

RDF Primer

W3C Recommendation 10 February 2004

This version:
http://www.w3.org/TR/2004/REC-rdf-primer-20040210/
Latest version:
http://www.w3.org/TR/rdf-primer/
Previous version:
http://www.w3.org/TR/2003/PR-rdf-primer-20031215/
Editors:
Frank Manola, fmanola@acm.org
Eric Miller, W3C, em@w3.org
Series Editor:
Brian McBride, Hewlett-Packard Laboratories, bwm@hplb.hpl.hp.com

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

See also translations.

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

Abstract

The Resource Description Framework (RDF) is a language for representing information about resources in the World Wide Web. This Primer is designed to provide the reader with the basic knowledge required to effectively use RDF. It introduces the basic concepts of RDF and describes its XML syntax. It describes how to define RDF vocabularies using the RDF Vocabulary Description Language, and gives an overview of some deployed RDF applications. It also describes the content and purpose of other RDF specification documents.

Status of this Document

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

This is one document in a set of six (Primer, Concepts, Syntax, Semantics, Vocabulary, and Test Cases) intended to jointly replace the original Resource Description Framework specifications, RDF Model and Syntax (1999 Recommendation) and RDF Schema (2000 Candidate Recommendation). It has been developed by the RDF Core Working Group as part of the W3C Semantic Web Activity (Activity Statement, Group Charter) for publication on 10 February 2004.

Changes to this document since the Proposed Recommendation Working Draft are detailed in the change log.

The public is invited to send comments to www-rdf-comments@w3.org (archive) and to participate in general discussion of related technology on www-rdf-interest@w3.org (archive).

A list of implementations is available.

The W3C maintains a list of any patent disclosures related to this work.

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at

http://www.w3.org/TR/.

Table of Contents

1. Introduction
2. Making Statements About Resources
   2.1 Basic Concepts
   2.2 The RDF Model
   2.3 Structured Property Values and Blank Nodes
   2.4 Typed Literals
   2.5 Concepts Summary
3. An XML Syntax for RDF: RDF/XML
   3.1 Basic Principles
   3.2 Abbreviating and Organizing RDF URIrefs
   3.3 RDF/XML Summary
4. Other RDF Capabilities
   4.1 RDF Containers
   4.2 RDF Collections
   4.3 RDF Reification
   4.4 More on Structured Values: rdf:value
   4.5 XML Literals
5. Defining RDF Vocabularies: RDF Schema
   5.1 Describing Classes
   5.2 Describing Properties
   5.3 Interpreting RDF Schema Declarations
   5.4 Other Schema Information
   5.5 Richer Schema Languages
6. Some RDF Applications: RDF in the Field
   6.1 Dublin Core Metadata Initiative
   6.2 PRISM
   6.3 XPackage
   6.4 RSS 1.0: RDF Site Summary
   6.5 CIM/XML
   6.6 Gene Ontology Consortium
   6.7 Describing Device Capabilities and User Preferences
7. Other Parts of the RDF Specification
   7.1 RDF Semantics
   7.2 Test Cases
8. References
   8.1 Normative References
   8.2 Informational References
9. Acknowledgments

Appendices

A. More on Uniform Resource Identifiers (URIs)
B. More on the Extensible Markup Language (XML)
C. Changes

---

1. Introduction

The Resource Description Framework (RDF) is a language for representing information about resources in the World Wide Web. It is particularly intended for representing metadata about Web resources, such as the title, author, and modification date of a Web page, copyright and licensing information about a Web document, or the availability schedule for some shared resource. However, by generalizing the concept of a "Web resource", RDF can also be used to represent information about things that can be *identified* on the Web, even when they cannot be directly *retrieved* on the Web. Examples include information about items available from on-line shopping facilities (e.g., information about specifications, prices, and availability), or the description of a Web user's preferences for information delivery.

RDF is intended for situations in which this information needs to be processed by applications, rather than being only displayed to people. RDF provides a common framework for expressing this information so it can be exchanged between applications without loss of meaning. Since it is a common framework, application designers can leverage the availability of common RDF parsers and processing tools. The ability to exchange information between different applications means that the information may be made available to applications other than those for which it was originally created.

RDF is based on the idea of identifying things using Web identifiers (called *Uniform Resource Identifiers*, or *URIs*), and describing resources in terms of simple properties and property values. This enables RDF to represent simple statements about resources as a *graph* of nodes and arcs representing the resources, and their properties and values. To make this discussion somewhat more concrete as soon as possible, the group of statements "there is a Person identified by `http://www.w3.org/People/EM/contact#me`, whose name is Eric Miller, whose email address is em@w3.org, and whose title is Dr." could be represented as the RDF graph in Figure 1:



Figure 1: An RDF Graph Describing Eric Miller

Figure 1 illustrates that RDF uses URIs to identify:

- individuals, e.g., Eric Miller, identified by `http://www.w3.org/People/EM/contact#me`
- kinds of things, e.g., Person, identified by `http://www.w3.org/2000/10/swap/pim/contact#Person`
- properties of those things, e.g., mailbox, identified by `http://www.w3.org/2000/10/swap/pim/contact#mailbox`
- values of those properties, e.g. `mailto:em@w3.org` as the value of the mailbox property (RDF also uses character strings such as "Eric Miller", and values from other datatypes such as integers and dates, as the values of properties)

RDF also provides an XML-based syntax (called *RDF/XML*) for recording and exchanging these graphs. Example 1 is a small chunk of RDF in RDF/XML corresponding to the graph in Figure 1:

Example 1: RDF/XML Describing Eric Miller

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
             xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#">

  <contact:Person rdf:about="http://www.w3.org/People/EM/contact#me">
    <contact:fullName>Eric Miller</contact:fullName>
    <contact:mailbox rdf:resource="mailto:em@w3.org"/>
    <contact:personalTitle>Dr.</contact:personalTitle>
  </contact:Person>

</rdf:RDF>
```

Note that this RDF/XML also contains URIs, as well as properties like `mailbox` and `fullName` (in an abbreviated form), and their respective values `em@w3.org`, and `Eric Miller`.

Like HTML, this RDF/XML is machine processable and, using URIs, can link pieces of information across the Web. However, unlike conventional hypertext, RDF URIs can refer to any identifiable thing, including things that may not be directly retrievable on the Web (such as the person Eric Miller). The result is that in addition to describing such things as Web pages, RDF can also describe cars, businesses, people, news events, etc. In addition, RDF properties themselves have URIs, to precisely identify the relationships that exist between the linked items.

The following documents contribute to the specification of RDF:

- RDF Concepts and Abstract Syntax [RDF-CONCEPTS]
- RDF/XML Syntax Specification [RDF-SYNTAX]
- RDF Vocabulary Description Language 1.0: RDF Schema [RDF-VOCABULARY]
- RDF Semantics [RDF-SEMANTICS]
- RDF Test Cases [RDF-TESTS]
- RDF Primer (this document)

This Primer is intended to provide an introduction to RDF and describe some existing RDF applications, to help information system designers and application developers understand the features of RDF and how to use them. In particular, the Primer is intended to answer such questions as:

- What does RDF look like?
- What information can RDF represent?
- How is RDF information created, accessed, and processed?
- How can existing information be combined with RDF?

The Primer is a *non-normative* document, which means that it does not provide a definitive specification of RDF. The examples and other explanatory material in the Primer are provided to help readers understand RDF, but they may not always provide definitive or fully-complete answers. In such cases, the relevant normative parts of the RDF specification should be consulted. To help in doing this, the Primer describes the roles these other documents play in the complete specification of RDF, and provides links pointing to the relevant parts of the normative specifications, at appropriate places in the discussion.

It should also be noted that these RDF documents update and clarify previously-published RDF specifications, the Resource Description Framework (RDF) Model and Syntax Specification [RDF-MS] and the Resource Description Framework (RDF) Schema Specification 1.0 [RDF-S]. As a result, there have been some changes in terminology, syntax, and concepts. This Primer reflects the newer set of RDF specifications given in the bulleted list of RDF documents cited above. Hence, readers familiar with the older specifications, and with earlier tutorial and introductory articles based on them, should be aware that there may be differences between the current specifications and those previous documents. The RDF Issue Tracking document [RDFISSUE] can be consulted for a list of issues raised concerning the previous RDF specifications, and their resolution in the current specifications.

2. Making Statements About Resources

RDF is intended to provide a simple way to make statements about Web resources, e.g., Web pages. This section describes the basic ideas behind the way RDF provides these capabilities (the normative specification describing these concepts is RDF Concepts and Abstract Syntax [RDF-CONCEPTS]).

2.1 Basic Concepts

Imagine trying to state that someone named John Smith created a particular Web page. A straightforward way to state this in a natural language such as English would be in the form of a simple statement such as:

```
http://www.example.org/index.html has a creator whose value is John Smith
```

Parts of this statement are emphasized to illustrate that, in order to describe the properties of something, there need to be ways to name, or identify, a number of things:

- the thing the statement describes (the Web page, in this case)
- a specific property (creator, in this case) of the thing the statement describes
- the thing the statement says is the value of this property (who the creator is), for the thing the statement describes

In this statement, the Web page's URL (Uniform Resource Locator) is used to identify it. In addition, the word "creator" is used to identify the property, and the two words "John Smith" to identify the thing (a person) that is the value of this property.

Other properties of this Web page could be described by writing additional English statements of the same general form, using the URL to identify the page, and words (or other expressions) to identify the properties and their values. For example, the date the page was created, and the language in which the page is written, could be described using the additional statements:

```
http://www.example.org/index.html has a creation-date whose value is August 16, 1999
http://www.example.org/index.html has a language whose value is English
```

RDF is based on the idea that the things being described have properties which have values, and that resources can be described by making statements, similar to those above, that specify those properties and values. RDF uses a particular terminology for talking about the various parts of statements. Specifically, the part that identifies the thing the statement is about (the Web page in this example) is called the subject. The part that identifies the property or characteristic of the subject that the statement specifies (creator, creation-date, or language in these examples) is called the predicate, and the part that identifies the value of that property is called the object. So, taking the English statement

```
http://www.example.org/index.html has a creator whose value is John Smith
```

the RDF terms for the various parts of the statement are:

- the *subject* is the URL `http://www.example.org/index.html`
- the *predicate* is the word "creator"
- the *object* is the phrase "John Smith"

However, while English is good for communicating between (English-speaking) humans, RDF is about making *machine-processable* statements. To make these kinds of statements suitable for processing by machines, two things are needed:

- a system of machine-processable identifiers for identifying a subject, predicate, or object in a statement without any possibility of confusion with a similar-looking identifier that might be used by someone else on the Web.
- a machine-processable language for representing these statements and exchanging them between machines.

Fortunately, the existing Web architecture provides both these necessary facilities.

As illustrated earlier, the Web already provides one form of identifier, the *Uniform Resource Locator* (URL). A URL was used in the original example to identify the Web page that John Smith created. A URL is a character string that identifies a Web resource by representing its primary access mechanism (essentially, its network "location"). However, it is also important to be able to record information about many things that, unlike Web pages, do not have network locations or URLs.

The Web provides a more general form of identifier for these purposes, called the Uniform Resource Identifier (URI). URLs are a particular kind of URI. All URIs share the property that different persons or organizations can independently create them, and use them to identify things. However, URIs are not limited to identifying things that have network locations, or use other computer access mechanisms. In fact, a URI can be created to refer to anything that needs to be referred to in a statement, including

- network-accessible things, such as an electronic document, an image, a service (e.g., "today's weather report for Los Angeles"), or a group of other resources.
- things that are not network-accessible, such as human beings, corporations, and bound books in a library.
- abstract concepts that do not physically exist, such as the concept of a "creator".

Because of this generality, RDF uses URIs as the basis of its mechanism for identifying the subjects, predicates, and objects in statements. To be more precise, RDF uses URI references [URIS]. A URI reference (or *URIref*) is a URI, together with an optional *fragment identifier* at the end. For example, the URI reference `http://www.example.org/index.html#section2` consists of the URI `http://www.example.org/index.html` and (separated by the "#" character) the fragment identifier `Section2`. RDF URIrefs can contain Unicode [UNICODE] characters (see [RDF-CONCEPTS]), allowing many languages to be reflected in URIrefs. RDF defines a *resource* as anything that is identifiable by a URI reference, so using URIrefs allows RDF to describe practically anything, and to state relationships between such things as well. URIrefs and fragment identifiers are discussed further in Appendix A, and in [RDF-CONCEPTS].

To represent RDF statements in a machine-processable way, RDF uses the Extensible Markup Language [XML]. XML was designed to allow anyone to design their own document format and then write a document in that format. RDF defines a specific XML markup language, referred to as *RDF/XML*, for use in representing RDF information, and for exchanging it between machines. An example of RDF/XML was given in Section 1. That example (Example 1) used tags such as `<contact:fullName>` and `<contact:personalTitle>` to delimit the text content `Eric Miller` and `Dr.`, respectively. Such tags allow programs written with an understanding of what the tags mean to properly interpret that content. Both XML content and (with certain exceptions) tags can contain Unicode [UNICODE] characters, allowing information from many languages to be directly represented. Appendix B provides further background on XML in general. The specific RDF/XML syntax used for RDF is described in more detail in Section 3, and is normatively defined in [RDF-SYNTAX]

2.2 The RDF Model

Section 2.1 has introduced RDF's basic statement concepts, the idea of using URI references to identify the things referred to in RDF statements, and RDF/XML as a machine-processable way to represent RDF statements. With that background, this section describes how RDF uses URIs to make statements about resources. The introduction said that RDF was based on the idea of expressing simple statements about resources, where each statement consists of a subject, a predicate, and an object. In RDF, the English statement:

```
http://www.example.org/index.html has a creator whose value is John Smith
```

could be represented by an RDF statement having:

- a subject `http://www.example.org/index.html`
- a predicate `http://purl.org/dc/elements/1.1/creator`
- and an object `http://www.example.org/staffid/85740`

Note how URIrefs are used to identify not only the subject of the original statement, but also the predicate and object, instead of using the words "creator" and "John Smith", respectively (some of the effects of using URIrefs in this way will be discussed later in this section).

RDF models statements as nodes and arcs in a graph. RDF's graph model is defined in [RDF-CONCEPTS]. In this notation, a statement is represented by:

- a node for the subject
- a node for the object
- an arc for the predicate, directed from the subject node to the object node.

So the RDF statement above would be represented by the graph shown in Figure 2:



Figure 2: A Simple RDF Statement

Groups of statements are represented by corresponding groups of nodes and arcs. So, to reflect the additional English statements

```
http://www.example.org/index.html has a creation-date whose value is August 16, 1999
http://www.example.org/index.html has a language whose value is English
```

in the RDF graph, the graph shown in Figure 3 could be used (using suitable URIrefs to name the properties "creation-date" and "language"):



Figure 3: Several Statements About the Same Resource

Figure 3 illustrates that objects in RDF statements may be either URIrefs, or constant values (called literals) represented by character strings, in order to represent certain kinds of property values. (In the case of the predicate `http://purl.org/dc/elements/1.1/language` the literal is an international standard two-letter code for English.) Literals may not be used as subjects or predicates in RDF statements. In drawing RDF graphs, nodes that are URIrefs are shown as ellipses, while nodes that are literals are shown as boxes. (The simple character string literals used in these examples are called plain literals, to distinguish them from the typed literals to be introduced in Section 2.4. The various kinds of literals that can be used in RDF statements are defined in [RDF-CONCEPTS]. Both plain and typed literals can contain Unicode [UNICODE] characters, allowing information from many languages to be directly represented.)

Sometimes it is not convenient to draw graphs when discussing them, so an alternative way of writing down the statements, called triples, is also used. In the triples notation, each statement in the graph is written as a simple triple of subject, predicate, and object, in that order. For example, the three statements shown in Figure 3 would be written in the triples notation as:

```
<http://www.example.org/index.html> <http://purl.org/dc/elements/1.1/creator>
<http://www.example.org/staffid/85740> .

<http://www.example.org/index.html> <http://www.example.org/terms/creation-date>
"August 16, 1999" .

<http://www.example.org/index.html> <http://purl.org/dc/elements/1.1/language> "en" .
```

Each triple corresponds to a single arc in the graph, complete with the arc's beginning and ending nodes (the subject and object of the statement). Unlike the drawn graph (but like the original statements), the triples notation requires that a node be separately identified for each statement it appears in. So, for example, `http://www.example.org/index.html` appears three times (once in each triple) in the triples representation of the graph, but only once in the drawn graph. However, the triples represent exactly the same information as the drawn graph, and this is a key point: what is fundamental to RDF is the *graph model* of the statements. The

notation used to represent or depict the graph is secondary.

The full triples notation requires that URI references be written out completely, in angle brackets, which, as the example above illustrates, can result in very long lines on a page. For convenience, the Primer uses a shorthand way of writing triples (the same shorthand is also used in other RDF specifications). This shorthand substitutes an XML *qualified name* (or *QName*) without angle brackets as an abbreviation for a full URI reference (QNames are discussed further in Appendix B). A QName contains a *prefix* that has been assigned to a namespace URI, followed by a colon, and then a *local name*. The full URIref is formed from the QName by appending the local name to the namespace URI assigned to the prefix. So, for example, if the QName prefix `foo` is assigned to the namespace URI `http://example.org/somewhere/`, then the QName `foo:bar` is shorthand for the URIref `http://example.org/somewhere/bar`. Primer examples will also use several "well-known" QName prefixes (without explicitly specifying them each time), defined as follows:

prefix `rdf:`, namespace URI: `http://www.w3.org/1999/02/22-rdf-syntax-ns#`
prefix `rdfs:`, namespace URI: `http://www.w3.org/2000/01/rdf-schema#`
prefix `dc:`, namespace URI: `http://purl.org/dc/elements/1.1/`
prefix `owl:`, namespace URI: `http://www.w3.org/2002/07/owl#`
prefix `ex:`, namespace URI: `http://www.example.org/` (or `http://www.example.com/`)
prefix `xsd:`, namespace URI: `http://www.w3.org/2001/XMLSchema#`

Obvious variations on the "example" prefix `ex:` will also be used as needed in the examples, for instance,

prefix `exterms:`, namespace URI: `http://www.example.org/terms/` (for terms used by an example organization),
prefix `exstaff:`, namespace URI: `http://www.example.org/staffid/` (for the example organization's staff identifiers),
prefix `ex2:`, namespace URI: `http://www.domain2.example.org/` (for a second example organization), and so on.

Using this new shorthand, the previous set of triples can be written as:

```
ex:index.html  dc:creator             exstaff:85740 .

ex:index.html  exterms:creation-date  "August 16, 1999" .

ex:index.html  dc:language            "en" .
```

Since RDF uses URIrefs instead of words to name things in statements, RDF refers to a set of URIrefs (particularly a set intended for a specific purpose) as a *vocabulary*. Often, the URIrefs in such vocabularies are organized so that they can be represented as a set of QNames using a common prefix. That is, a common namespace URIref will be chosen for all terms in a vocabulary, typically a URIref under the control of whoever is defining the vocabulary. URIrefs that are contained in the vocabulary are formed by appending individual local names to the end of the common URIref. This forms a set of URIrefs with a common prefix. For instance, as illustrated by the previous examples, an organization such as example.org might define a vocabulary consisting of URIrefs starting with the prefix `http://www.example.org/terms/` for terms it uses in its business, such as "creation-date" or "product", and another vocabulary of URIrefs starting with `http://www.example.org/staffid/` to identify its employees. RDF uses this same approach to define its own vocabulary of terms with special meanings in RDF. The URIrefs in this RDF vocabulary all begin with `http://www.w3.org/1999/02/22-rdf-syntax-ns#`, conventionally associated with the QName prefix `rdf:`. The RDF Vocabulary Description Language (described in Section 5) defines an additional set of terms having URIrefs that begin with `http://www.w3.org/2000/01/rdf-schema#`, conventionally associated with the QName prefix `rdfs:`. (Where a specific QName prefix is commonly used in connection with a given set of terms in this way, the QName prefix itself is sometimes used as the name of the vocabulary. For example, someone might refer to "the `rdfs:` vocabulary".)

Using common URI prefixes provides a convenient way to organize the URIrefs for a related set of terms. However, this is just a convention. The RDF model only recognizes full URIrefs; it does not "look inside" URIrefs or use any knowledge about their structure. In particular, RDF does not assume there is any relationship between URIrefs just because they have a common leading prefix (see Appendix A for further discussion). Moreover, there is nothing that says that URIrefs with different leading prefixes cannot be considered part of the same vocabulary. A particular organization, process, tool, etc. can define a vocabulary that is significant for it, using URIrefs from any number of other vocabularies as part of its vocabulary.

In addition, sometimes an organization will use a vocabulary's namespace URIref as the URL of a Web resource that provides further information about that vocabulary. For example, as noted earlier, the QName prefix `dc:` will be used in Primer examples, associated with the namespace URIref `http://purl.org/dc/elements/1.1/`. In fact, this refers to the Dublin Core vocabulary described in Section 6.1. Accessing this namespace URIref in a Web browser will retrieve additional information about the Dublin Core vocabulary (specifically, an RDF schema). However, this is also just a convention. RDF does not assume that a namespace URI identifies a retrievable Web resource (see Appendix B for further discussion).

In the rest of the Primer, the term *vocabulary* will be used when referring to a set of URIrefs defined for some specific purpose, such as the set of URIrefs defined by RDF for its own use, or the set of URIrefs defined by example.org to identify its employees. The term *namespace* will be used only when referring specifically to the syntactic concept of an XML namespace (or in describing the URI

assigned to a prefix in a QName).

URIrefs from different vocabularies can be freely mixed in RDF graphs. For example, the graph in Figure 3 uses URIrefs from the `exterms:`, `exstaff:`, and `dc:` vocabularies. Also, RDF imposes no restrictions on how many statements using a given URIref as predicate can appear in a graph to describe the same resource. For example, if the resource `ex:index.html` had been created by the cooperative efforts of several staff members in addition to John Smith, example.org might have written the statements:

```
ex:index.html  dc:creator  exstaff:85740 .

ex:index.html  dc:creator  exstaff:27354 .

ex:index.html  dc:creator  exstaff:00816 .
```

These examples of RDF statements begin to illustrate some of the advantages of using URIrefs as RDF's basic way of identifying things. For instance, in the first statement, instead of identifying the creator of the Web page by the character string "John Smith", he has been assigned a URIref, in this case (using a URIref based on his employee number) `http://www.example.org/staffid/85740` . An advantage of using a URIref in this case is that the identification of the statement's subject can be more precise. That is, the creator of the page is not the character string "John Smith", or any one of the thousands of people named John Smith, but the particular John Smith associated with that URIref (whoever created the URIref defines the association). Moreover, since there is a URIref to refer to John Smith, he is a full-fledged resource, and additional information can be recorded about him, simply by adding additional RDF statements with John's URIref as the subject. For example, Figure 4 shows some additional statements giving John's name and age.



Figure 4: More Information About John Smith

These examples also illustrate that RDF uses URIrefs as *predicates* in RDF statements. That is, rather than using character strings (or words) such as "creator" or "name" to identify properties, RDF uses URIrefs. Using URIrefs to identify properties is important for a number of reasons. First, it distinguishes the properties one person may use from different properties someone else may use that would otherwise be identified by the same character string. For instance, in the example in Figure 4, example.org uses "name" to mean someone's full name written out as a character string literal (e.g., "John Smith"), but someone else may intend "name" to mean something different (e.g., the name of a variable in a piece of program text). A program encountering "name" as a property identifier on the Web (or merging data from multiple sources) would not necessarily be able to distinguish these uses. However, if example.org writes `http://www.example.org/terms/name` for its "name" property, and the other person writes `http://www.domain2.example.org/genealogy/terms/name` for hers, it is clear that there are distinct properties involved (even if a program cannot automatically determine the distinct meanings). Also, using URIrefs to identify properties enables the properties to be treated as resources themselves. Since properties are resources, additional information can be recorded about them (e.g., the English description of what example.org means by "name"), simply by adding additional RDF statements with the property's URIref as the subject.

Using URIrefs as subjects, predicates, and objects in RDF statements supports the development and use of shared vocabularies on the Web, since people can discover and begin using vocabularies already used by others to describe things, reflecting a shared understanding of those concepts. For example, in the triple

```
ex:index.html   dc:creator   exstaff:85740 .
```

the predicate `dc:creator`, when fully expanded as a URIref, is an unambiguous reference to the "creator" attribute in the Dublin Core metadata attribute set (discussed further in Section 6.1), a widely-used set of attributes (properties) for describing information of all kinds. The writer of this triple is effectively saying that the relationship between the Web page (identified by `http://www.example.org/index.html`) and the creator of the page (a distinct person, identified by `http://www.example.org/staffid/85740`) is exactly the concept identified by `http://purl.org/dc/elements/1.1/creator`. Another person familiar with the Dublin Core vocabulary, or who finds out what `dc:creator` means (say by looking up its definition on the Web) will know what is meant by this relationship. In addition, based on this understanding, people can write programs to behave in accordance with that meaning when processing triples containing the predicate `dc:creator`.

Of course, this depends on increasing the general use of URIrefs to refer to things instead of using literals; e.g., using URIrefs like `exstaff:85740` and `dc:creator` instead of character string literals like `John Smith` and `creator`. Even then, RDF's use of URIrefs does not solve all identification problems because, for example, people can still use different URIrefs to refer to the same thing. For this reason, it is a good idea to try to use terms from existing vocabularies (such as the Dublin Core) where possible, rather than making up new terms that might overlap with those of some other vocabulary. Appropriate vocabularies for use in specific application areas are being developed all the time, as illustrated by the applications described in Section 6. However, even when synonyms are created, the fact that these different URIrefs are used in the commonly-accessible "Web space" provides the opportunity both to identify equivalences among these different references, and to migrate toward the use of common references.

In addition, it is important to distinguish between any meaning that *RDF itself* associates with terms (such as `dc:creator` in the previous example) used in RDF statements and additional, *externally-defined* meaning that people (or programs written by those people) might associate with those terms. As a language, RDF directly defines only the graph syntax of subject, predicate, and object triples, certain meanings associated with URIrefs in the `rdf:` vocabulary, and certain other concepts to be described later. These things are normatively defined in [RDF-CONCEPTS] and [RDF-SEMANTICS]. However, RDF does not define the meanings of terms from other vocabularies, such as `dc:creator`, that might be used in RDF statements. Specific vocabularies will be created, with specific meanings assigned to the URIrefs defined in them, externally to RDF. RDF statements using URIrefs from these vocabularies may convey the specific meanings associated with those terms to people familiar with these vocabularies, or to RDF applications written to process these vocabularies, without conveying any of these meanings to an arbitrary RDF application *not* specifically written to process these vocabularies.

For example, people can associate meaning with a triple such as

```
ex:index.html  dc:creator  exstaff:85740 .
```

based on the meaning they associate with the appearance of the word "creator" as part of the URIref `dc:creator`, or based on their understanding of the specific definition of `dc:creator` in the Dublin Core vocabulary. However, as far as an arbitrary RDF application is concerned the triple might as well be something like

```
fy:joefy.iunm  ed:dsfbups  fytubgg:85740 .
```

as far as any built-in meaning is concerned. Similarly, any natural language text describing the meaning of `dc:creator` that might be found on the Web provides no additional meaning that an arbitrary RDF application can directly use.

Of course, URIrefs from a particular vocabulary can be used in RDF statements even though a given application may not be able to associate any special meanings with them. For example, generic RDF software would recognize that the above expression is an RDF statement, that `ed:dsfbups` is the predicate, and so on. It will simply not associate with the triple any special meaning that the vocabulary developer might have associated with a URIref like `ed:dsfbups`. Moreover, based on their understanding of a given vocabulary, people can write RDF applications to behave in accordance with the special meanings assigned to URIrefs from that vocabulary, even though that meaning will not be accessible to RDF applications not written in that way.

The result of all this is that RDF provides a way to make statements that applications can more easily process. An application cannot actually "understand" such statements, as noted already, any more than a database system "understands" terms like "employee" or "salary" in processing a query like `SELECT NAME FROM EMPLOYEE WHERE SALARY > 35000`. However, if an application is appropriately written, it can deal with RDF statements in a way that makes it seem like it does understand them, just as a database system and its applications can do useful work in processing employee and payroll information without understanding "employee" and "payroll". For example, a user could search the Web for all book reviews and create an average rating for each book. Then, the user could put that information back on the Web. Another Web site could take that list of book rating averages and create a "Top Ten Highest Rated Books" page. Here, the availability and use of a shared vocabulary about ratings, and a shared group of URIrefs identifying the books they apply to, allows individuals to build a mutually-understood and increasingly-powerful (as additional contributions are made) "information base" about books on the Web. The same principle applies to the vast amounts of information that people create about thousands of subjects every day on the Web.

RDF statements are similar to a number of other formats for recording information, such as:

- entries in a simple record or catalog listing describing the resource in a data processing system.
- rows in a simple relational database.
- simple assertions in formal logic

and information in these formats can be treated as RDF statements, allowing RDF to be used to integrate data from many sources.

2.3 Structured Property Values and Blank Nodes

Things would be very simple if the only types of information to be recorded about things were obviously in the form of the simple RDF statements illustrated so far. However, most real-world data involves structures that are more complicated than that, at least on the surface. For instance, in the original example, the date the Web page was created is recorded as a single `exterms:creation-date` property, with a plain literal as its value. However, suppose the value of the `exterms:creation-date` property needed to record the month, day, and year as separate pieces of information? Or, in the case of John Smith's personal information, suppose John's address was being described. The whole address could be written out as a plain literal, as in the triple

```
exstaff:85740   exterms:address   "1501 Grant Avenue, Bedford, Massachusetts 01730" .
```

However, suppose John's address needed to be recorded as a *structure* consisting of separate street, city, state, and postal code

values? How would this be done in RDF?

Structured information like this is represented in RDF by considering the aggregate thing to be described (like John Smith's address) as a resource, and then making statements about that new resource. So, in the RDF graph, in order to break up John Smith's address into its component parts, a new node is created to represent the concept of John Smith's address, with a new URIref to identify it, say `http://www.example.org/addressid/85740` (abbreviated as `exaddressid:85740`). RDF statements (additional arcs and nodes) can then be written with that node as the subject, to represent the additional information, producing the graph shown in Figure 5:

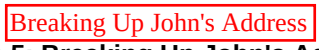


Figure 5: Breaking Up John's Address

or the triples:

```
exstaff:85740       exterms:address        exaddressid:85740 .
exaddressid:85740   exterms:street         "1501 Grant Avenue" .
exaddressid:85740   exterms:city           "Bedford" .
exaddressid:85740   exterms:state          "Massachusetts" .
exaddressid:85740   exterms:postalCode     "01730" .
```

This way of representing structured information in RDF can involve generating numerous "intermediate" URIrefs such as `exaddressid:85740` to represent aggregate concepts such as John's address. Such concepts may never need to be referred to directly from outside a particular graph, and hence may not require "universal" identifiers. In addition, in the *drawing* of the graph representing the group of statements shown in Figure 5, the URIref assigned to identify "John Smith's address" is not really needed, since the graph could just as easily have been drawn as in Figure 6:



Figure 6: Using a Blank Node

Figure 6, which is a perfectly good RDF graph, uses a node without a URIref to stand for the concept of "John Smith's address". This blank node serves its purpose in the drawing without needing a URIref, since the node itself provides the necessary connectivity between the various other parts of the graph. (Blank nodes were called *anonymous resources* in [RDF-MS].) However, some form of explicit identifier for that node is needed in order to represent this graph as triples. To see this, trying to write the triples corresponding to what is shown in Figure 6 would produce something like:

```
exstaff:85740   exterms:address        ??? .
???             exterms:street         "1501 Grant Avenue" .
???             exterms:city           "Bedford" .
???             exterms:state          "Massachusetts" .
???             exterms:postalCode     "01730" .
```

where ??? stands for something that indicates the presence of the blank node. Since a complex graph might contain more than one blank node, there also needs to be a way to differentiate between these different blank nodes in a triples representation of the graph. As a result, triples use blank node identifiers, having the form `_:name`, to indicate the presence of blank nodes. For instance, in this example a blank node identifier `_:johnaddress` might be used to refer to the blank node, in which case the resulting triples might be:

```
exstaff:85740   exterms:address        _:johnaddress .
_:johnaddress   exterms:street         "1501 Grant Avenue" .
_:johnaddress   exterms:city           "Bedford" .
_:johnaddress   exterms:state          "Massachusetts" .
_:johnaddress   exterms:postalCode     "01730" .
```

In a triples representation of a graph, each distinct blank node in the graph is given a different blank node identifier. Unlike URIrefs and literals, blank node identifiers are not considered to be actual parts of the RDF graph (this can be seen by looking at the drawn graph in Figure 6 and noting that the blank node has no blank node identifier). Blank node identifiers are just a way of representing the blank nodes in a graph (and distinguishing one blank node from another) when the graph is written in triple form. Blank node identifiers also have significance only within the triples representing a *single* graph (two different graphs with the same number of blank nodes might independently use the same blank node identifiers to distinguish them, and it would be incorrect to assume that blank nodes from different graphs having the same blank node identifiers are the same). If it is expected that a node in a graph will need to be referenced from outside the graph, a URIref should be assigned to identify it. Finally, because blank node identifiers represent (blank) *nodes*, rather than arcs, in the triple form of an RDF graph, blank node identifiers may only appear as subjects or objects in triples; blank node identifiers may not be used as predicates in triples.

The beginning of this section noted that aggregate structures, like John Smith's address, can be represented by considering the aggregate thing to be described as a separate resource, and then making statements about that new resource. This example illustrates an important aspect of RDF: RDF directly represents only *binary* relationships, e.g. the relationship between John Smith and the literal representing his address. Representing the relationship between John and the group of separate *components* of this address involves dealing with an *n-ary* (n-way) relationship (in this case, n=5) between John and the street, city, state, and postal code components. In order to represent such structures directly in RDF (e.g., considering the address as a group of street, city, state, and postal code components), this n-way relationship must be broken up into a group of separate binary relationships. Blank nodes provide one way to do this. For each n-ary relationship, one of the participants is chosen as the subject of the relationship (John in this case), and a blank node is created to represent the rest of the relationship (John's address in this case). The remaining participants in the relationship (such as the city in this example) are then represented as separate properties of the new resource represented by the blank node.

Blank nodes also provide a way to more accurately make statements about resources that may not have URIs, but that are described in terms of relationships with other resources that *do* have URIs. For example, when making statements about a person, say Jane Smith, it may seem natural to use a URI based on that person's email address as her URI, e.g., `mailto:jane@example.org`. However, this approach can cause problems. For example, it may be necessary to record information both about *Jane's mailbox* (e.g., the server it is on) as well as about *Jane herself* (e.g., her current physical address), and using a URIref for Jane based on her email address makes it difficult to know whether it is Jane or her mailbox that is being described. The same problem exists when a company's Web page URL, say `http://www.example.com/`, is used as the URI of the company itself. Once again, it may be necessary to record information about the Web page itself (e.g., who created it and when) as well as about the company, and using `http://www.example.com/` as an identifier for both makes it difficult to know which of these is the actual subject.

The fundamental problem is that using Jane's *mailbox* as a stand-in for *Jane* is not really accurate: Jane and her mailbox are not the same thing, and hence they should be identified differently. When Jane herself does not have a URI, a blank node provides a more accurate way of modeling this situation. Jane can be represented by a blank node, and that blank node used as the subject of a statement with `exterms:mailbox` as the property and the URIref `mailto:jane@example.org` as its value. The blank node could also be described with an `rdf:type` property having a value of `exterms:Person` (types are discussed in more detail in the following sections), an `exterms:name` property having a value of `"Jane Smith"`, and any other descriptive information that might be useful, as shown in the following triples:

```
_:jane   exterms:mailbox   <mailto:jane@example.org> .
_:jane   rdf:type          exterms:Person .
_:jane   exterms:name      "Jane Smith" .
_:jane   exterms:empID     "23748"  .
_:jane   exterms:age       "26" .
```

(Note that `mailto:jane@example.org` is written within angle brackets in the first triple. This is because `mailto:jane@example.org` is a full URIref in the `mailto` URI scheme, rather than a QName abbreviation, and full URIrefs must be enclosed in angle brackets in the triples notation.)

This says, accurately, that "there is a resource of type `exterms:Person`, whose electronic mailbox is identified by `mailto:jane@example.org`, whose name is `Jane Smith`, etc." That is, the blank node can be read as "there is a resource". Statements with that blank node as subject then provide information about the characteristics of that resource.

In practice, using blank nodes instead of URIrefs in these cases does not change the way this kind of information is handled very much. For example, if it is known that an email address uniquely identifies someone at example.org (particularly if the address is unlikely to be reused), that fact can still be used to associate information about that person from multiple sources, even though the email address is not the person's URI. In this case, if some RDF is found on the Web that describes a book, and gives the author's contact information as `mailto:jane@example.org`, it might be reasonable, combining this new information with the previous set of triples, to conclude that the author's name is Jane Smith. The point is that saying something like "the author of the book is `mailto:jane@example.org`" is typically a shorthand for "the author of the book is someone whose mailbox is `mailto:jane@example.org`". Using a blank node to represent this "someone" is just a more accurate way to represent the real world situation. (Incidentally, some RDF-based schema languages allow specifying that certain properties are *unique identifiers* of the resources they describe. This is discussed further in Section 5.5.)

Using blank nodes in this way can also help avoid the use of literals in what might be inappropriate situations. For example, in describing Jane's book, lacking a URIref to identify the author, the publisher might have written (using the publisher's own `ex2terms:` vocabulary):

```
ex2terms:book78354   rdf:type          ex2terms:Book .
ex2terms:book78354   ex2terms:author   "Jane Smith" .
```

However, the author of the book is not really the character string "Jane Smith", but a person whose *name* is Jane Smith. The same information might be more accurately given by the publisher using a blank node, as:

```
ex2terms:book78354   rdf:type          ex2terms:Book .
ex2terms:book78354   ex2terms:author   _:author78354 .
_:author78354        rdf:type          ex2terms:Person .
_:author78354        ex2terms:name     "Jane Smith" .
```

This essentially says "resource `ex2terms:book78354` is of type `ex2terms:Book`, and its author is a resource of type `ex2terms:Person`, whose name is `Jane Smith`." Of course, in this particular case the publisher might instead have assigned its own URIrefs to its authors instead of using blank nodes to identify them, in order to encourage external references to its authors.

Finally, the example above giving Jane's age as 26 illustrates the fact that sometimes the value of a property may appear to be simple, but actually may be more complex. In this case, Jane's age is actually 26 *years*, but the units information (years) is not explicitly given. Such information is often omitted in contexts where it can be safely assumed that anyone accessing the property value will understand the units being used. However, in the wider context of the Web, it is generally *not* safe to make this assumption. For example, a U.S. site might give a weight value in pounds, but someone accessing that data from outside the U.S. might assume that weights are given in kilograms. In general, careful consideration should be given to explicitly representing units and similar information. This issue is discussed further in Section 4.4, which describes an RDF feature for representing such information as structured values, as well as some other techniques for representing such information.

2.4 Typed Literals

The last section described how to handle situations in which property values represented by plain literals had to be broken up into structured values to represent the individual parts of those literals. Using this approach, instead of, say, recording the date a Web page was created as a single `exterms:creation-date` property, with a single plain literal as its value, the value would be represented as a structure consisting of the month, day, and year as separate pieces of information, using separate plain literals to represent the corresponding values. However, so far, all constant values that serve as objects in RDF statements have been represented by these plain (untyped) literals, even when the intent is probably for the value of the property to be a number (e.g., the value of a `year` or `age` property) or some other kind of more specialized value.

For example, Figure 4 illustrated an RDF graph recording information about John Smith. That graph recorded the value of John Smith's `exterms:age` property as the plain literal "27", as shown in Figure 7:



Figure 7: Representing John Smith's Age

In this case, the hypothetical organization example.org probably intends for "27" to be interpreted as a number, rather than as the string consisting of the character "2" followed by the character "7" (since the literal represents the value of an "age" property). However, there is no information in Figure 7's graph that explicitly indicates that "27" should be interpreted as a number. Similarly, example.org also probably intends for "27" to be interpreted as a *decimal* number, i.e., the value *twenty seven*, rather than, say, as an *octal* number, i.e., the value *twenty three*. However, once again there is no information in Figure 7's graph that explicitly indicates this. Specific applications might be written with the understanding that they should interpret values of the `exterms:age` property as decimal numbers, but this would mean that proper interpretation of this RDF would depend on information not explicitly provided in the RDF graph, and hence on information that would not necessarily be available to other applications that might need to interpret this RDF.

The common practice in programming languages or database systems is to provide this additional information about how to interpret a literal by associating a *datatype* with the literal, in this case, a datatype like `decimal` or `integer`. An application that understands the datatype then knows, for example, whether the literal "10" is intended to represent the number *ten*, the number *two*, or the string consisting of the character "1" followed by the character "0", depending on whether the specified datatype is `integer`, `binary`, or `string`. (More specialized datatypes could also be used to include the units information mentioned at the end of Section 2.3, e.g., a datatype `integerYears`, although the Primer will not elaborate on this idea.) In RDF, typed literals are used to provide this kind of information.

An RDF typed literal is formed by pairing a string with a URIref that identifies a particular datatype. This results in a single literal node in the RDF graph with the pair as the literal. The value represented by the typed literal is the value that the specified datatype associates with the specified string. For example, using a typed literal, John Smith's age could be described as being the integer number *27* using the triple:

```
<http://www.example.org/staffid/85740>  <http://www.example.org/terms/age>
"27"^^<http://www.w3.org/2001/XMLSchema#integer> .
```

or, using the QName simplification for writing long URIs:

```
exstaff:85740  exterms:age  "27"^^xsd:integer .
```

or as shown in Figure 8:

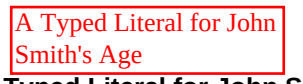


Figure 8: A Typed Literal for John Smith's Age

Similarly, in the graph shown in Figure 3 describing information about a Web page, the value of the page's `exterms:creation-date` property was written as the plain literal "August 16, 1999". However, using a typed literal, the creation date of the Web page could be explicitly described as being the date *August 16, 1999*, using the triple:

```
ex:index.html  exterms:creation-date  "1999-08-16"^^xsd:date .
```

or as shown in Figure 9:

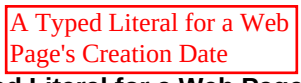


Figure 9: A Typed Literal for a Web Page's Creation Date

Unlike typical programming languages and database systems, RDF has no built-in set of datatypes of its own, such as datatypes for integers, reals, strings, or dates. Instead, RDF typed literals simply provide a way to explicitly indicate, for a given literal, what datatype should be used to interpret it. The datatypes used in typed literals are defined externally to RDF, and identified by their datatype URIs. (There is one exception: RDF defines a built-in datatype with the URIref `rdf:XMLLiteral` to represent XML content as a literal value. This datatype is defined in [RDF-CONCEPTS], and its use is described in Section 4.5.) For instance, the examples in Figure 8 and Figure 9 use the datatypes `integer` and `date` from the XML Schema datatypes defined in XML Schema Part 2: Datatypes [XML-SCHEMA2]. An advantage of this approach is that it gives RDF the flexibility to directly represent information coming from different sources without the need to perform type conversions between these sources and a native set of RDF datatypes. (Type conversions would still be required when moving information between systems having different sets of datatypes, but RDF would impose no extra conversions into and out of a native set of RDF datatypes.)

RDF datatype concepts are based on a conceptual framework from XML Schema datatypes [XML-SCHEMA2], as described in RDF Concepts and Abstract Syntax [RDF-CONCEPTS]. This conceptual framework defines a datatype as consisting of:

- A set of values, called the *value space*, that literals of the datatype are intended to represent. For example, for the XML Schema datatype `xsd:date`, this set of values is a set of dates.
- A set of character strings, called the *lexical space*, that the datatype uses to represent its values. This set determines which character strings can legally be used to represent literals of this datatype. For example, the datatype `xsd:date` defines `1999-08-16` as being a legal way to write a literal of this type (as opposed, say, to `August 16, 1999`). As defined in [RDF-CONCEPTS], the lexical space of a datatype is a set of Unicode [UNICODE] strings, allowing information from many languages to be directly represented.
- A *lexical-to-value mapping* from the lexical space to the value space. This determines the value that a given character string from the lexical space represents for this particular datatype. For example, the lexical-to-value mapping for datatype `xsd:date` determines that, for this datatype, the string `1999-08-16` represents the date *August 16, 1999*. The lexical-to-value mapping is a factor because the same character string may represent different values for different datatypes.

Not all datatypes are suitable for use in RDF. For a datatype to be suitable for use in RDF, it must conform to the conceptual framework just described. This basically means that, given a character string, the datatype must unambiguously define whether or not the string is in its lexical space, and what value in its value space the string represents. For example, the basic XML Schema datatypes such as `xsd:string`, `xsd:boolean`, `xsd:date`, etc. are suitable for use in RDF. However, some of the built-in XML Schema datatypes are not suitable for use in RDF. For example, `xsd:duration` does not have a well-defined value space, and `xsd:QName` requires an enclosing XML document context. Lists of the XML Schema datatypes that are currently considered suitable and unsuitable for use in RDF are given in [RDF-SEMANTICS].

Since the value that a given typed literal denotes is defined by the typed literal's datatype, and, with the exception of `rdf:XMLLiteral`, RDF does not define any datatypes, the actual interpretation of a typed literal appearing in an RDF graph (e.g., determining the value it denotes) must be performed by software that is written to correctly process not only RDF, but the typed literal's datatype as well. Effectively, this software must be written to process an extended language that includes not only RDF, but also the datatype, as part of its built-in vocabulary. This raises the issue of which datatypes will be generally available in RDF software. Generally, the XML Schema datatypes that are listed as suitable for use in RDF in [RDF-SEMANTICS] have a "first among equals" status in RDF. As noted already, the examples in Figure 8 and Figure 9 used some of these XML Schema datatypes, and the Primer will be using these datatypes in most of its other examples of typed literals as well (for one thing, XML Schema datatypes already have assigned URIrefs that can be used to refer to them, specified in [XML-SCHEMA2]). These XML Schema datatypes are treated no differently than any other datatype, but they are expected to be the most widely used, and therefore the most likely to be

interoperable among different software. As a result, it is expected that much RDF software will also be written to process these datatypes. However, RDF software could be written to process other sets of datatypes as well, assuming they were determined to be suitable for use with RDF, as described already.

In general, RDF software may be called on to process RDF data that contains references to datatypes that the software has not been written to process, in which case there are some things the software will not be able to do. For one thing, with the exception of `rdf:XMLLiteral`, RDF itself does not define the URIrefs that identify datatypes. As a result, RDF software, unless it has been written to recognize specific URIrefs, will not be able to determine whether or not a URIref written in a typed literal actually identifies a datatype. Moreover, even when a URIref does identify a datatype, RDF itself does not define the validity of pairing that datatype with a particular literal. This validity can only be determined by software written to correctly process that particular datatype.

For example, the typed literal in the triple:

```
exstaff:85740  exterms:age  "pumpkin"^^xsd:integer .
```

or the graph shown in Figure 10:



Figure 10: An Invalid Typed Literal for John Smith's Age

is valid RDF, but obviously an error as far as the `xsd:integer` datatype is concerned, since `"pumpkin"` is not defined as being in the lexical space of `xsd:integer`. RDF software not written to process the `xsd:integer` datatype would not be able to recognize this error.

However, proper use of RDF typed literals provides more information about the intended interpretation of literal values, and hence makes RDF statements a better means of information exchange among applications.

2.5 Concepts Summary

Taken as a whole, RDF is basically simple: nodes-and-arcs diagrams interpreted as statements about things identified by URIrefs. This section has presented an introduction to these concepts. As noted earlier, the normative (i.e., definitive) RDF specification describing these concepts is RDF Concepts and Abstract Syntax [RDF-CONCEPTS], which should be consulted for further information. The formal semantics (meaning) of these concepts is defined in the (normative) RDF Semantics [RDF-SEMANTICS] document.

However, in addition to the basic techniques for describing things using RDF statements discussed so far, it should be clear that people or organizations also need a way to describe the *vocabularies* (terms) they intend to use in those statements, specifically, vocabularies for:

- describing types of things (like `exterms:Person`)
- describing properties (like `exterms:age` and `exterms:creation-date`), and
- describing the types of things that can serve as the subjects or objects of statements involving those properties (such as specifying that the value of an `exterms:age` property should always be an `xsd:integer`).

The basis for describing such vocabularies in RDF is the RDF Vocabulary Description Language 1.0: RDF Schema [RDF-VOCABULARY], which will be described in Section 5.

Additional background on the basic ideas underlying RDF, and its role in providing a general language for describing Web information, can be found in [WEBDATA]. RDF draws upon ideas from knowledge representation, artificial intelligence, and data management, including Conceptual Graphs, logic-based knowledge representation, frames, and relational databases. Some possible sources of background information on these subjects include [SOWA], [CG], [KIF], [HAYES], [LUGER], and [GRAY].

3. An XML Syntax for RDF: RDF/XML

As described in Section 2, RDF's conceptual model is a graph. RDF provides an XML syntax for writing down and exchanging RDF graphs, called *RDF/XML*. Unlike triples, which are intended as a shorthand notation, RDF/XML is the normative syntax for writing RDF. RDF/XML is defined in the RDF/XML Syntax Specification [RDF-SYNTAX]. This section describes this RDF/XML syntax.

3.1 Basic Principles

The basic ideas behind the RDF/XML syntax can be illustrated using some of the examples presented already. Take as an example

the English statement:

> **http://www.example.org/index.html** has a **creation-date** whose value is **August 16, 1999**

The RDF graph for this single statement, after assigning a URIref to the `creation-date` property, is shown in Figure 11:



Figure 11: Describing a Web Page's Creation Date

with a triple representation of:

```
ex:index.html   exterms:creation-date   "August 16, 1999" .
```

(Note that a typed literal is not used for the date value in this example. Representing typed literals in RDF/XML will be described later in this section.)

Example 2 shows the RDF/XML syntax corresponding to the graph in Figure 11:

Example 2: RDF/XML for the Web Page's Creation Date

```
1. <?xml version="1.0"?>
2. <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3.             xmlns:exterms="http://www.example.org/terms/">

4.   <rdf:Description rdf:about="http://www.example.org/index.html">
5.       <exterms:creation-date>August 16, 1999</exterms:creation-date>
6.   </rdf:Description>

7. </rdf:RDF>
```

(Line numbers are added to help in explaining the example.)

This seems like a lot of overhead. It is easier to understand what is going on by considering each part of this XML in turn (a brief introduction to XML is provided in Appendix B).

Line 1, `<?xml version="1.0"?>`, is the *XML declaration*, which indicates that the following content is XML, and what version of XML it is.

Line 2 begins an `rdf:RDF` element. This indicates that the following XML content (starting here and ending with the `</rdf:RDF>` in line 7) is intended to represent RDF. Following the `rdf:RDF` on this same line is an XML namespace declaration, represented as an `xmlns` attribute of the `rdf:RDF` start-tag. This declaration specifies that all tags in this content prefixed with `rdf:` are part of the namespace identified by the URIref `http://www.w3.org/1999/02/22-rdf-syntax-ns#`. URIrefs beginning with the string `http://www.w3.org/1999/02/22-rdf-syntax-ns#` are used for terms from the RDF vocabulary.

Line 3 specifies another XML namespace declaration, this time for the prefix `exterms:`. This is expressed as another `xmlns` attribute of the `rdf:RDF` element, and specifies that the namespace URIref `http://www.example.org/terms/` is to be associated with the `exterms:` prefix. URIrefs beginning with the string `http://www.example.org/terms/` are used for terms from the vocabulary defined by the example organization, example.org. The ">" at the end of line 3 indicates the end of the `rdf:RDF` start-tag. Lines 1-3 are general "housekeeping" necessary to indicate that this is RDF/XML content, and to identify the namespaces being used within the RDF/XML content.

Lines 4-6 provide the RDF/XML for the specific statement shown in Figure 11. An obvious way to talk about any RDF statement is to say it is a *description*, and that it is *about* the subject of the statement (in this case, about http://www.example.org/index.html), and this is the way RDF/XML represents the statement. The `rdf:Description` start-tag in line 4 indicates the start of a *description* of a resource, and goes on to identify the resource the statement is *about* (the subject of the statement) using the `rdf:about` attribute to specify the URIref of the subject resource. Line 5 provides a *property element*, with the QName `exterms:creation-date` as its tag, to represent the predicate and object of the statement. The QName `exterms:creation-date` is chosen so that appending the local name `creation-date` to the URIref of the `exterms:` prefix (`http://www.example.org/terms/`) gives the statement's predicate URIref `http://www.example.org/terms/creation-date`. The content of this property element is the object of the statement, the plain literal `August 19, 1999` (the value of the creation-date property of the subject resource). The property element is nested within the containing `rdf:Description` element, indicating that this property applies to the resource specified in the

`rdf:about` attribute of the `rdf:Description` element. Line 6 indicates the end of this particular `rdf:Description` element.

Finally, Line 7 indicates the end of the `rdf:RDF` element started on line 2. Using an `rdf:RDF` element to enclose RDF/XML content is optional in situations where the XML can be identified as RDF/XML by context. This is discussed further in [RDF-SYNTAX]. However, it does not hurt to provide the `rdf:RDF` element in any case, and Primer examples will generally (but not always) provide one.

Example 2 illustrates the basic ideas used by RDF/XML to encode an RDF graph as XML elements, attributes, element content, and attribute values. The URIrefs of predicates (as well as some nodes) are written as XML *QNames*, consisting of a short *prefix* denoting a namespace URI, together with a *local name* denoting a namespace-qualified element or attribute, as described in Appendix B. The (namespace URIref, local name) pair is chosen so that concatenating them forms the URIref of the original node or predicate. The URIrefs of subject nodes are written as XML attribute values (URIrefs of object nodes may sometimes be written as attribute values as well). Literal nodes (which are always object nodes) become element text content or attribute values. (Many of these options are described later in the Primer; all of these options are described in [RDF-SYNTAX].)

An RDF graph consisting of multiple statements can be represented in RDF/XML by using RDF/XML similar to Lines 4-6 in Example 2 to separately represent each statement. For example, to write the following two statements:

```
ex:index.html   exterms:creation-date   "August 16, 1999" .
ex:index.html   dc:language             "en" .
```

the RDF/XML in Example 3 could be used:

Example 3: RDF/XML for Two Statements

```
1.  <?xml version="1.0"?>
2.  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3.             xmlns:dc="http://purl.org/dc/elements/1.1/"
4.             xmlns:exterms="http://www.example.org/terms/">

5.    <rdf:Description rdf:about="http://www.example.org/index.html">
6.        <exterms:creation-date>August 16, 1999</exterms:creation-date>
7.    </rdf:Description>

8.    <rdf:Description rdf:about="http://www.example.org/index.html">
9.        <dc:language>en</dc:language>
10.   </rdf:Description>

11. </rdf:RDF>
```

Example 3 is the same as Example 2, with the addition of a second `rdf:Description` element (in lines 8-10) to represent the second statement. (An additional namespace declaration is also given in line 3 to identify the additional namespace used in this statement.) An arbitrary number of additional statements could be written in the same way, using a separate `rdf:Description` element for each additional statement. As Example 3 illustrates, once the overhead of writing the XML and namespace declarations is dealt with, writing each additional RDF statement in RDF/XML is both straightforward and not too complicated.

The RDF/XML syntax provides a number of abbreviations to make common uses easier to write. For example, it is typical for the same resource to be described with several properties and values at the same time, as in Example 3, where the resource `ex:index.html` is the subject of several statements. To handle such cases, RDF/XML allows multiple property elements representing those properties to be nested within the `rdf:Description` element that identifies the subject resource. For example, to represent the following group of statements about `http://www.example.org/index.html`:

```
ex:index.html   dc:creator              exstaff:85740 .
ex:index.html   exterms:creation-date   "August 16, 1999" .
ex:index.html   dc:language             "en" .
```

whose graph (the same as Figure 3) is shown in Figure 12:



Figure 12: Several Statements About the Same Resource

the RDF/XML shown in Example 4 could be written:

Example 4: Abbreviating Multiple Properties

```
1.  <?xml version="1.0"?>
2.  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3.             xmlns:dc="http://purl.org/dc/elements/1.1/"
4.             xmlns:exterms="http://www.example.org/terms/">

5.    <rdf:Description rdf:about="http://www.example.org/index.html">
6.         <exterms:creation-date>August 16, 1999</exterms:creation-date>
7.         <dc:language>en</dc:language>
8.         <dc:creator rdf:resource="http://www.example.org/staffid/85740"/>
9.    </rdf:Description>

10. </rdf:RDF>
```

Compared with the previous two examples, Example 4 adds an additional `dc:creator` property element (in line 8). In addition, the property elements for the three properties whose subject is `http://www.example.org/index.html` are nested within a single `rdf:Description` element identifying that subject, rather than writing a separate `rdf:Description` element for each statement.

Line 8 also introduces a new form of property element. The `dc:language` element in line 7 is similar to the `exterms:creation-date` element used in Example 2. Both these elements represent properties with plain literals as property values, and such elements are written by enclosing the literal within start- and end-tags corresponding to the property name. However, the `dc:creator` element on line 8 represents a property whose value is *another resource*, rather than a literal. If the URIref of this resource were written as a plain literal within start- and end-tags in the same way as the literal values of the other elements, this would say that the value of the `dc:creator` element was the *character string* `http://www.example.org/staffid/85740`, rather than the resource identified by that literal interpreted as a URIref. In order to indicate the difference, the `dc:creator` element is written using what XML calls an *empty-element tag* (it has no separate end-tag), and the property value is written using an `rdf:resource` attribute within that empty element. The `rdf:resource` attribute indicates that the property element's value is another resource, identified by its URIref. Because the URIref is being used as an attribute *value*, RDF/XML requires the URIref to be written out (as an absolute or relative URIref), rather than abbreviating it as a QName as was done in writing element and attribute *names* (absolute and relative URIrefs are discussed in Appendix A).

It is important to understand that the RDF/XML in Example 4 is an *abbreviation*. The RDF/XML in Example 5, in which each statement is written separately, describes exactly the same RDF graph (the graph of Figure 12):

Example 5: Writing Example 4 as Separate Statements

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns:dc="http://purl.org/dc/elements/1.1/"
           xmlns:exterms="http://www.example.org/terms/">

  <rdf:Description rdf:about="http://www.example.org/index.html">
      <exterms:creation-date>August 16, 1999</exterms:creation-date>
  </rdf:Description>

  <rdf:Description rdf:about="http://www.example.org/index.html">
      <dc:language>en</dc:language>
  </rdf:Description>

  <rdf:Description rdf:about="http://www.example.org/index.html">
      <dc:creator rdf:resource="http://www.example.org/staffid/85740"/>
  </rdf:Description>

</rdf:RDF>
```

The following sections will describe a few additional RDF/XML abbreviations. [RDF-SYNTAX] provides a more thorough description of the abbreviations that are available.

RDF/XML can also represent graphs that include nodes that have no URIrefs, i.e., the *blank nodes* described in Section 2.3. For example, Figure 13 (taken from [RDF-SYNTAX]) shows a graph saying "the document 'http://www.w3.org/TR/rdf-syntax-grammar' has a title 'RDF/XML Syntax Specification (Revised)' and has an editor, the editor has a name 'Dave Beckett' and a home page 'http://purl.org/net/dajobe/' ".



Figure 13: A Graph Containing a Blank Node

This illustrates an idea discussed in Section 2.3: the use of a blank node to represent something that does not have a URIref, but can be described in terms of other information. In this case, the blank node represents a person, the editor of the document, and the person is described by his name and home page.

RDF/XML provides several ways to represent graphs containing blank nodes. These are all described in [RDF-SYNTAX]. The approach illustrated here, which is the most direct approach, is to assign a *blank node identifier* to each blank node. A blank node identifier serves to identify a blank node within a particular RDF/XML document but, unlike a URIref, is unknown outside the document in which it is assigned. A blank node is referred to in RDF/XML using an `rdf:nodeID` attribute, with a blank node identifier as its value, in places where the URIref of a resource would otherwise appear. Specifically, a statement with a blank node as its *subject* can be written in RDF/XML using an `rdf:Description` element with an `rdf:nodeID` attribute instead of an `rdf:about` attribute. Similarly, a statement with a blank node as its *object* can be written using a property element with an `rdf:nodeID` attribute instead of an `rdf:resource` attribute. Using `rdf:nodeID`, Example 6 shows the RDF/XML corresponding to Figure 13:

Example 6: RDF/XML Describing a Blank Node

```
1.  <?xml version="1.0"?>
2.  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3.             xmlns:dc="http://purl.org/dc/elements/1.1/"
4.             xmlns:exterms="http://example.org/stuff/1.0/">

5.     <rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
6.       <dc:title>RDF/XML Syntax Specification (Revised)</dc:title>
7.       <exterms:editor rdf:nodeID="abc"/>
8.     </rdf:Description>

9.     <rdf:Description rdf:nodeID="abc">
10.        <exterms:fullName>Dave Beckett</exterms:fullName>
11.        <exterms:homePage rdf:resource="http://purl.org/net/dajobe/"/>
12.    </rdf:Description>

13. </rdf:RDF>
```

In Example 6, the blank node identifier `abc` is used in line 9 to identify the blank node as the subject of several statements, and is used in line 7 to indicate that the blank node is the value of a resource's `exterms:editor` property. The advantage of using a blank node identifier over some of the other approaches described in [RDF-SYNTAX] is that using a blank node identifier allows the same blank node to be referred to in more than one place in the same RDF/XML document.

Finally, the *typed literals* described in Section 2.4 may be used as property values instead of the plain literals used in the examples so far. A typed literal is represented in RDF/XML by adding an `rdf:datatype` attribute specifying a datatype URIref to the property element containing the literal.

For example, to change the statement in Example 2 to use a typed literal instead of a plain literal for the `exterms:creation-date` property, the triple representation would be:

```
ex:index.html   exterms:creation-date   "1999-08-16"^^xsd:date .
```

with corresponding RDF/XML syntax shown in Example 7:

Example 7: RDF/XML Using a Typed Literal

```
1. <?xml version="1.0"?>
2. <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3.            xmlns:exterms="http://www.example.org/terms/">

4.   <rdf:Description rdf:about="http://www.example.org/index.html">
5.     <exterms:creation-date rdf:datatype=
         "http://www.w3.org/2001/XMLSchema#date">1999-08-16
       </exterms:creation-date>
6.   </rdf:Description>

7. </rdf:RDF>
```

In line 5 of Example 7, a typed literal is given as the value of the `exterms:creation-date` property element by adding an `rdf:datatype` attribute to the element's start-tag to specify the datatype. The value of this attribute is the URIref of the datatype, in this case, the URIref of the XML Schema `date` datatype. Since this is an attribute value, the URIref must be written out, rather than

using the QName abbreviation `xsd:date` used in the triple. A literal appropriate to this datatype is then written as the element content, in this case, the literal `1999-08-16`, which is the literal representation for August 16, 1999 in the XML Schema `date` datatype.

In the rest of the Primer, the examples will use typed literals from appropriate datatypes rather than plain (untyped) literals, in order to emphasize the value of typed literals in conveying more information about the intended interpretation of literal values. (The exceptions will be that plain literals will continue to be used in examples taken from actual applications that do not currently use typed literals, in order to accurately reflect the usage in those applications.) In RDF/XML, both plain and typed literals (and, with certain exceptions, tags) can contain Unicode [UNICODE] characters, allowing information from many languages to be directly represented.

Example 7 illustrates that using typed literals requires writing an `rdf:datatype` attribute with a URIref identifying the datatype for each element whose value is a typed literal. As noted earlier, RDF/XML requires that URIrefs used as attribute values must be written out, rather than abbreviated as a QName. XML *entities* can be used in RDF/XML to improve readability in such cases, by providing an additional abbreviation facility for URIrefs. Essentially, an XML entity declaration associates a name with a string of characters. When the entity name is referenced elsewhere within an XML document, XML processors replace the reference with the corresponding string. For example, the `ENTITY` declaration (specified as part of a `DOCTYPE` declaration at the beginning of the RDF/XML document):

```
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
```

defines the entity `xsd` to be the string representing the namespace URIref for XML Schema datatypes. This declaration allows the full namespace URIref to be abbreviated elsewhere in the XML document by the *entity reference* `&xsd;`. Using this abbreviation, Example 7 could also be written as shown in Example 8.

Example 8: RDF/XML Using a Typed Literal and an XML Entity

```
1. <?xml version="1.0"?>
2. <!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>

3. <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4.           xmlns:exterms="http://www.example.org/terms/">

5.   <rdf:Description rdf:about="http://www.example.org/index.html">
6.     <exterms:creation-date rdf:datatype="&xsd;date">1999-08-16
       </exterms:creation-date>
7.   </rdf:Description>

8. </rdf:RDF>
```

The `DOCTYPE` declaration in line 2 defines the entity `xsd`, which is used in line 6.

The use of XML entities as an abbreviation mechanism is optional in RDF/XML, and hence the use of an XML `DOCTYPE` declaration is also optional in RDF/XML. (For readers familiar with XML, RDF/XML is only required to be "well-formed" XML. RDF/XML is not designed to be validated against a DTD by a validating XML processor. This is discussed more fully in Appendix B, which provides additional information about XML.)

For readability purposes, examples in the rest of the Primer will use the XML entity `xsd` as just described. XML entities are discussed further in Appendix B. As illustrated in Appendix B, other URIrefs (and, more generally, other strings) can also be abbreviated using XML entities. However, the URIrefs for XML Schema datatypes are the only ones that will be abbreviated in this way in Primer examples.

Although additional abbreviated forms for writing RDF/XML are available, the facilities illustrated so far provide a simple but general way to express graphs in RDF/XML. Using these facilities, an RDF graph is written in RDF/XML as follows:

- All blank nodes are assigned blank node identifiers.
- Each node is listed in turn as the subject of an un-nested `rdf:Description` element, using an `rdf:about` attribute if the node has a URIref, or an `rdf:nodeID` attribute if the node is blank.
  For each triple with this node as subject, an appropriate property element is created, with either literal content (possibly empty), an `rdf:resource` attribute specifying the object of the triple (if the object node has a URIref), or an `rdf:nodeID` attribute specifying the object of the triple (if the object node is blank).

Compared to some of the more abbreviated approaches described in [RDF-SYNTAX], this simple approach provides the most direct representation of the actual graph structure, and is particularly recommended for applications in which the output RDF/XML is to be used in further RDF processing.

3.2 Abbreviating and Organizing RDF URIrefs

So far, the examples have assumed that the resources being described have been given URIrefs already. For instance, the initial examples provided descriptive information about example.org's Web page, whose URIref was `http://www.example.org/index.html`. This resource was identified in RDF/XML using an `rdf:about` attribute citing its full URIref. Although RDF does not specify or control how URIrefs are assigned to resources, sometimes it is desirable to achieve the effect of assigning URIrefs to resources that are part of an organized group of resources. For example, suppose a sporting goods company, example.com, wanted to provide an RDF-based catalog of its products, such as tents, hiking boots, and so on, as an RDF/XML document, identified by (and located at) `http://www.example.com/2002/04/products`. In that resource, each product might be given a separate RDF description. This catalog, along with one of these descriptions, the catalog entry for a model of tent called the "Overnighter", might be written in RDF/XML as shown in Example 9:

Example 9: RDF/XML for example.com's Catalog

```
1.   <?xml version="1.0"?>
2.   <!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
3.   <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4.               xmlns:exterms="http://www.example.com/terms/">

5.     <rdf:Description rdf:ID="item10245">
6.          <exterms:model rdf:datatype="&xsd;string">Overnighter</exterms:model>
7.          <exterms:sleeps rdf:datatype="&xsd;integer">2</exterms:sleeps>
8.          <exterms:weight rdf:datatype="&xsd;decimal">2.4</exterms:weight>
9.          <exterms:packedSize rdf:datatype="&xsd;integer">784</exterms:packedSize>
10.    </rdf:Description>

  ...other product descriptions...

11.  </rdf:RDF>
```

Example 9 is similar to previous examples in the way it represents the properties (model, sleeping capacity, weight) of the resource (the tent) being described. (The surrounding xml, DOCTYPE, RDF, and namespace information is included in lines 1 through 4, and line 11, but this information would only need to be provided once for the whole catalog, not repeated for each entry in the catalog. Note also that although the *datatypes* associated with the various property values are given explicitly, the *units* associated with some of these property values are not, even though this information should be available to properly interpret the values. Representing units and similar information that may be associated with property values is discussed in Section 4.4. In this example, the value of `exterms:sleeps` is the number of persons the tent can sleep, the value of `exterms:weight` is given in kilograms, and the value of `exterms:packedSize` is given in square centimeters, the area the tent occupies on a backpack.)

An important *difference* from previous examples is that, in line 5, the `rdf:Description` element has an `rdf:ID` attribute instead of an `rdf:about` attribute. Using `rdf:ID` specifies a *fragment identifier*, given by the value of the `rdf:ID` attribute (`item10245` in this case, which might be the catalog number assigned by example.com), as an abbreviation of the complete URIref of the resource being described. The fragment identifier `item10245` will be interpreted relative to a *base URI*, in this case, the URI of the containing catalog document. The full URIref for the tent is formed by taking the base URI (of the catalog), and appending the character "#" (to indicate that what follows is a fragment identifier) and then `item10245` to it, giving the absolute URIref `http://www.example.com/2002/04/products#item10245`.

The `rdf:ID` attribute is somewhat similar to the `ID` attribute in XML and HTML, in that it defines a name which must be unique relative to the current base URI (in this example, that of the catalog). In this case, the `rdf:ID` attribute appears to be assigning a name (`item10245`) to this particular kind of tent. Any other RDF/XML within this catalog could refer to the tent by using either the absolute URIref `http://www.example.com/2002/04/products#item10245`, or the *relative URIref* `#item10245`. The relative URIref would be understood as being a URIref defined relative to the base URIref of the catalog. Using a similar abbreviation, the URIref of the tent could also be given by specifying `rdf:about="#item10245"` in the catalog entry (i.e., by specifying the relative URIref directly) instead of `rdf:ID="item10245"` . As an abbreviation mechanism, the two forms are essentially synonyms: the full URIref formed by RDF/XML is the same in either case: `http://www.example.com/2002/04/products#item10245`. However, using `rdf:ID` provides an additional check when assigning a set of distinct names, since a given value of the `rdf:ID` attribute can only appear once relative to the same base URI (the catalog document, in this example). Using either form, example.com would be giving the URIref for the tent in a two-stage process, first assigning the URIref for the whole catalog, and then using a relative URIref in the description of the tent in the catalog to indicate the URIref that has been assigned to this particular kind of tent. Moreover, this use of a relative URIref can be thought of either as being an abbreviation for a full URIref that has been assigned to the tent independently of the RDF, or as being the assignment of the URIref to the tent within the catalog.

RDF located *outside* the catalog could refer to this tent by using the full URIref, i.e., by concatenating the relative URIref `#item10245` of the tent to the base URI of the catalog, forming the absolute URIref `http://www.example.com/2002/04/products#item10245`. For example, an outdoor sports Web site exampleRatings.com might use RDF to provide ratings of various tents. The (5-star) rating given to the tent described in Example 9 might then be represented on exampleRatings.com's Web site as shown in Example 10:

Example 10: exampleRatings.com's Rating of the Tent

```
1.  <?xml version="1.0"?>
2.  <!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
3.  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4.             xmlns:sportex="http://www.exampleRatings.com/terms/">

5.    <rdf:Description rdf:about="http://www.example.com/2002/04/products#item10245">
6.         <sportex:ratingBy rdf:datatype="&xsd;string">Richard
Roe</sportex:ratingBy>
7.         <sportex:numberStars rdf:datatype="&xsd;integer">5</sportex:numberStars>
8.    </rdf:Description>
9.  </rdf:RDF>
```

In Example 10, line 5 uses an `rdf:Description` element with an `rdf:about` attribute whose value is the full URIref of the tent. The use of this URIref allows the tent being referred to in the rating to be precisely identified.

These examples illustrate several points. First, even though RDF does not specify or control how URIrefs are assigned to resources (in this case, the various tents and other items in the catalog), the *effect* of assigning URIrefs to resources in RDF can be achieved by combining a process (external to RDF) that identifies a single document (the catalog in this case) as the source for descriptions of those resources, with the use of relative URIrefs in descriptions of those resources within that document. For instance, example.com could use this catalog as the central source where its products are described, with the understanding that if a product's item number is not in an entry in this catalog, it is not a product known to example.com. (Note that RDF does not assume any particular relationship exists between two resources just because their URIrefs have the same base, or are otherwise similar. This relationship may be known to example.com, but it is not directly defined by RDF.)

These examples also illustrate one of the basic architectural principles of the Web, which is that anyone should be able to freely add information about an existing resource, using any vocabulary they please [BERNERS-LEE98]. The examples further illustrate that the RDF describing a particular resource does not need to be located all in one place; instead, it may be distributed throughout the Web. This is true not only for situations like this one, in which one organization is rating or commenting on a resource defined by another, but also for situations in which the original definer of a resource (or anyone else) wishes to amplify the description of that resource by providing additional information about it. This may be done by modifying the RDF document in which the resource was originally described, to add the properties and values needed to describe the additional information. Or, as this example illustrates, a separate document could be created, providing the additional properties and values in `rdf:Description` elements that refer to the original resource via its URIref using `rdf:about`.

The discussion above indicated that relative URIrefs such as `#item10245` will be interpreted relative to a *base URI*. By default, this base URI would be the URI of the resource in which the relative URIref is used. However, in some cases it is desirable to be able to explicitly specify this base URI. For instance, suppose that in addition to the catalog located at `http://www.example.com/2002/04/products`, example.org wanted to provide a duplicate catalog on a mirror site, say at `http://mirror.example.com/2002/04/products`. This could create a problem, since if the catalog was accessed from the mirror site, the URIref for the example tent would be generated from the URI of the containing document, forming `http://mirror.example.com/2002/04/products#item10245`, rather than `http://www.example.com/2002/04/products#item10245`, and hence would apparently refer to a different resource than the one intended. Alternatively, example.org might want to assign a base URIref for its set of product URIrefs *without* publishing a single source document whose location defines the base.

To deal with such cases, RDF/XML supports XML Base [XML-BASE], which allows an XML document to specify a base URI other than the URI of the document itself. Example 11 shows how the catalog would be described using XML Base:

Example 11: Using XML Base in example.com's Catalog

```
1.   <?xml version="1.0"?>
2.   <!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
3.   <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4.              xmlns:exterms="http://www.example.com/terms/"
5.              xml:base="http://www.example.com/2002/04/products">

6.     <rdf:Description rdf:ID="item10245">
7.          <exterms:model rdf:datatype="&xsd;string">Overnighter</exterms:model>
8.          <exterms:sleeps rdf:datatype="&xsd;integer">2</exterms:sleeps>
9.          <exterms:weight rdf:datatype="&xsd;decimal">2.4</exterms:weight>
10.         <exterms:packedSize rdf:datatype="&xsd;integer">784</exterms:packedSize>
11.    </rdf:Description>

  ...other product descriptions...
```

```
12.  </rdf:RDF>
```

In Example 11, the `xml:base` declaration in line 5 specifies that the base URI for the content within the `rdf:RDF` element (until another `xml:base` attribute is specified) is `http://www.example.com/2002/04/products`, and all relative URIrefs cited within that content will be interpreted relative to that base, no matter what the URI of the containing document is. As a result, the relative URIref of the tent, `#item10245`, will be interpreted as the same absolute URIref, `http://www.example.com/2002/04/products#item10245`, no matter what the actual URI of the catalog document is, or whether the base URIref actually identifies a particular document at all.

So far, the examples have used a single product description, a particular model of tent, from example.com's catalog. However, example.com will probably offer several different models of tents, as well as multiple instances of other categories of products, such as backpacks, hiking boots, and so on. This idea of things being classified into different *kinds* or *categories* is similar to the programming language concept of objects having different *types* or *classes*. RDF supports this concept by providing a predefined property, `rdf:type`. When an RDF resource is described with an `rdf:type` property, the value of that property is considered to be a resource that represents a category or *class* of things, and the subject of that property is considered to be an *instance* of that category or class. Using `rdf:type`, Example 12 shows how example.com might indicate that the product description is that of a tent:

Example 12: Describing a Tent with `rdf:type`

```
1.   <?xml version="1.0"?>
2.   <!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
3.   <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4.              xmlns:exterms="http://www.example.com/terms/"
5.              xml:base="http://www.example.com/2002/04/products">

6.     <rdf:Description rdf:ID="item10245">
7.          <rdf:type rdf:resource="http://www.example.com/terms/Tent"/>
8.          <exterms:model rdf:datatype="&xsd;string">Overnighter</exterms:model>
9.          <exterms:sleeps rdf:datatype="&xsd;integer">2</exterms:sleeps>
10.         <exterms:weight rdf:datatype="&xsd;decimal">2.4</exterms:weight>
11.         <exterms:packedSize rdf:datatype="&xsd;integer">784</exterms:packedSize>
12.    </rdf:Description>

  ...other product descriptions...

13.  </rdf:RDF>
```

In Example 12, the `rdf:type` property in line 7 indicates that the resource being described is an instance of the class identified by the URIref `http://www.example.com/terms/Tent`. This assumes that example.com has described its classes as part of the same vocabulary that it uses to describe its other terms (such as the property `exterms:weight`), so the absolute URIref of the class is used to refer to it. If example.com had described these classes as part of the product catalog itself, the relative URIref `#Tent` could have been used to refer to it.

RDF itself does not provide facilities for defining application-specific classes of things, such as `Tent` in this example, or their properties, such as `exterms:weight`. Instead, such classes would be described in an *RDF schema*, using the *RDF Schema* language discussed in Section 5. Other such facilities for describing classes can also be defined, such as the *DAML+OIL* and *OWL* languages described in Section 5.5.

It is fairly common in RDF for resources to have `rdf:type` properties that describe the resources as instances of specific types or classes. Such resources are called *typed nodes* in the graph, or *typed node elements* in the RDF/XML. RDF/XML provides a special abbreviation for describing these typed nodes. In this abbreviation, the `rdf:type` property and its value are removed, and the `rdf:Description` element for the node is replaced by an element whose name is the QName corresponding to the value of the removed `rdf:type` property (a URIref that names a class). Using this abbreviation, example.com's tent from Example 12 could also be described as shown in Example 13:

Example 13: Abbreviating the Tent's Type

```
1.   <?xml version="1.0"?>
2.   <!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
3.   <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4.              xmlns:exterms="http://www.example.com/terms/"
5.              xml:base="http://www.example.com/2002/04/products">

6.     <exterms:Tent rdf:ID="item10245">
7.          <exterms:model rdf:datatype="&xsd;string">Overnighter</exterms:model>
8.          <exterms:sleeps rdf:datatype="&xsd;integer">2</exterms:sleeps>
```

```
9.          <exterms:weight rdf:datatype="&xsd;decimal">2.4</exterms:weight>
10.         <exterms:packedSize rdf:datatype="&xsd;integer">784</exterms:packedSize>
11.    </exterms:Tent>

  ...other product descriptions...

12.  </rdf:RDF>
```

Since a resource may be described as an instance of more than one class, a resource may have more than one `rdf:type` property. However, only one of these `rdf:type` properties can be abbreviated in this way. The others must be written out using `rdf:type` properties, in the manner illustrated by the `rdf:type` property in Example 12.

In addition to its use in describing instances of user-defined classes such as `exterms:Tent`, the typed node abbreviation is also commonly used in RDF/XML when describing instances of the built-in RDF classes (such as `rdf:Bag`) to be described in Section 4, and the built-in RDF Schema classes (such as `rdfs:Class`) to be described in Section 5.

Both Example 12 and Example 13 illustrate that RDF statements can be written in RDF/XML in a way that closely resembles descriptions that might have been written directly in (non-RDF) XML. This is an important consideration, given the increasing use of XML in all kinds of applications, since it suggests that RDF could be used in these applications without requiring major changes in the way their information is structured.

3.3 RDF/XML Summary

The examples above have illustrated some of the basic ideas behind the RDF/XML syntax. These examples provide enough information to begin writing useful RDF/XML. A more thorough discussion of the principles behind the modeling of RDF statements in XML (known as *striping*), together with a presentation of the other RDF/XML abbreviations available, and other details and examples about writing RDF in XML, is given in the (normative) RDF/XML Syntax Specification [RDF-SYNTAX].

4. Other RDF Capabilities

RDF provides a number of additional capabilities, such as built-in types and properties for representing groups of resources and RDF statements, and capabilities for representing XML fragments as property values. These additional capabilities are described in the following sections.

4.1 RDF Containers

There is often a need to describe *groups* of things: for example, to say that a book was created by several authors, or to list the students in a course, or the software modules in a package. RDF provides several predefined (built-in) types and properties that can be used to describe such groups.

First, RDF provides a *container vocabulary* consisting of three predefined types (together with some associated predefined properties). A *container* is a resource that contains things. The contained things are called *members*. The members of a container may be resources (including blank nodes) or literals. RDF defines three types of containers:

- `rdf:Bag`
- `rdf:Seq`
- `rdf:Alt`

A *Bag* (a resource having type `rdf:Bag`) represents a group of resources or literals, possibly including duplicate members, where there is no significance in the order of the members. For example, a Bag might be used to describe a group of part numbers in which the order of entry or processing of the part numbers does not matter.

A *Sequence* or *Seq* (a resource having type `rdf:Seq`) represents a group of resources or literals, possibly including duplicate members, where the order of the members is significant. For example, a Sequence might be used to describe a group that must be maintained in alphabetical order.

An *Alternative* or *Alt* (a resource having type `rdf:Alt`) represents a group of resources or literals that are *alternatives* (typically for a single value of a property). For example, an Alt might be used to describe alternative language translations for the title of a book, or to describe a list of alternative Internet sites at which a resource might be found. An application using a property whose value is an Alt container should be aware that it can choose any one of the members of the group as appropriate.

To describe a resource as being one of these types of containers, the resource is given an `rdf:type` property whose value is one of the predefined resources `rdf:Bag`, `rdf:Seq`, or `rdf:Alt` (whichever is appropriate). The container resource (which may either be a

blank node or a resource with a URIref) denotes the group as a whole. The *members* of the container can be described by defining a *container membership property* for each member with the container resource as its subject and the member as its object. These container membership properties have names of the form `rdf:_n`, where *n* is a decimal integer greater than zero, with no leading zeros, e.g., `rdf:_1`, `rdf:_2`, `rdf:_3`, and so on, and are used specifically for describing the members of containers. Container resources may also have other properties that describe the container, in addition to the container membership properties and the `rdf:type` property.

It is important to understand that while these types of containers are described using predefined RDF types and properties, any special meanings associated with these containers, e.g., that the members of an Alt container are alternative values, are only *intended* meanings. These specific container types, and their definitions, are provided with the aim of establishing a shared convention among those who need to describe groups of things. All RDF does is provide the types and properties that can be used to construct the RDF graphs to describe each type of container. RDF has no more built-in understanding of what a resource of type `rdf:Bag` is than it has of what a resource of type `ex:Tent` (discussed in Section 3.2) is. In each case, applications must be written to behave according to the particular meaning involved for each type. This point will be expanded on in the following examples.

A typical use of a container is to indicate that the value of a property is a group of things. For example, to represent the sentence "Course 6.001 has the students Amy, Mohamed, Johann, Maria, and Phuong", the course could be described by giving it a `s:students` property (from an appropriate vocabulary) whose value is a container of type `rdf:Bag` (representing the group of students). Then, using the container membership properties, individual students could be identified as being members of that group, as in the RDF graph shown in Figure 14:

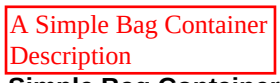


Figure 14: A Simple Bag Container Description

Since the value of the `s:students` property in this example is described as a Bag, there is no intended significance in the order given for the URIrefs of the students, even though the membership properties in the graph have integers in their names. It is up to applications creating and processing graphs that include `rdf:Bag` containers to ignore any (apparent) order in the names of the membership properties.

RDF/XML provides some special syntax and abbreviations to make it simpler to describe such containers. For example, Example 14 describes the graph shown in Figure 14:

Example 14: RDF/XML for a Bag of Students

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:s="http://example.org/students/vocab#">

   <rdf:Description rdf:about="http://example.org/courses/6.001">
      <s:students>
         <rdf:Bag>
            <rdf:li rdf:resource="http://example.org/students/Amy"/>
            <rdf:li rdf:resource="http://example.org/students/Mohamed"/>
            <rdf:li rdf:resource="http://example.org/students/Johann"/>
            <rdf:li rdf:resource="http://example.org/students/Maria"/>
            <rdf:li rdf:resource="http://example.org/students/Phuong"/>
         </rdf:Bag>
      </s:students>
   </rdf:Description>
</rdf:RDF>
```

Example 14 shows that RDF/XML provides `rdf:li` as a convenience element to avoid having to explicitly number each membership property. The numbered properties `rdf:_1`, `rdf:_2`, and so on are generated from the `rdf:li` elements in forming the corresponding graph. The element name `rdf:li` was chosen to be mnemonic with the term "list item" from HTML. Note also the use of a `<rdf:Bag>` element nested within the `<s:students>` property element. The `<rdf:Bag>` element is another example of the abbreviation used in Example 13 that replaces both an `rdf:Description` element and an `rdf:type` element with a single element when describing an instance of a type (an instance of `rdf:Bag` in this case). Since no URIref is specified, the Bag is a blank node. Its nesting within the `<s:students>` property element is an abbreviated way of indicating that the blank node is the value of this property. These abbreviations are described further in [RDF-SYNTAX].

The graph structure for an `rdf:Seq` container, and the corresponding RDF/XML, are similar to those for an `rdf:Bag` (the only difference is in the type, `rdf:Seq`). Once again, although an `rdf:Seq` container is intended to describe a sequence, it is up to applications creating and processing the graph to appropriately interpret the sequence of integer-valued property names.

To illustrate an Alt container, the sentence "The source code for X11 may be found at ftp.example.org, ftp1.example.org, or ftp2.example.org" could be expressed in the RDF graph shown in Figure 15:



Figure 15: A Simple Alt Container Description

Example 15 shows how the graph in Figure 15 could be written in RDF/XML:

Example 15: RDF/XML for an Alt Container

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:s="http://example.org/packages/vocab#">

   <rdf:Description rdf:about="http://example.org/packages/X11">
      <s:DistributionSite>
         <rdf:Alt>
            <rdf:li rdf:resource="ftp://ftp.example.org"/>
            <rdf:li rdf:resource="ftp://ftp1.example.org"/>
            <rdf:li rdf:resource="ftp://ftp2.example.org"/>
         </rdf:Alt>
      </s:DistributionSite>
   </rdf:Description>
</rdf:RDF>
```

An Alt container is intended to have at least one member, identified by the property `rdf:_1`. This member is intended to be considered as the default or preferred value. Other than the member identified as `rdf:_1`, the order of the remaining elements is not significant.

The RDF in Figure 15 *as written* states simply that the value of the `s:DistributionSite` site property is the Alt container resource itself. Any additional meaning that is to be read into this graph, e.g., that one of the *members* of the Alt container is to be considered as the value of the `s:DistributionSite` site property, or that `ftp://ftp.example.org` is the default or preferred value, must be built into an application's understanding of the intended meaning of an Alt container, and/or into the meaning defined for the particular property (`s:DistributionSite` in this case), which also must be understood by the application.

Alt containers are frequently used in conjunction with language tagging. (RDF/XML permits the use of the `xml:lang` attribute defined in [XML] to indicate that the element content is in a specified language. The use of `xml:lang` is described in [RDF-SYNTAX], and illustrated later in Section 6.2.) For example, a work whose title has been translated into several languages might have its `title` property pointing to an Alt container holding literals representing the titles expressed in each of the language variants.

The distinction between the intended meanings of a Bag and an Alt can be further illustrated by considering the authorship of the book "Huckleberry Finn". The book has exactly one author, but the author has two names (Mark Twain and Samuel Clemens). Either name is sufficient to specify the author. Thus using an Alt container for the author's names more accurately represents the relationship than using a Bag (which might suggest there are two *different* authors).

Users are free to choose their own ways to describe groups of resources, rather than using the RDF container vocabulary. These RDF containers are merely provided as common definitions that, if generally used, could help make data involving groups of resources more interoperable.

Sometimes there are clear alternatives to using these RDF container types. For example, a relationship between a particular resource and a group of other resources could be indicated by making the first resource the subject of multiple statements using the same property. This is structurally different from the resource being the subject of a single statement whose object is a container containing multiple members. In some cases, these two structures may have equivalent meaning, but in other cases they may not. The choice of which to use in a given situation should be made with this in mind.

Consider as an example the relationship between a writer and her publications, as in the sentence:

> Sue has written "Anthology of Time", "Zoological Reasoning", and "Gravitational Reflections".

In this case, there are three resources each of which was written independently by the same writer. This could be expressed using repeated properties as:

```
exstaff:Sue   exterms:publication   ex:AnthologyOfTime .
exstaff:Sue   exterms:publication   ex:ZoologicalReasoning .
```

```
exstaff:Sue   exterms:publication   ex:GravitationalReflections .
```

In this example there is no stated relationship between the publications other than that they were written by the same person. Each of the statements is an independent fact, and so using repeated properties would be a reasonable choice. However, this could just as reasonably be represented as a statement about the group of resources written by Sue:

```
exstaff:Sue   exterms:publication   _:z .
_:z           rdf:type              rdf:Bag .
_:z           rdf:_1                ex:AnthologyOfTime .
_:z           rdf:_2                ex:ZoologicalReasoning .
_:z           rdf:_3                ex:GravitationalReflections .
```

On the other hand, the sentence:

> The resolution was approved by the Rules Committee, having members Fred, Wilma, and Dino.

says that the committee *as a whole* approved the resolution; it does not necessarily state that each committee member *individually* voted in favor of the resolution. In this case, it would be potentially misleading to model this sentence as three separate `exterms:approvedBy` statements, one for each committee member, as shown below:

```
ex:resolution   exterms:approvedBy   ex:Fred .
ex:resolution   exterms:approvedBy   ex:Wilma .
ex:resolution   exterms:approvedBy   ex:Dino .
```

since these statements say that each member individually approved the resolution.

In this case, it would be better to model the sentence as a single `exterms:approvedBy` statement whose subject is the resolution and whose object is the committee itself. The committee resource could then be described as a Bag whose members are the members of the committee, as in the following triples:

```
ex:resolution       exterms:approvedBy   ex:rulesCommittee .
ex:rulesCommittee   rdf:type             rdf:Bag .
ex:rulesCommittee   rdf:_1               ex:Fred .
ex:rulesCommittee   rdf:_2               ex:Wilma .
ex:rulesCommittee   rdf:_3               ex:Dino .
```

When using RDF containers, it is important to understand that the statements are not *constructing* containers, as in a programming language data structure. Instead, the statements are *describing* containers (groups of things) that presumably exist. For instance, in the Rules Committee example just given, the Rules Committee is an unordered group of people, whether it is described in RDF that way or not. Saying that the resource `ex:rulesCommittee` has type `rdf:Bag` is not saying that the Rules Committee is a data structure, or constructing a particular data structure to hold the members of the group (the Rules Committee could be described as a Bag without describing any members at all). Instead, it is describing the Rules Committee as having characteristics corresponding to those associated with a Bag container, namely that it has members, and their order of description is not significant. Similarly, using the container membership properties simply describes a container resource as having certain things as members. This does not necessarily say that the things described as members are the *only* members that exist. For example, the triples given above to describe the Rules Committee say only that Fred, Wilma, and Dino are members of the committee, not that they are the *only* members of the committee.

Also, Example 14 and Example 15 illustrated a common "pattern" in describing containers, regardless of the type of container involved (e.g., use of a blank node with an appropriate `rdf:type` property to represent the container itself, and use of `rdf:li` to generate sequentially-numbered container membership properties). However, it is important to understand that RDF does not *enforce* this particular way of using the RDF container vocabulary, and so it is possible to use this vocabulary in other ways. For example, in some cases it might be appropriate to use a container resource having a URIref rather than using a blank node. Moreover, it is possible to use the container vocabulary in ways that may not describe graphs with the "well-formed" structures shown in the previous examples. For example, Example 16 shows the RDF/XML for a graph similar to the Alt container shown in Figure 15, but which writes the container membership properties explicitly, rather than using `rdf:li` to generate them:

Example 16: RDF/XML for an "Ill-Formed" Alt Container

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:s="http://example.org/packages/vocab#">

 <rdf:Description rdf:about="http://example.org/packages/X11">
    <s:DistributionSite>
```

```
      <rdf:Alt>
         <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag"/>
         <rdf:_2 rdf:resource="ftp://ftp.example.org"/>
         <rdf:_2 rdf:resource="ftp://ftp1.example.org"/>
         <rdf:_5 rdf:resource="ftp://ftp2.example.org"/>
      </rdf:Alt>
    </s:DistributionSite>
 </rdf:Description>
</rdf:RDF>
```

As noted in [RDF-SEMANTICS], RDF imposes no "well-formedness" conditions on the use of the container vocabulary, so Example 16 is perfectly legal, even though the container is described as *both* a Bag and an Alt, it is described as having two distinct values of the `rdf:_2` property, and it does not have `rdf:_1`, `rdf:_3`, or `rdf:_4` properties.

As a result, RDF applications that require containers to be "well-formed" should be written to check that the container vocabulary is being used appropriately, in order to be fully robust.

4.2 RDF Collections

A limitation of the containers described in Section 4.1 is that there is no way to *close* them, i.e., to say "these are all the members of the container". As noted in Section 4.1, a container only says that certain identified resources are members; it does not say that other members do not exist. Also, while one graph may describe some of the members, there is no way to exclude the possibility that there is another graph somewhere that describes additional members. RDF provides support for describing groups containing only the specified members, in the form of RDF *collections*. An RDF collection is a group of things represented as a list structure in the RDF graph. This list structure is constructed using a predefined *collection vocabulary* consisting of the predefined type `rdf:List`, the predefined properties `rdf:first` and `rdf:rest`, and the predefined resource `rdf:nil`.

To illustrate this, the sentence "The students in course 6.001 are Amy, Mohamed, and Johann" could be represented using the graph shown in Figure 16:



Figure 16: An RDF Collection (list structure)

In this graph, each member of the collection, such as `s:Amy`, is the object of an `rdf:first` property whose subject is a resource (a blank node in this example) that represents a list. This list resource is linked to the rest of the list by an `rdf:rest` property. The end of the list is indicated by the `rdf:rest` property having as its object the resource `rdf:nil` (the resource `rdf:nil` represents the empty list, and is defined as being of type `rdf:List`). This structure will be familiar to those who know the Lisp programming language. As in Lisp, the `rdf:first` and `rdf:rest` properties allow applications to traverse the structure. Each of the blank nodes forming this list structure is implicitly of type `rdf:List` (that is, each of these nodes implicitly has an `rdf:type` property whose value is the predefined type `rdf:List`), although this is not explicitly shown in the graph. The RDF Schema language [RDF-VOCABULARY] defines the properties `rdf:first` and `rdf:rest` as having subjects of type `rdf:List`, so the information about these nodes being lists can generally be inferred, rather than the corresponding `rdf:type` triples being written out all the time.

RDF/XML provides a special notation to make it easy to describe collections using graphs of this form. In RDF/XML, a collection can be described by a property element that has the attribute `rdf:parseType="Collection"`, and that contains a group of nested elements representing the members of the collection. RDF/XML provides the `rdf:parseType` attribute to indicate that the contents of an element are to be interpreted in a special way. In this case, the `rdf:parseType="Collection"` attribute indicates that the enclosed elements are to be used to create the corresponding list structure in the RDF graph (other values of the `rdf:parseType` attribute will be described in later sections of the Primer).

To illustrate how `rdf:parseType="Collection"` works, the RDF/XML from Example 17 would result in the RDF graph shown in Figure 16:

Example 17: RDF/XML for a Collection of Students

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:s="http://example.org/students/vocab#">

   <rdf:Description rdf:about="http://example.org/courses/6.001">
      <s:students rdf:parseType="Collection">
            <rdf:Description rdf:about="http://example.org/students/Amy"/>
            <rdf:Description rdf:about="http://example.org/students/Mohamed"/>
```

```
            <rdf:Description rdf:about="http://example.org/students/Johann"/>
      </s:students>
   </rdf:Description>
</rdf:RDF>
```

The use of `rdf:parseType="Collection"` in RDF/XML always defines a list structure like the one shown in Figure 16, i.e., a fixed finite list of items with a given length and terminated by `rdf:nil`, and which uses "new" blank nodes that are unique to the list structure itself. However, RDF does not *enforce* this particular way of using the RDF collection vocabulary, and so it is possible to use this vocabulary in other ways, some of which may not describe lists or closed collections. To see why, note that the graph shown in Figure 16 could also be written in RDF/XML by writing out the same triples "in longhand" (without using `rdf:parseType="Collection"`) using the collection vocabulary, as in Example 18:

Example 18: RDF/XML for a Collection of Students in "Longhand"

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:s="http://example.org/students/vocab#">

<rdf:Description rdf:about="http://example.org/courses/6.001">
   <s:students rdf:nodeID="sch1"/>
</rdf:Description>

<rdf:Description rdf:nodeID="sch1">
   <rdf:first rdf:resource="http://example.org/students/Amy"/>
   <rdf:rest rdf:nodeID="sch2"/>
</rdf:Description>

<rdf:Description rdf:nodeID="sch2">
   <rdf:first rdf:resource="http://example.org/students/Mohamed"/>
   <rdf:rest rdf:nodeID="sch3"/>
</rdf:Description>

<rdf:Description rdf:nodeID="sch3">
   <rdf:first rdf:resource="http://example.org/students/Johann"/>
   <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
</rdf:Description>
</rdf:RDF>
```

As noted in [RDF-SEMANTICS] (and as was the case for the container vocabulary described in Section 4.1), RDF imposes no "well-formedness" conditions on the use of the collection vocabulary so, when writing triples in longhand, it is possible to define RDF graphs with structures other than the well-structured graphs that would be automatically generated by using `rdf:parseType="Collection"`. For example, it is not illegal to assert that a given node has two distinct values of the `rdf:first` property, to create structures that have forked or non-list tails, or to simply omit part of the description of a collection. Also, graphs defined by using the collection vocabulary in longhand could use URIrefs to identify the components of the list instead of blank nodes unique to the list structure. In this case, it would be possible to create triples in other graphs that effectively added elements to the collection, making it non-closed.

As a result, RDF applications that require collections to be well-formed should be written to check that the collection vocabulary is being used appropriately, in order to be fully robust. In addition, languages such as OWL [OWL], which can define additional constraints on the structure of RDF graphs, can rule out some of these cases.

4.3 RDF Reification

RDF applications sometimes need to describe other RDF statements using RDF, for instance, to record information about when statements were made, who made them, or other similar information (this is sometimes referred to as "provenance" information). For example, Example 9 in Section 3.2 described a particular tent with URIref `exproducts:item10245`, offered for sale by example.com. One of the triples from that description, describing the weight of the tent, was:

```
exproducts:item10245   exterms:weight   "2.4"^^xsd:decimal .
```

and it might be useful for example.com to record who provided that particular piece of information.

RDF provides a built-in vocabulary intended for describing RDF statements. A description of a statement using this vocabulary is called a *reification* of the statement. The RDF reification vocabulary consists of the type `rdf:Statement`, and the properties `rdf:subject`, `rdf:predicate`, and `rdf:object`. However, while RDF provides this reification vocabulary, care is needed in

using it, because it is easy to imagine that the vocabulary defines some things that are not actually defined. This point will be discussed further later in this section.

Using the reification vocabulary, a *reification* of the statement about the tent's weight would be given by assigning the statement a URIref such as `exproducts:triple12345` (so statements can be written describing it), and then describing the statement using the statements:

```
exproducts:triple12345   rdf:type        rdf:Statement .
exproducts:triple12345   rdf:subject     exproducts:item10245 .
exproducts:triple12345   rdf:predicate   exterms:weight .
exproducts:triple12345   rdf:object      "2.4"^^xsd:decimal .
```

These statements say that the resource identified by the URIref `exproducts:triple12345` is an RDF statement, that the subject of the statement refers to the resource identified by `exproducts:item10245`, the predicate of the statement refers to the resource identified by `exterms:weight`, and the object of the statement refers to the decimal value identified by the typed literal `"2.4"^^xsd:decimal`. Assuming that the original statement is actually identified by `exproducts:triple12345`, it should be clear by comparing the original statement with the reification that the reification actually does describe it. The conventional use of the RDF reification vocabulary always involves describing a statement using four statements in this pattern; the four statements are sometimes referred to as a "reification quad" for this reason.

Using reification according to this convention, example.com could record the fact that John Smith made the original statement about the tent's weight by first assigning the original statement a URIref (such as `exproducts:triple12345` as before), describing that statement using the reification just described, and then adding an additional statement that `exproducts:triple12345` was written by John Smith (using a URIref to identify which John Smith is being referred to). The resulting statements would be:

```
exproducts:triple12345   rdf:type        rdf:Statement .
exproducts:triple12345   rdf:subject     exproducts:item10245 .
exproducts:triple12345   rdf:predicate   exterms:weight .
exproducts:triple12345   rdf:object      "2.4"^^xsd:decimal .
exproducts:triple12345   dc:creator      exstaff:85740 .
```

The original statement, together with the reification and the attribution of the statement to John Smith, forms the graph shown in Figure 17:

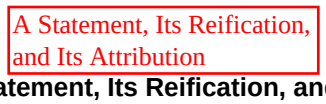


Figure 17: A Statement, Its Reification, and Its Attribution

This graph could be written in RDF/XML as shown in Example 19:

Example 19: RDF/XML for the Reification Example

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:dc="http://purl.org/dc/elements/1.1/"
            xmlns:exterms="http://www.example.com/terms/"
            xml:base="http://www.example.com/2002/04/products">

  <rdf:Description rdf:ID="item10245">
     <exterms:weight rdf:datatype="&xsd;decimal">2.4</exterms:weight>
  </rdf:Description>

  <rdf:Statement rdf:about="#triple12345">
     <rdf:subject rdf:resource="http://www.example.com/2002/04/products#item10245"/>
     <rdf:predicate rdf:resource="http://www.example.com/terms/weight"/>
     <rdf:object rdf:datatype="&xsd;decimal">2.4</rdf:object>

     <dc:creator rdf:resource="http://www.example.com/staffid/85740"/>
  </rdf:Statement>

</rdf:RDF>
```

Section 3.2 introduced the use of the `rdf:ID` attribute in RDF/XML in an `rdf:Description` element to abbreviate the URIref of the subject of a statement. `rdf:ID` can also be used in a property element to automatically produce a reification of the triple that the property element generates. Example 20 shows how this could be used to produce the same graph as Example 19:

Example 20: Generating Reifications using `rdf:ID`

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:dc="http://purl.org/dc/elements/1.1/"
            xmlns:exterms="http://www.example.com/terms/"
            xml:base="http://www.example.com/2002/04/products">

  <rdf:Description rdf:ID="item10245">
     <exterms:weight rdf:ID="triple12345" rdf:datatype="&xsd;decimal">2.4
     </exterms:weight>
  </rdf:Description>

  <rdf:Description rdf:about="#triple12345">
     <dc:creator rdf:resource="http://www.example.com/staffid/85740"/>
  </rdf:Description>

</rdf:RDF>
```

In this case, specifying the attribute `rdf:ID="triple12345"` in the `exterms:weight` element results in the original triple describing the tent's weight:

```
exproducts:item10245   exterms:weight   "2.4"^^xsd:decimal .
```

plus the reification triples:

```
exproducts:triple12345   rdf:type        rdf:Statement .
exproducts:triple12345   rdf:subject     exproducts:item10245 .
exproducts:triple12345   rdf:predicate   exterms:weight .
exproducts:triple12345   rdf:object      "2.4"^^xsd:decimal .
```

The subject of these reification triples is a URIref formed by concatenating the base URI of the document (given in the `xml:base` declaration), the character "#" (to indicate that what follows is a fragment identifier), and the value of the `rdf:ID` attribute; that is, the triples have the same subject `exproducts:triple12345` as in the previous examples.

Note that asserting the reification is not the same as asserting the original statement, and neither implies the other. That is, when someone says that John said something about the weight of a tent, they are not making a statement about the weight of a tent themselves, they are making a statement about something John said. Conversely, when someone describes the weight of a tent, they are not also making a statement about a statement they made (since they may have no intention of talking about things called "statements").

The text above deliberately referred in a number of places to "the conventional use of reification". As noted earlier, care is needed when using the RDF reification vocabulary because it is easy to imagine that the vocabulary defines some things that are not actually defined. While there are applications that successfully use reification, they do so by following some conventions, and making some assumptions, that are in addition to the actual meaning that RDF defines for the reification vocabulary, and the actual facilities that RDF provides to support it.

For one thing, it is important to note that in the conventional use of reification, the subject of the reification triples is assumed to identify a *particular instance* of a triple in a particular RDF document, rather than some arbitrary triple having the same subject, predicate, and object. This particular convention is used because reification is intended for expressing properties such as dates of composition and source information, as in the examples given already, and these properties need to be applied to specific instances of triples. There could be several triples that have the same subject, predicate, and object and, although a graph is defined as a *set* of triples, several instances with the same triple structure might occur in different documents. Thus, to fully support this convention, there needs to be some means of associating the subject of the reification triples with *an individual triple in some document*. However, RDF provides no way to do this.

For instance, in the examples above, there is no explicit information in either the triples or the RDF/XML that actually indicates that the original statement describing the tent's weight is the resource `exproducts:triple12345`, the resource that is the subject of the four reification statements and the statement that John Smith created it. This can be seen by looking at the drawn graph shown in Figure 17. The original statement is certainly part of this graph, but as far as the information in the graph is concerned, `exproducts:triple12345` is a separate resource, rather than identifying that part of the graph. RDF does not provide a built-in way of indicating how a URIref like `exproducts:triple12345` is associated with a particular statement or graph, any more than it provides a built-in way of indicating how a URIref like `exproducts:item10245` is associated with an actual tent. Associating specific URIrefs with specific resources (statements in this case) must be done using mechanisms outside of RDF.

Using `rdf:ID` as shown in Example 20 generates the reification automatically, and provides a convenient way of indicating the URIref to be used as the subject of the statements in the reification. Moreover, it provides a partial "hook" relating the triples in the reification with the piece of RDF/XML syntax that caused them to be created, since the value `triple12345` of the `rdf:ID` attribute is used to generate the URIref of the subject of the reification triples. However, this relationship is once again outside RDF, since there is nothing in the resulting triples that explicitly says that the original triple had the URIref `exproducts:triple12345` (RDF does not assume there is any relationship between a URIref and any RDF/XML that it might have been used or abbreviated in).

The lack of a built-in means for assigning URIrefs to statements does not mean that "provenance" information of this kind cannot be expressed in RDF, just that it cannot be done using only the meaning RDF associates with the reification vocabulary. For example, if an RDF document (say, a Web page) has a URI, statements could be made about the resource identified by that URI and, based on some application-dependent understanding of how those statements should be interpreted, an application could act as if those statements "distribute" over (apply equally to) all the statements in the document. Also, if some mechanism exists (outside of RDF) to assign URIs to individual RDF statements, then statements could certainly be made about those individual statements, using their URIs to identify them. However, in these cases, it would also not be strictly necessary to use the reification vocabulary in the conventional way.

To see this, assuming the original statement:

```
exproducts:item10245   exterms:weight   "2.4"^^xsd:decimal .
```

had a URIref of `exproducts:triple12345`, the statement could be attributed to John Smith simply by the statement:

```
exproducts:triple12345   dc:creator   exstaff:85740 .
```

with no use of the reification vocabulary (although the description of `exproducts:triple12345` as having `rdf:type rdf:Statement` might also be helpful).

In addition, the reification vocabulary could be used directly according to the convention described above, along with an application-dependent understanding as to how to associate specific triples with their reifications. However, other applications receiving this RDF would not necessarily share this application-dependent understanding, and thus would not necessarily interpret the graphs appropriately.

It is also important to note that the interpretation of reification described here is not the same as "quotation", as found in some languages. Instead, the reification describes the relationship between a particular instance of a triple and the resources the triple refers to. The reification can be read intuitively as saying "this RDF triple talks about these things", rather than (as in quotation) "this RDF triple has this form." For instance, in the reification example used in this section, the triple:

```
exproducts:triple12345   rdf:subject   exproducts:item10245 .
```

describing the `rdf:subject` of the original statement says that the subject of the statement is the resource (the tent) identified by the URIref `exproducts:item10245`. It does *not* say that the subject of the statement is the URIref itself (i.e., a string beginning with certain characters), as quotation would do.

4.4 More on Structured Values: rdf:value

Section 2.3 noted that the RDF model intrinsically supports only *binary* relations; that is, a statement specifies a relation between two resources. For example, the statement:

```
exstaff:85740   exterms:manager   exstaff:62345 .
```

states that the relation `exterms:manager` holds between two employees (presumably one manages the other).

However, in some cases it is necessary to represent information involving higher arity relations (relations between more than two resources) in RDF. Section 2.3 discussed one example of this, where the problem was to represent the relationship between John Smith and his address information, and the value of John's address was a structured value of his street, city, state, and postal code. Writing this as a relation shows that this address is a 5-ary relation of the form:

```
address(exstaff:85740, "1501 Grant Avenue", "Bedford", "Massachusetts", "01730")
```

Section 2.3 noted that this kind of structured information can be represented in RDF by considering the aggregate thing be described

(here, the group of components representing John's address) as a separate resource, and then making separate statements about that new resource, as in the triples:

```
exstaff:85740   exterms:address        _:johnaddress .
_:johnaddress   exterms:street         "1501 Grant Avenue" .
_:johnaddress   exterms:city           "Bedford" .
_:johnaddress   exterms:state          "Massachusetts" .
_:johnaddress   exterms:postalCode     "01730" .
```

(where `_:johnaddress` is the blank node identifier of the blank node representing John's address.)

This is a general way to represent any n-ary relation in RDF: select one of the participants (John in this case) to serve as the subject of the original relation (`address` in this case), then specify an intermediate resource to represent the rest of the relation (either with or without assigning it a URI), then give that new resource properties representing the remaining components of the relation.

In the case of John's address, none of the individual parts of the structured value could be considered the "main" value of the `exterms:address` property; all of the parts contribute equally to the value. However, in some cases one of the parts of the structured value is often thought of as the "main" value, with the other parts of the relation providing additional contextual or other information that qualifies the main value. For instance, in Example 9 in Section 3.2, the weight of a particular tent was given as the decimal value *2.4* using a typed literal, i.e.,

```
exproduct:item10245   exterms:weight   "2.4"^^xsd:decimal .
```

In fact, a more complete description of the weight would have been *2.4 kilograms* rather than just the decimal value *2.4*. To state this, the value of the `exterms:weight` property would need to have two components, the typed literal for the decimal value and an indication of the unit of measure (kilograms). In this situation the decimal value could be considered the "main" value of the `exterms:weight` property, because frequently the value would be recorded simply as the typed literal (as in the triple above), relying on an understanding of the context to fill in the unstated units information.

In the RDF model a qualified property value of this kind can be considered as simply another kind of structured value. To represent this, a separate resource could be used to represent the structured value as a whole (the weight, in this case), and to serve as the object of the original statement. That resource could then be given properties representing the individual parts of the structured value. In this case, there should be a property for the typed literal representing the decimal value, and a property for the unit. RDF provides a predefined `rdf:value` property to describe the main value (if there is one) of a structured value. So in this case, the typed literal could be given as the value of the `rdf:value` property, and the resource `exunits:kilograms` as the value of an `exterms:units` property (assuming the resource `exunits:kilograms` is defined as part of example.org's vocabulary). The resulting triples would be:

```
exproduct:item10245   exterms:weight   _:weight10245 .
_:weight10245         rdf:value        "2.4"^^xsd:decimal .
_:weight10245         exterms:units    exunits:kilograms .
```

which can be expressed using the RDF/XML shown in Example 21:

Example 21: RDF/XML using `rdf:value`

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:exterms="http://www.example.org/terms/">

  <rdf:Description rdf:about="http://www.example.com/2002/04/products#item10245">
     <exterms:weight rdf:parseType="Resource">
       <rdf:value rdf:datatype="&xsd;decimal">2.4</rdf:value>
       <exterms:units rdf:resource="http://www.example.org/units/kilograms"/>
     </exterms:weight>
  </rdf:Description>

</rdf:RDF>
```

Example 21 also illustrates a second use of the `rdf:parseType` attribute introduced in Section 4.2, in this case, `rdf:parseType="Resource"`. An `rdf:parseType="Resource"` attribute is used to indicate that the contents of an element are to be interpreted as the description of a new (blank node) resource, without actually having to write a nested `rdf:Description` element. In this case, the `rdf:parseType="Resource"` attribute used in the `exterms:weight` property element indicates that a blank node is to be created as the value of the `exterms:weight` property, and that the enclosed elements (`rdf:value` and

exterms:units) describe properties of that blank node. Further details on rdf:parseType="Resource" are given in [RDF-SYNTAX].

The same approach can be used to represent quantities using any units of measure, as well as values taken from different classification schemes or rating systems, by using the rdf:value property to give the main value, and using additional properties to identify the classification scheme or other information that further describes the value.

There is no need to use rdf:value for these purposes (e.g., a user-defined property name, such as exterms:amount, could have been used instead of rdf:value in Example 21), and RDF does not associate any special meaning with rdf:value. rdf:value is simply provided as a convenience for use in these commonly-occurring situations.

However, even though much existing data in databases and on the Web (and in later Primer examples) takes the form of simple values for properties such as weights, costs, etc., the principle that such simple values are often insufficient to adequately describe these values is an important one. In a global environment such as the Web, it is generally *not* safe to make the assumption that anyone accessing a property value will understand the units being used (or other contextually-dependent information that may be involved). For example, a U.S. site might give a weight value in pounds, but someone accessing that data from outside the U.S. might assume that weights are given in kilograms. The correct interpretation of data in the Web environment may require that additional information (such as units information) be explicitly recorded. This can be done in many ways, such as using rdf:value, building units into property names (e.g., exterms:weightInKg), defining specialized datatypes that include units information (e.g., extypes:kilograms), or adding additional user-defined properties to specify this information (e.g., exterms:unitOfWeight), either in descriptions of individual items or products, in descriptions of sets of data (e.g., all the data in a catalog or on a site), or in schemas (see Section 5).

4.5 XML Literals

Sometimes the value of a property needs to be a fragment of XML, or text that might contain XML markup. For example, a publisher might maintain RDF metadata that includes the titles of books and articles. While such titles are often just simple strings of characters, this is not always the case. For instance, the titles of books on mathematics may contain mathematical formulas that could be represented using MathML [MATHML]. Titles might also include markup for other reasons, such as for Ruby annotations [RUBY], or for bidirectional rendering or special glyph variants (see, e.g., [CHARMOD]).

RDF/XML provides a special notation to make it easy to write literals of this kind. This is done using a third value of the rdf:parseType attribute. Giving an element the attribute rdf:parseType="Literal" indicates that the contents of the element are to be interpreted as an XML fragment. Example 22 illustrates the use of rdf:parseType="Literal":

Example 22: RDF/XML for an XML Literal

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:dc="http://purl.org/dc/elements/1.1/"
            xml:base="http://www.example.com/books">

  <rdf:Description rdf:ID="book12345">
     <dc:title rdf:parseType="Literal">
       <span xml:lang="en">
         The <em>&lt;br /&gt;</em> Element Considered Harmful.
       </span>
     </dc:title>
  </rdf:Description>

</rdf:RDF>
```

The RDF/XML in Example 22 describes a graph containing a single triple with subject ex:book12345, and predicate dc:title. The rdf:parseType="Literal" attribute in the RDF/XML indicates that all the XML within the <dc:title> element is an XML fragment that is the value of the dc:title property. In the graph, this value is a typed literal, whose datatype, rdf:XMLLiteral, is defined in [RDF-CONCEPTS] specifically to represent fragments of XML (including character sequences that may or may not include XML markup). The XML fragment is canonicalized according to the XML Exclusive Canonicalization recommendation [XML-XC14N]. This causes declarations of used namespaces to be added to the fragment, the uniform escaping or unescaping of characters, the expansion of empty-element tags, and other transformations. (For these reasons, and the fact that the triples notation itself requires further escaping, the actual typed literal is not shown here. RDF/XML provides the rdf:parseType="Literal" attribute so that RDF users will *not* have to deal directly with these transformations. Those interested in the details should consult [RDF-CONCEPTS] and [RDF-SYNTAX].) Contextual attributes, such as xml:lang and xml:base are not inherited from the RDF/XML document, and, if required, must, as shown in the example, be explicitly specified in the XML fragment.

This example illustrates that care must be taken in designing RDF data. It might appear at first glance that titles are simple strings

best represented as plain literals, and only later might it be discovered that some titles contain markup. In cases where the value of a property may sometimes contain markup and sometimes not, either `rdf:parseType="Literal"` should be used throughout, or software must handle both plain literals and literals of type `rdf:XMLLiteral` as values of the property.

5. Defining RDF Vocabularies: RDF Schema

RDF provides a way to express simple statements about resources, using named properties and values. However, RDF user communities also need the ability to define the *vocabularies* (terms) they intend to use in those statements, specifically, to indicate that they are describing specific kinds or classes of resources, and will use specific properties in describing those resources. For example, the company example.com from the examples in Section 3.2 would want to describe classes such as `exterms:Tent`, and use properties such as `exterms:model`, `exterms:weightInKg`, and `exterms:packedSize` to describe them (QNames with various "example" namespace prefixes are used as the names of classes and properties here as a reminder that in RDF these names are actually *URI references*, as discussed in Section 2.1). Similarly, people interested in describing bibliographic resources would want to describe classes such as `ex2:Book` or `ex2:MagazineArticle`, and use properties such as `ex2:author`, `ex2:title`, and `ex2:subject` to describe them. Other applications might need to describe classes such as `ex3:Person` and `ex3:Company`, and properties such as `ex3:age`, `ex3:jobTitle`, `ex3:stockSymbol`, and `ex3:numberOfEmployees`. RDF itself provides no means for defining such application-specific classes and properties. Instead, such classes and properties are described as an RDF vocabulary, using extensions to RDF provided by the RDF Vocabulary Description Language 1.0: RDF Schema [RDF-VOCABULARY], referred to here as *RDF Schema*.

RDF Schema does not provide a vocabulary of application-specific classes like `exterms:Tent`, `ex2:Book`, or `ex3:Person`, and properties like `exterms:weightInKg`, `ex2:author` or `ex3:JobTitle`. Instead, it provides the facilities needed to *describe* such classes and properties, and to indicate which classes and properties are expected to be used together (for example, to say that the property `ex3:jobTitle` will be used in describing a `ex3:Person`). In other words, RDF Schema provides a *type system* for RDF. The RDF Schema type system is similar in some respects to the type systems of object-oriented programming languages such as Java. For example, RDF Schema allows resources to be defined as instances of one or more *classes*. In addition, it allows classes to be organized in a hierarchical fashion; for example a class `ex:Dog` might be defined as a subclass of `ex:Mammal` which is a subclass of `ex:Animal`, meaning that any resource which is in class `ex:Dog` is also implicitly in class `ex:Animal` as well. However, RDF classes and properties are in some respects very different from programming language types. RDF class and property descriptions do not create a straightjacket into which information must be forced, but instead provide additional information about the RDF resources they describe. This information can be used in a variety of ways, which will be discussed in Section 5.3.

The RDF Schema facilities are themselves provided in the form of an RDF vocabulary; that is, as a specialized set of predefined RDF resources with their own special meanings. The resources in the RDF Schema vocabulary have URIrefs with the prefix `http://www.w3.org/2000/01/rdf-schema#` (conventionally associated with the QName prefix `rdfs:`). Vocabulary descriptions (schemas) written in the RDF Schema language are legal RDF graphs. Hence, RDF software that is not written to also process the additional RDF Schema vocabulary can still interpret a schema as a legal RDF graph consisting of various resources and properties, but will not "understand" the additional built-in meanings of the RDF Schema terms. To understand these additional meanings, RDF software must be written to process an extended language that includes not only the `rdf:` vocabulary, but also the `rdfs:` vocabulary, together with their built-in meanings. This point will be illustrated in the next section.

The following sections will illustrate RDF Schema's basic resources and properties.

5.1 Describing Classes

A basic step in any kind of description process is identifying the various kinds of things to be described. RDF Schema refers to these "kinds of things" as *classes*. A *class* in RDF Schema corresponds to the generic concept of a *Type* or *Category*, somewhat like the notion of a class in object-oriented programming languages such as Java. RDF classes can be used to represent almost any category of thing, such as Web pages, people, document types, databases or abstract concepts. Classes are described using the RDF Schema resources `rdfs:Class` and `rdfs:Resource`, and the properties `rdf:type` and `rdfs:subClassOf`.

For example, suppose an organization `example.org` wanted to use RDF to provide information about different kinds of motor vehicles. In RDF Schema, `example.org` would first need a class to represent the category of things that are motor vehicles. The resources that belong to a class are called its *instances*. In this case, `example.org` intends for the instances of this class to be resources that are motor vehicles.

In RDF Schema, a *class* is any resource having an `rdf:type` property whose value is the resource `rdfs:Class`. So the motor vehicle class would be described by assigning the class a URIref, say `ex:MotorVehicle` (using `ex:` to stand for the URIref `http://www.example.org/schemas/vehicles`, which is used as the prefix for URIrefs from example.org's vocabulary) and describing that resource with an `rdf:type` property whose value is the resource `rdfs:Class`. That is, `example.org` would write the RDF statement:

```
ex:MotorVehicle   rdf:type   rdfs:Class .
```

As indicated in Section 3.2, the property `rdf:type` is used to indicate that a resource is an instance of a class. So, having described `ex:MotorVehicle` as a class, resource `exthings:companyCar` would be described as a motor vehicle by the RDF statement:

```
exthings:companyCar   rdf:type   ex:MotorVehicle .
```

(This statement uses a common convention that class names are written with an initial uppercase letter, while property and instance names are written with an initial lowercase letter. However, this convention is not required in RDF Schema. The statement also assumes that `example.org` has decided to define separate vocabularies for classes of things, and instances of things.)

The resource `rdfs:Class` itself has an `rdf:type` of `rdfs:Class`. A resource may be an instance of more than one class.

After describing class `ex:MotorVehicle`, `example.org` might want to describe additional classes representing various specialized kinds of motor vehicle, e.g., passenger vehicles, vans, minivans, and so on. These classes can be described in the same way as class `ex:MotorVehicle`, by assigning a URIref for each new class, and writing RDF statements describing these resources as classes, e.g., writing:

```
ex:Van     rdf:type   rdfs:Class .
ex:Truck   rdf:type   rdfs:Class .
```

and so on. However, these statements by themselves only describe the individual classes. `example.org` may also want to indicate their special relationship to class `ex:MotorVehicle`, i.e., that they are specialized *kinds* of MotorVehicle.

This kind of specialization relationship between two classes is described using the predefined `rdfs:subClassOf` property to relate the two classes. For example, to state that `ex:Van` is a specialized kind of `ex:MotorVehicle`, `example.org` would write the RDF statement:

```
ex:Van   rdfs:subClassOf   ex:MotorVehicle .
```

The meaning of this `rdfs:subClassOf` relationship is that any instance of class `ex:Van` is also an instance of class `ex:MotorVehicle`. So if resource `exthings:companyVan` is an instance of `ex:Van` then, based on the declared `rdfs:subClassOf` relationship, RDF software written to understand the RDF Schema vocabulary can infer the additional information that `exthings:companyVan` is also an instance of `ex:MotorVehicle`.

This example of `exthings:companyVan` illustrates the point made earlier about RDF Schema defining an extended language. RDF itself does not define the special meaning of terms from the RDF Schema vocabulary such as `rdfs:subClassOf`. So if an RDF schema defines this `rdfs:subClassOf` relationship between `ex:Van` and `ex:MotorVehicle`, RDF software not written to understand the RDF Schema terms would recognize this as a triple, with predicate `rdfs:subClassOf`, but it would not understand the special significance of `rdfs:subClassOf`, and not be able to draw the additional inference that `exthings:companyVan` is also an instance of `ex:MotorVehicle`.

The `rdfs:subClassOf` property is *transitive*. This means, for example, that given the RDF statements:

```
ex:Van       rdfs:subClassOf   ex:MotorVehicle .
ex:MiniVan   rdfs:subClassOf   ex:Van .
```

RDF Schema defines `ex:MiniVan` as also being a subclass of `ex:MotorVehicle`. As a result, RDF Schema defines resources that are instances of class `ex:MiniVan` as also being instances of class `ex:MotorVehicle` (as well as being instances of class `ex:Van`). A class may be a subclass of more than one class (for example, `ex:MiniVan` may be a subclass of both `ex:Van` and `ex:PassengerVehicle`). RDF Schema defines all classes as subclasses of class `rdfs:Resource` (since the instances belonging to all classes are resources).

Figure 18 shows the full class hierarchy being discussed in these examples.

A Vehicle Class Hierarchy

Figure 18: A Vehicle Class Hierarchy

(To simplify the figure, the `rdf:type` properties relating each of the classes to `rdfs:Class` are omitted in Figure 18. In fact, RDF Schema defines both the subjects and objects of statements that use the `rdfs:subClassOf` property to be resources of type `rdfs:Class`, so this information could be inferred. However, in actually writing schemas, it is good practice to explicitly provide this information.)

This schema could also be described by the triples:

```
ex:MotorVehicle       rdf:type          rdfs:Class .
ex:PassengerVehicle   rdf:type          rdfs:Class .
ex:Van                rdf:type          rdfs:Class .
ex:Truck              rdf:type          rdfs:Class .
ex:MiniVan            rdf:type          rdfs:Class .

ex:PassengerVehicle   rdfs:subClassOf   ex:MotorVehicle .
ex:Van                rdfs:subClassOf   ex:MotorVehicle .
ex:Truck              rdfs:subClassOf   ex:MotorVehicle .

ex:MiniVan            rdfs:subClassOf   ex:Van .
ex:MiniVan            rdfs:subClassOf   ex:PassengerVehicle .
```

Example 23 shows how this schema could be written in RDF/XML.

Example 23: The Vehicle Class Hierarchy in RDF/XML

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://example.org/schemas/vehicles">

<rdf:Description rdf:ID="MotorVehicle">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdf:Description>

<rdf:Description rdf:ID="PassengerVehicle">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>

<rdf:Description rdf:ID="Truck">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>

<rdf:Description rdf:ID="Van">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>

<rdf:Description rdf:ID="MiniVan">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#Van"/>
  <rdfs:subClassOf rdf:resource="#PassengerVehicle"/>
</rdf:Description>

</rdf:RDF>
```

As discussed in Section 3.2 in connection with Example 13, RDF/XML provides an abbreviation for describing resources having an `rdf:type` property (*typed nodes*). Since RDF Schema classes are RDF resources, this abbreviation can be applied to the description of classes. Using this abbreviation, the schema could also be described as shown in Example 24:

Example 24: The Vehicle Class Hierarchy Using the Typed Node Abbreviation

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://example.org/schemas/vehicles">

<rdfs:Class rdf:ID="MotorVehicle"/>

<rdfs:Class rdf:ID="PassengerVehicle">
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdfs:Class>
```

```
<rdfs:Class rdf:ID="Truck">
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Van">
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdfs:Class>

<rdfs:Class rdf:ID="MiniVan">
  <rdfs:subClassOf rdf:resource="#Van"/>
  <rdfs:subClassOf rdf:resource="#PassengerVehicle"/>
</rdfs:Class>

</rdf:RDF>
```

Similar typed node abbreviations will be used throughout the rest of this section.

The RDF/XML in Example 23 and Example 24 introduces names, such as `MotorVehicle`, for the resources (classes) that it describes using `rdf:ID`, to give the effect of "assigning" URIrefs relative to the schema document as described in Section 3.2. `rdf:ID` is useful here because it both abbreviates the URIrefs, and also provides an additional check that the value of the `rdf:ID` attribute is unique against the current base URI (usually the document URI). This helps pick up repeated `rdf:ID` values when defining the names of classes and properties in RDF schemas. Relative URIrefs based on these names can then be used in other class definitions within the same schema (e.g., as `#MotorVehicle` is used in the description of the other classes). The full URIref of this class, assuming that the schema itself was the resource `http://example.org/schemas/vehicles`, would be `http://example.org/schemas/vehicles#MotorVehicle` (shown in Figure 18). As noted in Section 3.2, to ensure that the references to these schema classes would be consistently maintained even if the schema were relocated or copied (or to simply assign a base URIref for the schema classes without assuming they are all published at a single location), the class descriptions could also include an explicit `xml:base="http://example.org/schemas/vehicles"` declaration. Use of an explicit `xml:base` declaration is considered good practice, and one is provided in both examples.

To refer to these classes in RDF instance data (e.g., data describing individual vehicles of these classes) located elsewhere, `example.org` would need to identify the classes either by writing absolute URIrefs, by using relative URIrefs together with an appropriate `xml:base` declaration, or by using QNames together with an appropriate namespace declaration that allows the QNames to be expanded to the proper URIrefs. For example, the resource `exthings:companyCar` could be described as an instance of the class `ex:MotorVehicle` described in the schema of Example 24 by the RDF/XML shown in Example 25 :

Example 25: An Instance of `ex:MotorVehicle`

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:ex="http://example.org/schemas/vehicles#"
  xml:base="http://example.org/things">

   <ex:MotorVehicle rdf:ID="companyCar"/>

</rdf:RDF>
```

Note that the QName `ex:MotorVehicle`, when expanded using the namespace declaration `xmlns:ex="http://example.org/schemas/vehicles#"`, becomes the full URIref `http://example.org/schemas/vehicles#MotorVehicle`, which is the correct URIref for the `MotorVehicle` class as shown in Figure 18. The `xml:base` declaration `xml:base="http://example.org/things"` is provided to allow the `rdf:ID="companyCar"` to expand to the proper `exthings:companyCar` URIref (since a QName cannot be used as the value of the `rdf:ID` attribute).

5.2 Describing Properties

In addition to describing the specific *classes* of things they want to describe, user communities also need to be able to describe specific *properties* that characterize those classes of things (such as `rearSeatLegRoom` to describe a passenger vehicle). In RDF Schema, properties are described using the RDF class `rdf:Property`, and the RDF Schema properties `rdfs:domain`, `rdfs:range`, and `rdfs:subPropertyOf`.

All properties in RDF are described as instances of class `rdf:Property`. So a new property, such as `exterms:weightInKg`, is described by assigning the property a URIref, and describing that resource with an `rdf:type` property whose value is the resource `rdf:Property`, for example, by writing the RDF statement:

```
exterms:weightInKg   rdf:type   rdf:Property .
```

RDF Schema also provides vocabulary for describing how properties and classes are intended to be used together in RDF data. The most important information of this kind is supplied by using the RDF Schema properties `rdfs:range` and `rdfs:domain` to further describe application-specific properties.

The `rdfs:range` property is used to indicate that the values of a particular property are instances of a designated class. For example, if `example.org` wanted to indicate that the property `ex:author` had values that are instances of class `ex:Person`, it would write the RDF statements:

```
ex:Person   rdf:type     rdfs:Class .
ex:author   rdf:type     rdf:Property .
ex:author   rdfs:range   ex:Person .
```

These statements indicate that `ex:Person` is a class, `ex:author` is a property, and that RDF statements using the `ex:author` property have instances of `ex:Person` as objects.

A property, say `ex:hasMother`, can have zero, one, or more than one range property. If `ex:hasMother` has no range property, then nothing is said about the values of the `ex:hasMother` property. If `ex:hasMother` has one range property, say one specifying `ex:Person` as the range, this says that the values of the `ex:hasMother` property are instances of class `ex:Person`. If `ex:hasMother` has more than one range property, say one specifying `ex:Person` as its range, and another specifying `ex:Female` as its range, this says that the values of the `ex:hasMother` property are resources that are instances of *all* of the classes specified as the ranges, i.e., that any value of `ex:hasMother` is *both* a `ex:Female` *and* a `ex:Person`.

This last point may not be obvious. However, stating that the property `ex:hasMother` has the two ranges `ex:Female` and `ex:Person` involves making two separate statements:

```
ex:hasMother   rdfs:range   ex:Female .
ex:hasMother   rdfs:range   ex:Person .
```

For any given statement using this property, say:

```
exstaff:frank   ex:hasMother   exstaff:frances .
```

in order for *both* the `rdfs:range` statements to be correct, it must be the case that `exstaff:frances` is *both* an instance of `ex:Female` and of `ex:Person`.

The `rdfs:range` property can also be used to indicate that the value of a property is given by a typed literal, as discussed in Section 2.4. For example, if `example.org` wanted to indicate that the property `ex:age` had values from the XML Schema datatype `xsd:integer`, it would write the RDF statements:

```
ex:age   rdf:type     rdf:Property .
ex:age   rdfs:range   xsd:integer .
```

The datatype `xsd:integer` is identified by its URIref (the full URIref being `http://www.w3.org/2001/XMLSchema#integer`). This URIref can be used without explicitly stating in the schema that it identifies a datatype. However, it is often useful to explicitly state that a given URIref identifies a datatype. This can be done using the RDF Schema class `rdfs:Datatype`. To state that `xsd:integer` is a datatype, `example.org` would write the RDF statement:

```
xsd:integer   rdf:type   rdfs:Datatype .
```

This statement says that `xsd:integer` is the URIref of a datatype (which is assumed to conform to the requirements for RDF datatypes described in [RDF-CONCEPTS]). Such a statement does *not* constitute a *definition* of a datatype, e.g., in the sense that `example.org` is defining a new datatype. There is no way to define datatypes in RDF Schema. As noted in Section 2.4, datatypes are defined externally to RDF (and to RDF Schema), and *referred to* in RDF statements by their URIrefs. This statement simply serves to document the existence of the datatype, and indicate explicitly that it is being used in this schema.

The `rdfs:domain` property is used to indicate that a particular property applies to a designated class. For example, if `example.org` wanted to indicate that the property `ex:author` applies to instances of class `ex:Book`, it would write the RDF statements:

```
ex:Book     rdf:type      rdfs:Class .
ex:author   rdf:type      rdf:Property .
```

```
ex:author   rdfs:domain   ex:Book .
```

These statements indicate that `ex:Book` is a class, `ex:author` is a property, and that RDF statements using the `ex:author` property have instances of `ex:Book` as subjects.

A given property, say `exterms:weight`, may have zero, one, or more than one domain property. If `exterms:weight` has no domain property, then nothing is said about the resources that `exterms:weight` properties may be used with (any resource could have a `exterms:weight` property). If `exterms:weight` has one domain property, say one specifying `ex:Book` as the domain, this says that the `exterms:weight` property applies to instances of class `ex:Book`. If `exterms:weight` has more than one domain property, say one specifying `ex:Book` as the domain and another one specifying `ex:MotorVehicle` as the domain, this says that any resource that has a `exterms:weight` property is an instance of *all* of the classes specified as the domains, i.e., that any resource that has a `exterms:weight` property is both a `ex:Book` *and* a `ex:MotorVehicle` (illustrating the need for care in specifying domains and ranges).

As in the case of `rdfs:range`, this last point may not be obvious. However, stating that the property `exterms:weight` has the two domains `ex:Book` and `ex:MotorVehicle` involves making two separate statements:

```
exterms:weight   rdfs:domain   ex:Book .
exterms:weight   rdfs:domain   ex:MotorVehicle .
```

For any given statement using this property, say:

```
exthings:companyCar   exterms:weight   "2500"^^xsd:integer .
```

in order for *both* the `rdfs:domain` statements to be correct, it must be the case that `exthings:companyCar` is *both* an instance of `ex:Book` and of `ex:MotorVehicle`.

The use of these range and domain descriptions can be illustrated by extending the vehicle schema, adding two properties `ex:registeredTo` and `ex:rearSeatLegRoom`, a new class `ex:Person`, and explicitly describing the datatype `xsd:integer` as a datatype. The `ex:registeredTo` property applies to any `ex:MotorVehicle` and its value is a `ex:Person`. For the sake of this example, `ex:rearSeatLegRoom` applies only to instances of class `ex:PassengerVehicle`. The value is an `xsd:integer` giving the number of centimeters of rear seat legroom. These descriptions are shown in Example 26:

Example 26: Some Property Descriptions for the Vehicle Schema

```
<rdf:Property rdf:ID="registeredTo">
  <rdfs:domain rdf:resource="#MotorVehicle"/>
  <rdfs:range rdf:resource="#Person"/>
</rdf:Property>

<rdf:Property rdf:ID="rearSeatLegRoom">
  <rdfs:domain rdf:resource="#PassengerVehicle"/>
  <rdfs:range rdf:resource="&xsd;integer"/>
</rdf:Property>

<rdfs:Class rdf:ID="Person"/>

<rdfs:Datatype rdf:about="&xsd;integer"/>
```

Note that an `<rdf:RDF>` element is not used in Example 26, because it is assumed this RDF/XML is being added to the vehicle schema described in Example 24. This same assumption also allows the use of relative URIrefs like `#MotorVehicle` to refer to other classes from that schema.

RDF Schema provides a way to specialize *properties* as well as classes. This specialization relationship between two properties is described using the predefined `rdfs:subPropertyOf` property. For example, if `ex:primaryDriver` and `ex:driver` are both properties, `example.org` could describe these properties, and the fact that `ex:primaryDriver` is a specialization of `ex:driver`, by writing the RDF statements:

```
ex:driver          rdf:type             rdf:Property .
ex:primaryDriver   rdf:type             rdf:Property .
ex:primaryDriver   rdfs:subPropertyOf   ex:driver .
```

The meaning of this `rdfs:subPropertyOf` relationship is that if an instance `exstaff:fred` is an `ex:primaryDriver` of the instance `ex:companyVan`, then RDF Schema defines `exstaff:fred` as also being an `ex:driver` of `ex:companyVan`. The RDF/XML describing these properties (assuming again that it is being added to the vehicle schema described in Example 24) is

shown in Example 27.

Example 27: More Properties for the Vehicle Schema

```
<rdf:Property rdf:ID="driver">
  <rdfs:domain rdf:resource="#MotorVehicle"/>
</rdf:Property>

<rdf:Property rdf:ID="primaryDriver">
  <rdfs:subPropertyOf rdf:resource="#driver"/>
</rdf:Property>
```

A property may be a subproperty of zero, one or more properties. All RDF Schema `rdfs:range` and `rdfs:domain` properties that apply to an RDF property also apply to each of its subproperties. So, in the above example, RDF Schema defines `ex:primaryDriver` as also having an `rdfs:domain` of `ex:MotorVehicle`, because of its subproperty relationship to `ex:driver`.

Example 28 shows the RDF/XML for the full vehicle schema, containing all the descriptions given so far:

Example 28: The Full Vehicle Schema

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://example.org/schemas/vehicles">

<rdfs:Class rdf:ID="MotorVehicle"/>

<rdfs:Class rdf:ID="PassengerVehicle">
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Truck">
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Van">
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdfs:Class>

<rdfs:Class rdf:ID="MiniVan">
  <rdfs:subClassOf rdf:resource="#Van"/>
  <rdfs:subClassOf rdf:resource="#PassengerVehicle"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Person"/>

<rdfs:Datatype rdf:about="&xsd;integer"/>

<rdf:Property rdf:ID="registeredTo">
  <rdfs:domain rdf:resource="#MotorVehicle"/>
  <rdfs:range rdf:resource="#Person"/>
</rdf:Property>

<rdf:Property rdf:ID="rearSeatLegRoom">
  <rdfs:domain rdf:resource="#PassengerVehicle"/>
  <rdfs:range rdf:resource="&xsd;integer"/>
</rdf:Property>

<rdf:Property rdf:ID="driver">
  <rdfs:domain rdf:resource="#MotorVehicle"/>
</rdf:Property>

<rdf:Property rdf:ID="primaryDriver">
  <rdfs:subPropertyOf rdf:resource="#driver"/>
</rdf:Property>
```

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

Having shown how to describe classes and properties using RDF Schema, instances using those classes and properties can now be illustrated. For example, Example 29 describes an instance of the `ex:PassengerVehicle` class described in Example 28, together with some hypothetical values for its properties.

Example 29: An Instance of `ex:PassengerVehicle`

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:ex="http://example.org/schemas/vehicles#"
            xml:base="http://example.org/things">

  <ex:PassengerVehicle rdf:ID="johnSmithsCar">
       <ex:registeredTo rdf:resource="http://www.example.org/staffid/85740"/>
       <ex:rearSeatLegRoom
           rdf:datatype="&xsd;integer">127</ex:rearSeatLegRoom>
       <ex:primaryDriver rdf:resource="http://www.example.org/staffid/85740"/>
  </ex:PassengerVehicle>
</rdf:RDF>
```

This example assumes that the instance is described in a separate document from the schema. Since the schema has an `xml:base` of `http://example.org/schemas/vehicles`, the namespace declaration `xmlns:ex="http://example.org/schemas/vehicles#"` is provided to allow QNames such as `ex:registeredTo` in the instance data to be properly expanded to the URIrefs of the classes and properties described in that schema. An `xml:base` declaration is also provided for this instance, to allow `rdf:ID="johnSmithsCar"` to expand to the proper URIref independently of the location of the actual document.

Note that an `ex:registeredTo` property can be used in describing this instance of `ex:PassengerVehicle`, because `ex:PassengerVehicle` is a subclass of `ex:MotorVehicle`. Note also that a typed literal is used for the value of the `ex:rearSetLegRoom` property in this instance, rather than a plain literal (i.e., rather than stating the value as `<ex:rearSeatLegRoom>127</ex:rearSeatLegRoom>`). Because the schema describes the range of this property as an `xsd:integer`, the value of the property should be a typed literal of that datatype in order to match the range description (i.e., the range declaration does not automatically "assign" a datatype to a plain literal, and so a typed literal of the appropriate datatype must be explicitly provided). Additional information, either in the schema, or in additional instance data, could also be provided to explicitly specify the *units* of the `ex:rearSetLegRoom` property (centimeters), as discussed in Section 4.4.

5.3 Interpreting RDF Schema Declarations

As noted earlier, the RDF Schema type system is similar in some respects to the type systems of object-oriented programming languages such as Java. However, RDF differs from most programming language type systems in several important respects.

One important difference is that instead of describing a class as having a collection of specific properties, an RDF schema describes properties as applying to specific classes of resources, using *domain* and *range* properties. For example, a typical object-oriented programming language might define a class `Book` with an attribute called `author` having values of type `Person`. A corresponding RDF schema would describe a class `ex:Book`, and, in a separate description, a property `ex:author` having a domain of `ex:Book` and a range of `ex:Person`.

The difference between these approaches may seem to be only syntactic, but in fact there is an important difference. In the programming language class description, the attribute `author` is part of the description of class `Book`, and applies *only* to instances of class `Book`. Another class (say, `softwareModule`) might also have an attribute called `author`, but this would be considered a *different* attribute. In other words, the *scope* of an attribute description in most programming languages is restricted to the class or type in which it is defined. In RDF, on the other hand, property descriptions are, by default, *independent* of class definitions, and have, by default, *global* scope (although they may optionally be declared to apply only to certain classes using domain specifications).

As a result, an RDF schema could describe a property `exterms:weight` without a domain being specified. This property could then be used to describe instances of any class that might be considered to have a weight. One benefit of the RDF property-based approach is that it becomes easier to extend the use of property definitions to situations that might not have been anticipated in the original description. At the same time, this is a "benefit" which must be used with care, to insure that properties are not mis-applied in inappropriate situations.

Another result of the global scope of RDF property descriptions is that it is not possible in an RDF schema to define a specific property as having locally-different ranges depending on the class of the resource it is applied to. For example, in defining the property `ex:hasParent`, it would be desirable to be able to say that if the property is used to describe a resource of class

ex:Human, then the range of the property is also a resource of class ex:Human, while if the property is used to describe a resource of class ex:Tiger, then the range of the property is also a resource of class ex:Tiger. This kind of definition is not possible in RDF Schema. Instead, any range defined for an RDF property applies to *all* uses of the property, and so ranges should be defined with care. However, while such locally-different ranges cannot be defined in RDF Schema, they can be defined in some of the richer schema languages discussed in Section 5.5.

Another important difference is that RDF Schema descriptions are not necessarily *prescriptive* in the way programming language type declarations typically are. For example, if a programming language declares a class Book with an author attribute having values of type Person, this is usually interpreted as a group of *constraints*. The language will not allow the creation of an instance of Book without an author attribute, and it will not allow an instance of Book with an author attribute that does not have a Person as its value. Moreover, if author is the *only* attribute defined for class Book, the language will not allow an instance of Book with some other attribute.

RDF Schema, on the other hand, provides schema information as additional *descriptions* of resources, but does not prescribe how these descriptions should be used by an application. For example, suppose an RDF schema states that an ex:author property has an rdfs:range of class ex:Person. This is simply an RDF statement that RDF statements containing ex:author properties have instances of ex:Person as objects.

This schema-supplied information might be used in different ways. One application might interpret this statement as specifying part of a template for RDF data it is creating, and use it to ensure that any ex:author property has a value of the indicated (ex:Person) class. That is, this application interprets the schema description as a *constraint* in the same way that a programming language might. However, another application might interpret this statement as providing additional information about data it is receiving, information which may not be provided explicitly in the original data. For example, this second application might receive some RDF data that includes an ex:author property whose value is a resource of unspecified class, and use this schema-provided statement to conclude that the resource must be an instance of class ex:Person. A third application might receive some RDF data that includes an ex:author property whose value is a resource of class ex:Corporation, and use this schema information as the basis of a warning that "there may be an inconsistency here, but on the other hand there may not be". Somewhere else there may be a declaration that resolves the apparent inconsistency (e.g., a declaration to the effect that "a Corporation is a (legal) Person").

Moreover, depending on how the application interprets the property descriptions, a description of an instance might be considered valid either *without* some of the schema-specified properties (e.g., there might be an instance of ex:Book without an ex:author property, even if ex:author is described as having a domain of ex:Book), or with *additional* properties (there might be an instance of ex:Book with an ex:technicalEditor property, even though the schema describing class ex:Book does not describe such a property).

In other words, statements in an RDF schema are always *descriptions*. They may also be *prescriptive* (introduce constraints), but only if the application interpreting those statements wants to treat them that way. All RDF Schema does is provide a way of stating this additional information. Whether this information conflicts with explicitly specified instance data is up to the application to determine and act upon.

5.4 Other Schema Information

RDF Schema provides a number of other built-in properties, which can be used to provide documentation and other information about an RDF schema or about instances. For example the rdfs:comment property can be used to provide a human-readable description of a resource. The rdfs:label property can be used to provide a more human-readable version of a resource's name. The rdfs:seeAlso property can be used to indicate a resource that might provide additional information about the subject resource. The rdfs:isDefinedBy property is a subproperty of rdfs:seeAlso, and can be used to indicate a resource that (in a sense not specified by RDF; e.g., the resource may not be an RDF schema) "defines" the subject resource. RDF Vocabulary Description Language 1.0: RDF Schema [RDF-VOCABULARY] should be consulted for further discussion of these properties.

As with a number of the built-in RDF properties such as rdf:value, the uses described for these RDF Schema properties are only their *intended* uses. [RDF-SEMANTICS] defines no special meanings for these properties, and RDF Schema does not define any constraints based on these intended uses. For example, there is no constraint specified that the object of a rdfs:seeAlso property *must* provide additional information about the subject of the statement in which it appears.

5.5 Richer Schema Languages

RDF Schema provides basic capabilities for describing RDF vocabularies, but additional capabilities are also possible, and can be useful. These capabilities may be provided through further development of RDF Schema, or in other languages based on RDF. Other richer schema capabilities that have been identified as useful (but that are not provided by RDF Schema) include:

- *cardinality constraints* on properties, e.g., that a Person has *exactly one* biological father.
- specifying that a given property (such as ex:hasAncestor) is *transitive*, e.g., that if A ex:hasAncestor B, and B ex:hasAncestor C, then A ex:hasAncestor C.

- specifying that a given property is a unique identifier (or *key*) for instances of a particular class.
- specifying that two different classes (having different URIrefs) actually represent the same class.
- specifying that two different instances (having different URIrefs) actually represent the same individual.
- specifying constraints on the range or cardinality of a property that depend on the class of resource to which a property is applied, e.g., being able to say that for a soccer team the `ex:hasPlayers` property has 11 values, while for a basketball team the same property should have only 5 values.
- the ability to describe new classes in terms of combinations (e.g., unions and intersections) of other classes, or to say that two classes are disjoint (i.e., that no resource is an instance of both classes).

The additional capabilities mentioned above, in addition to others, are the targets of *ontology* languages such as DAML+OIL [DAML+OIL] and OWL [OWL]. Both these languages are based on RDF and RDF Schema (and both currently provide all the additional capabilities mentioned above). The intent of such languages is to provide additional machine-processable *semantics* for resources, that is, to make the machine representations of resources more closely resemble their intended real world counterparts. While such capabilities are not necessarily needed to build useful applications using RDF (see Section 6 for a description of a number of existing RDF applications), the development of such languages is a very active subject of work as part of the development of the Semantic Web.

6. Some RDF Applications: RDF in the Field

The previous sections have described the general capabilities of RDF and RDF Schema. While examples were used in those sections to illustrate those capabilities, and some of those examples may have suggested potential RDF applications, those sections did not actually discuss any *real* applications. This section will describe some actual deployed RDF applications, showing how RDF supports various real-world requirements to represent and manipulate information about a wide variety of things.

6.1 Dublin Core Metadata Initiative

Metadata is *data about data*. Specifically, the term refers to data used to identify, describe, or locate information resources, whether these resources are physical or electronic. While structured metadata processed by computers is relatively new, the basic concept of metadata has been used for many years in helping manage and use large collections of information. Library card catalogs are a familiar example of such metadata.

The Dublin Core is a set of "elements" (properties) for describing documents (and hence, for recording metadata). The element set was originally developed at the March 1995 Metadata Workshop in Dublin, Ohio. The Dublin Core has subsequently been modified on the basis of later Dublin Core Metadata workshops, and is currently maintained by the Dublin Core Metadata Initiative. The goal of the Dublin Core is to provide a minimal set of descriptive elements that facilitate the description and the automated indexing of document-like networked objects, in a manner similar to a library card catalog. The Dublin Core metadata set is intended to be suitable for use by resource discovery tools on the Internet, such as the "Webcrawlers" employed by popular World Wide Web search engines. In addition, the Dublin Core is meant to be sufficiently simple to be understood and used by the wide range of authors and casual publishers who contribute information to the Internet. Dublin Core elements have become widely used in documenting Internet resources (the Dublin Core `creator` element has already been used in earlier examples). The current elements of the Dublin Core are defined in the Dublin Core Metadata Element Set, Version 1.1: Reference Description [DC], and contain definitions for the following properties:

- **Title**: A name given to the resource.
- **Creator**: An entity primarily responsible for making the content of the resource.
- **Subject**: The topic of the content of the resource.
- **Description**: An account of the content of the resource.
- **Publisher**: An entity responsible for making the resource available
- **Contributor**: An entity responsible for making contributions to the content of the resource.
- **Date**: A date associated with an event in the life cycle of the resource.
- **Type**: The nature or genre of the content of the resource.
- **Format**: The physical or digital manifestation of the resource.
- **Identifier**: An unambiguous reference to the resource within a given context.
- **Source**: A reference to a resource from which the present resource is derived.
- **Language**: A language of the intellectual content of the resource.
- **Relation**: A reference to a related resource.
- **Coverage**: The extent or scope of the content of the resource.
- **Rights**: Information about rights held in and over the resource.

Information using the Dublin Core elements may be represented in any suitable language (e.g., in HTML `meta` elements). However, RDF is an ideal representation for Dublin Core information. The examples below represent the simple description of a set of resources in RDF using the Dublin Core vocabulary. Note that the specific Dublin Core RDF vocabulary shown here is not intended to be authoritative. The Dublin Core Reference Description [DC] is the authoritative reference.

The first example, Example 30, describes a Web site home page using Dublin Core properties:

Example 30: A Web Page Described using Dublin Core Properties

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/elements/1.1/">
    <rdf:Description rdf:about="http://www.dlib.org">
      <dc:title>D-Lib Program - Research in Digital Libraries</dc:title>
      <dc:description>The D-Lib program supports the community of people
       with research interests in digital libraries and electronic
       publishing.</dc:description>
      <dc:publisher>Corporation For National Research Initiatives</dc:publisher>
      <dc:date>1995-01-07</dc:date>
      <dc:subject>
        <rdf:Bag>
          <rdf:li>Research; statistical methods</rdf:li>
          <rdf:li>Education, research, related topics</rdf:li>
          <rdf:li>Library use Studies</rdf:li>
        </rdf:Bag>
      </dc:subject>
      <dc:type>World Wide Web Home Page</dc:type>
      <dc:format>text/html</dc:format>
      <dc:language>en</dc:language>
    </rdf:Description>
</rdf:RDF>
```

Note that both RDF and the Dublin Core define an (XML) element called "Description" (although the Dublin Core element name is written in lowercase). Even if the initial letter were identically uppercase, the XML namespace mechanism enables these two elements to be distinguished (one is `rdf:Description`, and the other is `dc:description`). Also, as a matter of interest, accessing http://purl.org/dc/elements/1.1/ (the namespace URI used to identify the Dublin Core vocabulary in this example) in a Web browser (as of the current writing) will retrieve an RDF Schema declaration for [DC].

The second example, Example 31, describes a published magazine:

Example 31: Describing A Magazine Using Dublin Core

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:dcterms="http://purl.org/dc/terms/">
    <rdf:Description rdf:about="http://www.dlib.org/dlib/may98/05contents.html">
      <dc:title>DLIB Magazine - The Magazine for Digital Library Research
        - May 1998</dc:title>
      <dc:description>D-LIB magazine is a monthly compilation of
       contributed stories, commentary, and briefings.</dc:description>
      <dc:contributor>Amy Friedlander</dc:contributor>
      <dc:publisher>Corporation for National Research Initiatives</dc:publisher>
      <dc:date>1998-01-05</dc:date>
      <dc:type>electronic journal</dc:type>
      <dc:subject>
        <rdf:Bag>
          <rdf:li>library use studies</rdf:li>
          <rdf:li>magazines and newspapers</rdf:li>
        </rdf:Bag>
      </dc:subject>
      <dc:format>text/html</dc:format>
      <dc:identifier rdf:resource="urn:issn:1082-9873"/>
      <dcterms:isPartOf rdf:resource="http://www.dlib.org"/>
    </rdf:Description>
 </rdf:RDF>
```

Example 31 uses (in the third line from the bottom) the Dublin Core *qualifier* `isPartOf` (from a separate vocabulary) to indicate that this magazine is "part of" the previously-described Web site.

The third example, Example 32, describes a specific article in the magazine described in Example 31.

Example 32: Describing a Magazine Article

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:dcterms="http://purl.org/dc/terms/">
    <rdf:Description rdf:about="http://www.dlib.org/dlib/may98/miller/05miller.html">
      <dc:title>An Introduction to the Resource Description Framework</dc:title>
      <dc:creator>Eric J. Miller</dc:creator>
      <dc:description>The Resource Description Framework (RDF) is an
       infrastructure that enables the encoding, exchange and reuse of
       structured metadata. rdf is an application of xml that imposes needed
       structural constraints to provide unambiguous methods of expressing
       semantics. rdf additionally provides a means for publishing both
       human-readable and machine-processable vocabularies designed to
       encourage the reuse and extension of metadata semantics among
       disparate information communities. the structural constraints rdf
       imposes to support the consistent encoding and exchange of
       standardized metadata provides for the interchangeability of separate
       packages of metadata defined by different resource description
       communities. </dc:description>
      <dc:publisher>Corporation for National Research Initiatives</dc:publisher>
      <dc:subject>
        <rdf:Bag>
          <rdf:li>machine-readable catalog record formats</rdf:li>
          <rdf:li>applications of computer file organization and
           access methods</rdf:li>
        </rdf:Bag>
      </dc:subject>
      <dc:rights>Copyright © 1998 Eric Miller</dc:rights>
      <dc:type>Electronic Document</dc:type>
      <dc:format>text/html</dc:format>
      <dc:language>en</dc:language>
      <dcterms:isPartOf
rdf:resource="http://www.dlib.org/dlib/may98/05contents.html"/>
    </rdf:Description>
</rdf:RDF>
```

Example 32 also uses the qualifier `isPartOf`, this time to indicate that this article is "part of" the previously-described magazine.

Computer languages and file formats do not always make explicit provision for embedding metadata with the data it describes. In many cases, the metadata has to be specified as a separate resource and explicitly linked to the data (this has been done for the RDF metadata that describes the Primer; there is an explicit link to this metadata at the end of the Primer). However, applications and languages are increasingly making explicit provision for embedding metadata directly with the data. For example, the W3C's Scalable Vector Graphics language [SVG] (another XML-based language) provides an explicit `metadata` element for recording metadata along with other SVG data. Any XML-based metadata language can be used inside this element. [SVG] includes the example shown in Example 33 of how to embed metadata describing an SVG document in the SVG document itself. The example uses the Dublin Core vocabulary, and RDF/XML for recording the metadata.

Example 33: Including Metadata in an SVG Document

```
<?xml version="1.0"?>
<svg width="4in" height="3in" version="1.1"
    xmlns = 'http://www.w3.org/2000/svg'>
    <desc xmlns:myfoo="http://example.org/myfoo">
      <myfoo:title>This is a financial report</myfoo:title>
      <myfoo:descr>The global description uses markup from the
        <myfoo:emph>myfoo</myfoo:emph> namespace.</myfoo:descr>
      <myfoo:scene><myfoo:what>widget $growth</myfoo:what>
      <myfoo:contains>$three $graph-bar</myfoo:contains>
        <myfoo:when>1998 $through 2000</myfoo:when> </myfoo:scene>
   </desc>
    <metadata>
      <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:dc = "http://purl.org/dc/elements/1.1/" >
        <rdf:Description rdf:about="http://example.org/myfoo"
             dc:title="MyFoo Financial Report"
             dc:description="$three $bar $thousands $dollars $from 1998 $through
2000"
             dc:publisher="Example Organization"
             dc:date="2000-04-11"
             dc:format="image/svg+xml"
             dc:language="en" >
          <dc:creator>
            <rdf:Bag>
              <rdf:li>Irving Bird</rdf:li>
              <rdf:li>Mary Lambert</rdf:li>
            </rdf:Bag>
          </dc:creator>
        </rdf:Description>
      </rdf:RDF>
    </metadata>
</svg>
```

Adobe's Extensible Metadata Platform (XMP) is another example of technology that allows metadata about a file to be embedded into the file itself. XMP uses RDF/XML as the basis of its metadata representation. A number of Adobe products already support XMP.

6.2 PRISM

PRISM: Publishing Requirements for Industry Standard Metadata [PRISM] is a metadata specification developed in the publishing industry. Magazine publishers and their vendors formed the PRISM Working Group to identify the industry's needs for metadata and define a specification to meet them. Publishers want to use existing content in many ways in order to get a greater return on the investment made in creating it. Converting magazine articles to HTML for posting on the Web is one example. Licensing it to aggregators like LexisNexis is another. All of these are "first uses" of the content; typically they all go live at the time the magazine hits the stands. The publishers also want their content to be "evergreen". It might be used in new issues, such as in a retrospective article. It could be used by other divisions in the company, such as in a book compiled from the magazine's photos, recipes, etc. Another use is to license it to outsiders, such as in a reprint of a product review, or in a retrospective produced by a different publisher. This overall goal requires a metadata approach that emphasizes *discovery*, *rights tracking*, and *end-to-end metadata*.

Discovery: Discovery is a general term for finding content which encompasses searching, browsing, content routing, and other techniques. Discussions of discovery frequently center on a consumer searching a public Web site. However, discovering content is much broader than that. The audience may consist of consumers, or it may consist of internal users such as researchers, designers, photo editors, licensing agents, etc. To assist discovery, PRISM provides properties to describe the topics, formats, genre, origin, and contexts of a resource. It also provides means for categorizing resources using multiple subject description taxonomies.

Rights Tracking: Magazines frequently contain material licensed from others. Photos from a stock photo agency are the most common type of licensed material, but articles, sidebars, and all other types of content may be licensed. Simply knowing if content was licensed for one-time use, requires royalty payments, or is wholly-owned by the publisher is a struggle. PRISM provides elements for basic tracking of such rights. A separate vocabulary defined in the PRISM specification supports description of places, times, and industries where content may or may not be used.

End-to-end metadata: Most published content already has metadata created for it. Unfortunately, when content moves between systems, the metadata is frequently discarded, only to be re-created later in the production process at considerable expense. PRISM aims to reduce this problem by providing a specification that can be used in multiple stages in the content production pipeline. An important feature of the PRISM specification is its use of other existing specifications. Rather than create an entirely new thing, the group decided to use existing specifications as much as possible, and only define new things where needed. For this reason, the PRISM specification uses XML, RDF, Dublin Core, and well as various ISO formats and vocabularies.

A PRISM description may be as simple as a few Dublin Core properties with plain literal values. Example 34 describes a photograph, giving basic information on its title, photographer, format, etc.

Example 34: A PRISM Description of a Photograph

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/"
         xml:lang="en-US">

 <rdf:Description rdf:about="http://travel.example.com/2000/08/Corfu.jpg">
  <dc:title>Walking on the Beach in Corfu</dc:title>
```

```
  <dc:description>Photograph taken at 6:00 am on Corfu with two models
  </dc:description>
  <dc:creator>John Peterson</dc:creator>
  <dc:contributor>Sally Smith, lighting</dc:contributor>
  <dc:format>image/jpeg</dc:format>
 </rdf:Description>
</rdf:RDF>
```

PRISM also augments the Dublin Core to allow more detailed descriptions. The augmentations are defined as three new vocabularies, generally cited using the prefixes `prism:`, `pcv:`, and `prl:`.

`prism:` This prefix refers to the main PRISM vocabulary, whose terms use the URI prefix `http://prismstandard.org/namespaces/basic/1.0/`. Most of the properties in this vocabulary are more specific versions of properties from the Dublin Core. For example, more specific versions of `dc:date` are provided by properties like `prism:publicationTime`, `prism:releaseTime`, `prism:expirationTime`, etc.

`pcv:` This prefix refers to the PRISM Controlled Vocabulary (pcv) vocabulary, whose terms use the URI prefix `http://prismstandard.org/namespaces/pcv/1.0/`. Currently, common practice for describing the subject(s) of an article is by supplying descriptive keywords. Unfortunately, simple keywords do not make a great difference in retrieval performance, due to the fact that different people will use different keywords [BATES96]. Best practice is to code the articles with subject terms from a "controlled vocabulary". The vocabulary should provide as many synonyms as possible for its terms in the vocabulary. This way the controlled terms provide a meeting ground for the keywords supplied by the searcher and the indexer. The pcv vocabulary provides properties for specifying terms in a vocabulary, the relations between terms, and alternate names for the terms.

`prl:` This prefix refers to the PRISM Rights Language vocabulary, whose terms use the URI prefix `http://prismstandard.org/namespaces/prl/1.0/`. Digital Rights Management is an area undergoing considerable upheaval. There are a number of proposals for rights management languages, but none are clearly favored throughout the industry. Because there was no clear choice to recommend, the PRISM Rights Language (PRL) was defined as an interim measure. It provides properties which let people say if an item can or cannot be "used", depending on conditions of time, geography, and industry. This is believed to be an 80/20 trade-off which will help publishers begin to save money when tracking rights. It is not intended to be a general rights language, or allow publishers to automatically enforce limits on consumer uses of the content.

PRISM uses RDF because of its abilities for dealing with descriptions of varying complexity. Currently, a great deal of metadata uses simple character string (plain literal) values, such as:

```
<dc:coverage>Greece</dc:coverage>
```

Over time the developers of PRISM expect uses of the PRISM specification to become more sophisticated, moving from simple literal values to more structured values. In fact, that range of values is a situation being faced now. Some publishers already use sophisticated controlled vocabularies, others are barely using manually-supplied keywords. To illustrate this, some examples of the different kinds of values that can be given for the `dc:coverage` property are:

```
<dc:coverage>Greece</dc:coverage>

<dc:coverage rdf:resource="http://prismstandard.org/vocabs/ISO-3166/GR"/>
```

(i.e., using either a plain literal or a URIref to identify the country) and

```
<dc:coverage>
  <pcv:Descriptor rdf:about="http://prismstandard.org/vocabs/ISO-3166/GR">
    <pcv:label xml:lang="en">Greece</pcv:label>
    <pcv:label xml:lang="fr">Grèce</pcv:label>
  </pcv:Descriptor>
</dc:coverage>
```

(using a structured value to provide both a URIref and names in various languages).

Note also that there are properties whose meanings are similar, or subsets of other properties. For example, the geographic subject of a resource could be given with

```
<prism:subject>Greece</prism:subject>
<dc:coverage>Greece</dc:coverage>
```

or

```
<prism:location>Greece</prism:location>
```

Any of those properties might use the simple literal value, or a more complex structured value. Such a range of possibilities cannot be adequately described by DTDs, or even by the newer XML Schemas. While there is a wide range of syntactic variations to deal with, RDF's graph model has a simple structure - a set of triples. Dealing with the metadata in the triples domain makes it much easier for older software to accommodate content with new extensions.

This section closes with two final examples. Example 35 says that the image (`.../Corfu.jpg`) cannot be used (`#none`) in the tobacco industry (code 21 in SIC, the Standard Industrial Classifications).

Example 35: A PRISM Description of an Image

```
<rdf:RDF xmlns:prism="http://prismstandard.org/namespaces/basic/1.0/"
         xmlns:prl="http://prismstandard.org/namespaces/prl/1.0/"
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/">

 <rdf:Description rdf:about="http://travel.example.com/2000/08/Corfu.jpg">
  <dc:rights rdf:parseType="Resource"
         xml:base="http://prismstandard.org/vocabularies/1.0/usage.xml">
     <prl:usage rdf:resource="#none"/>
     <prl:industry rdf:resource="http://prismstandard.org/vocabs/SIC/21"/>
  </dc:rights>
 </rdf:Description>
</rdf:RDF>
```

Example 36 says that the photographer for the Corfu image was employee 3845, better known as John Peterson. It also says that the geographic coverage of the photo is Greece. It does so by providing, not just a code from a controlled vocabulary, but a cached version of the information for that term in the vocabulary.

Example 36: Additional Information about the Image from Example 35

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:pcv="http://prismstandard.org/namespaces/pcv/1.0/"
         xmlns:dc="http://purl.org/dc/elements/1.1/"
         xml:base="http://travel.example.com/">

  <rdf:Description rdf:about="/2000/08/Corfu.jpg">
    <dc:identifier rdf:resource="/content/2357845" />
    <dc:creator>
      <pcv:Descriptor rdf:about="/emp3845">
        <pcv:label>John Peterson</pcv:label>
      </pcv:Descriptor>
    </dc:creator>
    <dc:coverage>
      <pcv:Descriptor
          rdf:about="http://prismstandard.org/vocabs/ISO-3166/GR">
        <pcv:label xml:lang="en">Greece</pcv:label>
        <pcv:label xml:lang="fr">Grece</pcv:label>
      </pcv:Descriptor>
    </dc:coverage>
  </rdf:Description>
</rdf:RDF>
```

6.3 XPackage

Many situations involve the need to maintain information about structured groupings of resources and their associations that are, or may be, used as a unit. The XML Package (XPackage) specification [XPACKAGE] provides a framework for defining such groupings, called *packages*. XPackage specifies a framework for describing the resources included in such packages, the properties of those resources, their method of inclusion, and their relationships with each other. XPackage applications include specifying the style sheets used by a document, declaring the images shared by multiple documents, indicating the author and other metadata of a document, describing how namespaces are used by XML resources, and providing a manifest for bundling resources into a single archive file.

The XPackage framework is based upon XML, RDF, and the XML Linking Language [XLINK], and provides multiple RDF

vocabularies: one for general packaging descriptions, and several other vocabularies for providing supplemental resource information useful to package processors.

One application of XPackage is the description of XHTML documents and their supporting resources. An XHTML document retrieved from a Web site may rely on other resources such as style sheets and image files that also need to be retrieved. However, the identities of these supporting resources may not be obvious without processing the entire document. Other information about the document, such as the name of its author, may also not be available without processing the document. XPackage allows such descriptive information to be stored in a standard way in a package description document containing RDF. The outer elements of a package description document describing such an XHTML document might look like Example 37 (with namespace declarations removed for simplicity):

Example 37: Outer Elements of an XPackage Package Description Document

```
<?xml version="1.0"?>
<xpackage:description>
  <rdf:RDF>

    (description of individual resources go here)

  </rdf:RDF>
</xpackage:description>
```

Resources (such as the XHTML document, style sheets, and images) are described within this package description document using standard RDF/XML syntax. Each resource description element may include RDF properties from various vocabularies (XPackage uses the term "ontology" for what RDF calls a "vocabulary"). Besides the main packaging vocabulary, XPackage itself specifies several supplemental vocabularies, including:

- a vocabulary (using prefix `file:`) for describing files (with properties such as `file:size`)
- a vocabulary (using prefix `mime:`) for providing MIME information (with properties such as `mime:contentType`)
- a vocabulary (using prefix `unicode:`) for providing character usage information (with properties such as `unicode:script`)
- a vocabulary (using prefix `x:`) for describing XML-based resources (with properties such as `x:namespace` and `x:style`)

In Example 38, the document's MIME content type ("application/xhtml+xml") is defined using a standard XPackage property from the XPackage MIME vocabulary, `mime:contentType`. Another property, the document's author (in this case, "Garret Wilson"), is described using a property from the Dublin Core vocabulary, defined outside of XPackage, resulting in a `dc:creator` property.

Example 38: A Description of an XHTML Document

```
<?xml version="1.0"?>
<xpackage:description
          xmlns:xpackage="http://xpackage.org/namespaces/2003/xpackage#"
          xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
          xmlns:dc="http://purl.org/dc/elements/1.1/"
          xmlns:mime="http://xpackage.org/namespaces/2003/mime#"
          xmlns:x="http://xpackage.org/namespaces/2003/xml#"
          xmlns:xlink="http://www.w3.org/1999/xlink">
    <rdf:RDF>

    <!--doc.html-->
    <rdf:Description rdf:about="urn:example:xhtmldocument-doc">
      <rdfs:comment>The XHTML document.</rdfs:comment>
      <xpackage:location xlink:href="doc.html"/>
      <mime:contentType>application/xhtml+xml</mime:contentType>
      <x:namespace rdf:resource="http://www.w3.org/1999/xhtml"/>
      <x:style rdf:resource="urn:example:xhtmldocument-stylesheet"/>
      <dc:creator>Garret Wilson</dc:creator>
      <xpackage:manifest rdf:parseType="Collection">
         <rdf:Description rdf:about="urn:example:xhtmldocument-stylesheet"/>
         <rdf:Description rdf:about="urn:example:xhtmldocument-image"/>
      </xpackage:manifest>
    </rdf:Description>

    </rdf:RDF>
</xpackage:description>
```

The `xpackage:manifest` property indicates that both the style sheet and image resources are necessary for processing; those

resources are described separately within the package description document. The example style sheet resource description in Example 39 lists its location within the package ("stylesheet.css") using the general XPackage vocabulary `xpackage:location` property (which is compatible with XLink), and shows through use of the XPackage MIME vocabulary `mime:contentType` property that it is a CSS style sheet ("text/css").

Example 39: A Style Sheet Resource Description

```
<?xml version="1.0"?>
<xpackage:description
          xmlns:xpackage="http://xpackage.org/namespaces/2003/xpackage#"
          xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
          xmlns:dc="http://purl.org/dc/elements/1.1/"
          xmlns:mime="http://xpackage.org/namespaces/2003/mime#"
          xmlns:x="http://xpackage.org/namespaces/2003/xml#"
          xmlns:xlink="http://www.w3.org/1999/xlink">
    <rdf:RDF>

    <!--stylesheet.css-->
    <rdf:Description rdf:about="urn:example:xhtmldocument-css">
      <rdfs:comment>The document style sheet.</rdfs:comment>
      <xpackage:location xlink:href="stylesheet.css"/>
      <mime:contentType>text/css</mime:contentType>
    </rdf:Description>

    </rdf:RDF>
</xpackage:description>
```

The full version of this example may be found in [XPACKAGE].

6.4 RSS 1.0: RDF Site Summary

People sometimes need to access a wide variety of information on the Web on a day-to-day basis, such as schedules, to-do lists, news headlines, search results, "What's New", etc. As the sources and diversity of the information on the Web increases, it becomes increasingly difficult to manage this information and integrate it into a coherent whole. RSS 1.0 ("RDF Site Summary") is an RDF vocabulary that provides a lightweight, yet powerful way of describing information for timely, large-scale distribution and reuse. RSS 1.0 is also perhaps the most widely deployed RDF application on the Web.

To give a simple example, the W3C home page is a primary point of contact with the public and serves in part to disseminate information about the deliverables of the Consortium. An example of the W3C home page as of a certain date is shown in Figure 19. The center column of news items changes frequently. To support the timely dissemination of this information, the W3C Team has implemented an RDF Site Summary (RSS 1.0) news feed that makes the content in the center column available to others to reuse as they will. News syndication sites may merge the headlines into a summary of the day's latest news, others may display the headlines as links as a service to their readers, and, increasingly, individuals may subscribe to this feed with a desktop application. These desktop *RSS readers* allow their users to keep track of potentially hundreds of sites, without having to visit each one in their browser.



Figure 19: The W3C Home Page

Numerous sites all over the Web provide RSS 1.0 feeds. Example 40 is an example of the W3C feed (from a different date):

Example 40: An Example of the W3C RSS 1.0 Feed

```
<?xml version="1.0" encoding="utf-8"?>

<rdf:RDF xmlns="http://purl.org/rss/1.0/"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

  <channel rdf:about="http://www.w3.org/2000/08/w3c-synd/home.rss">
    <title>The World Wide Web Consortium</title>
    <description>Leading the Web to its Full Potential...</description>
    <link>http://www.w3.org/</link>

    <dc:date>2002-10-28T08:07:21Z</dc:date>

    <items>
        <rdf:Seq>
            <rdf:li rdf:resource="http://www.w3.org/News/2002#item164"/>
            <rdf:li rdf:resource="http://www.w3.org/News/2002#item168"/>
            <rdf:li rdf:resource="http://www.w3.org/News/2002#item167"/>
        </rdf:Seq>
    </items>

  </channel>

  <item rdf:about="http://www.w3.org/News/2002#item164">
    <title>User Agent Accessibility Guidelines Become a W3C
       Proposed Recommendation</title>
    <description>17 October 2002: W3C is pleased to announce the
       advancement of User Agent Accessibility Guidelines 1.0 to
       Proposed Recommendation. Comments are welcome through 14 November.
       Written for developers of user agents, the guidelines lower
       barriers to Web accessibility for people with disabilities
       (visual, hearing, physical, cognitive, and neurological).
       The companion Techniques Working Draft is updated. Read about
       the Web Accessibility Initiative. (News archive)</description>
    <link>http://www.w3.org/News/2002#item164</link>
    <dc:date>2002-10-17</dc:date>
  </item>

  <item rdf:about="http://www.w3.org/News/2002#item168">
    <title>Working Draft of Authoring Challenges for Device
       Independence Published</title>
    <description>25 October 2002: The Device Independence
       Working Group has released the first public Working Draft of
       Authoring Challenges for Device Independence. The draft describes
       the considerations that Web authors face in supporting access to
       their sites from a variety of different devices. It is written
       for authors, language developers, device experts and developers
       of Web applications and authoring systems. Read about the Device
       Independence Activity (News archive)</description>
    <link>http://www.w3.org/News/2002#item168</link>
    <dc:date>2002-10-25</dc:date>
  </item>

  <item rdf:about="http://www.w3.org/News/2002#item167">
    <title>CSS3 Last Call Working Drafts Published</title>
    <description>24 October 2002: The CSS Working Group has
       released two Last Call Working Drafts and welcomes comments
       on them through 27 November. CSS3 module: text is a set of
       text formatting properties and addresses international contexts.
       CSS3 module: Ruby is properties for ruby, a short run of text
       alongside base text typically used in East Asia. CSS3 module:
       The box model for the layout of textual documents in visual
```

```
      media is also updated. Cascading Style Sheets (CSS) is a
      language used to render structured documents like HTML and
      XML on screen, on paper, and in speech. Visit the CSS home
      page. (News archive)</description>
    <link>http://www.w3.org/News/2002#item167</link>
    <dc:date>2002-10-24</dc:date>
  </item>

</rdf:RDF>
```

As Example 40 shows, the format is designed for content that can be packaged into easily distinguishable sections. News sites, Web logs, sports scores, stock quotes, and the like are all use-cases for RSS 1.0.

The RSS feed can be requested by any application able to "speak" HTTP. More recently, however, RSS 1.0 applications are splitting into three different categories:

- On-line aggregators - Sites such as Meerkat and NewsIsFree, shown side-by-side in Figure 20 (each mirroring W3C's column of news). These gather feeds from thousands of sources, separate each of the `<item>`s out, and add them together again into one large group. The whole group is then made searchable. In this way, one can search for the latest news on, for example, "Java" from perhaps thousands of sites, without having to search them all.
- Desktop Readers - Utilities such as Amphetadesk and NetNewsWire Lite allow their users to subscribe to hundreds of feeds from their desktop. Readers customarily refresh each feed once an hour, allowing users to stay up to date.
- Scripts - RSS's original purpose was to allow Webmasters to include the content of another's site within their own. RSS 1.0 is still used in this way, with many sites (Slashdot for example) incorporating RSS feeds on their front page.



Figure 20: MeerKat and NewsIsFree

RSS 1.0 is extensible by design. By importing additional RDF vocabularies (or *modules* as they are known within the RSS development community), the RSS 1.0 author can provide large amounts of metadata and handling instructions to the recipient of the file. Modules can, as with more general RDF vocabularies, be written by anyone. Currently there are 3 official modules and 19 proposed modules readily recognized by the community at large. These modules range from the complete Dublin Core module to more specialized RSS-centric modules such as the Aggregation module.

Care should be taken when discussing "RSS" in the scope of RDF. There are currently two RSS specification strands. One strand (RSS 0.91,0.92,0.93,0.94 and 2.0) does not use RDF. The other strand (RSS 0.9 and 1.0) does.

6.5 CIM/XML

Electric utilities use power system models for a number of different purposes. For example, simulations of power systems are necessary for planning and security analysis. Power system models are also used in actual operations, e.g., by the Energy Management Systems (EMS) used in energy control centers. An operational power system model can consist of thousands of classes of information. In addition to using these models in-house, utilities need to exchange system modeling information, both in planning, and for operational purposes, e.g., for coordinating transmission and ensuring reliable operations. However, individual utilities use different software for these purposes, and as a result the system models are stored in different formats, making the exchange of these models difficult.

In order to support the exchange of power system models, utilities needed to agree on common definitions of power system entities and relationships. To support this, the Electric Power Research Institute (EPRI) a non-profit energy research consortium, developed a Common Information Model (CIM) [CIM]. The CIM specifies common semantics for power system resources, their attributes, and relationships. In addition, to further support the ability to electronically exchange CIM models, the power industry has developed CIM/XML, a language for expressing CIM models in XML. CIM/XML is an RDF application, using RDF and RDF Schema to organize its XML structures. The North American Electric Reliability Council (NERC) (an industry-supported organization formed to promote the

reliability of electricity delivery in North America) has adopted CIM/XML as the standard for exchanging models between power transmission system operators. The CIM/XML format is also going through an IEC international standardization process. An excellent discussion of CIM/XML can be found in [DWZ01]. [NB: This power industry CIM should not be confused with the CIM developed by the Distributed Management Task Force for representing management information for distributed software, network, and enterprise environments. The DMTF CIM also has an XML representation, but does not currently use RDF, although independent research is underway in that direction.]

The CIM can represent all of the major objects of an electric utility as object classes and attributes, as well as their relationships. CIM uses these object classes and attributes to support the integration of independently developed applications between vendor specific EMS systems, or between an EMS system and other systems that are concerned with different aspects of power system operations, such as generation or distribution management.

The CIM is specified as a set of class diagrams using the Unified Modeling Language (UML). The base class of the CIM is the `PowerSystemResource` class, with other more specialized classes such as `Substation`, `Switch`, and `Breaker` being defined as subclasses. CIM/XML represents the CIM as an RDF Schema vocabulary, and uses RDF/XML as the language for exchanging specific system models. Example 41 shows examples of CIM/XML class and property definitions:

Example 41: Examples of CIM/XML Class and Property Definitions

```
<rdfs:Class rdf:ID="PowerSystemResource">
  <rdfs:label xml:lang="en">PowerSystemResource</rdfs:label>
  <rdfs:comment>"A power system component that can be either an
    individual element such as a switch or a set of elements
    such as a substation. PowerSystemResources that are sets
    could be members of other sets. For example a Switch is a
    member of a Substation and a Substation could be a member
    of a division of a Company"</rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="Breaker">
  <rdfs:label xml:lang="en">Breaker</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Switch" />
  <rdfs:comment>"A mechanical switching device capable of making,
     carrying, and breaking currents under normal circuit conditions
     and also making, carrying for a specified time, and breaking
     currents under specified abnormal circuit conditions e.g. those
     of short circuit. The typeName is the type of breaker, e.g.,
     oil, air blast, vacuum, SF6."</rdfs:comment>
</rdfs:Class>

<rdf:Property rdf:ID="Breaker.ampRating">
   <rdfs:label xml:lang="en">ampRating</rdfs:label>
   <rdfs:domain rdf:resource="#Breaker" />
   <rdfs:range rdf:resource="#CurrentFlow" />
   <rdfs:comment>"Fault interrupting rating in amperes"</rdfs:comment>
</rdf:Property>
```

CIM/XML uses only a subset of the complete RDF/XML syntax, in order to simplify expressing the models. In addition, CIM/XML implements some extensions to the RDF Schema vocabulary. These extensions support the description of inverse roles and multiplicity (cardinality) constraints describing how many instances of a given property are allowed for a given resource (allowable values for a multiplicity declaration are zero-or-one, exactly-one, zero-or-more, one-or-more). The properties in Example 42 illustrate these extensions (which are identified by a `cims:` QName prefix):

Example 42: Some CIM/XML Extensions of RDF Schema

```
<rdf:Property rdf:ID="Breaker.OperatedBy">
   <rdfs:label xml:lang="en">OperatedBy</rdfs:label>
   <rdfs:domain rdf:resource="#Breaker" />
   <rdfs:range rdf:resource="#ProtectionEquipment" />
   <cims:inverseRoleName rdf:resource="#ProtectionEquipment.Operates" />
   <cims:multiplicity rdf:resource="http://www.cim-logic.com/schema/990530#M:0..n" />
   <rdfs:comment>"Circuit breakers may be operated by
       protection relays."</rdfs:comment>
</rdf:Property>

<rdf:Property rdf:ID="ProtectionEquipment.Operates">
   <rdfs:label xml:lang="en">Operates</rdfs:label>
```

```
   <rdfs:domain rdf:resource="#ProtectionEquipment" />
   <rdfs:range rdf:resource="#Breaker" />
   <cims:inverseRoleName rdf:resource="#Breaker.OperatedBy" />
   <cims:multiplicity rdf:resource="http://www.cim-logic.com/schema/990530#M:0..n" />
   <rdfs:comment>"Circuit breakers may be operated by
       protection relays."</rdfs:comment>
</rdf:Property>
```

EPRI has conducted successful interoperability tests using CIM/XML to exchange real-life, large-scale models (involving, in the case of one test, data describing over 2000 substations) between a variety of vendor products, and validating that these models would be correctly interpreted by typical utility applications. Although the CIM was originally intended for EMS systems, it is also being extended to support power distribution and other applications as well.

The Object Management Group has adopted an object interface standard to access CIM power system models called the Data Access Facility [DAF]. Like the CIM/XML language, the DAF is based on the RDF model and shares the same CIM schema. However, while CIM/XML enables a model to be exchanged as a document, DAF enables an application to access the model as a set of objects.

CIM/XML illustrates the useful role RDF can play in supporting XML-based exchange of information that is naturally expressed as entity-relationship or object-oriented classes, attributes, and relationships (even when that information will not necessarily be Web-accessible). In these cases, RDF provides a basic structure for the XML in support of identifying objects, and using them in structured relationships. This connection is illustrated by a number of applications using RDF/XML for information interchange, as well as a number of projects investigating linkages between RDF (or ontology languages such as OWL) and UML (and its XML representations). CIM/XML's need to extend RDF Schema to support cardinality constraints and inverse relationships also illustrates the kinds of requirements that have led to the development of more powerful RDF-based schema/ontology languages such as DAML+OIL and OWL described in Section 5.5. Such languages may be appropriate in supporting many similar modeling applications in the future.

Finally, CIM/XML also illustrates an important fact for those looking for additional examples of "RDF in the Field": sometimes languages are described as "XML" languages, or systems are described as using "XML", and the "XML" they are actually using is RDF/XML, i.e., they are RDF applications. Sometimes it is necessary to go fairly far into the description of the language or system in order to find this out (in some examples that have been found, RDF is never explicitly mentioned at all, but sample data clearly shows it is RDF/XML). Moreover, in applications such as CIM/XML, the RDF that is created will not be readily found on the Web, since it is intended for information exchange between software components rather than for general access (although future scenarios could be imagined in which more of this type of RDF would become Web-accessible).

6.6 Gene Ontology Consortium

Structured metadata using controlled vocabularies such as SNOMED RT (Systematized Nomenclature of Medicine Reference Terminology) and MeSH (Medical Subject Headings) plays an important role in medicine, enabling more efficient literature searches and aiding in the distribution and exchange of medical knowledge [COWAN]. At the same time, the field of medicine is rapidly changing, and with that comes the need to develop additional vocabularies.

The objective of the Gene Ontology (GO) Consortium [GO] is to provide controlled vocabularies to describe specific aspects of gene products. Collaborating databases annotate their gene products (or genes) with GO terms, providing references and indicating what kind of evidence is available to support the annotations. The use of common GO terms by these databases facilitates uniform queries across them. The GO ontologies are structured to allow both attribution and querying to be performed at different levels of granularity. The GO vocabularies are dynamic, since knowledge of gene and protein roles in cells is accumulating and changing.

The three organizing principles of the GO are *molecular function*, *biological process*, and *cellular component*. A gene product has one or more molecular functions and is used in one or more biological processes; it may be, or may be associated with, one or more cellular components. Definitions of the terms within all three of these ontologies are contained in a single (text) definition file. XML formatted versions, containing all three ontology files and all available definitions, are generated monthly.

Function, process and component are represented as directed acyclic graphs (DAGs) or networks. A child term may be an "instance" of its parent term (isa relationship) or a component of its parent term (part-of relationship). A child term may have more than one parent term and may have a different class of relationship with its different parents. Synonyms and cross-references to external databases are also represented in the ontologies. GO uses RDF/XML facilities to represent the relationships between terms in the XML versions of the ontologies, because of its flexibility in representing these graph structures, as well as its widespread tool support. At the same time, GO currently uses *non*-RDF nested XML structures within the term descriptions, so the language used is not pure RDF/XML.

Example 43 shows some sample GO information from the GO documentation:

Example 43: Sample GO Information

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE go:go>
<go:go xmlns:go="http://www.geneontology.org/xml-dtd/go.dtd#"
       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <go:version timestamp="Wed May 9 23:55:02 2001" />

  <rdf:RDF>
     <go:term rdf:about="http://www.geneontology.org/go#GO:0003673">
        <go:accession>GO:0003673</go:accession>
        <go:name>Gene_Ontology</go:name>
        <go:definition></go:definition>
     </go:term>

     <go:term rdf:about="http://www.geneontology.org/go#GO:0003674">
        <go:accession>GO:0003674</go:accession>
        <go:name>molecular_function</go:name>
        <go:definition>The action characteristic of a gene product.</go:definition>
        <go:part-of rdf:resource="http://www.geneontology.org/go#GO:0003673" />
        <go:dbxref>
           <go:database_symbol>go</go:database_symbol>
           <go:reference>curators</go:reference>
        </go:dbxref>
     </go:term>

     <go:term rdf:about="http://www.geneontology.org/go#GO:0016209">
        <go:accession>GO:0016209</go:accession>
        <go:name>antioxidant</go:name>
        <go:definition></go:definition>
        <go:isa rdf:resource="http://www.geneontology.org/go#GO:0003674" />
        <go:association>
           <go:evidence evidence_code="ISS">
              <go:dbxref>
                 <go:database_symbol>fb</go:database_symbol>
                 <go:reference>fbrf0105495</go:reference>
              </go:dbxref>
           </go:evidence>
           <go:gene_product>
              <go:name>CG7217</go:name>
              <go:dbxref>
                 <go:database_symbol>fb</go:database_symbol>
                 <go:reference>FBgn0038570</go:reference>
              </go:dbxref>
           </go:gene_product>
        </go:association>
        <go:association>
           <go:evidence evidence_code="ISS">
              <go:dbxref>
                 <go:database_symbol>fb</go:database_symbol>
                 <go:reference>fbrf0105495</go:reference>
              </go:dbxref>
           </go:evidence>
           <go:gene_product>
              <go:name>Jafrac1</go:name>
              <go:dbxref>
                 <go:database_symbol>fb</go:database_symbol>
                 <go:reference>FBgn0040309</go:reference>
              </go:dbxref>
           </go:gene_product>
        </go:association>
      </go:term>
  </rdf:RDF>
</go:go>
```

Example 43 illustrates that `go:term` is the basic element. In some cases, the GO has defined its own terms rather than using RDF Schema. For example, term `GO:0016209` has the element `<go:isa`

`rdf:resource="http://www.geneontology.org/go#GO:0003674" />`. This tag represents the relationship "`GO:0016209` isa `GO:0003674`", or, in English, "Antioxidant is a molecular function." Another specialized relationship is `go:part-of`. For example, `GO:0003674` has the element `<go:part-of rdf:resource="http://www.geneontology.org/go#GO:0003673" />`. This says that "Molecular function is part of the Gene Ontology".

Every annotation must be attributed to a source, which may be a literature reference, another database or a computational analysis. The annotation must indicate what kind of evidence is found in the cited source to support the association between the gene product and the GO term. A simple controlled vocabulary is used to record evidence. Examples include:

- ISS means "inferred from sequence similarity [with <database:sequence_id>]"
- IDA means "inferred from direct assay"
- TAS means "traceable author statement"

The `go:dbxref` element represents the term in an external database, and `go:association` represents the gene associations of each term. `go:association` can have both `go:evidence`, which holds a `go:dbxref` to the evidence supporting the association, and a `go:gene_product`, which contains the gene symbol and `go:dbxref`. These elements illustrate that the GO XML syntax is not "pure" RDF/XML, since the nesting of other elements within these elements does not conform to the alternate node/predicate arc "stripes" described in Sections 2.1 and 2.2 of [RDF-SYNTAX].

The GO illustrates a number of interesting points. First, it shows that the value of using XML for information exchange can be enhanced by structuring that XML using RDF. This is particularly true for data that has an overall graph or network structure, rather than being a strict hierarchy. The GO is also another example in which data using RDF will not necessarily appear for direct use on the Web (although the files are Web-accessible). It is also another example of data which is, on the surface, described as "XML", but on closer examination uses RDF/XML facilities (albeit not "pure" RDF/XML). Finally, the GO illustrates the role RDF can play as a basis for representing ontologies. This role will be further enhanced once richer RDF-based languages for specifying ontologies, such as the DAML+OIL or OWL languages discussed in Section 5.5, become more widely used. In fact, a Gene Ontology Next Generation project is currently developing a representation of the GO ontologies in these richer languages.

6.7 Describing Device Capabilities and User Preferences

In recent years a large number of new mobile devices for browsing the Web have appeared. Many of these devices have highly divergent capabilities including a wide range of input and output capabilities as well as different levels of language support. Mobile devices may also have widely differing network connectivity capabilities. Users of these new devices expect a usable presentation regardless of the device's capabilities or the current network characteristics. Likewise, users want their dynamically changing preferences (e.g. turn audio on/off) to be considered when content or an application is presented. The reality, however, is that device heterogeneity, and the lack of a standard way for users to convey their preferences to the server, may result in: content that cannot be stored on the device, content that cannot be displayed, or content that violates the desires of the user. Additionally, the resulting content may take too long to convey over the network to the client device.

A solution for addressing these problems is for a client to encode its *delivery context* - the device's capabilities, the user's preferences, the network characteristics, etc. - in such a way that a server can use the context to customize content for the device and user (see [DIPRINC] for a definition of delivery context). The W3C's Composite Capabilities/Preferences Profile (CC/PP) specification [CC/PP] helps to address this problem by defining a generic framework for describing a delivery context.

The CC/PP framework defines a relatively simple structure - a two-level hierarchy of components and attribute/value pairs. A *component* may be used to capture a part of a delivery context (e.g. network characteristics, software supported by a device, or the hardware characteristics of a device). A component may contain one or more *attributes*. For example a component that encodes user preferences may contain an attribute to specify whether or not *AudioOutput* is desired.

CC/PP defines its structure (the hierarchy described above) using RDF Schema (see [CC/PP] for details of the structure schema). A CC/PP *vocabulary* defines specific components and their attributes. [CC/PP], however, does not define such vocabularies. Instead, vocabularies are defined by other organizations or applications (as described below). [CC/PP] also does not define a protocol for transporting an instance of a CC/PP vocabulary.

An instance of a CC/PP vocabulary is called a *profile*. CC/PP attributes are encoded as RDF properties in a profile. Example 44 shows a profile fragment of user preferences for a user that prefers an audio presentation:

Example 44: A CC/PP Profile Fragment

```
 <ccpp:component>
  <rdf:Description rdf:ID="UserPreferences">
   <rdf:type
rdf:resource="http://www.example.org/profiles/prefs/v1_0#UserPreferences"/>
   <ex:AudioOutput>Yes</ex:AudioOutput>
   <ex:Graphics>No</ex:Graphics>
```

```
   <ex:Languages>
    <rdf:Seq>
     <rdf:li>en-cockney</rdf:li>
     <rdf:li>en</rdf:li>
    </rdf:Seq>
   </ex:Languages>
  </rdf:Description>
 </ccpp:component>
```

There are several advantages to using RDF in this application. First, a profile encoded via CC/PP may include attributes that were defined in schemas created by different organizations. RDF is a natural fit for these profiles because no single organization is likely to create a *super* schema for the aggregated profile data. A second advantage of RDF is that it facilitates (by virtue of its graph-based data model) the insertion of arbitrary attributes (RDF properties) into a profile. This is particularly useful for profiles that include frequently changing data such as location information.

The Open Mobile Alliance has defined the User Agent Profile (UAProf) [UAPROF] - a CC/PP-based framework that includes a vocabulary for describing device capabilities, user agent capabilities, network characteristics, etc., as well as a protocol for transporting a profile. UAProf defines six components including: *HardwarePlatform*, *SoftwarePlatform*, *NetworkCharacteristics* and *BrowserUA*. It also defines several attributes for each of its components although a component's attributes are not fixed - they may be supplemented or overridden. Example 45 shows a fragment of UAProf's *HardwarePlatform* component:

Example 45: A Fragment of UAProf's HardwarePlatform Component

```
 <prf:component>
  <rdf:Description rdf:ID="HardwarePlatform">
   <rdf:type
rdf:resource="http://www.openmobilealliance.org/profiles/UAPROF/ccppschema-
20021113#HardwarePlatform"/>
   <prf:ScreenSizeChar>15x6</prf:ScreenSizeChar>
   <prf:BitsPerPixel>2</prf:BitsPerPixel>
   <prf:ColorCapable>No</prf:ColorCapable>
   <prf:BluetoothProfile>
    <rdf:Bag>
     <rdf:li>headset</rdf:li>
     <rdf:li>dialup</rdf:li>
     <rdf:li>lanaccess</rdf:li>
    </rdf:Bag>
   </prf:BluetoothProfile>
  </rdf:Description>
 </prf:component>
```

The UAProf protocol supports both *static* profiles and *dynamic* profiles. A *static* profile is accessed via a URI. This has several advantages: a client's request to a server only contains a URI rather a potentially verbose XML document (thus minimizing over the air traffic); the client does not have to store and/or create the profile; the implementation burden on a client is relatively light-weight. *Dynamic* profiles are created on-the-fly and consequently do not have an associated URI. They may consist of a profile fragment containing a *difference* from a static profile, but they may also contain unique data that is not included in the client's static profile. A request may contain any number of static profiles and dynamic profiles. However, the ordering of the profiles is important as later profiles override earlier profiles in the request. See [UAPROF] for more information about UAProf's protocol and its rules for resolving multiple profiles.

Several other communities (i.e. 3GPP's TS 26.234 [3GPP] and the WAP Forum's Multimedia Messaging Service Client Transactions Specification [MMS-CTR]) have defined vocabularies based on CC/PP. As a result, a profile may take advantage of the distributed nature of RDF and include components defined from various vocabularies. Example 46 shows such a profile:

Example 46: A Profile Using Several Vocabularies

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:prf="http://www.wapforum.org/profiles/UAPROF/ccppschema-20010330#"
         xmlns:mms="http://www.wapforum.org/profiles/MMS/ccppschema-20010111#"
     xmlns:pss="http://www.3gpp.org/profiles/PSS/ccppschema-YYYYMMDD#">

 <rdf:Description rdf:ID="SomeDevice">
  <prf:component>
   <rdf:Description rdf:ID="Streaming">
    <rdf:type rdf:resource="http://www.3gpp.org/profiles/PSS/ccppschema-
PSS5#Streaming"/>
```

```
    <pss:AudioChannels>Stereo</pss:AudioChannels>
    <pss:VideoPreDecoderBufferSize>30720</pss:VideoPreDecoderBufferSize>

<pss:VideoInitialPostDecoderBufferingPeriod>0</pss:VideoInitialPostDecoderBufferingPeriod>
    <pss:VideoDecodingByteRate>16000</pss:VideoDecodingByteRate>
   </rdf:Description>
  </prf:component>

  <prf:component>
   <rdf:Description rdf:ID="MmsCharacteristics">
    <rdf:type rdf:resource="http://www.wapforum.org/profiles/MMS/ccppschema-
20010111#Streaming"/>
    <mms:MmsMaxMessageSize>2048</mms:MmsMaxMessageSize>
    <mms:MmsMaxImageResolution>80x60</mms:MmsMaxImageResolution>
    <mms:MmsVersion>2.0</mms:MmsVersion>
   </rdf:Description>
  </prf:component>

  <prf:component>
   <rdf:Description rdf:ID="PushCharacteristics">
    <rdf:type
rdf:resource="http://www.openmobilealliance.org/profiles/UAPROF/ccppschema-
20010330#PushCharacteristics"/>
    <prf:Push-MsgSize>1024</prf:Push-MsgSize>
    <prf:Push-MaxPushReq>5</prf:Push-MaxPushReq>
    <prf:Push-Accept>
     <rdf:Bag>
      <rdf:li>text/html</rdf:li>
      <rdf:li>text/plain</rdf:li>
      <rdf:li>image/gif</rdf:li>
     </rdf:Bag>
    </prf:Push-Accept>
   </rdf:Description>
  </prf:component>

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

The definition of a delivery context and the data within a context will continually evolve. Consequently, RDF's inherent extensibility, and thus support for dynamically changing vocabularies, make RDF a good framework for encoding a delivery context.

7. Other Parts of the RDF Specification

Section 1 indicated that the RDF Specification consists of a number of documents (in addition to this Primer):

- RDF Concepts and Abstract Syntax [RDF-CONCEPTS]
- RDF/XML Syntax Specification [RDF-SYNTAX]
- RDF Vocabulary Description Language 1.0: RDF Schema [RDF-VOCABULARY]
- RDF Semantics [RDF-SEMANTICS]
- RDF Test Cases [RDF-TESTS]

The Primer has already discussed the subjects of several of these documents, basic RDF concepts (in Section 2), the RDF/XML syntax (in Section 3) and RDF Schema (in Section 5). This section briefly describes the remaining documents (even though there have already been numerous references to [RDF-SEMANTICS] as well), in order to explain their role in the complete specification of RDF.

7.1 RDF Semantics

As discussed in the preceding sections, RDF is intended to be used to express statements about resources in the form of a graph, using specific vocabularies (names of resources, properties, classes, etc.). RDF is also intended to be the foundation for more advanced languages, such as those discussed in Section 5.5. In order to serve these purposes, the "meaning" of an RDF graph must be defined in a very precise manner.

Exactly what constitutes the "meaning" of an RDF graph in a very general sense may depend on many factors, including conventions

within a user community to interpret user-defined RDF classes and properties in specific ways, comments in natural language, or links to other content-bearing documents. As noted briefly in Section 2.2, much of the meaning conveyed in these forms will not be directly accessible to machine processing, although this meaning may be used by human interpreters of the RDF information, or by programmers writing software to perform various kinds of processing on that RDF information. However, RDF statements also have a *formal* meaning which determines, with mathematical precision, the conclusions (or *entailments*) that machines can draw from a given RDF graph. The RDF Semantics [RDF-SEMANTICS] document defines this formal meaning, using a technique called *model theory* for specifying the semantics of a formal language. [RDF-SEMANTICS] also defines the semantic extensions to the RDF language represented by RDF Schema, and by individual datatypes. In other words, the RDF model theory provides the formal underpinnings for all RDF concepts. Based on the semantics defined in the model theory, it is simple to translate an RDF graph into a logical expression with essentially the same meaning.

7.2 Test Cases

The RDF Test Cases [RDF-TESTS] supplement the textual RDF specifications with test cases (examples) corresponding to particular technical issues addressed by the RDF Core Working Group. To help describe these examples, the Test Cases document introduces a notation called N-Triples, which provides the basis for the triples notation used throughout this Primer. The test cases are published in machine-readable form at Web locations referenced by the Test Cases document, so developers can use these as the basis for automated testing of RDF software.

The test cases are divided into a number of categories:

- Positive and Negative Parser Tests: These test whether RDF/XML parsers produce a correct N-Triples output graph from legal RDF/XML input documents, or correctly report errors if the input documents are not legal RDF/XML.
- Positive and Negative Entailment Tests: These test whether proper entailments (conclusions) are or are not drawn from sets of specified RDF statements.
- Datatype-aware Entailment Tests: These are positive or negative entailment tests that involve the use of datatypes, and hence require additional support for the specific datatypes involved in the tests.
- Miscellaneous Tests: These are tests that do not fall into one of the other categories.

The test cases are not a complete specification of RDF, and are not intended to take precedence over the other specification documents. However, they are intended to illustrate the intent of the RDF Core Working Group with respect to the design of RDF, and developers may find these test cases helpful should the wording of the specifications be unclear on any point of detail.

8. References

8.1 Normative References

[RDF-CONCEPTS]
Resource Description Framework (RDF): Concepts and Abstract Syntax, Klyne G., Carroll J. (Editors), W3C Recommendation, 10 February 2004. This version is http://www.w3.org/TR/2004/REC-rdf-primer-20040210/. The latest version is http://www.w3.org/TR/rdf-concepts/.

[RDF-MIME-TYPE]
MIME Media Types, The Internet Assigned Numbers Authority (IANA). This document is http://www.iana.org/assignments/media-types/ . The registration for `application/rdf+xml` is archived at http://www.w3.org/2001/sw/RDFCore/mediatype-registration .

[RDF-MS]
Resource Description Framework (RDF) Model and Syntax Specification, Lassila O., Swick R. (Editors), World Wide Web Consortium, 22 February 1999. This version is http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/. The latest version is http://www.w3.org/TR/REC-rdf-syntax/.

[RDF-SEMANTICS]
RDF Semantics, Hayes P. (Editor), W3C Recommendation, 10 February 2004. This version is http://www.w3.org/TR/2004/REC-rdf-mt-20040210/. The latest version is http://www.w3.org/TR/rdf-mt/.

[RDF-SYNTAX]
RDF/XML Syntax Specification (Revised), Beckett D. (Editor), W3C Recommendation, 10 February 2004. This version http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/. The latest version is http://www.w3.org/TR/rdf-syntax-grammar/.

[RDF-TESTS]
RDF Test Cases, Grant J., Beckett D. (Editors), W3C Recommendation, 10 February 2004. This version is http://www.w3.org/TR/2004/REC-rdf-testcases-20040210/. The latest version is http://www.w3.org/TR/rdf-testcases/.

[RDF-VOCABULARY]
RDF Vocabulary Description Language 1.0: RDF Schema, Brickley D., Guha R.V. (Editors), W3C Recommendation, 10 February 2004. This version is http://www.w3.org/TR/2004/REC-rdf-schema-20040210/. The latest version is http://www.w3.org/TR/rdf-schema/.

[UNICODE]
The Unicode Standard, Version 3, The Unicode Consortium, Addison-Wesley, 2000. ISBN 0-201-61633-5, as updated from time to time by the publication of new versions. (See http://www.unicode.org/unicode/standard/versions/ for the latest version and additional information on versions of the standard and of the Unicode Character Database).

[URIS]
RFC 2396 - Uniform Resource Identifiers (URI): Generic Syntax, Berners-Lee T., Fielding R., Masinter L., IETF, August 1998, http://www.isi.edu/in-notes/rfc2396.txt.

[XML]
Extensible Markup Language (XML) 1.0, Second Edition, Bray T., Paoli J., Sperberg-McQueen C.M., Maler E. (Editors), World Wide Web Consortium, 6 October 2000. This version is http://www.w3.org/TR/2000/REC-xml-20001006. The latest version is http://www.w3.org/TR/REC-xml.

[XML-BASE]
XML Base, Marsh J. (Editor), World Wide Web Consortium, 27 June 2001. This version is http://www.w3.org/TR/2001/REC-xmlbase-20010627/. The latest version is http://www.w3.org/TR/xmlbase/.

[XML-NS]
Namespaces in XML, Bray T., Hollander D., Layman A. (Editors), World Wide Web Consortium, 14 January 1999. This version is http://www.w3.org/TR/1999/REC-xml-names-19990114/. The latest version is http://www.w3.org/TR/REC-xml-names/.

[XML-XC14N]
Exclusive XML Canonicalization Version 1.0, Boyer J., Eastlake D.E. 3rd, Reagle J. (Authors/Editors), World Wide Web Consortium, 18 July 2002. This version is http://www.w3.org/TR/2002/REC-xml-exc-c14n-20020718/. The latest version is http://www.w3.org/TR/xml-exc-c14n/.

8.2 Informational References

[3GPP]
3GPP TS 26.234. 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Transparent end-to-end packet switched streaming service; Protocols and codecs V5.2.0 (2002-09). This document is available at http://www.3gpp.org/specs/specs.htm via directory ftp://ftp.3gpp.org/specs/2002-09/Rel-5/26_series/.

[ADDRESS-SCHEMES]
Addressing Schemes, Connolly D., 2001. This document is http://www.w3.org/Addressing/schemes.html.

[BATES96]
Indexing and Access for Digital Libraries and the Internet: Human, Database, and Domain Factors, Bates M.J., 1996. This document is http://is.gseis.ucla.edu/research/mjbates.html.

[BERNERS-LEE98]
What the Semantic Web can represent, Berners-Lee T., 1998. This document is http://www.w3.org/DesignIssues/RDFnot.html.

[CC/PP]
Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies, Klyne G., Reynolds F., Woodrow C., Ohto H., Hjelm J., Butler M., Tran, L., W3C Recommendation, 15 January 2004. This version is http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/. The latest version is http://www.w3.org/TR/CCPP-struct-vocab/.

[CG]
Conceptual Graphs, Sowa J., ISO working document ISO/JTC1/SC32/WG2 N 000, 2 April 2001 (work in progress). Available at http://users.bestweb.net/~sowa/cg/cgstand.htm.

[CHARMOD]
Character Model for the World Wide Web 1.0, Dürst M., Yergeau F., Ishida R., Wolf M., Freytag A., Texin T. (Editors), World Wide Web Consortium, 20 February 2002 (work in progress). This version is http://www.w3.org/TR/2002/WD-charmod-20020220/. The latest version is http://www.w3.org/TR/charmod/.

[CIM]
Common Information Model (CIM): CIM 10 Version, EPRI, Palo Alto, CA: 2001, 1001976. This document is available at http://www.epri.com/attachments/286161_1001976(1).pdf (267pp.).

[COWAN]
Metadata, Reuters Health Information, and Cross-Media Publishing , Cowan J., 2002. Presentation at Seybold New York 2002 Enterprise Publishing Conference. This document is http://seminars.seyboldreports.com/seminars/2002_new_york/presentations/014/cowan_john.ppt. An accompanying transcript is http://seminars.seyboldreports.com/2002_new_york/files/transcripts/doc/transcript_EP7.doc

[DAF]
Utility Management System (UMS) Data Access Facility, version 2.0, Object Management Group, November 2002. This document is available at http://www.omg.org/technology/documents/formal/UMS_Data_Access_Facility.htm.

[DAML+OIL]
DAML+OIL (March 2001) Reference Description, Connolly D., van Harmelen F., Horrocks I., McGuinness D.L., Patel-Schneider P.F., Stein L.A., World Wide Web Consortium, 18 December 2001. This document is http://www.w3.org/TR/daml+oil-reference.

[DC]
Dublin Core Metadata Element Set, Version 1.1: Reference Description, 02 June 2003. This version is

http://dublincore.org/documents/2003/06/02/dces/. The latest version is http://dublincore.org/documents/dces/.

[DIPRINC]
Device Independence Principles. Gimson, R., Finkelstein, S., Maes, S., Suryanarayana, L., World Wide Web Consortium, 18 September 2001 (work in progress). This version is http://www.w3.org/TR/2001/WD-di-princ-20010918. The latest version is http://www.w3.org/TR/di-princ/.

[DWZ01]
XML for CIM Model Exchange , deVos A., Widergreen S.E., Zhu J., Proc. IEEE Conference on Power Industry Computer Systems, Sydney, Australia, 2001. This document is http://www.langdale.com.au/PICA/.

[GO]
Gene Ontology: tool for the unification of biology, The Gene Ontology Consortium, *Nature Genetics*, Vol. 25: 25-29, May 2000. Available at http://www.geneontology.org/GO_nature_genetics_2000.pdf

[GRAY]
Logic, Algebra and Databases, Gray P., Ellis Horwood Ltd., 1984. ISBN 0-85312-709-3, 0-85312-803-0, 0-470-20103-7, 0-470-20259-9.

[HAYES]
In Defense of Logic, Hayes P., Proceedings from the International Joint Conference on Artificial Intelligence, 1975, San Francisco. Morgan Kaufmann Inc., 1977. Also in *Computation and Intelligence: Collected Readings*, Luger G. (ed), AAAI press/MIT press, 1995. ISBN 0-262-62101-0.

[KIF]
Knowledge Interchange Format, Genesereth M., draft proposed American National Standard NCITS.T2/98-004. Available at http://logic.stanford.edu/kif/dpans.html.

[LBASE]
LBase: Semantics for Languages of the Semantic Web, Guha R. V., Hayes P., W3C Note, 10 October 2003. This version is http://www.w3.org/TR/2003/NOTE-lbase-20031010/. The latest version is http://www.w3.org/TR/lbase/.

[LUGER]
Artificial Intelligence: Structures and Strategies for Complex Problem Solving (3rd ed.), Luger G., Stubblefield W., Addison Wesley Longman, 1998. ISBN 0-805-31196-3.

[MATHML]
Mathematical Markup Language (MathML) Version 2.0, Carlisle D., Ion P., Miner R., Poppelier N. (Editors); Ausbrooks R., Buswell S., Dalmas S., Devitt S., Diaz A., Hunter R., Smith B., Soiffer N., Sutor R., Watt S. (Principal Authors), World Wide Web Consortium, 21 February 2001. This version is http://www.w3.org/TR/2001/REC-MathML2-20010221/. The latest version is http://www.w3.org/TR/MathML2/.

[MMS-CTR]
Multimedia Messaging Service Client Transactions Specification. WAP-206-MMSCTR-20020115-a. This document is available at http://www.openmobilealliance.org/.

[NAMEADDRESS]
Naming and Addressing: URIs, URLs, ..., Connolly D., 2002. This document is http://www.w3.org/Addressing/.

[OWL]
OWL Web Ontology Language Reference, Dean M., Schreiber G (Editors); van Harmelen F., Hendler J., Horrocks I., McGuinness D.L., Patel-Schneider P.F., Stein L.A. (Authors), W3C Recommendation, 10 February 2004. The latest version is http://www.w3.org/TR/owl-ref/.

[PRISM]
PRISM: Publishing Requirements for Industry Standard Metadata, Version 1.1, 19 February 2002. The latest version of the PRISM specification is available at http://www.prismstandard.org/.

[RDFISSUE]
RDF Issue Tracking, McBride B., 2002. This document is http://www.w3.org/2000/03/rdf-tracking/.

[RDF-S]
Resource Description Framework (RDF) Schema Specification 1.0 , Brickley D., Guha, R.V. (Editors), World Wide Web Consortium. 27 March 2000. This version is http://www.w3.org/TR/2000/CR-rdf-schema-20000327/.

[RSS]
RDF Site Summary (RSS) 1.0, Beged-Dov G., Brickley D., Dornfest R., Davis I., Dodds L., Eisenzopf J., Galbraith D., Guha R.V., MacLeod K., Miller E., Swartz A., van der Vlist E., 2000. This document is http://purl.org/rss/1.0/spec.

[RUBY]
Ruby Annotation, Sawicki M., Suignard M., Ishikawa M., Dürst M., Texin T. (Editors), World Wide Web Consortium, 31 May 2001. This version is http://www.w3.org/TR/2001/REC-ruby-20010531/. The latest version is http://www.w3.org/TR/ruby/.

[SOWA]
Knowledge Representation: Logical, Philosophical and Computational Foundations, Sowa J., Brookes/Cole, 2000. ISBN 0-534-94965-7.

[SVG]
Scalable Vector Graphics (SVG) 1.1 Specification, Ferraiolo J., Fujisawa J., Jackson D. (Editors), World Wide Web Consortium, 14 January 2003. This version is http://www.w3.org/TR/2003/REC-SVG11-20030114/. The latest version is http://www.w3.org/TR/SVG11/.

[UAPROF]

User Agent Profile. OMA-WAP-UAProf-v1_1. This document is available at http://www.openmobilealliance.org/.

[WEBDATA]
Web Architecture: Describing and Exchanging Data, Berners-Lee T., Connolly D., Swick R., World Wide Web Consortium, 7 June 1999. This document is http://www.w3.org/1999/04/WebData.

[XLINK]
XML Linking Language (XLink) Version 1.0, DeRose S., Maler E., Orchard D. (Editors), World Wide Web Consortium, 27 June 2001. This version is http://www.w3.org/TR/2001/REC-xlink-20010627/. The latest version is http://www.w3.org/TR/xlink/.

[XML-SCHEMA2]
XML Schema Part 2: Datatypes, Biron P., Malhotra A. (Editors), World Wide Web Consortium. 2 May 2001. This version is http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/. The latest version is http://www.w3.org/TR/xmlschema-2/.

[XPACKAGE]
XML Package (XPackage) 1.0 , Wilson G., Editor's Working Draft, 6 March 2003. This version is http://www.xpackage.org/specification/xpackage-draft-20030306.html. The latest version is http://www.xpackage.org/specification/.

9. Acknowledgments

This document has benefited from inputs from many members of the RDF Core Working Group. Specific thanks are due to Art Barstow, Dave Beckett, Dan Brickley, Ron Daniel, Ben Hammersley, Martyn Horner, Graham Klyne, Sean Palmer, Patrick Stickler, Aaron Swartz, Ralph Swick, and Garret Wilson who, together with the many people who commented on earlier versions of the Primer, provided valuable contributions to this document.

In addition, this document contains a significant contribution from Pat Hayes, Sergey Melnik, and Patrick Stickler, who led the development of the RDF datatype facilities described in the RDF family of specifications.

Frank Manola also thanks The MITRE Corporation, Frank's employer during most of the preparation of this document, for its support of his RDF Core Working Group activities under a MITRE Sponsored Research grant.

Appendix A: More on Uniform Resource Identifiers (URIs)

Note: This section is intended to provide a brief introduction to URIs. The definitive specification of URIs is RFC 2396 [URIS], which should be consulted for further details. Additional discussion of URIs can also be found in Naming and Addressing: URIs, URLs, ... [NAMEADDRESS].

As discussed in Section 2.1, the Web provides a general form of identifier, called the Uniform Resource Identifier (URI), for identifying (naming) resources on the Web. Unlike URLs, URIs are not limited to identifying things that have network locations, or use other computer access mechanisms. A number of different *URI schemes* (URI forms) have been already been developed, and are being used, for various purposes. Examples include:

- `http:` (Hypertext Transfer Protocol, for Web pages)
- `mailto:` (email addresses), e.g., `mailto:em@w3.org`
- `ftp:` (File Transfer Protocol)
- `urn:` (Uniform Resource Names, intended to be persistent location-independent resource identifiers), e.g., `urn:isbn:0-520-02356-0` (for a book)

A list of existing URI schemes can be found in Addressing Schemes [ADDRESS-SCHEMES], and it is a good idea to consider adapting one of the existing schemes for any specialized identification purposes, rather than trying to invent a new one.

No one person or organization controls who makes URIs or how they can be used. While some URI schemes, such as URL's `http:`, depend on centralized systems such as DNS, other schemes, such as `freenet:`, are completely decentralized. This means that, as with any other kind of name, no one needs special authority or permission to create a URI for something. Also, anyone can create URIs to refer to things they do not own, just as in ordinary language anyone can use whatever name they like for things they do not own.

As also noted in Section 2.1, RDF uses *URI references* [URIS] to name subjects, predicates, and objects in RDF statements. A URI reference (or *URIref*) is a URI, together with an optional *fragment identifier* at the end. For example, the URI reference `http://www.example.org/index.html#section2` consists of the URI `http://www.example.org/index.html` and (separated by the "#" character) the fragment identifier `Section2`. RDF URIrefs can contain Unicode [UNICODE] characters (see [RDF-CONCEPTS]), allowing many languages to be reflected in URIrefs.

URIrefs may be either *absolute* or *relative*. An *absolute* URIref refers to a resource independently of the context in which the URIref appears, e.g., the URIref `http://www.example.org/index.html`. A *relative* URIref is a shorthand form of an absolute URIref, where some prefix of the URIref is missing, and information from the context in which the URIref appears is required to fill in the missing information. For example, the relative URIref `otherpage.html`, when appearing in a resource `http://www.example.org/index.html`, would be filled out to the absolute URIref `http://www.example.org/otherpage.html`. A URIref without a URI part is considered a reference to the current document (the document in which it appears). So, an empty URIref within a document is considered equivalent to the URIref of the document itself. A URIref consisting of just a fragment identifier is considered equivalent to the URIref of the document in which it appears, with the fragment identifier appended to it. For example, within `http://www.example.org/index.html`, if `#section2` appeared as a URIref, it would be considered equivalent to the absolute URIref `http://www.example.org/index.html#section2`.

[RDF-CONCEPTS] notes that RDF graphs (the abstract models) do not use relative URIrefs, i.e., the subjects, predicates, and objects (and datatypes in typed literals) in RDF statements must always be identified independently of any context. However, a specific concrete RDF syntax, such as RDF/XML, may allow relative URIrefs to be used as a shorthand for absolute URIrefs in certain situations. RDF/XML does permit such use of relative URIrefs, and some of the RDF/XML examples in this Primer illustrate such uses. [RDF-SYNTAX] should be consulted for further details.

Both RDF and Web browsers use URIrefs to identify things. However, RDF and browsers interpret URIrefs in slightly different ways. This is because RDF uses URIrefs *only* to identify things, while browsers also use URIrefs to *retrieve* things. Often there is no effective difference, but in some cases the difference can be significant. One obvious difference is that when a URIref is used in a browser, there is the expectation that it identifies a resource that can actually be retrieved: that something is actually "at" the location identified by the URI. However, in RDF a URIref may be used to identify something, such as a person, that *cannot* be retrieved on the Web. People sometimes use RDF together with a convention that, when a URIref is used to identify an RDF resource, a page containing descriptive information about that resource will be placed on the Web "at" that URI, so that the URIref can be used in a browser to retrieve that information. This can be a useful convention in some circumstances, although it creates a difficulty in distinguishing the identity of the original resource from the identity of the Web page describing it (a subject discussed further in Section 2.3). However, this convention is not an explicit part of the definition of RDF, and RDF itself does not assume that a URIref identifies something that can be retrieved.

Another difference is in the way URIrefs with fragment identifiers are handled. Fragment identifiers are often seen in the URLs that identify HTML documents, where they serve to identify a specific place within the document identified by the URL. In normal HTML usage, where URI references are used to retrieve the indicated resources, the two URIrefs:

```
http://www.example.org/index.html
http://www.example.org/index.html#Section2
```

are related (they both refer to the same document, the second one identifying a location within the first one). However, as noted already, RDF uses URI references purely to *identify* resources, not to retrieve them, and RDF assumes no particular relationship between these two URIrefs. As far as RDF is concerned, they are syntactically different URI references, and hence may refer to unrelated things. This does not mean that the HTML-defined containment relationship might not exist, just that RDF does not assume that a relationship exists based only on the fact that the URI parts of the URI references are the same.

Carrying this point further, RDF does not assume that there is any relationship between URI references that share a common leading string, whether there is a fragment identifier or not. For example, as far as RDF is concerned, the two URIrefs:

```
http://www.example.org/foo.html
http://www.example.org/bar.html
```

have no particular relationship even though both of them start with the string `http://www.example.org/`. To RDF, they are simply different resources, because their URIrefs are different. (They may in fact be two files located in the same directory, but RDF does not assume this or any other relationship exists.)

Appendix B: More on the Extensible Markup Language (XML)

Note: This section is intended to provide a brief introduction to XML. The definitive specification of XML is [XML], which should be consulted for further details.

The Extensible Markup Language [XML] was designed to allow anyone to design their own document format and then write a document in that format. Like HTML documents (Web pages), XML documents contain text. This text consists primarily of plain text content, and markup in the form of *tags*. This markup allows a processing program to interpret the various pieces of content (called *elements*). Both XML content and (with certain exceptions) tags can contain Unicode [UNICODE] characters, allowing information from many languages to be directly represented. In HTML, the set of permissible tags, and their interpretation, is defined by the HTML specification. However, XML allows users to define their own markup languages (tags and the structures in which they can appear) adapted to their own specific requirements (the RDF/XML language described in Section 3 is one such XML markup language). For

example, the following is a simple passage marked up using an XML-based markup language:

```
<sentence><person webid="http://example.com/#johnsmith">I</person>
just got a new pet <animal>dog</animal>.</sentence>
```

Elements delimited by tags (`<sentence>`, `<person>`, etc.) are introduced to reflect a particular structure associated with the passage. The tags allow a program written with an understanding of these particular elements, and the way they are structured, to properly interpret the passage. For example, one of the elements in this example is `<animal>dog</animal>`. This consists of the *start-tag* `<animal>`, the element *content*, and a matching *end-tag* `</animal>`. This `animal` element, together with the `person` element, are nested as part of the content of the `sentence` element. The nesting is possibly clearer (and closer to some of the more "structured" XML contained in the rest of this Primer) if the sentence is written:

```
<sentence>
    <person webid="http://example.com/#johnsmith">I</person>
    just got a new pet
    <animal>dog</animal>.
</sentence>
```

In some cases, an element may have no content. This can be written either by enclosing no content within the pair of delimiting start- and end-tags, as in `<animal></animal>`, or by using a shorthand form of tag called an *empty-element tag*, as in `<animal/>`.

In some cases, a start-tag (or empty-element tag) may contain qualifying information other than the tag name, in the form of *attributes*. For example, the start-tag of the `<person>` element contains the attribute `webid="http://example.com/#johnsmith"` (presumably identifying the specific person referred to). An attribute consists of a name, an equal sign, and a value (enclosed in quotes).

This particular markup language uses the words "sentence," "person," and "animal" as tag names in an attempt to convey some of the meaning of the elements; and they *would* convey meaning to an English-speaking person reading it, or to a program specifically written to interpret this vocabulary. However, there is no built-in meaning here. For example, to non-English speakers, or to a program not written to understand this markup, the element `<person>` may mean absolutely nothing. Take the following passage, for example:

```
<dfgre><reghh bjhbw="http://example.com/#johnsmith">I</reghh>
just got a new pet <yudis>dog</yudis>.</dfgre>
```

To a machine, this passage has exactly the same structure as the previous example. However, it is no longer clear to an English-speaker what is being said, because the tags are no longer English words. Moreover, others may have used the same words as tags in their own markup languages, but with completely different intended meanings. For example, "sentence" in another markup language might refer to the amount of time that a convicted criminal must serve in a penal institution. So additional mechanisms must be provided to help keep XML vocabulary straight.

To prevent confusion, it is necessary to uniquely identify markup elements. This is done in XML using XML Namespaces [XML-NS]. A *namespace* is just a way of identifying a part of the Web (space) which acts as a qualifier for a specific set of names. A namespace is created for an XML markup language by creating a URI for it. By qualifying tag names with the URIs of their namespaces, anyone can create their own tags and properly distinguish them from tags with identical spellings created by others. A convention that is sometimes followed is to create a Web page to describe the markup language (and the intended meaning of the tags) and use the URL of that Web page as the URI for its namespace. However, this is just a convention, and neither XML nor RDF assumes that a namespace URI identifies a retrievable Web resource. The following example illustrates the use of an XML namespace.

```
<user:sentence xmlns:user="http://example.com/xml/documents/">
   <user:person user:webid="http://example.com/#johnsmith">I</user:person>
just got a new pet <user:animal>dog</user:animal>.
</user:sentence>
```

In this example, the attribute `xmlns:user="http://example.com/xml/documents/"` declares a namespace for use in this piece of XML. It maps the *prefix* `user` to the namespace URI `http://example.com/xml/documents/`. The XML content can then use *qualified names* (or *QNames*) like `user:person` as tags. A QName contains a prefix that identifies a namespace, followed by a colon, and then a *local name* for an XML tag or attribute name. By using namespace URIs to distinguish specific groups of names, and qualifying tags with the URIs of the namespaces they come from, as in this example, there is no need to worry about tag names conflicting. Two tags having the same spelling are considered the same only if they also have the same namespace URIs.

Every XML document is required to be *well-formed*. This means the XML document must satisfy a number of syntactic conditions, for example, that every start-tag must have a matching end-tag, and that elements must be properly nested within other elements (elements may not overlap). The complete set of well-formedness conditions is defined in [XML].

In addition, an XML document may optionally include an XML *document type declaration* to define additional constraints on the structure of the document, and to support the use of predefined units of text within the document. The document type declaration (introduced with `DOCTYPE`) contains or points to declarations that define a grammar for the document. This grammar is known as a *document type definition*, or *DTD*. The declarations in a DTD specify such things as which XML elements and attributes may appear in XML documents corresponding to the DTD, the relationships of these elements and attributes (e.g., which elements can be nested within which other elements, or which attributes may appear with which elements), and whether elements or attributes are required or optional. The document type declaration can point to a set of declarations located outside the document (called the *external subset*, which can be used to allow common declarations to be shared among multiple documents), can include the declarations directly in the document (called the *internal subset*), or can have both internal and external DTD subsets. The complete DTD for a document consists of both subsets taken together. A simple example of an XML document with a document type declaration is shown in Example 47:

Example 47: An XML Document With a Document Type Declaration

```
<?xml version="1.0"?>
<!DOCTYPE greeting SYSTEM "http://www.example.org/dtds/hello.dtd">
<greeting>Hello, world!</greeting>
```

In this case, the document has only an external DTD subset, and the *system identifier* `http://www.example.org/dtds/hello.dtd` provides its location (a URIref).

An XML document is *valid* if it has an associated document type declaration and the document complies with the constraints defined by the document type declaration.

An RDF/XML document is only required to be well-formed XML; it is not intended to be validated against an XML DTD (or an XML Schema), and [RDF-SYNTAX] does not specify a normative DTD that could be used for validating arbitrary RDF/XML (an appendix of [RDF-SYNTAX] does provide a non-normative example schema for RDF/XML). As a result, more detailed discussion of XML DTD grammars is beyond the scope of this Primer. Further information on XML DTDs and XML validation can be found in [XML], and the numerous books on XML.

However, there is one use of XML document type declarations that *is* relevant to RDF/XML, and that is their use in defining XML *entities*. An XML entity declaration essentially associates a name with a string of characters. When the entity name is used elsewhere within an XML document, XML processors replace the entity name with the corresponding string. This provides a way to abbreviate long strings such as URIrefs, and can help make XML documents containing such strings more readable. Using a document type declaration just to declare XML entities is allowed, and can be useful, even when (as in RDF/XML) the documents are not intended to be validated.

In RDF/XML documents, entities are generally declared within the document itself, i.e., using only an internal DTD subset (one reason for this is that RDF/XML is not intended to be validated, and non-validating XML processors are not required to process external DTD subsets). For example, providing the document type declaration shown in Example 48 at the beginning of an RDF/XML document allows the URIrefs in that document for the `rdf`, `rdfs`, and `xsd` namespaces to be abbreviated as `&rdf;`, `&rdfs;`, and `&xsd;` respectively, as shown in the example.

Example 48: Some XML Entity Declarations

```
<?xml version='1.0'?>

<!DOCTYPE rdf:RDF [
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
]>

<rdf:RDF
    xmlns:rdf = "&rdf;"
    xmlns:rdfs = "&rdfs;"
    xmlns:xsd = "&xsd;">

...RDF statements...

</rdf:RDF>
```

Appendix C: Changes

Only minor editorial and typographic changes have been made since the Proposed Recommendation version. Older changes are detailed in its change log.


RDF
Metadata