
Definition:	Section 6. Notations
skos:note	
URI:	http://www.w3.org/2008/05/skos#note
Definition:	Section 7. Documentation Properties
skos:prefLabel	
URI:	http://www.w3.org/2008/05/skos#prefLabel
Definition:	Section 5. Lexical Labels
Super-properties:	http://www.w3.org/2000/01/rdf-schema#label
skos:related	
URI:	http://www.w3.org/2008/05/skos#related
Definition:	Section 8. Semantic Relations

Super-properties:	<code>skos:semanticRelation</code>
Other characteristics:	Symmetric
<code>skos:relatedMatch</code>	
URI:	http://www.w3.org/2008/05/skos#relatedMatch
Definition:	Section 10. Mapping Properties
Super-properties:	<code>skos:mappingRelation</code> <code>skos:related</code>
Other characteristics:	Symmetric
<code>skos:scopeNote</code>	
URI:	http://www.w3.org/2008/05/skos#scopeNote
Definition:	Section 7. Documentation Properties
Super-properties:	<code>skos:note</code>
<code>skos:semanticRelation</code>	
URI:	http://www.w3.org/2008/05/skos#semanticRelation
Definition:	Section 8. Semantic Relations
Domain:	<code>skos:Concept</code>
Range:	<code>skos:Concept</code>
<code>skos:topConceptOf</code>	
URI:	http://www.w3.org/2008/05/skos#topConceptOf
Definition:	Section 4. Concept Schemes
Super-properties:	<code>skos:inScheme</code>
Inverse of:	<code>skos:hasTopConcept</code>

Appendix C. SKOS Data Model as RDF Triples

The SKOS Data Model as RDF Triples can be found at <http://www.w3.org/2008/05/skos>.

Note that it is not possible to express all of the statements of the SKOS data model as RDF triples.

The SKOS eXtension for Labels (XL) Data Model as RDF Triples can be found at <http://www.w3.org/2008/05/skos-xl>.

Note also that it is not possible to express all of the statements of the XL data model as RDF triples.
