Resource Description Framework (RDF) Model and Syntax Specification

W3C Recommendation 22 February 1999

This Version:
http://www.w3.org/TR/1999/REC-rdf-syntax-19990222
Newest Version:
http://www.w3.org/TR/REC-rdf-syntax
Editors:
Ora Lassila <ora.lassila@research.nokia.com>, Nokia Research Center
Ralph R. Swick <swick@w3.org>, World Wide Web Consortium

Document Status

Status of This Document

This document has been reviewed by W3C Members and other interested parties and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from other documents. 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.

The list of know errors in this specification is available at http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/errata.

Comments on this specification may be sent to <www-rdf-comments@w3.org>. The archive of public comments is available at http://www.w3.org/Archives/Public/www-rdf-comments.

Table of Contents

---

1. Introduction

The World Wide Web was originally built for human consumption, and although everything on it is *machine-readable*, this data is not *machine-understandable*. It is very hard to automate anything on the Web, and because of the volume of information the Web contains, it is not possible to manage it manually. The solution proposed here is to use *metadata* to describe the data contained on the Web. Metadata is "data about data" (for example, a library catalog is metadata, since it describes publications) or specifically in the context of this specification "data describing Web resources". The distinction between "data" and "metadata" is not an absolute one; it is a distinction created primarily by a particular application, and many times the same resource will be interpreted in both ways simultaneously.

Resource Description Framework (RDF) is a foundation for processing metadata; it provides interoperability between applications that exchange machine-understandable information on the Web. RDF emphasizes facilities to enable automated processing of Web resources. RDF can be used in a variety of application areas; for example: in *resource discovery* to provide better search engine capabilities, in *cataloging* for describing the content and content relationships available at a particular Web site, page, or digital library, by *intelligent software agents* to facilitate knowledge sharing and exchange, in *content rating*, in describing *collections of pages* that represent a single logical "document", for describing *intellectual property rights* of Web pages, and for expressing the *privacy preferences* of a user as well as the *privacy policies* of a Web site. RDF with *digital signatures* will be key to building the "Web of Trust" for electronic commerce, collaboration, and other applications.

This document introduces a model for representing RDF metadata as well as a syntax for encoding and transporting this metadata in a manner that maximizes the interoperability of independently developed Web servers and clients. The syntax presented here uses the Extensible Markup Language [XML]: one of the goals of RDF is to make it possible to specify semantics for data based on XML in a standardized, interoperable manner. RDF and XML are complementary: RDF is a model of metadata and only addresses by reference many of the encoding issues that transportation and file storage require (such as internationalization, character sets, etc.). For these issues, RDF relies on the support of XML. It is also important to understand that this XML syntax is only one possible syntax for RDF and that alternate ways to represent the same RDF data model may emerge.

The broad goal of RDF is to define a mechanism for describing resources that makes no assumptions about a particular application domain, nor defines (a priori) the semantics of any application domain. The definition of the mechanism should be domain neutral, yet the mechanism should be suitable for describing information about any domain.

This specification will be followed by other documents that will complete the framework. Most importantly, to facilitate the definition of metadata, RDF will have a class system much like many object-oriented programming and modeling systems. A collection of classes (typically authored for a specific purpose or domain) is called a *schema*. Classes are organized in a hierarchy, and offer extensibility through subclass refinement. This way, in order to create a schema slightly different from an existing one it is not necessary to "reinvent the wheel" but one can just provide incremental modifications to the base schema. Through the sharability of schemas RDF will support the reusability of metadata definitions. Due to RDF's incremental extensibility, agents processing metadata will be able to trace the origins of schemata they are unfamiliar with back to known schemata and perform meaningful actions on metadata they weren't originally designed to process. The sharability and extensibility of RDF also allows metadata authors to use multiple inheritance to "mix" definitions, to provide multiple views to their data, leveraging work done by others. In addition, it is possible to create RDF instance data based on multiple schemata from multiple sources (i.e., "interleaving" different types of metadata). Schemas may themselves be written in RDF; a companion document to this specification, [RDFSchema], describes one set of properties and classes for describing RDF schemas.

As a result of many communities coming together and agreeing on basic principles of metadata representation and transport, RDF has drawn influence from several different sources. The main influences have come from the *Web standardization community* itself in the form of HTML metadata and PICS, the *library community*, the *structured document community* in the form of SGML and more importantly XML, and also the *knowledge representation (KR) community.* There are also other areas of technology that contributed to the RDF design; these include object-oriented programming and modeling languages, as well as databases. While RDF draws from the KR community, readers familiar with that field are cautioned that RDF does not specify a mechanism for *reasoning*. RDF can be characterized as a simple frame system. A reasoning mechanism could be built on top of this frame system.

2. Basic RDF

2.1. Basic RDF Model

The foundation of RDF is a model for representing named properties and property values. The RDF model draws on well-established principles from various data representation communities. RDF properties may be thought of as attributes of resources and in this sense correspond to traditional attribute-value pairs. RDF properties also represent relationships between resources and an RDF model can therefore resemble an entity-relationship diagram. (More precisely, RDF Schemas — which are themselves instances of RDF data models — are ER diagrams.) In object-oriented design terminology, resources correspond to objects and properties correspond to instance variables.

The RDF data model is a syntax-neutral way of representing RDF expressions. The data model representation is used to evaluate equivalence in meaning. Two RDF expressions are equivalent if and only if their data model representations are the same. This definition of equivalence permits some syntactic variation in expression without altering the meaning. (See Section 6. for additional discussion of string comparison issues.)

The basic data model consists of three object types:

Resources	All things being described by RDF expressions are called *resources*. A resource may be an entire Web page; such as the HTML document "http://www.w3.org/Overview.html" for example. A resource may be a part of a Web page; e.g. a specific HTML or XML element within the document source. A resource may also be a whole collection of pages; e.g. an entire Web site. A resource may also be an object that is not directly accessible via the Web; e.g. a printed book. Resources are always named by URIs plus optional anchor ids (see [URI]). Anything can have a URI; the extensibility of URIs allows the introduction of identifiers for any entity imaginable.
Properties	A *property* is a specific aspect, characteristic, attribute, or relation used to describe a resource. Each property has a specific meaning, defines its permitted values, the types of resources it can describe, and its relationship with other properties. This document does not address how the characteristics of properties are expressed; for such information, refer to the RDF Schema specification).
Statements	A specific resource together with a named property plus the value of that property for that resource is an RDF *statement*. These three individual parts of a statement are called, respectively, the *subject*, the *predicate*, and the *object*. The object of a statement (i.e., the property value) can be another resource or it can be a literal; i.e., a resource (specified by a URI) or a simple string or other primitive datatype defined by XML. In RDF terms, a *literal* may have content that is XML markup but is not further evaluated by the RDF processor. There are some syntactic restrictions on how markup in literals may be expressed; see Section 2.2.1.

2.1.1. Examples

Resources are identified by a *resource identifier*. A resource identifier is a URI plus an optional anchor id (see Section 2.2.1.). For the purposes of this section, properties will be referred to by a simple name.

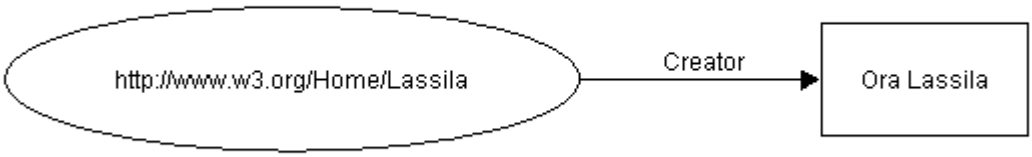
Consider as a simple example the sentence:

> *Ora Lassila is the creator of the resource http://www.w3.org/Home/Lassila.*

This sentence has the following parts:

Subject (Resource)	http://www.w3.org/Home/Lassila
Predicate (Property)	Creator
Object (literal)	"Ora Lassila"

In this document we will diagram an RDF statement pictorially using directed labeled graphs (also called "nodes and arcs diagrams"). In these diagrams, the nodes (drawn as ovals) represent resources and arcs represent named properties. Nodes that represent string literals will be drawn as rectangles. The sentence above would thus be diagrammed as:



D

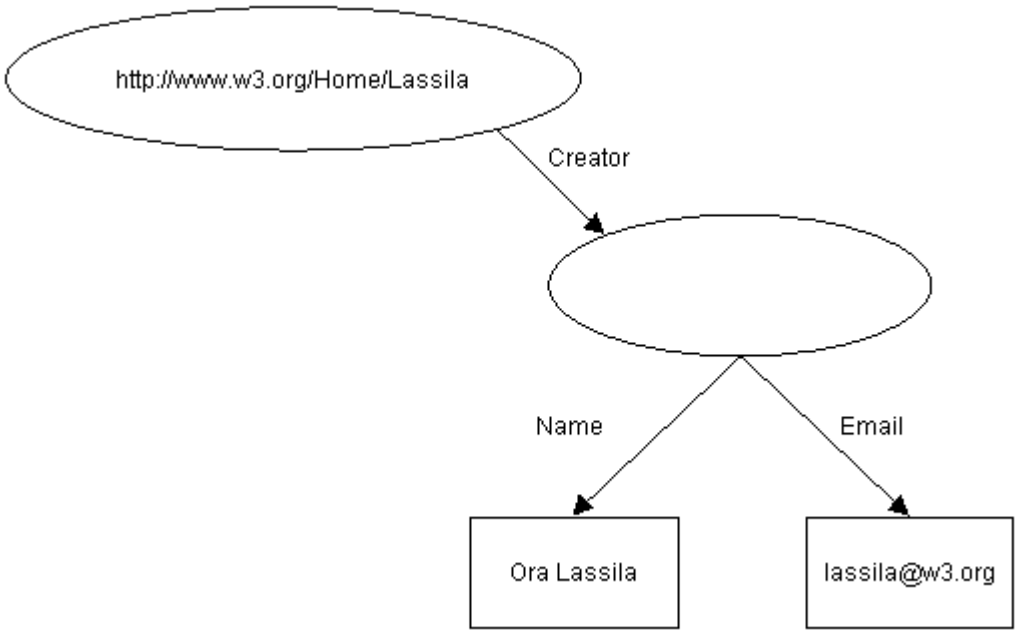
Figure 1: Simple node and arc diagram

Note: The direction of the arrow is important. The arc always starts at the subject and points to the object of the statement. The simple diagram above may also be read "*http://www.w3.org/Home/Lassila has creator Ora Lassila*", or in general "*<subject> HAS <predicate> <object>*".

Now, consider the case that we want to say something more about the characteristics of the creator of this resource. In prose, such a sentence would be:

The individual whose name is Ora Lassila, email <lassila@w3.org>, is the creator of http://www.w3.org/Home/Lassila.

The intention of this sentence is to make the value of the Creator property a structured entity. In RDF such an entity is represented as another resource. The sentence above does not give a name to that resource; it is anonymous, so in the diagram below we represent it with an empty oval:



D

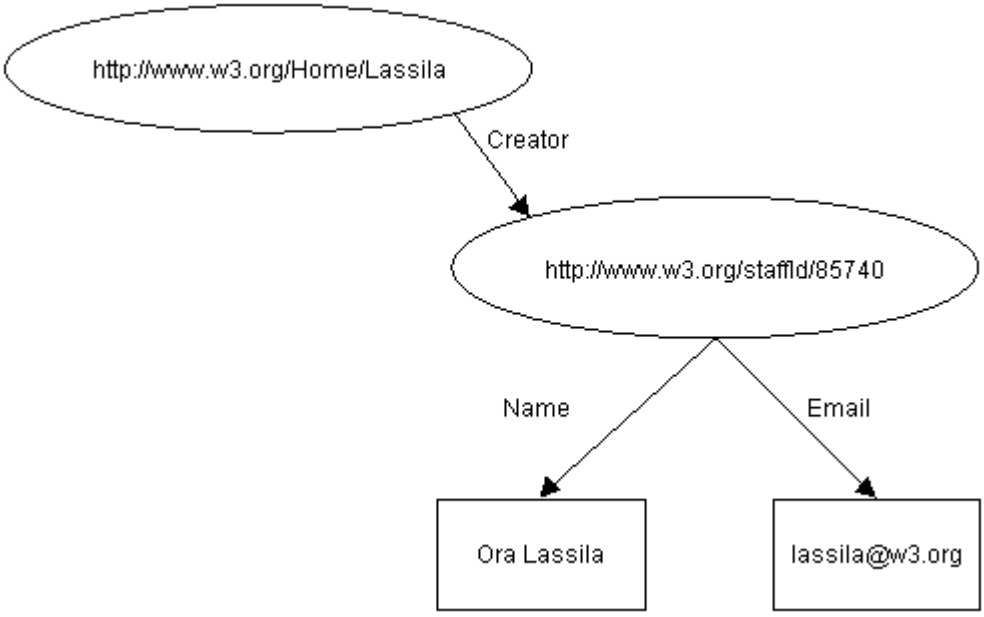
Figure 2: Property with structured value

Note: corresponding to the reading in the previous note, this diagram could be read "*http://www.w3.org/Home/Lassila has creator* something *and* something *has name Ora Lassila and email lassila@w3.org*".

The structured entity of the previous example can also be assigned a unique identifier. The choice of identifier is made by the application database designer. To continue the example, imagine that an employee id is used as the unique identifier for a "person" resource. The URIs that serve as the unique keys for each employee (as defined by the organization) might then be something like `http://www.w3.org/staffId/85740`. Now we can write the two sentences:

> *The individual referred to by employee id 85740 is named Ora Lassila and has the email address lassila@w3.org. The resource http://www.w3.org/Home/Lassila was created by this individual.*

The RDF model for these sentences is:



D

Figure 3: Structured value with identifier

Note that this diagram is identical to the previous one with the addition of the URI for the previously anonymous resource. From the point of view of a second application querying this model, there is no distinction between the statements made in a single sentence and the statements made in separate sentences. Some applications will need to be able to make such a distinction however, and RDF supports this; see Section 4, Statements about Statements, for further details.

2.2. Basic RDF Syntax

The RDF data model provides an abstract, conceptual framework for defining and using metadata. A concrete syntax is also needed for the purposes of creating and exchanging this metadata. This

specification of RDF uses the Extensible Markup Language [XML] encoding as its interchange syntax. RDF also requires the XML namespace facility to precisely associate each property with the schema that defines the property; see Section 2.2.3., Schemas and Namespaces.

The syntax descriptions in this document use the Extended Backus-Naur Form notation as defined in Section 6, Notation, of [XML] to describe the essential RDF syntax elements. The EBNF here is condensed for human readability; in particular, the italicized "*rdf*" is used to represent a variable namespace prefix rather than the more precise BNF notation "'<' NSprefix ':...'". The requirement that the property and type names in end-tags exactly match the names in the corresponding start-tags is implied by the XML rules. All syntactic flexibilities of XML are also implicitly included; e.g. whitespace rules, quoting using either single quote (') or double quote ("), character escaping, case sensitivity, and language tagging.

This specification defines two XML syntaxes for encoding an RDF data model instance. The *serialization syntax* expresses the full capabilities of the data model in a very regular fashion. The *abbreviated syntax* includes additional constructs that provide a more compact form to represent a subset of the data model. RDF interpreters are expected to implement both the full serialization syntax and the abbreviated syntax. Consequently, metadata authors are free to mix the two.

2.2.1. Basic Serialization Syntax

A single RDF statement seldom appears in isolation; most commonly several properties of a resource will be given together. The RDF XML syntax has been designed to accomodate this easily by grouping multiple statements for the same resource into a `Description` element. The `Description` element names, in an `about` attribute, the resource to which each of the statements apply. If the resource does not yet exist (i.e., does not yet have a resource identifier) then a `Description` element can supply the identifer for the resource using an `ID` attribute.

Basic RDF serialization syntax takes the form:

```
 [1] RDF             ::= ['<rdf:RDF>'] description* ['</rdf:RDF>']
 [2] description     ::= '<rdf:Description' idAboutAttr? '>' propertyElt*
                         '</rdf:Description>'
 [3] idAboutAttr     ::= idAttr | aboutAttr
 [4] aboutAttr       ::= 'about="' URI-reference '"'
 [5] idAttr          ::= 'ID="' IDsymbol '"'
 [6] propertyElt     ::= '<' propName '>' value '</' propName '>'
                       | '<' propName resourceAttr '/>'
 [7] propName        ::= Qname
 [8] value           ::= description | string
 [9] resourceAttr    ::= 'resource="' URI-reference '"'
[10] Qname           ::= [ NSprefix ':' ] name
[11] URI-reference   ::= string, interpreted per [URI]
[12] IDsymbol        ::= (any legal XML name symbol)
[13] name            ::= (any legal XML name symbol)
[14] NSprefix        ::= (any legal XML namespace prefix)
[15] string          ::= (any XML text, with "<", ">", and "&" escaped)
```

The `RDF` element is a simple wrapper that marks the boundaries in an XML document between which the content is explicitly intended to be mappable into an RDF data model instance. The `RDF` element is optional if the content can be known to be RDF from the application context.

`Description` contains the remaining elements that cause the creation of statements in the model

instance. The `Description` element may be thought of (for purposes of the basic RDF syntax) as simply a place to hold the identification of the resource being described. Typically there will be more than one statement made about a resource; `Description` provides a way to give the resource name just once for several statements.

When the `about` attribute is specified with `Description`, the statements in the `Description` refer to the resource whose identifier is determined from the `about`. The value of the `about` attribute is interpreted as a URI-reference per Section 4 of [URI]. The corresponding resource identifier is obtained by resolving the URI-reference to absolute form as specified by [URI]. If a fragment identifier is included in the URI-reference then the resource identifier refers only to the subcomponent of the containing resource that is identifed by the corresponding fragment id internal to that containing resource (see anchor in [Dexter94]), otherwise the identifier refers to the entire resource specified by the URI. A `Description` element without an `about` attribute represents a new resource. Such a resource might be a surrogate, or proxy, for some other physical resource that does not have a recognizable URI. The value of the `ID` attribute of the `Description` element, if present, is the anchor id of this "in-line" resource.

If another `Description` or property value needs to refer to the in-line resource it will use the value of the `ID` of that resource in its own `about` attribute. The `ID` attribute signals the creation of a new resource and the `about` attribute refers to an existing resource; therefore either `ID` or `about` may be specified on `Description` but not both together in the same element. The values for each `ID` attribute must not appear in more than one `ID` attribute within a single document.

A single `Description` may contain more than one *propertyElt* element with the same property name. Each such propertyElt adds one arc to the graph. The interpretation of this graph is defined by the schema designer.

Within a *propertyElt*, the `resource` attribute specifies that some other resource is the value of this property; that is, the object of the statement is another resource identified by URI rather than a literal. The resource identifier of the object is obtained by resolving the `resource` attribute URI-reference in the same manner as given above for the `about` attribute. *`String`*s must be well-formed XML; the usual XML content quoting and escaping mechanisms may be used if the string contains character sequences (e.g. "<" and "&") that violate the well-formedness rules or that otherwise might look like markup. See Section 6. for additional syntax to specify a property value with well-formed XML content containing markup such that the markup is not interpreted by RDF.

Property names must be associated with a schema. This can be done by qualifying the element names with a namespace prefix to unambiguously connect the property definition with the corresponding RDF schema or by declaring a default namespace as specified in [NAMESPACES].

The example sentence from Section 2.1.1

> *Ora Lassila is the creator of the resource http://www.w3.org/Home/Lassila.*

is represented in RDF/XML as:

```
<rdf:RDF>
  <rdf:Description about="http://www.w3.org/Home/Lassila">
    <s:Creator>Ora Lassila</s:Creator>
  </rdf:Description>
```

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

Here the namespace prefix 's' refers to a specific namespace prefix chosen by the author of this RDF expression and defined in an XML namespace declaration such as:

```
xmlns:s="http://description.org/schema/"
```

This namespace declaration would typically be included as an XML attribute on the *rdf*:RDF element but may also be included with a particular Description element or even an individual propertyElt expression. The namespace name URI in the namespace declaration is a globally unique identifier for the particular schema this metadata author is using to define the use of the Creator property. Other schemas may also define a property named Creator and the two properties will be distinguished via their schema identifiers. Note also that a schema usually defines several properties; a single namespace declaration will suffice to make a large vocabulary of properties available for use.

The complete XML document containing the description above would be:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s="http://description.org/schema/">
  <rdf:Description about="http://www.w3.org/Home/Lassila">
    <s:Creator>Ora Lassila</s:Creator>
  </rdf:Description>
</rdf:RDF>
```

Using the default namespace syntax defined in [NAMESPACES] for the RDF namespace itself, this document could also be written as:

```
<?xml version="1.0"?>
<RDF
  xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s="http://description.org/schema/">
  <Description about="http://www.w3.org/Home/Lassila">
    <s:Creator>Ora Lassila</s:Creator>
  </Description>
</RDF>
```

Furthermore, namespace declarations can be associated with an individual Description element or even an individual propertyElt element as in:

```
<?xml version="1.0"?>
<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <Description about="http://www.w3.org/Home/Lassila">
    <s:Creator xmlns:s="http://description.org/schema/">Ora Lassila</s:Creator>
  </Description>
</RDF>
```

As XML namespace declarations may be nested, the previous example may be further condensed to:

```
<?xml version="1.0"?>
<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <Description about="http://www.w3.org/Home/Lassila">
    <Creator xmlns="http://description.org/schema/">Ora Lassila</Creator>
  </Description>
</RDF>
```

Highly condensed expressions such as this are discouraged, however, when the RDF/XML encoding is written by hand or expected to be edited in a plain text editor. Though unambiguous, the possibility of error is greater than if explicit prefixes are used consistently. Note that an RDF/XML fragment that is intended to be inserted in other documents should declare all the namespaces it uses so that it is completely self-contained. For readability, the introductory examples in the remainder of this section omit the namespace declarations in order to not obscure the specific points being illustrated.

2.2.2. Basic Abbreviated Syntax

While the serialization syntax shows the structure of an RDF model most clearly, often it is desirable to use a more compact XML form. The RDF *abbreviated syntax* accomplishes this. As a further benefit, the abbreviated syntax allows documents obeying certain well-structured XML DTDs to be directly interpreted as RDF models.

Three forms of abbreviation are defined for the basic serialization syntax. The first is usable for properties that are not repeated within a `Description` and where the values of those properties are literals. In this case, the properties may be written as XML attributes of the `Description` element. The previous example then becomes:

```
<rdf:RDF>
  <rdf:Description about="http://www.w3.org/Home/Lassila"
                   s:Creator="Ora Lassila" />
</rdf:RDF>
```

Note that since the `Description` element has no other content once the Creator property is written in XML attribute form, the XML empty element syntax is employed to elide the `Description` end-tag.

Here is another example of the use of this same abbreviation form:

```
<rdf:RDF>
  <rdf:Description about="http://www.w3.org">
    <s:Publisher>World Wide Web Consortium</s:Publisher>
    <s:Title>W3C Home Page</s:Title>
    <s:Date>1998-10-03T02:27</s:Date>
  </rdf:Description>
</rdf:RDF>
```

is equivalent for RDF purposes to

```
<rdf:RDF>
  <rdf:Description about="http://www.w3.org"
        s:Publisher="World Wide Web Consortium"
        s:Title="W3C Home Page"
        s:Date="1998-10-03T02:27"/>
</rdf:RDF>
```

Note that while these two RDF expressions are equivalent, they may be treated differently by other processing engines. In particular, if these two expressions were embedded into an HTML document then the default behavior of a non-RDF-aware browser would be to display the values of the properties in the first case while in the second case there should be no text displayed (or at most a whitespace character).

The second RDF abbreviation form works on nested `Description` elements. This abbreviation form can be employed for specific statements when the object of the statement is another resource and the values of any properties given in-line for this second resource are strings. In this case, a similar transformation of XML element names into XML attributes is used: the properties of the resource in the nested `Description` may be written as XML attributes of the propertyElt element in which that `Description` was contained.

The second example sentence from Section 2.1.1

> *The individual referred to by employee id 85740 is named Ora Lassila and has the email address lassila@w3.org. The resource http://www.w3.org/Home/Lassila was created by this individual.*

is written in RDF/XML using explicit serialization form as

```
<rdf:RDF>
  <rdf:Description about="http://www.w3.org/Home/Lassila">
    <s:Creator rdf:resource="http://www.w3.org/staffId/85740"/>
  </rdf:Description>

  <rdf:Description about="http://www.w3.org/staffId/85740">
    <v:Name>Ora Lassila</v:Name>
    <v:Email>lassila@w3.org</v:Email>
  </rdf:Description>
</rdf:RDF>
```

This form makes it clear to a reader that two separate resources are being described but it is less clear that the second resource is used within the first description. This same expression could be written in the following way to make this relationship more obvious to the human reader. Note that to the machine, there is no difference:

```
<rdf:RDF>
  <rdf:Description about="http://www.w3.org/Home/Lassila">
    <s:Creator>
      <rdf:Description about="http://www.w3.org/staffId/85740">
        <v:Name>Ora Lassila</v:Name>
        <v:Email>lassila@w3.org</v:Email>
      </rdf:Description>
    </s:Creator>
  </rdf:Description>
</rdf:RDF>
```

Using the second basic abbreviation syntax, the inner `Description` element and its contained property expressions can be written as attributes of the Creator element:

```
<rdf:RDF>
  <rdf:Description about="http://www.w3.org/Home/Lassila">
    <s:Creator rdf:resource="http://www.w3.org/staffId/85740"
       v:Name="Ora Lassila"
       v:Email="lassila@w3.org" />
  </rdf:Description>
</rdf:RDF>
```

When using this abbreviation form the `about` attribute of the nested `Description` element becomes a `resource` attribute on the propertyElt element, as the resource named by the URI is in both cases the value of the Creator property. It is entirely a matter of writer's preference which of the three forms above are used in the RDF source. They all produce the same internal RDF models.

> *Note: The observant reader who has studied the remainder of this document will see that there are some additional relationships represented by a `Description` element to preserve the specific syntactic grouping of statements. Consequently the three forms above are slightly different in ways not important to the discussion in this section. These differences become important only when making higher-order statements as described in Section 4.*

The third basic abbreviation applies to the common case of a `Description` element containing a `type` property (see Section 4.1 for the meaning of `type`). In this case, the resource type defined in the schema corresponding to the value of the `type` property can be used directly as an element name. For example, using the previous RDF fragment if we wanted to add the fact that the resource http://www.w3.org/staffId/85740 represents an instance of a Person, we would write this in full serialization syntax as:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s="http://description.org/schema/">
  <rdf:Description about="http://www.w3.org/Home/Lassila">
    <s:Creator>
      <rdf:Description about="http://www.w3.org/staffId/85740">
        <rdf:type resource="http://description.org/schema/Person"/>
        <v:Name>Ora Lassila</v:Name>
        <v:Email>lassila@w3.org</v:Email>
      </rdf:Description>
    </s:Creator>
  </rdf:Description>
</rdf:RDF>
```

and using this third abbreviated form as:

```
<rdf:RDF>
  <rdf:Description about="http://www.w3.org/Home/Lassila">
    <s:Creator>
      <s:Person about="http://www.w3.org/staffId/85740">
        <v:Name>Ora Lassila</v:Name>
        <v:Email>lassila@w3.org</v:Email>
      </s:Person>
    </s:Creator>
  </rdf:Description>
</rdf:RDF>
```

The EBNF for the basic abbreviated syntax replaces productions [2] and [6] of the grammar for the basic serialization syntax in the following manner:

```
[2a] description    ::= '<rdf:Description' idAboutAttr? propAttr* '/>'
                      | '<rdf:Description' idAboutAttr? propAttr* '>'
                            propertyElt* '</rdf:Description>'
                      | typedNode
[6a] propertyElt    ::= '<' propName '>' value '</' propName '>'
```

```
                        | '<' propName resourceAttr? propAttr* '/>'
[16] propAttr       ::= propName '="' string '"'
                        (with embedded quotes escaped)
[17] typedNode      ::= '<' typeName idAboutAttr? propAttr* '/>'
                        | '<' typeName idAboutAttr? propAttr* '>'
                            property* '</' typeName '>'
```

2.2.3. Schemas and Namespaces

When we write a sentence in natural language we use words that are meant to convey a certain meaning. That meaning is crucial to understanding the statements and, in the case of applications of RDF, is crucial to establishing that the correct processing occurs as intended. It is crucial that *both* the writer and the reader of a statement understand the same meaning for the terms used, such as Creator, approvedBy, Copyright, etc. or confusion will result. In a medium of global scale such as the World Wide Web it is not sufficient to rely on shared cultural understanding of concepts such as "creatorship"; it pays to be as precise as possible.

Meaning in RDF is expressed through reference to a *schema*. You can think of a schema as a kind of dictionary. A schema defines the terms that will be used in RDF statements and gives specific meanings to them. A variety of schema forms can be used with RDF, including a specific form defined in a separate document [RDFSchema] that has some specific characteristics to help with automating tasks using RDF.

A schema is the place where definitions and restrictions of usage for properties are documented. In order to avoid confusion between independent -- and possibly conflicting -- definitions of the same term, RDF uses the XML namespace facility. Namespaces are simply a way to tie a specific use of a word in context to the dictionary (schema) where the intended definition is to be found. In RDF, each predicate used in a statement must be identified with exactly one namespace, or schema. However, a `Description` element may contain statements with predicates from many schemas. Examples of RDF Descriptions that use more than one schema appear in Section 7.

2.3. Qualified Property Values

Often the value of a property is something that has additional contextual information that is considered "part of" that value. In other words, there is a need to qualify property values. Examples of such qualification include naming a unit of measure, a particular restricted vocabulary, or some other annotation. For some uses it is appropriate to use the property value without the qualifiers. For example, in the statement "the price of that pencil is 75 U.S. cents" it is often sufficient to say simply "the price of that pencil is 75".

In the RDF model a qualified property value is simply another instance of a structured value. The object of the original statement is this structured value and the qualifiers are further properties of this common resource. The principal value being qualified is given as the value of the *value* property of this common resource. See Section 7.3. Non-Binary Relations for an example of the use of the *value* property.

3. Containers

Frequently it is necessary to refer to a collection of resources; for example, to say that a work was created by more than one person, or to list the students in a course, or the software modules in a

package. RDF containers are used to hold such lists of resources or literals.

3.1. Container Model

RDF defines three types of container objects:

Bag	An unordered list of resources or literals. *Bag*s are used to declare that a property has multiple values and that there is no significance to the order in which the values are given. *Bag* might be used to give a list of part numbers where the order of processing the parts does not matter. Duplicate values are permitted.
Sequence	An ordered list of resources or literals. *Sequence* is used to declare that a property has multiple values and that the order of the values is significant. *Sequence* might be used, for example, to preserve an alphabetical ordering of values. Duplicate values are permitted.
Alternative	A list of resources or literals that represent alternatives for the (single) value of a property. *Alternative* might be used to provide alternative language translations for the title of a work, or to provide a list of Internet mirror sites at which a resource might be found. An application using a property whose value is an *Alternative* collection is aware that it can choose any one of the items in the list as appropriate.

> *Note: The definitions of* Bag *and* Sequence *explicitly permit duplicate values. RDF does not define a core concept of* Set*, which would be a* Bag *with no duplicates, because the RDF core does not mandate an enforcement mechanism in the event of violations of such constraints. Future work layered on the RDF core may define such facilities.*

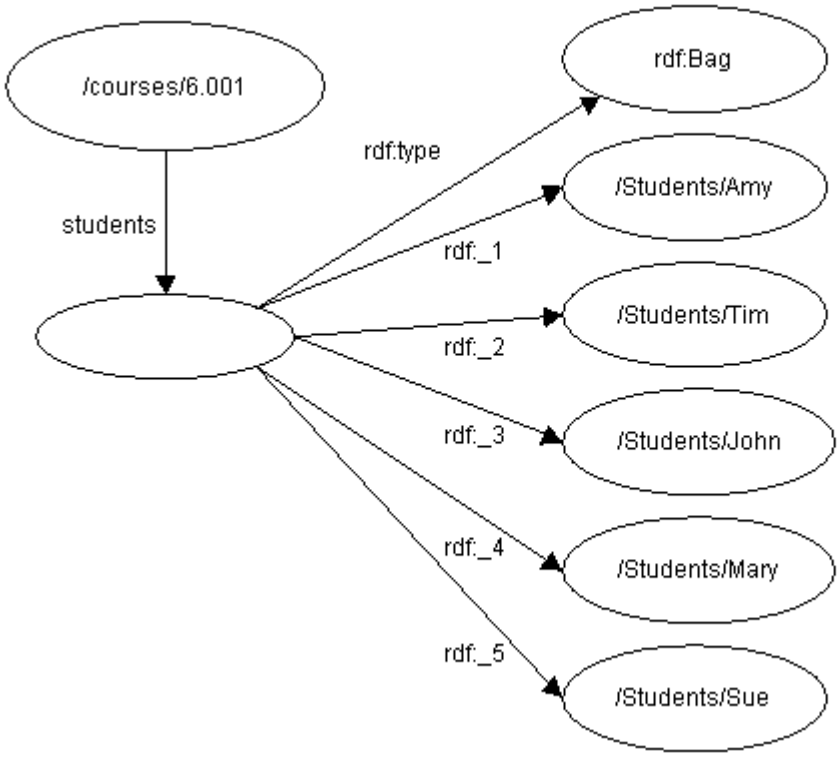
To represent a collection of resources, RDF uses an additional resource that identifies the specific collection (an *instance* of a collection, in object modeling terminology). This resource must be declared to be an instance of one of the container object types defined above. The *type* property, defined below, is used to make this declaration. The membership relation between this container resource and the resources that belong in the collection is defined by a set of properties defined expressly for this purpose. These membership properties are named simply "_1", "_2", "_3", etc. Container resources may have other properties in addition to the membership properties and the *type* property. Any such additional statements describe the container; see Section 3.3, Distributive Referents, for discussion of statements about each of the members themselves.

A common use of containers is as the value of a property. When used in this way, the statement still has a single statement object regardless of the number of members in the container; the container resource itself is the object of the statement.

For example, to represent the sentence

> *The students in course 6.001 are Amy, Tim, John, Mary, and Sue.*

the RDF model is



D

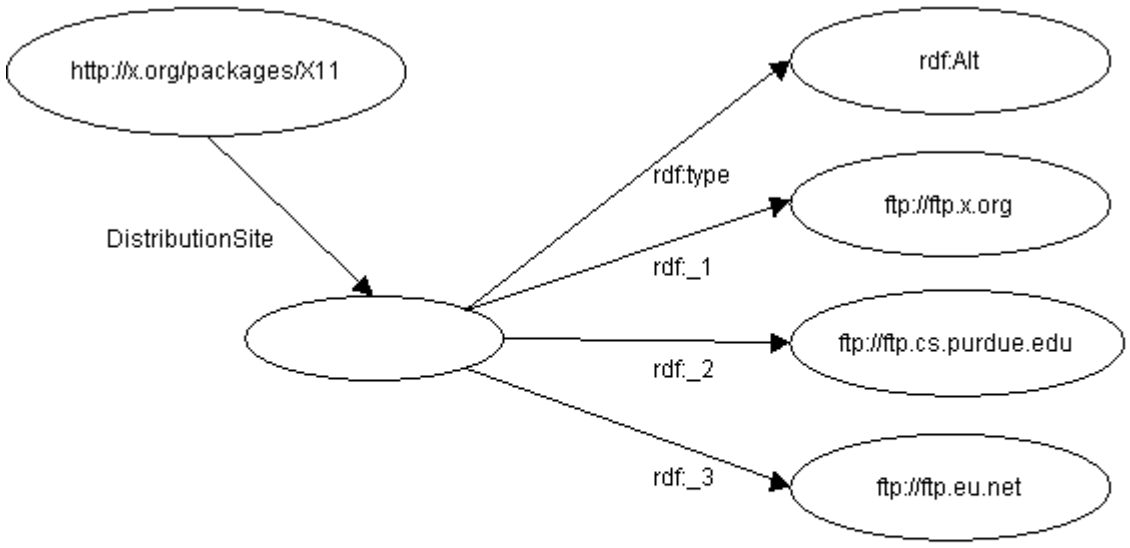
Figure 4: Simple Bag container

Bag containers are not equivalent to repeated properties of the same type; see Section 3.5. for a discussion of the difference. Authors will need to decide on a case-by-case basis which one (repeated property statement or Bag) is more appropriate to use.

The sentence

> *The source code for X11 may be found at ftp.x.org, ftp.cs.purdue.edu, or ftp.eu.net.*

is modeled in RDF as



D

Figure 5: Simple Alternative container

Alternative containers are frequently used in conjunction with language tagging. A work whose title has been translated into several languages might have its Title property pointing to an Alternative container holding each of the language variants.

3.2. Container Syntax

RDF container syntax takes the form:

```
[18] container       ::= sequence | bag | alternative
[19] sequence        ::= '<rdf:Seq' idAttr? '>' member* '</rdf:Seq>'
[20] bag             ::= '<rdf:Bag' idAttr? '>' member* '</rdf:Bag>'
[21] alternative     ::= '<rdf:Alt' idAttr? '>' member+ '</rdf:Alt>'
[22] member          ::= referencedItem | inlineItem
[23] referencedItem  ::= '<rdf:li' resourceAttr '/>'
[24] inlineItem      ::= '<rdf:li>' value '</rdf:li>'
```

Containers may be used everywhere a `Description` is permitted:

```
[1a] RDF             ::= '<rdf:RDF>' obj* '</rdf:RDF>'
[8a] value           ::= obj | string
[25] obj             ::= description | container
```

Note that RDF/XML uses `li` as a convenience element to avoid having to explicitly number each member. The `li` element assigns the properties `_1`, `_2`, and so on as necessary. The element name `li` was chosen to be mnemonic with the term "list item" from HTML.

An `Alt` container is required to have at least one member. This member will be identified by the property `_1` and is the default or preferred value.

> *Note: The RDF Schema specification* [RDFSCHEMA] *also defines a mechanism to declare additional subclasses of these container types, in which case production [18] is extended to include the names of those declared subclasses. There is also a syntax for writing literal values in attribute form; see the full grammar in Section 6.*

3.2.1. Examples

The model for the sentence

> *The students in course 6.001 are Amy, Tim, John, Mary, and Sue.*

is written in RDF/XML as

```
<rdf:RDF>
  <rdf:Description about="http://mycollege.edu/courses/6.001">
    <s:students>
      <rdf:Bag>
        <rdf:li resource="http://mycollege.edu/students/Amy"/>
        <rdf:li resource="http://mycollege.edu/students/Tim"/>
        <rdf:li resource="http://mycollege.edu/students/John"/>
        <rdf:li resource="http://mycollege.edu/students/Mary"/>
        <rdf:li resource="http://mycollege.edu/students/Sue"/>
      </rdf:Bag>
    </s:students>
  </rdf:Description>
</rdf:RDF>
```

In this case, since the value of the students property is expressed as a Bag there is no significance to the order given here for the URIs of each student.

The model for the sentence

> *The source code for X11 may be found at ftp.x.org, ftp.cs.purdue.edu, or ftp.eu.net.*

is written in RDF/XML as

```
<rdf:RDF>
  <rdf:Description about="http://x.org/packages/X11">
    <s:DistributionSite>
      <rdf:Alt>
        <rdf:li resource="ftp://ftp.x.org"/>
        <rdf:li resource="ftp://ftp.cs.purdue.edu"/>
        <rdf:li resource="ftp://ftp.eu.net"/>
      </rdf:Alt>
    </s:DistributionSite>
  </rdf:Description>
</rdf:RDF>
```

Here, any one of the items listed in the container value for DistributionSite is an acceptable value without regard to the other items.

3.3. Distributive Referents: Statements about Members of a Container

Container structures give rise to an issue about statements: when a statement is made referring to a collection, what "thing" is the statement describing? Or in other words, to what object is the statement is referring? Is the statement describing the container itself or is the statement describing the members of the container? The object being described (in the XML syntax indicated by the `about` attribute) is in RDF called the *referent*.

The following example:

```
<rdf:Bag ID="pages">
  <rdf:li resource="http://foo.org/foo.html" />
  <rdf:li resource="http://bar.org/bar.html" />
</rdf:Bag>

<rdf:Description about="#pages">
  <s:Creator>Ora Lassila</s:Creator>
</rdf:Description>
```

expresses that "Ora Lassila" is the creator of the Bag "pages". It does not, however, say anything about the individual pages, the members of the Bag. The referent of the `Description` is the container (the Bag), not its members. One would sometimes like to write a statement about each of the contained objects individually, instead of the container itself. In order to express that "Ora Lassila" is the creator of each of the pages, a different kind of referent is called for, one that *distributes* over the members of the container. This referent in RDF is expressed using the `aboutEach` attribute:

```
[3a] idAboutAttr    ::= idAttr | aboutAttr | aboutEachAttr
[26] aboutEachAttr  ::= 'aboutEach="' URI-reference '"'
```

As an example, if we wrote

```
<rdf:Description aboutEach="#pages">
  <s:Creator>Ora Lassila</s:Creator>
</rdf:Description>
```

we would get the desired meaning. We will call the new referent type a *distributive referent*. Distributive referents allow us to "share structure" in an RDF `Description`. For example, when writing several `Descriptions` that all have a number of common statement parts (predicates and objects), the common parts can be shared among all the `Descriptions`, possibly resulting in space savings and more maintainable metadata. The value of an `aboutEach` attribute must be a container. Using a distributive referent on a container is the same as making all the statements about each of the members separately.

No explicit graph representation of distributive referents is defined. Instead, in terms of the statements made, distributive referents are expanded into the individual statements about the individual container members (internally, implementations are free to retain information about the distributive referents - in order to save space, for example - as long as any querying functions work as if all of the statements were made individually). Thus, with respect to the resources "foo" and "bar", the above example is equivalent to

```
<rdf:Description about="http://foo.org/foo.html">
```

```
  <s:Creator>Ora Lassila</s:Creator>
</rdf:Description>

<rdf:Description about="http://bar.org/bar.html">
  <s:Creator>Ora Lassila</s:Creator>
</rdf:Description>
```

3.4. Containers Defined By A URI Pattern

One very frequent use of metadata is to make statements about "all pages at my Web site", or "all pages in this branch of my Web site". In many cases it is impractical or even undesirable to try to list each such resource explicitly and identify it as a member of a container. RDF therefore has a second distributive referent type. This second distributive referent type is a shorthand syntax that represents an instance of a Bag whose members are by definition all resources whose resource identifiers begin with a specified string:

```
[26a] aboutEachAttr  ::= 'aboutEach="' URI-reference '"'
                       | 'aboutEachPrefix="' string '"'
```

The `aboutEachPrefix` attribute declares that there is a Bag whose members are all the resources whose fully resolved resource identifiers begin with the character string given as the value of the attribute. The statements in a `Description` that has the `aboutEachPrefix` attribute apply individually to each of the members of this Bag.

For example, if the two resources http://foo.org/doc/page1 and http://foo.org/doc/page2 exist then we can say that each of them has a copyright property by writing

```
<rdf:Description aboutEachPrefix="http://foo.org/doc">
  <s:Copyright>© 1998, The Foo Organization</s:Copyright>
</rdf:Description>
```

If these are the only two resources whose URIs start with that string then the above is equivalent to both of the following alternatives:

```
<rdf:Description about="http://foo.org/doc/page1">
  <s:Copyright>© 1998, The Foo Organization</s:Copyright>
</rdf:Description>
<rdf:Description about="http://foo.org/doc/page2">
  <s:Copyright>© 1998, The Foo Organization</s:Copyright>
</rdf:Description>
```

and

```
<rdf:Description aboutEach="#docpages">
  <s:Copyright>© 1998, The Foo Organization</s:Copyright>
</rdf:Description>
<rdf:Bag ID="docpages">
  <rdf:li resource="http://foo.org/doc/page1"/>
  <rdf:li resource="http://foo.org/doc/page2"/>
</rdf:Bag>
```

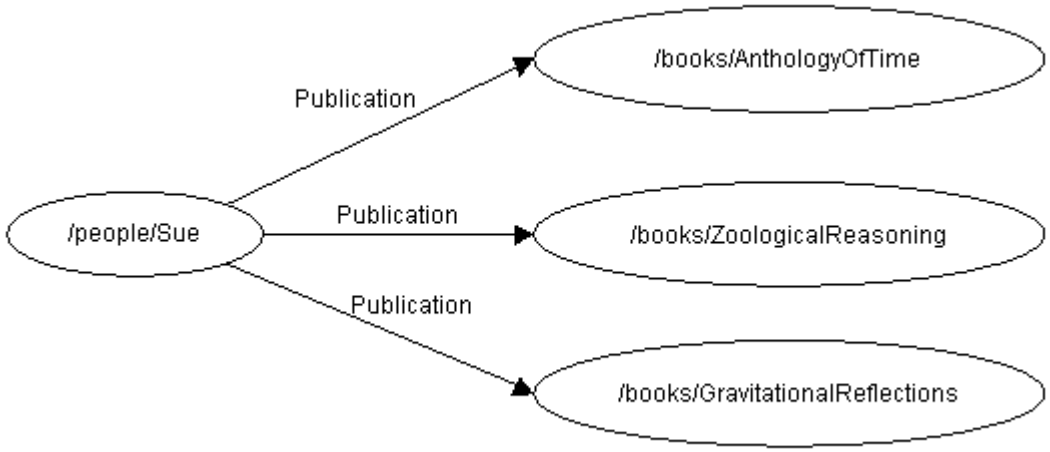
3.5. Containers Versus Repeated Properties

A resource may have multiple statements with the same predicate (i.e., using the same property). This is not the same as having a single statement whose object is a container containing multiple members. The choice of which to use in any particular circumstance is in part made by the person who designs the schema and in part made by the person who writes the specific RDF statements.

Consider as an example the relationship between a writer and her publications. We might have the sentence

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

That is, there are three resources each of which was written independently by the same writer.



D

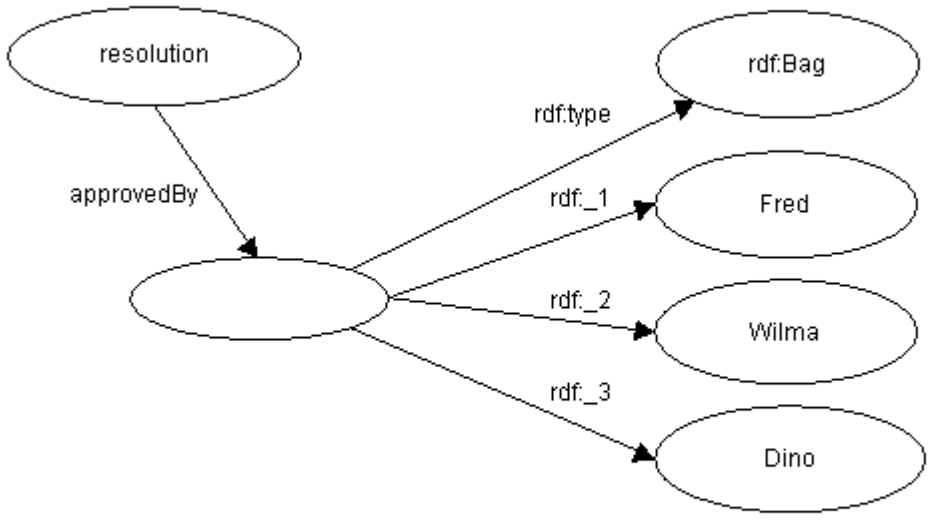
Figure 6: Repeated property

In this example there is no stated relationship between the publications other than that they were written by the same person.

On the other hand, the sentence

> *The committee of Fred, Wilma, and Dino approved the resolution.*

says that the three committee members as a whole voted in a certain manner; it does not necessarily state that each committee member voted in favor of the article. It would be incorrect to model this sentence as three separate approvedBy statements, one for each committee member, as this would state the vote of each individual member. Rather, it is better to model this as a single approvedBy statement whose object is a Bag containing the committee members' identities:



D

Figure 7: Using Bag to indicate a collective opinion

The choice of which representation to use, Bag or repeated property, is made by the person creating the metadata after considering the schema. If, for example, in the publications example above we wished to say that those were the complete set of publications then the schema might include a property called *publications* for that purpose. The value of the *publications* property would be a Bag listing all of Sue's works.

4. Statements about Statements

In addition to making statements about Web resources, RDF can be used for making statements about other RDF statements; we will refer to these as *higher-order statements*. In order to make a statement about another statement, we actually have to build a model of the original statement; this model is a new resource to which we can attach additional properties.

4.1. Modeling Statements

Statements are made about resources. A model of a statement is the resource we need in order to be able to make new statements (higher-order statements) about the modeled statement.

For example, let us consider the sentence

> *Ora Lassila is the creator of the resource http://www.w3.org/Home/Lassila.*

RDF would regard this sentence as a fact. If, instead, we write the sentence

> *Ralph Swick says that Ora Lassila is the creator of the resource http://www.w3.org/Home/Lassila.*

we have said nothing about the resource http://www.w3.org/Home/Lassila; instead, we have expressed a fact about a statement Ralph has made. In order to express this fact to RDF, we have to model the original statement as a resource with four properties. This process is formally called *reification* in the Knowledge Representation community. A model of a statement is called a *reified statement*.

To model statements RDF defines the following properties:

subject	The *subject* property identifies the resource being described by the modeled statement; that is, the value of the *subject* property is the resource about which the original statement was made (in our example, http://www.w3.org/Home/Lassila).
predicate	The *predicate* property identifies the original property in the modeled statement. The value of the *predicate* property is a resource representing the specific property in the original statement (in our example, creator).
object	The *object* property identifies the property value in the modeled statement. The value of the *object* property is the object in the original statement (in our example, "Ora Lassila").
type	The value of the *type* property describes the type of the new resource. All reified statements are instances of RDF:Statement; that is, they have a *type* property whose object is RDF:Statement. The *type* property is also used more generally to declare the type of any resource, as was shown in Section 3, "Containers".

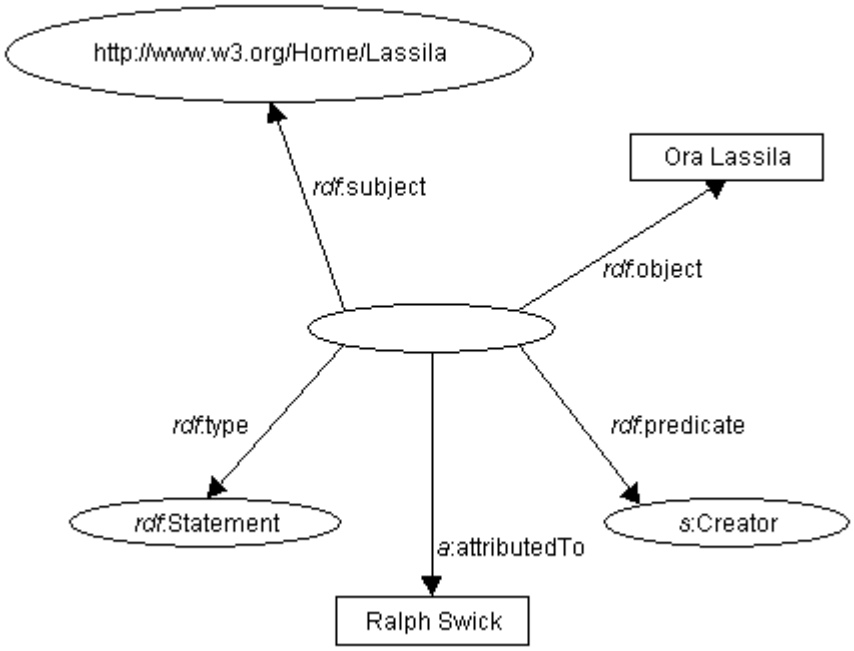
A new resource with the above four properties represents the original statement and can both be used as the object of other statements and have additional statements made about it. The resource with these four properties is not a replacement for the original statement, it is a model of the statement. A statement and its corresponding reified statement exist independently in an RDF graph and either may be present without the other. The RDF graph is said to contain the fact given in the statement if and only if the statement is present in the graph, irrespective of whether the corresponding reified statement is present.

To model the example above, we could attach another property to the reified statement (say, "attributedTo") with an appropriate value (in this case, "Ralph Swick"). Using base-level RDF/XML syntax, this could be written as

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:a="http://description.org/schema/">
  <rdf:Description>
    <rdf:subject resource="http://www.w3.org/Home/Lassila" />
    <rdf:predicate resource="http://description.org/schema/Creator" />
    <rdf:object>Ora Lassila</rdf:object>
    <rdf:type resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement" />
```

```
    <a:attributedTo>Ralph Swick</a:attributedTo>
  </rdf:Description>
</rdf:RDF>
```

Figure 8 represents this in graph form. Syntactically this is rather verbose; in Section 4.2. we present a shorthand for making statements about statements.



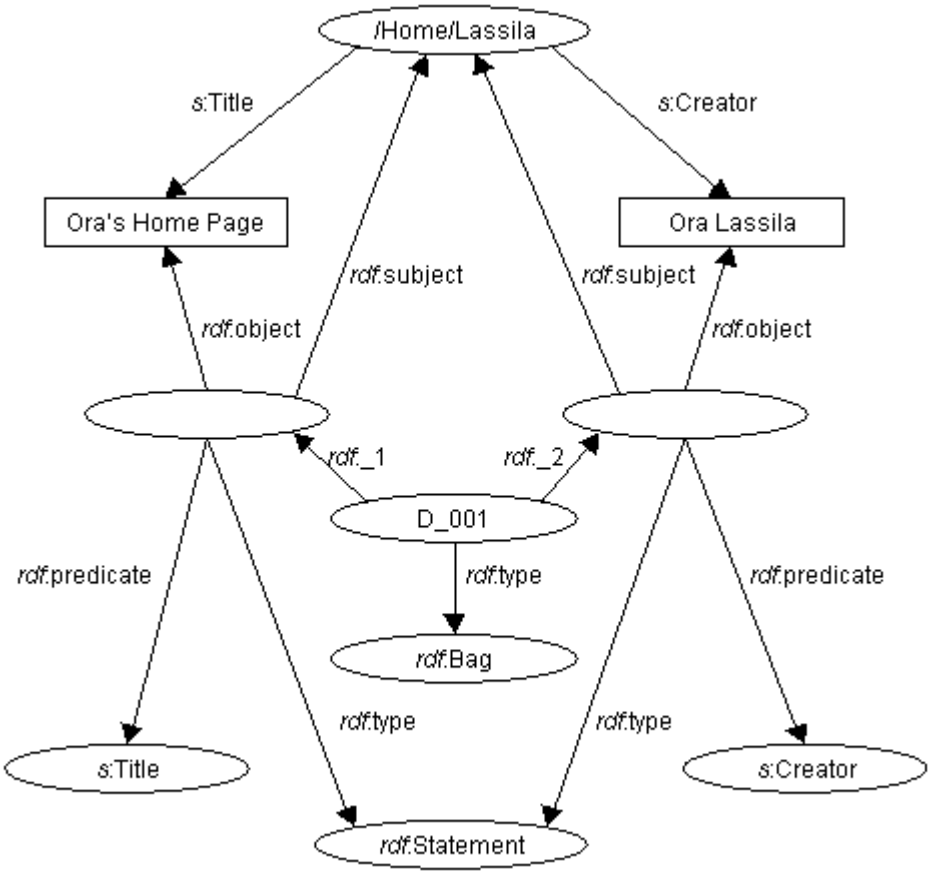
D

Figure 8: Representation of a reified statement

Reification is also needed to represent explicitly in the model the statement grouping implied by `Description` elements. The RDF graph model does not need a special construct for `Descriptions`; since `Descriptions` really are collections of statements, a `Bag` container is used to indicate that a set of statements came from the same (syntactic) `Description`. Each statement within a `Description` is reified and each of the reified statements is a member of the Bag representing that `Description`. As an example, the RDF fragment

```
<rdf:RDF>
  <rdf:Description about="http://www.w3.org/Home/Lassila" bagID="D_001">
    <s:Creator>Ora Lassila</s:Creator>
    <s:Title>Ora's Home Page</s:Title>
  </rdf:Description>
</rdf:RDF>
```

would result in the graph shown in Figure 9.



D

Figure 9: Using Bag to represent statement grouping

Note the new attribute `bagID`. This attribute specifies the resource id of the container resource:

```
[2b] description    ::= '<rdf:Description' idAboutAttr? bagIDAttr? propAttr* '/>'
                      | '<rdf:Description' idAboutAttr? bagIDAttr? propAttr* '>'
                            propertyElt* '</rdf:Description>'
[27] bagIDAttr      ::= 'bagID="' IDsymbol '"'
```

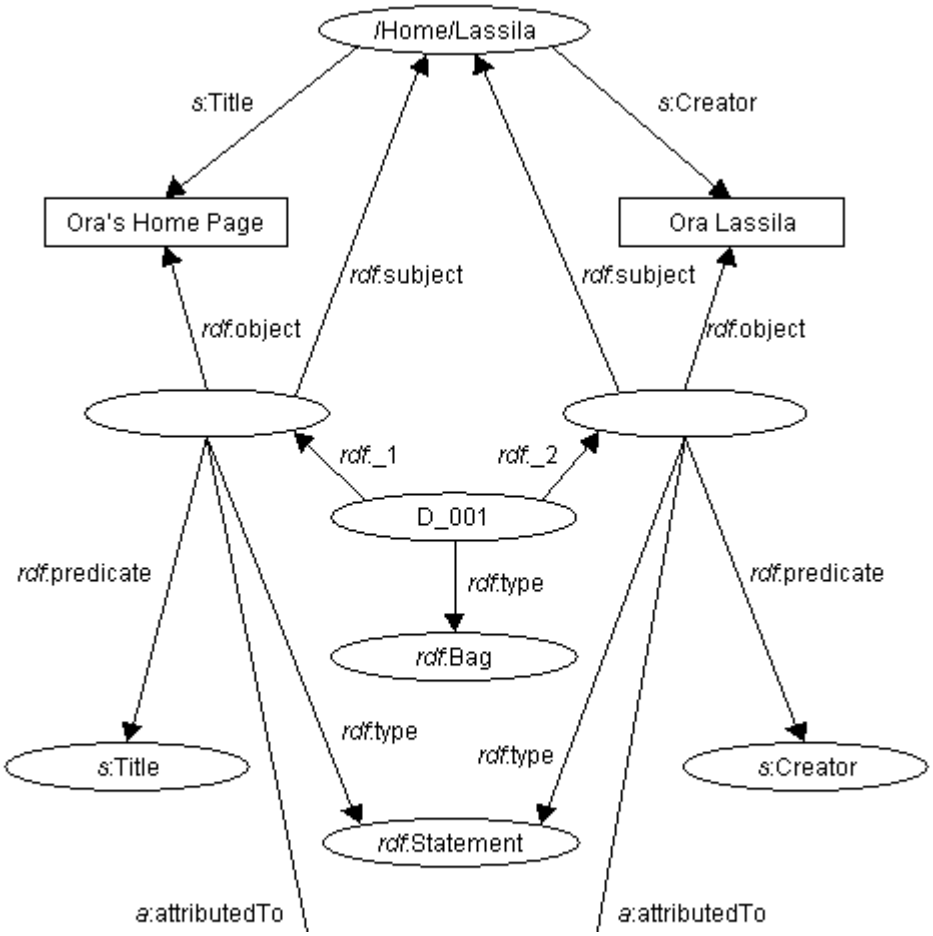
`BagID` and `ID` should not be confused. `ID` specifies the identification of an in-line resource whose properties are further detailed in the Description. `BagID` specifies the identification of the container resource whose members are the reified statements about another resource. A `Description` may have both an `ID` attribute and a `bagID` attribute.

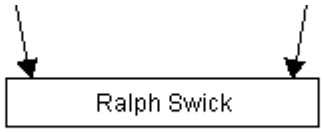
4.2. Syntactic Shorthand for Statements About Statements

Since attaching a `bagID` to a `Description` results in including in the model a Bag of the reified statements of the `Description`, we can use this as a syntactic shorthand when making statements about statements. For example, if we wanted to say that Ralph states that Ora is the creator of http://www.w3.org/Home/Lassila and that he also states that the title of that resource is "Ora's Home Page", we can simply add to the example above

```
<rdf:Description aboutEach="#D_001">
  <a:attributedTo>Ralph Swick</a:attributedTo>
</rdf:Description>
```

Note that this shorthand example includes additional facts in the model not represented by the example in Figure 8. This shorthand usage expresses facts about Ralph's statements and also facts about Ora's home page.





D

Figure 10: Representing statements about statements

The reader is referred to Section 5 ("Formal Model") of this specification for a more formal treatment of higher-order statements and reification.

5. Formal Model for RDF

This specification shows three representations of the data model; as 3-tuples (triples), as a graph, and in XML. These representations have equivalent meaning. The mapping between the representations used in this specification is not intended to constrain in any way the internal representation used by implementations.

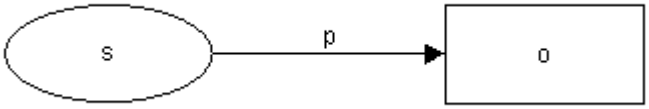
The RDF data model is defined formally as follows:

1. There is a set called *Resources*.
2. There is a set called *Literals*.
3. There is a subset of *Resources* called *Properties*.
4. There is a set called *Statements*, each element of which is a triple of the form

   {pred, sub, obj}

   Where pred is a property (member of *Properties*), sub is a resource (member of *Resources*), and obj is either a resource or a literal (member of *Literals*).

We can view a set of statements (members of *Statements*) as a directed labeled graph: each resource and literal is a vertex; a triple {p, s, o} is an arc from s to o, labeled by p. This is illustrated in figure 11.



D

Figure 11: Simple statement graph template

This can be read either

o is the value of p for s

or (left to right)

s has a property p with a value o

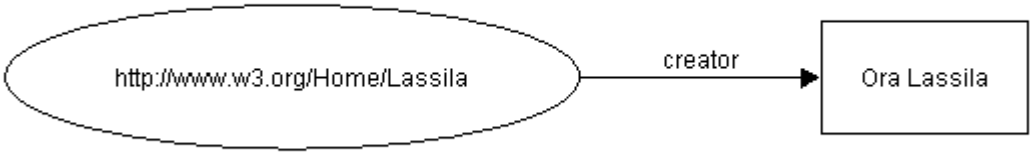
or even

the p of s is o

For example, the sentence

Ora Lassila is the creator of the resource http://www.w3.org/Home/Lassila

would be represented graphically as follows:



D

Figure 12: Simple statement graph

and the corresponding triple (member of *Statements*) would be

{creator, [http://www.w3.org/Home/Lassila], "Ora Lassila"}

The notation [*I*] denotes the resource identified by the URI *I* and quotation marks denote a literal.

Using the triples, we can explain how statements are reified (as introduced in Section 4). Given a statement

{creator, [http://www.w3.org/Home/Lassila], "Ora Lassila"}

we can express the reification of this as a new resource X as follows:

{type, [X], [RDF:Statement]}
{predicate, [X], [creator]}
{subject, [X], [http://www.w3.org/Home/Lassila]}
{object, [X], "Ora Lassila"}

From the standpoint of an RDF processor, facts (that is, statements) are triples that are members of *Statements*. Therefore, the original statement remains a fact despite it being reified since the triple

representing the original statement remains in *Statements*. We have merely added four more triples.

The property named "type" is defined to provide primitive typing. The formal definition of type is:

5. There is an element of *Properties* known as RDF:type.
6. Members of *Statements* of the form {RDF:type, sub, obj} must satisfy the following: sub and obj are members of *Resources.* [RDFSchema] places additional restrictions on the use of type.

Furthermore, the formal specification of reification is:

7. There is an element of *Resources*, not contained in *Properties*, known as RDF:Statement.
8. There are three elements in *Properties* known as RDF:predicate, RDF:subject and RDF:object.
9. Reification of a triple {pred, sub, obj} of *Statements* is an element r of *Resources* representing the reified triple and the elements s_1, s_2, s_3, and s_4 of *Statements* such that

   s_1: {RDF:predicate, r, pred}

   s_2: {RDF:subject, r, subj}

   s_3: {RDF:object, r, obj}

   s_4: {RDF:type, r, [RDF:Statement]}

The resource r in the definition above is called the *reified statement*. When a resource represents a reified statement; that is, it has an RDF:type property with a value of RDF:Statement, then that resource must have exactly one RDF:subject property, one RDF:object property, and one RDF:predicate property.

As described in Section 3, it is frequently necessary to represent a collection of resources or literals; for example to state that a property has an ordered sequence of values. RDF defines three kinds of collections: ordered lists, called *Sequences*, unordered lists, called *Bags*, and lists that represent alternatives for the (single) value of a property, called *Alternatives*.

Formally, these three collection types are defined by:

10. There are three elements of *Resources*, not contained in *Properties*, known as RDF:Seq, RDF:Bag, and RDF:Alt.
11. There is a subset of *Properties* corresponding to the ordinals (1, 2, 3, ...) called *Ord*. We refer to elements of *Ord* as RDF:_1, RDF:_2, RDF:_3, ...

To represent a collection *c*, create a triple {RDF:type, *c*, *t*} where *t* is one of the three collection types RDF:Seq, RDF:Bag, or RDF:Alt. The remaining triples {RDF:_1, *c*, r_1}, ..., {RDF:_n, *c*, r_n}, ... point

to each of the members r_n of the collection. For a single collection resource there may be at most one triple whose predicate is any given element of *Ord* and the elements of *Ord* must be used in sequence starting with RDF:_1. For resources that are instances of the RDF:Alt collection type, there must be exactly one triple whose predicate is RDF:_1 and that is the default value for the Alternatives resource (that is, there must always be at least one alternative).

6. Formal Grammar for RDF

The complete BNF for RDF is reproduced here from previous sections. The precise interpretation of the grammar in terms of the formal model is also given. Syntactic features inherited from XML are not reproduced here. These include all well-formedness constraints, the use of whitespace around attributes and the '=', as well as the use of either double or single quotes around attribute values. This section is intended for implementors who are building tools that read and interpret RDF/XML syntax.

Where used below, the keywords "SHOULD", "MUST", and "MUST NOT" are to be interpreted as described in RFC 2119 [RFC2119]. However, for readability, these words do not appear in all uppercase letters in this specification.

```
 [6.1] RDF            ::= ['<rdf:RDF>'] obj* ['</rdf:RDF>']
 [6.2] obj            ::= description | container
 [6.3] description    ::= '<rdf:Description' idAboutAttr? bagIdAttr? propAttr* '/>
                        | '<rdf:Description' idAboutAttr? bagIdAttr? propAttr* '>'
                              propertyElt* '</rdf:Description>'
                        | typedNode
 [6.4] container      ::= sequence | bag | alternative
 [6.5] idAboutAttr    ::= idAttr | aboutAttr | aboutEachAttr
 [6.6] idAttr         ::= ' ID="' IDsymbol '"'
 [6.7] aboutAttr      ::= ' about="' URI-reference '"'
 [6.8] aboutEachAttr  ::= ' aboutEach="' URI-reference '"'
                        | ' aboutEachPrefix="' string '"'
 [6.9] bagIdAttr      ::= ' bagID="' IDsymbol '"'
[6.10] propAttr       ::= typeAttr
                        | propName '="' string '"' (with embedded quotes escaped)
[6.11] typeAttr       ::= ' type="' URI-reference '"'
[6.12] propertyElt    ::= '<' propName idAttr? '>' value '</' propName '>'
                        | '<' propName idAttr? parseLiteral '>'
                              literal '</' propName '>'
                        | '<' propName idAttr? parseResource '>'
                              propertyElt* '</' propName '>'
                        | '<' propName idRefAttr? bagIdAttr? propAttr* '/>'
[6.13] typedNode      ::= '<' typeName idAboutAttr? bagIdAttr? propAttr* '/>'
                        | '<' typeName idAboutAttr? bagIdAttr? propAttr* '>'
                              propertyElt* '</' typeName '>'
[6.14] propName       ::= Qname
[6.15] typeName       ::= Qname
[6.16] idRefAttr      ::= idAttr | resourceAttr
[6.17] value          ::= obj | string
[6.18] resourceAttr   ::= ' resource="' URI-reference '"'
[6.19] Qname          ::= [ NSprefix ':' ] name
[6.20] URI-reference  ::= string, interpreted per [URI]
[6.21] IDsymbol       ::= (any legal XML name symbol)
[6.22] name           ::= (any legal XML name symbol)
[6.23] NSprefix       ::= (any legal XML namespace prefix)
[6.24] string         ::= (any XML text, with "<", ">", and "&" escaped)
[6.25] sequence       ::= '<rdf:Seq' idAttr? '>' member* '</rdf:Seq>'
```

```
                        | '<rdf:Seq' idAttr? memberAttr* '/>'
[6.26] bag            ::= '<rdf:Bag' idAttr? '>' member* '</rdf:Bag>'
                        | '<rdf:Bag' idAttr? memberAttr* '/>'
[6.27] alternative    ::= '<rdf:Alt' idAttr? '>' member+ '</rdf:Alt>'
                        | '<rdf:Alt' idAttr? memberAttr? '/>'
[6.28] member         ::= referencedItem | inlineItem
[6.29] referencedItem ::= '<rdf:li' resourceAttr '/>'
[6.30] inlineItem     ::= '<rdf:li' '>' value </rdf:li>'
                        | '<rdf:li' parseLiteral '>' literal </rdf:li>'
                        | '<rdf:li' parseResource '>' propertyElt* </rdf:li>'
[6.31] memberAttr     ::= ' rdf:_n="' string '"' (where n is an integer)
[6.32] parseLiteral   ::= ' parseType="Literal"'
[6.33] parseResource  ::= ' parseType="Resource"'
[6.34] literal        ::= (any well-formed XML)
```

The formal namespace name for the properties and classes defined in this specification is `http://www.w3.org/1999/02/22-rdf-syntax-ns#`. When an RDF processor encounters an XML element or attribute name that is declared to be from a namespace whose name begins with the string "http://www.w3.org/TR/REC-rdf-syntax" and the processor does not recognize the semantics of that name then the processor is required to skip (i.e., generate no tuples for) the entire XML element, including its content, whose name is unrecognized or that has an attribute whose name is unrecognized.

Each propertyElt *E* contained by a `Description` element results in the creation of a triple {p,r,v} where:

1. p is the expansion of the namespace-qualified tag name (Generic Identifier) of *E*. This expansion is generated by concatenating the namespace name given in the namespace declaration with the LocalPart of the qualified name.
2. r is
   - the resource whose identifier is given by the value of the `about` attribute of the `Description` or
   - a new resource whose identifier is the value of the `ID` attribute of the `Description`, if present; else the new resource has no identifier.
3. If *E* is an empty element (no content), v is the resource whose identifier is given by the `resource` attribute of *E*. If the content of *E* contains no XML markup or if `parseType="Literal"` is specified in the start tag of *E* then v is the content of *E* (a literal). Otherwise, the content of *E* must be another `Description` or container and v is the resource named by the (possibly implicit) `ID` or `about` of that `Description` or container.

The `parseType` attribute changes the interpretation of the element content. The `parseType` attribute should have one of the values 'Literal' or 'Resource'. The value is case-sensitive. The value 'Literal' specifies that the element content is to be treated as an RDF/XML literal; that is, the content must not be interpreted by an RDF processor. The value 'Resource' specifies that the element content must be treated as if it were the content of a `Description` element. Other values of `parseType` are reserved for future specification by RDF. With RDF 1.0 other values must be treated as identical to 'Literal'. In all cases, the content of an element having a `parseType` attribute must be well-formed XML. The content of an element having a `parseType="Resource"` attribute must further match the production for the content of a `Description` element.

> The RDF Model and Syntax Working Group acknowledges that the parseType='Literal' mechanism is a minimum-level solution to the requirement to express an RDF statement

with a value that has XML markup. Additional complexities of XML such as canonicalization of whitespace are not yet well defined. Future work of the W3C is expected to resolve such issues in a uniform manner for all applications based on XML. Future versions of RDF will inherit this work and may extend it as we gain insight from further application experience.

URI-References are resolved to resource identifiers by first resolving the URI-reference to absolute form as specified by [URI] using the base URI of the document in which the RDF statements appear. If a fragment identifier is included in the URI-reference then the resource identifier refers only to a subcomponent of the containing resource; this subcomponent is identifed by the corresponding anchor id internal to that containing resource and the extent of the subcomponent is defined by the fragment identifier in conjunction with the content type of the containing resource, otherwise the resource identifier refers to the entire item specified by the URI.

> Note: Although non-ASCII characters in URIs are not allowed by [URI], [XML] specifies a convention to avoid unnecessary incompatibilities in extended URI syntax. Implementors of RDF are encouraged to avoid further incompatibility and use the XML convention for system identifiers. Namely, that a non-ASCII character in a URI be represented in UTF-8 as one or more bytes, and then these bytes be escaped with the URI escaping mechanism (i.e., by converting each byte to %HH, where HH is the hexadecimal notation of the byte value).

The `Description` element itself represents an instance of a Bag resource. The members of this Bag are the resources corresponding to the reification of each of the statements in the `Description`. If the `bagID` attribute is specified its value is the identifier of this Bag, else the Bag is anonymous.

When `about` is specified with `Description`, the statements in the `Description` refer to the resource named in the `about`. A `Description` element without an `about` attribute represents an in-line resource. This in-line resource has a resource identifier formed using the value of the base URI of the document containing the RDF statements plus an anchor id equal to the value of the `ID` attribute of the `Description` element, if present. When another `Description` or property value refers to the in-line resource it will use the value of the `ID` in an `about` attribute. When the other `Description` refers to the Bag of resources corresponding to the reified statements it will use the value of `bagID` in an `about` attribute. Either `ID` or `about` may be specified on `Description` but not both together in the same element. The values for each `ID` and `bagID` attribute must not appear in more than one such attribute within a document nor may the same value be used in an `ID` and a `bagID` within a single document.

When `aboutEach` is specified with `Description`, the statements in the `Description` refer to each of the members of the container named by `aboutEach`. The triples {p,r,v} represented by each contained propertyElt *E* as described above are duplicated for each r that is a member of the container.

When `aboutEachPrefix` is specified with `Description`, the statements in the `Description` refer to each of the members of an anonymous Bag container. The members of this Bag container are all the resources whose absolute form resource identifiers begin with the character string given as the value of `aboutEachPrefix`. The absolute form resource identifier is produced by resolving the URI according to the algorithm in Section 5.2., Resolving Relative References to Absolute Form, in [URI]. The triples {p,r,v} represented by each contained propertyElt *E* as described above are duplicated for each r that is a member of the container.

`Seq`, `Bag`, and `Alt` each represent an instance of a Sequence, Bag, or Alternative container resource type respectively. A triple {RDF:type,c,t} is created where c is the collection resource and t is one of RDF:Seq, RDF:Bag, or RDF:Alt. The members of the collection are denoted by `li`. Each `li` element *E* corresponds to one member of the collection and results in the creation of a triple {p,c,v} where:

1. p is assigned consecutively according to the (XML) order of lexical appearance of each member starting with "RDF:_1" for each container.
2. c is the collection resource. The `ID` attribute, if specified, provides the URI fragment identifier for c.
3. (same as rule 3 above) If *E* is an empty element (no content), v is the resource whose resource identifier is given by the `resource` attribute of *E*. If the content of *E* contains no XML markup or if `parseType="Literal"` is specified in the start tag of *E* then v is the content of *E* (a literal). Otherwise, the content of *E* must be another `Description` or container and v is the resource named by the(possibly implicit) `ID` or `about` of that `Description` or container.

The URI identifies (after resolution) the target resource; i.e., the resource to which the `Description` applies or the resource that is included in the container. The `bagID` attribute on a `Description` element and the `ID` attribute on a container element permit that `Description` or container to be referred to by other `Descriptions`. The `ID` on a container element is the name that is used in a `resource` attribute on a property element to make the collection the value of that property.

Within propertyElt (production [6.12]), the URI used in a `resource` attribute identifies (after resolution) the resource that is the object of the statement (i.e., the value of this property). The value of the `ID` attribute, if specified, is the identifier for the resource that represents the reification of the statement. If an RDF expression (that is, content with RDF/XML markup) is specified as a property value the object is the resource given by the `about` attribute of the contained `Description` or the (possibly implied) `ID` of the contained `Description` or container resource. `Strings` must be well-formed XML; the usual XML content quoting and escaping mechanisms may be used if the string contains character sequences (e.g. "<" and "&") that violate the well-formedness rules or that otherwise might look like markup. The attribute `parseType="Literal"` specifies that the element content is an RDF literal. Any markup that is part of this content is included as part of the literal and not interpreted by RDF.

It is recommended that property names always be qualified with a namespace prefix to unambiguously connect the property definition with the corresponding schema.

As defined by XML, the character repertoire of an RDF string is ISO/IEC 10646 [ISO10646]. An actual RDF string, whether in an XML document or in some other representation of the RDF data model, may be stored using a direct encoding of ISO/IEC 10646 or an encoding that can be mapped to ISO/IEC 10646. Language tagging is part of the string value; it is applied to sequences of characters within an RDF string and does not have an explicit manifestation in the data model.

Two RDF strings are deemed to be the same if their ISO/IEC 10646 representations match. Each RDF application must specify which one of the following definitions of ’match’ it uses:

a. the two representations are identical, or
b. the two representations are canonically equivalent as defined by The Unicode Standard [Unicode].

> Note: The W3C I18N WG is working on a definition for string identity matching. This definition will most probably be based on canonical equivalences according to the Unicode standard and on the principle of early uniform normalization. Users of RDF should not rely on any applications matching using the canonical equivalents, but should try to make sure that their data is in the normalized form according to the upcoming definitions.

This specification does not state a mechanism for determining equivalence between literals that contain markup, nor whether such a mechanism is guaranteed to exist.

The `xml:lang` attribute may be used as defined by [XML] to associate a language with the property value. There is no specific data model representation for `xml:lang` (i.e., it adds no triples to the data model); the language of a literal is considered by RDF to be a part of the literal. An application may ignore language tagging of a string. All RDF applications must specify whether or not language tagging in literals is significant; that is, whether or not language is considered when performing string matching or other processing.

Attributes whose names start with "`xmlns`" are namespace declarations and do not represent triples in the data model. There is no specific data model representation for such namespace declarations.

Each property and value expressed in XML attribute form by productions [6.3] and [6.10] is equivalent to the same property and value expressed as XML content of the corresponding `Description` according to production [6.12]. Specifically; each XML attribute *A* specified with a `Description` start tag other than the attributes `ID`, `about`, `aboutEach`, `aboutEachPrefix`, `bagID`, `xml:lang`, or any attribute starting with the characters `xmlns` results in the creation of a triple {p,r,v} where:

1. p is the expansion of the namespace-qualified attribute name of *A*. This expansion is generated by concatenating the namespace name given in the namespace declaration with the LocalPart of the qualified name and then resolving this URI according to the algorithm in Section 5.2., Resolving Relative References to Absolute Form, in [URI].
2. r is the resource whose resource identifer is given by the value of the `about` attribute, resolved as specified above, or whose anchor id is given by the value of the `ID` attribute of the Description or is a member of the collection specified by the `aboutEach` or `aboutEachPrefix` attribute.
3. v is the attribute value of *A* (a literal).

Grammatically, production [6.11] is just a special case of the propName production [6.10]. The value of the `type` attribute is interpreted as a URI-reference and expanded in the same way as the value of the `resource` attribute. Use of [6.11] is equivalent to using *`rdf`*`:type` as an element (property) name together with a `resource` attribute.

The typedNode form (production [6.13]) may be used to represent instances of resources of specific types and to further describe those resources. A `Description` expressed in typedNode form by production [6.13] is equivalent to the same `Description` expressed by production [6.3] with the same `ID`, `bagID`, and `about` attributes plus an additional type property in the `Description` where the value of the type property is the resource whose identifier is given by the fully expanded and resolved URI corresponding to the typeName of the typedNode. Specifically, a typedNode represents a triple

{RDF:type,n,t} where n is the resource whose identifier is given by the value of the `about` attribute (after resolution) or whose anchor id is given by the value of the `ID` attribute of the typedNode element, and t is the expansion of the namespace-qualified tag name. The remainder of the typedNode attributes and content is handled as for `Description` elements above.

Properties and values expressed in XML attribute form within an empty XML element *E* by productions [6.10] and [6.12] are equivalent to the same properties and values expressed as XML content of a single `Description` element *D* which would become the content of *E*. The referent of *D* is the value of the property identified by the XML element name of *E* according to productions [6.17], [6.2], and [6.3]. Specifically; each propertyElt start tag containing attribute specifications other than `ID`, `resource`, `bagID`, `xml:lang`, or any attribute starting with the characters `xmlns` results in the creation of the triples $\{p,r_1,r_2\}$, $\{p_{a1},r_2,v_{a1}\}$, ..., $\{p_{an},r_2,v_{an}\}$ where

1. p is the expansion of the namespace-qualified tag name.
2. r_1 is the resource being referred to by the element containing this propertyElt expression.
3. r_2 is the resource named by the `resource` attribute if present or a new resource. If the `ID` attribute is given it is the identifier of this new resource.
4. p_{a1} ... p_{an} are the expansion of the namespace-qualified attribute names.
5. v_{a1} ... v_{an} are the corresponding attribute values.

The value of the `bagID` attribute, if specified, is the identifier for the Bag corresponding to the `Description` *D*; else the Bag is anonymous.

7. Examples

The following examples further illustrate features of RDF explained above.

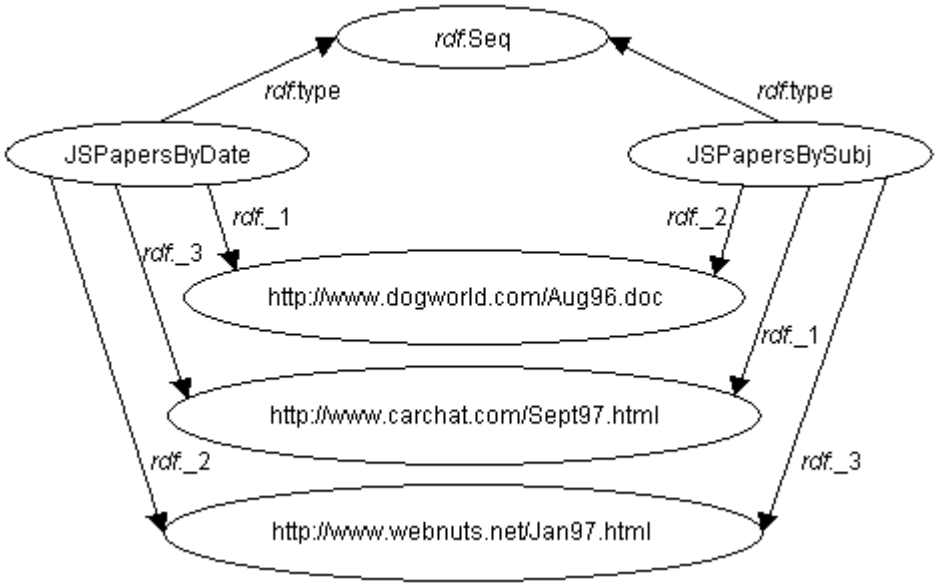
7.1. Sharing Values

A single resource can be the value of more than one property; that is, it can be the object of more than one statement and therefore pointed to by more than one arc. For example, a single Web page might be shared between several documents and might then be referenced more than once in a "sitemap". Or two different (ordered) sequences of the same resources may be given.

Consider the case of specifying the collected works of an author, sorted once by publication date and sorted again alphabetically by subject:

```
<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <Seq ID="JSPapersByDate">
    <li resource="http://www.dogworld.com/Aug96.doc"/>
    <li resource="http://www.webnuts.net/Jan97.html"/>
    <li resource="http://www.carchat.com/Sept97.html"/>
  </Seq>
  <Seq ID="JSPapersBySubj">
    <li resource="http://www.carchat.com/Sept97.html"/>
    <li resource="http://www.dogworld.com/Aug96.doc"/>
    <li resource="http://www.webnuts.net/Jan97.html"/>
  </Seq>
</RDF>
```

This XML example also uses the default namespace declaration syntax to elide the namespace prefix.



D

Figure 13: Sharing values between two sequences

7.2. Aggregates

To further illustrate aggregates, consider an example of a document with two authors specified alphabetically, a title specified in two different languages, and having two equivalent locations on the Web:

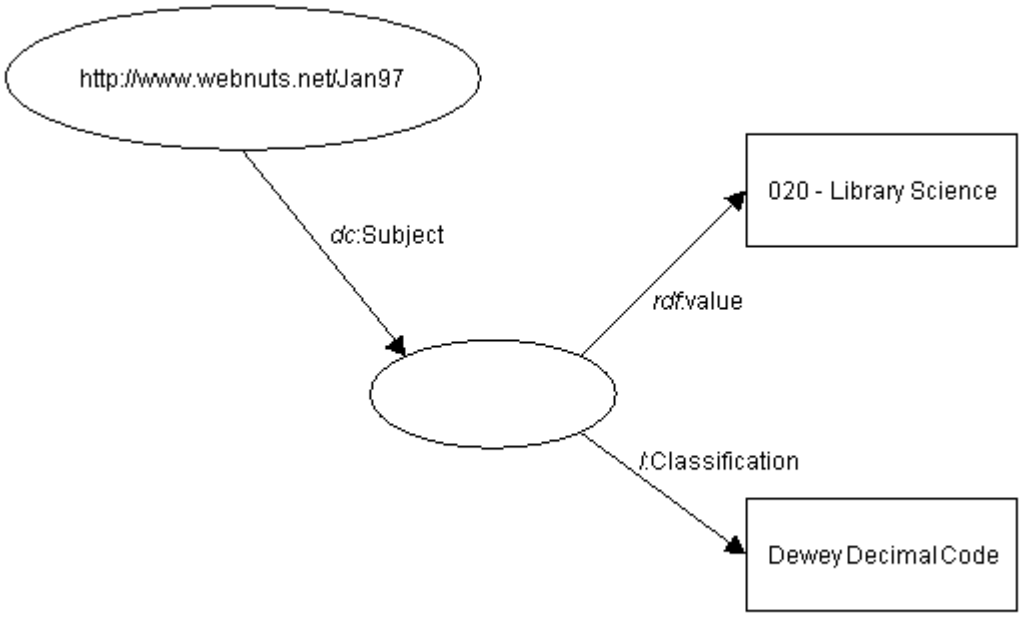
```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/metadata/dublin_core#">
  <rdf:Description about="http://www.foo.com/cool.html">
    <dc:Creator>
      <rdf:Seq ID="CreatorsAlphabeticalBySurname">
        <rdf:li>Mary Andrew</rdf:li>
        <rdf:li>Jacky Crystal</rdf:li>
      </rdf:Seq>
    </dc:Creator>

    <dc:Identifier>
      <rdf:Bag ID="MirroredSites">
        <rdf:li rdf:resource="http://www.foo.com.au/cool.html"/>
        <rdf:li rdf:resource="http://www.foo.com.it/cool.html"/>
      </rdf:Bag>
    </dc:Identifier>

    <dc:Title>
      <rdf:Alt>
        <rdf:li xml:lang="en">The Coolest Web Page</rdf:li>
        <rdf:li xml:lang="it">Il Pagio di Web Fuba</rdf:li>
      </rdf:Alt>
    </dc:Title>
  </rdf:Description>
</rdf:RDF>
```

This example illustrates the use of all three types of collection. The order of the creators is deemed significant so the *Sequence* container is used to hold them. The locations on the Web are equivalent; order is not significant, therefore a *Bag* is used. The document has only a single title and that title has two variants, so the *Alternatives* container is used.

> Note: In many cases, it is impossible to have a preferred language among various language alternatives; all languages are considered to be strictly equivalent. In these cases, the description author should use a `Bag` instead of an `Alt` container.

7.3. Non-Binary Relations

The RDF data model intrinsically only supports binary relations; that is, a statement specifies a relation between two resources. In the following examples we show the recommended way to represent higher arity relations in RDF using just binary relations. The recommended technique is to use an intermediate resource with additional properties of this resource giving the remaining relations. As an example, consider the subject of one of John Smith's recent articles -- library science. We could use the Dewey Decimal Code for library science to categorize that article. Dewey Decimal codes are far from the only subject categorization scheme, so to hold the classification system relation we identify an additional resource that is used as the value of the subject property and annotate this resource with an additional property that identifies the categorization scheme that was used. As specified in Section 2.3., the RDF core includes a *value* property to denote the principal value of the main relation. The resulting graph might look like:



D

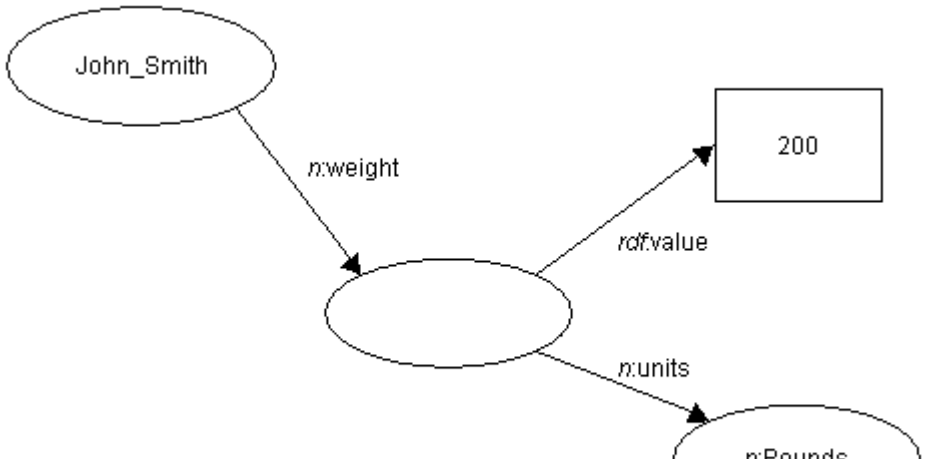
Figure 14: A ternary relation

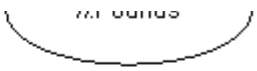
which could be exchanged as:

```
<RDF
  xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/metadata/dublin_core#"
  xmlns:l="http://mycorp.com/schemas/my-schema#">
  <Description about="http://www.webnuts.net/Jan97.html">
    <dc:Subject
      rdf:value="020 - Library Science"
      l:Classification="Dewey Decimal Code"/>
  </Description>
</RDF>
```

Note: In the example above two namespace declarations exist for the same namespace. This is frequently needed when default namespaces are declared so that attributes that do not come from the namespace of the element may be specified, as is the case with the rdf:value attribute in the dc:Subject element above.

A common use of this higher-arity capability is when dealing with units of measure. A person's weight is not just a number such as "200", it also includes the unit of measure used. In this case we might be using either pounds or kilograms. We could use a relationship with an additional arc to record the fact that John Smith is a rather strapping gentleman:





D

Figure 15: Unit of measure as a ternary relation

which can be exchanged as:

```
<RDF
  xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:n="http://www.nist.gov/units/">
  <Description about="John_Smith">
    <n:weight rdf:parseType="Resource">
      <rdf:value>200</rdf:value>
      <n:units rdf:resource="http://www.nist.gov/units/Pounds"/>
    </n:weight>
  </Description>
</RDF>
```

provided the resource "Pounds" is defined in a NIST schema with the URI http://www.nist.gov/units/Pounds.

7.4. Dublin Core Metadata

The Dublin Core metadata is designed to facilitate discovery of electronic resources in a manner similar to a library card catalog. These examples represent the simple description of a set of resources in RDF using vocabularies defined by the Dublin Core Initiative. *Note: the specific Dublin Core RDF vocabulary shown here is not intended to be authoritative. The Dublin Core Initiative is the authoritative reference.*

Here is a description of a Web site home page using Dublin Core properties:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/metadata/dublin_core#">
  <rdf:Description 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>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>
```

The second example is of a published magazine.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/metadata/dublin_core#"
  xmlns:dcq="http://purl.org/metadata/dublin_core_qualifiers#">
  <rdf:Description 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 rdf:parseType="Resource">
      <dcq:AgentType
        rdf:resource="http://purl.org/metadata/dublin_core_qualifiers#Editor"/>
      <rdf:value>Amy Friedlander</rdf:value>
    </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>urn:issn:1082-9873</dc:Identifier>
    <dc:Relation rdf:parseType="Resource">
      <dcq:RelationType
        rdf:resource="http://purl.org/metadata/dublin_core_qualifiers#IsPartOf"/>
      <rdf:value resource="http://www.dlib.org"/>
    </dc:Relation>
  </rdf:Description>
</rdf:RDF>
```

The third example is of a specific article in the magazine referred to in the previous example.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/metadata/dublin_core#"
  xmlns:dcq="http://purl.org/metadata/dublin_core_qualifiers#">
  <rdf:Description 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>
    <dc:Relation rdf:parseType="Resource">
      <dcq:RelationType
        rdf:resource="http://purl.org/metadata/dublin_core_qualifiers#IsPartOf"/>
      <rdf:value resource="http://www.dlib.org/dlib/may98/05contents.html"/>
    </dc:Relation>
  </rdf:Description>
</rdf:RDF>
```

Note: Schema developers may be tempted to declare the values of certain properties to use a syntax corresponding to the XML Namespace qualified name abbreviation. We advise against using these qualified names inside property values as this may cause incompatibilities with future XML datatyping mechanisms. Furthermore, those fully versed in XML 1.0 features may recognize that a similar abbreviation mechanism exists in user-defined entities. We also advise against relying on the use of entities as there is a proposal to define a future subset of XML that does not include user-defined entities.

7.5. Values Containing Markup

When a property value is a literal that contains XML markup, the following syntax is used to signal to the RDF interpreter not to interpret the markup but rather to retain it as part of the value. The precise representation of the resulting value is not specified here.

In the following example, the value of the Title property is a literal containing some MATHML markup.

```
<rdf:Description
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/metadata/dublin_core#"
  xmlns="http://www.w3.org/TR/REC-mathml"
  rdf:about="http://mycorp.com/papers/NobelPaper1">

  <dc:Title rdf:parseType="Literal">
    Ramifications of
       <apply>
      <power/>
      <apply>
        <plus/>
        <ci>a</ci>
        <ci>b</ci>
      </apply>
      <cn>2</cn>
```

```
      </apply>
      to World Peace
    </dc:Title>
    <dc:Creator>David Hume</dc:Creator>
  </rdf:Description>
```

7.6. PICS Labels

The Platform for Internet Content Selection (PICS) is a W3C Recommendation for exchanging descriptions of the content of Web pages and other material. PICS is a predecessor to RDF and it is an explicit requirement of RDF that it be able to express anything that can be expressed in a PICS label.

Here is an example of how a PICS label might be expressed in RDF form. *Note that work to re-specify PICS itself as an application of RDF may follow the completion of the RDF specification, thus the following example should not be considered an authoritative example of a future PICS schema.* This example comes directly from [PICS]. Note that a PICS Rating Service Description is exactly analogous to an RDF Schema; the categories described in such a Ratings Service description file are equivalent to properties in the RDF model.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:pics="http://www.w3.org/TR/xxxx/WD-PICS-labels#"
  xmlns:gcf="http://www.gcf.org/v2.5">
  <rdf:Description about="http://www.w3.org/PICS/Overview.html" bagID="L01"
    gcf:suds="0.5"
    gcf:density="0"
    gcf:color.hue="1"/>

  <rdf:Description about="http://www.w3.org/PICS/Underview.html" bagID="L02"
    gcf:subject="2"
    gcf:density="1"
    gcf:color.hue="1"/>

  <rdf:Description aboutEach="#L01"
    pics:by="John Doe"
    pics:on="1994.11.05T08:15-0500"
    pics:until="1995.12.31T23:59-0000"/>

  <rdf:Description aboutEach="#L02"
    pics:by="Jane Doe"
    pics:on="1994.11.05T08:15-0500"
    pics:until="1995.12.31T23:59-0000"/>
</rdf:RDF>
```

Note that `aboutEach` is used to indicate that the PICS label options refer to the individual (rating) statements and not to the container in which those statements happen to be supplied.

[PICS] also defines a type called a *generic label*. A PICS generic label is a label that applies to every page within a specified portion of the Web site.

Below is an example of how a PICS generic label would be written in RDF, using the `aboutEachPrefix` collection constructor. This example is drawn from the "Generic request" example in Appendix B of [PICS]:

```
<rdf:RDF
```

```
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:pics="http://www.w3.org/TR/xxxx/WD-PICS-labels#"
  xmlns:ages="http://www.ages.org/our-service/v1.0/">
  <rdf:Description aboutEachPrefix="http://www.w3.org/WWW/" bagID="L03"
    ages:age="11"/>

  <rdf:Description aboutEach="#L03"
    pics:by="abaird@w3.org"/>
</rdf:RDF>
```

The property age with the value "11" appears on every resource whose URI starts with the string "http://www.w3.org/WWW/". The reified statement corresponding to each such statement ("The age of [*I*] is 11") has a property stating that "abaird@w3.org" was responsible for creating those statements.

7.7. Content Hiding For RDF inside HTML

RDF, being well-formed XML, is suitable for direct inclusion in an HTML document when the user agent follows the HTML recommendations for error handling in invalid documents. When a fragment of RDF is incorporated into an HTML document some browsers will render any exposed string content. Exposed string content is anything that appears between the ">" that ends one tag and the "<" that begins the next tag. Generally, multiple consecutive whitespace characters including end-of-line characters are rendered as a single space.

The RDF abbreviated syntax can frequently be used to write property values that are strings in XML attribute form and leave only whitespace as exposed content. For example, the first part of the Dublin Core example from Section 7.4. could be written as:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/metadata/dublin_core#">
  <rdf:Description about="http://www.dlib.org"
    dc:Title="D-Lib Program - Research in Digital Libraries"
    dc:Description="The D-Lib program supports the community of people
     with research interests in digital libraries and electronic
     publishing."
    dc:Publisher="Corporation For National Research Initiatives"
    dc:Date="1995-01-07"/>
</rdf:RDF>
```

Rewriting to avoid exposed content will work for most common cases. One common but less obvious case is container descriptions. Consider the first part of the example in Section 7.2.:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/metadata/dublin_core#">
  <rdf:Description about="http://www.foo.com/cool.html">
    <dc:Creator>
      <rdf:Seq ID="CreatorsAlphabeticalBySurname">
        <rdf:li>Mary Andrew</rdf:li>
        <rdf:li>Jacky Crystal</rdf:li>
      </rdf:Seq>
    </dc:Creator>
  </rdf:Description>
</rdf:RDF>
```

To rewrite this with no exposed content, we use the following form:

To rewrite this with no exposed content, we use the following form:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/metadata/dublin_core#">
  <rdf:Description about="http://www.foo.com/cool.html">
    <dc:Creator>
      <rdf:Seq ID="CreatorsAlphabeticalBySurname"
        rdf:_1="Mary Andrew"
        rdf:_2="Jacky Crystal"/>
    </dc:Creator>
  </rdf:Description>
</rdf:RDF>
```

Note here that the `li` element cannot be used as an attribute due to the XML rule forbidding multiple occurrences of the same attribute name within a tag. Therefore we use the explicit RDF *Ord* properties; in effect manually expanding the `li` element.

A complete HTML document containing RDF metadata describing itself is:

```
<html>
<head>
  <rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/metadata/dublin_core#">
    <rdf:Description about="">
      <dc:Creator>
        <rdf:Seq ID="CreatorsAlphabeticalBySurname"
          rdf:_1="Mary Andrew"
          rdf:_2="Jacky Crystal"/>
      </dc:Creator>
    </rdf:Description>
  </rdf:RDF>
</head>
<body>
<P>This is a fine document.</P>
</body>
</html>
```

The HTML document above should be accepted by all browsers compliant with HTML 3.2 and later and should only render the characters "This is a fine document."

8. Acknowledgements

This specification is the work of the W3C RDF Model and Syntax Working Group. This Working Group has been most ably chaired by Eric Miller of the Online Computer Library Center and Bob Schloss of IBM. We thank Eric and Bob for their tireless efforts in keeping the group on track and we especially thank OCLC, IBM, and Nokia for supporting them and us in this endeavor.

The members of the Working Group who helped design this specfication, debate proposals, provide words, proofread numerous drafts and ultimately reach consensus are: Ron Daniel (DATAFUSION), Renato Iannella (DSTC), Tsuyoshi SAKATA (DVL), Murray Maloney (Grif), Bob Schloss (IBM), Naohiko URAMOTO (IBM), Bill Roberts (KnowledgeCite), Arthur van Hoff (Marimba), Charles Frankston (Microsoft), Andrew Layman (Microsoft), Chris McConnell (Microsoft), Jean Paoli (Microsoft), R.V. Guha (Netscape), Ora Lassila (Nokia), Ralph LeVan (OCLC), Eric Miller (OCLC), Charles Wicksteed (Reuters), Misha Wolf (Reuters), Wei Song (SISU), Lauren Wood (SoftQuad),

Tim Bray (Textuality), Paul Resnick (University of Michigan), Tim Berners-Lee (W3C), Dan Connolly (W3C), Jim Miller (W3C, emeritus), Ralph Swick (W3C). Dan Brickley (UK Bristol) joined the RDF Schema activity and brought us lots of sage advice in the final stages of this work. Martin Dürst (W3C) reviewed several working drafts and made a number of suggestions for improvement on behalf of the W3C Internationalization Working Group. Janne Saarela (W3C) performed a priceless service by creating a 'clean room' implementation from our working drafts.

This document is the collective work of the Working Group. The editors are indebted to the Working Group for helping to create and polish this specification.

Appendix A. Glossary

The following terms are used in this specification with varying degrees of intuitive meaning and precise meaning. The summary definitions here are for guidance only; they are non-normative. Where appropriate, the location in the document of the precise definition is given also.

Arc
: A representation of a property in a graph form; specifically the edges in a directed labeled graph.

Attribute
: A characteristic of an object. In Chapter 6 this term refers to a specific XML syntactic construct; the `name="value"` portions of an XML tag.

Element
: As used here, this term refers to a specific XML syntactic construct; i.e., the material between matching XML start and end tags.

Literal
: The most primitive value type represented in RDF, typically a string of characters. The content of a literal is not interpreted by RDF itself and may contain additional XML markup. Literals are distinguished from Resources in that the RDF model does not permit literals to be the subject of a statement.

Node
: A representation of a resource or a literal in a graph form; specifically, a vertex in a directed labeled graph.

Property
: A specific attribute with defined meaning that may be used to describe other resources. A property plus the value of that property for a specific resource is a *statement* about that resource. A property may define its permitted values as well as the types of resources that may be described with this property.

Resource
: An abstract object that represents either a physical object such as a person or a book or a conceptual object such as a color or the class of things that have colors. Web pages are usually considered to be physical objects, but the distinction between physical and conceptual or abstract objects is not important to RDF. A resource can also be a component of a larger object; for example, a resource can represent a specific person's left hand or a specific paragraph out of a document. As used in this specification, the term resource refers to the whole of an object if the URI does not contain a fragment (anchor) id or to the specific subunit named by the fragment or anchor id.

Statement
: An expression following a specified grammar that names a specific resource, a specific

property (attribute), and gives the value of that property for that resource. More specifically here, an *RDF statement* is a statement using the RDF/XML grammar specified in this document.

Triple
: A representation of a statement used by RDF, consisting of just the property, the resource identifier, and the property value in that order.

Appendix B. Transporting RDF

Descriptions may be associated with the resource they describe in one of four ways:

1. The Description may be contained within the resource ("embedded"; e.g. in HTML).
2. The Description may be external to the resource but supplied by the transfer mechanism in the same retrieval transaction as that which returns the resource ("along-with"; e.g. with HTTP GET or HEAD).
3. The Description may be retrieved independently from the resource, including from a different source ("service bureau"; e.g. using HTTP GET).
4. The Description may contain the resource ("wrapped"; e.g. RDF itself).

All resources will not support all association methods; in particular, many kinds of resources will not support embedding and only certain kinds of resources may be wrapped.

A human- or machine-understandable description of an RDF schema may be accessed through content negotiation by dereferencing the schema URI. If the schema is machine-understandable it may be possible for an application to learn some of the semantics of the properties named in the schema on demand. The logic and syntax of RDF schemas are described in a separate document, [RDFSchema].

The recommended technique for embedding RDF expressions in an HTML document is simply to insert the RDF in-line as shown in Example 7.7. This will make the resulting document non-conformant to HTML specifications up to and including HTML 4.0 but the W3C expects that the HTML specification will evolve to support this. Two practical issues will arise when this technique is employed with respect to browsers conforming to specifications of HTML up to and including HTML 4.0. Alternatives are available to authors in these cases; see [XMLinHTML]. It is up to the author to choose the appropriate alternative in each circumstance.

1. Some HTML 2.0 browsers will assume a </HEAD> tag immediately before the first RDF element that appears within <HEAD>.

   Authors concerned about very old browsers may place all RDF expressions at the end of the document head.

2. All HTML browsers conforming to specifications up to and including HTML 4.0 will render any content appearing in RDF property values expressed as XML elements (i.e., production [6.12]).

   Authors concerned about preventing their RDF content from rendering in old browsers may use the abbreviated syntax (propAttr form) to move the property value into an attribute. Not all properties can be expressed this way.

In the event that none of the alternatives above provides the capabilities desired by the author, the RDF expressions may be left external to the HTML document and linked with an HTML <LINK> element. The recommended relation type for this purpose is REL="meta"; e.g.

```
<LINK rel="meta" href="mydocMetadata.DC.RDF">
```

Appendix C: Notes about Usage

C.1. Property Names

The RDF serialization and abbreviated syntaxes use XML as their encoding. XML elements and attributes are case sensitive, so RDF property names are therefore also case sensitive. This specification does not require any specific format for property names other than that they be legal XML *names*. For its own identifiers, RDF has adopted the convention that all property names use "InterCap style"; that is, the first letter of the property name and the remainder of the word is lowercase; e.g. *subject*. When the property name is a composition of words or fragments of words, the words are concatenated with the first letter of each word (other than the first word) capitalized and no additional punctutation; e.g. *subClassOf*.

C.2. Namespace URIs

RDF uses the proposed XML namespace mechanism to implement globally unique identifiers for all properties. In addition, the namespace name serves as the identifier for the corresponding RDF schema. The namespace name is resolved to absolute form as specified by the algorithm in Section 5.2., Resolving Relative References to Absolute Form, in [URI]. An RDF processor can expect to use the schema URI to access the schema content. This specification places no further requirements on the content that might be supplied at that URI, nor how (if at all) the URI might be modified to obtain alternate forms or a fragment of the schema.

Appendix D: References

[Dexter94]
F. Halasz and M. Schwarz. The Dexter Hypertext Reference Model. Communications of the ACM, 37(2):30--39, February 1994. Edited by K. Grønbæck and R. Trigg. http://www.acm.org/pubs/citations/journals/cacm/1994-37-2/p30-halasz/

[HTML]
HTML 4.0 Specification, Raggett, Le Hors, Jacobs eds, World Wide Web Consortium Recommendation; http://www.w3.org/TR/REC-html40

[ISO10646]
ISO/IEC 10646. The applicable version of this standard is defined in the XML specification [XML].

[NAMESPACES]
Namespaces in XML; Bray, Hollander, Layman eds, World Wide Web Consortium Recommendation; http://www.w3.org/TR/1999/REC-xml-names-19990114.

[PICS]
PICS Label Distribution Label Syntax and Communication Protocols, Version 1.1, W3C Recommendation 31-October-96; http://www.w3.org/TR/REC-PICS-labels.

[RDFSchema]
Resource Description Framework (RDF) Schemas; Brickley, Guha, Layman eds., World Wide Web Consortium Working Draft; http://www.w3.org/TR/1998/WD-rdf-schema
[RFC2119]
Key words for use in RFCs to Indicate Requirement Levels; S. Bradner, March 1997; RFC2119.
[Unicode]
The Unicode Standard. The applicable version of this standard is the version defined by the XML specification [XML].
[URI]
Uniform Resource Identifiers (URI): Generic Syntax; Berners-Lee, Fielding, Masinter, Internet Draft Standard August, 1998; RFC2396.
[XML]
Extensible Markup Language (XML) 1.0; World Wide Web Consortium Recommendation; http://www.w3.org/TR/REC-xml.
[XMLinHTML]
XML in HTML Meeting Report; Connolly, Wood eds.; World Wide Web Consortium Note; http://www.w3.org/TR/NOTE-xh.

Appendix E: Changes

Some typographic changes were made after the Proposed Recommendation was published. The known errata in the previous version as of the time of publication have been corrected. A small clarifying change to the final paragraph of Section 6 was also made.

Ora Lassila <ora.lassila@research.nokia.com>
Ralph R. Swick <swick@w3.org>

Revision History:
17-February-1999: prepare for publication as W3C Recommendation.
5-January-1999: publish as W3C Proposed Recommendation.
16-December-1998: final draft intended as Proposed Recommendation.
30-October-1998: incorporate Last Call review comments, add parseType, improve the I18N wordings.
8-October-1998: final cleanup, move changes to Appendix E, publish as Last Call.
7-October-1998: reserve a bit of schema URI space for futureproofing, add rdf:value.
2-October-1998: major renaming; statements, predicates, subjects, objects.
4-September-1998: instanceOf -> type, revise higher-arity relations model, add node identifier.
19-August-1998: Add '_' to Ord property names.
12-August-1998: Update to newer XML namespace declaration syntax. Add content to Section 7.
20-July-1998: More typos fixed. Third public draft
15-July-1998: Incorporate comments and fix typos. Initial letter of property names changed to lowercase
15-June-1998: Major rewrite and reorganization
16-February-1998: Editorial cleanup, prep for second public distribution
6-February-1998: Editorial cleanup, add and revise some examples
11-January-1998: Renaming and collapsing of several elements
14-November-1997: Further refinement, especially regarding assertions
3-November-1997: Edits in preparation for second public distribution
2-October-1997: First public draft
1-October-1997: Edits in preparation for first public distribution
1-August-1997: First draft to Working Group

Last updated: $Date: 1999/02/24 14:45:07 $